

SDB - Simple Relational Database System
A User Guide

by David Betz
114 Davenport Ave.
Manchester, NH 03103

revised: Fall 1990

Robert C. Beckinger
John L. Pfaltz
Dept. of Computer Science
University of Virginia

1. Introduction

SDB is a simple database manager for small systems. It was developed to provide a relatively low overhead system for storing data on machines with limited disk and memory resources. The current version runs on a PDT-11/150 with 2 RX01 floppy disk drives and 60K bytes of memory under the RT-11 operating system. The version, currently used by the Department of Computer Science at the University of Virginia, is coded in C and runs in a Unix environment. (It also runs on the VAX under VMS.) Note, that SDB is not a "toy system". It has demonstrated good performance with relations of 50,000 tuples.

SDB was originally intended to be a relational database system, so many of the terms used in describing it are taken from the relational database literature. Within the context of SDB the user can safely make the following associations:

relation can be taken to mean *file*,
tuple can be taken to mean *record*, and
attribute can be taken to mean *field*.

It should be noted that SDB is not a relationally complete system. It provides the relational operations of *select*, *project*, *join*, *union*, and *difference*, but does not provide the set operation of *intersection*. The *intersection* may be derived, however, by performing two sequential *difference* operations.

2. Relation File Formats

SDB maintains a separate file (with '.sdb' suffix) for each relation that the user creates. This file contains a header block containing the definition of the relation including the names and types of all of the relation's attributes. The remainder of the file contains fixed length records each containing one tuple from the relation.

Tuples can be of three types:

active - tuples that contain actual active data
deleted - tuples that have been deleted
unused - tuples that haven't been used yet

Initially, all tuples are unused. When a new tuple is stored into a relation, the first unused tuple is found (they are all contiguous at the end of the relation file). The new tuple is stored as an active tuple.

When a tuple is deleted, it is marked as such. The space previously allocated to the deleted tuple is left unused until the relation is compressed.

It is possible that when attempting to store a new tuple, no unused tuple can be found even though the relation contains fewer than the maximum active tuples. This happens when tuples have been deleted since the time the relation file was last compressed.

The compress function allows all of the space lost by deleting tuples to be regained. It does this by copying all of the active tuples as far backward in the file as possible leaving all of the available space toward the end of the file.

3. Selection Expressions

A *selection expression* specifies a set of tuples over which some SDB operation is to be executed. The syntax for a selection expression is:

`<tse>` ::= `<r_names>` [where `<boolean>`]
`<r_names>` ::= `<rel_name>` [, `<rel_name>`] ...
`<rel_name>` ::= `<relation-name>` [`<alias>`]

Tuple selection expressions `<tse>` are used repeatedly in SDB; they are the basic mechanism for specifying subsets of tuples to be used in various operations.

When a single relation name is specified in a selection expression, each tuple within that relation becomes a candidate for selection.

When more than one relation name is specified, the tuples are formed by taking the cross product of all specified relations. If a relation is to be crossed with itself, an alias must be given to one or both of the occurrences of that relation name in the selection expression. This allows SDB to determine which relation occurrence is being referred to in the boolean part of the selection expression. However, selection from multiple relations is *not* recommended.

After the set of candidate tuples is determined, the boolean expression is evaluated for each candidate. The candidates for which the boolean expression evaluates to TRUE become the selected tuples.

Whenever a file name is allowed in the syntax for a command, it is possible to use either a SDB relation identifier or a quoted string. An identifier is interpreted as the (.sdb) file name; while a string is interpreted as a full file specification. The string form allows for the specification of an alternate device or extension.

4. Attribute, Relation Names, and Aliases

When a relation or attribute name is specified in statement, it is possible to provide an alternate name for that relation or attribute. This can be useful in relational operations when it is necessary to join a relation to itself. It is also useful for attributes when it is desired that the column headers in a table be different from the actual attribute names.

The syntax for specifying aliases is:

```
<name> ( <alias> )
```

Alternatively, attributes can be *renamed* as described later. Attribute names can also be qualified with a relation name, as in

```
<relation>.<attribute>
```

This is important when selecting tuples from multiple relations. Note that the period has a special meaning. Neither relation names nor attribute names can contain embedded periods.

5. Boolean Expressions

The syntax for boolean expressions is:

```
<expr>      ::= <land> [ ']' <land> ]
<land>      ::= <relat> [ '&' <relat> ]
<relat>     ::= <factor> [ <relop> <factor> ]
<factor>    ::= <operand> | '(' <expr> ')' | '~' '(' <expr> ')'
<operand>   ::= <number> | <string> | <attribute>
<attribute> ::= [ <rel_name> . ] <aname>
<relop>     ::= '=' | '<' | '>' | '<=' | '>='
```

where the semantics of the relational operators, <relop>, are

```
=      - equal
<>    - not equal
<      - less than
>      - greater than
<=    - less than or equal
>=    - greater than or equal
```

of the logical operators are

& - logical and
 | - logical or
 ~ - logical not

and the operands <operand> are

<number> - a literal string of digits containing at most one decimal point,
 <string> - a literal string of characters enclosed in double quotes, or
 <attribute> - an attribute name optionally qualified by a relation name.

6. Interactive Commands

The primary purpose of SDB is to create an interactive environment in which the user can create relations, insert tuples, select tuples, display subsets of relations and perform the basic relational operations on these relations.

In this mode, SDB issues the prompt

SDB>

to indicated it is waiting for an interactive command. If some string of characters, followed by a <CR>, is accepted, but SDB can not unambiguously recognize it as a complete command, it issues a secondary prompt

>

This allows multi-line input of commands. But it also can be confusing at first because some interactive commands must be terminated with a ; (semi-colon), while others need not. If the secondary prompt appears when you think the command is complete just enter a terminating ; .

In many of the following commands, we have inserted commas, for clarity. Commas are completely optional and are equivalent to blanks; as are tabs, new_lines, and other *white space*.

6.1. Basic Relation Management

In this section we describe those basic operations necessary to manage relations. They are: relation creation (definition); tuple insertion, update, and deletion; import and export of relations; and relation compression.

create

Create a relation file.

Format: create <rel_name> (<alist>) <max_size>

Rules:

1. <rel_name> is the name of the relation file
2. <alist> is a list of attribute definitions of the form:

<aname> { char | int | real } <bytes>

where:

<aname> is the name of the attribute,
 the type of the attribute is either *char int* or *real* and
 <bytes> is the number of bytes allocated to the attribute value.
3. <max_size> is the maximum number of tuples the file can represent.

Examples:

```
create films
(
  title char 30
  year int 4
  studio char 10
  type char 10
  color char 3
) 100

create parts
(
  name char 20
  title char 30
  year int 4
  role char 20
) 200
```

The first command creates a relation file named *films.sdb* with attributes *title*, *year*, *studio*, *type*, and *color* and space to store 100 tuples. The second command creates a similar relation file named *parts.sdb* with attributes *name*, *title*, *year*, *role* and space to store 200 tuples.

Note that, (1) the suffix *.sdb* is appended to all relation (file) names, and (2) attempts to create over existing sdb files will not be prevented.

These two relations, *films* and *parts* will be used as a running example throughout this manual.

insert

Accept tuple input from the terminal and insert into a relation. (Use *import* to insert tuples from a file.)

Format: insert <rel_name>

Rules:

1. <rel_name> is the name of a relation, a *.sdb* suffix is assumed;
2. the user will be prompted for the values of the attributes for the tuple to be inserted;
3. a null response to an attribute prompt will terminate tuple entry.

delete

Delete tuples from a set of relations.

Format: delete <tse> ;

Rules:

1. <tse> is a tuple selection expression;
2. selected tuples are deleted.

Example:

```
delete films where year = 1951 ;
```

update

Update the values of selected attributes in selected tuples.

Format: update { <attrs> | * } from <r_name> where <tse> ;

Rules:

1. <attrs> is a list of attribute names to be updated;
2. * means all attributes;
3. <tse> is a tuple selection expression;
4. for each set of selected tuples, the user is prompted for new values for the selected attributes;
5. a null response to an attribute prompt will retain the previous attribute value;
6. if a null value is desired, a single space can be entered;
7. values exceeding authorized size (as per create) will be truncated, and a warning to that effect posted;

Example:

```
update name from parts where name = "Casablanca" ;
```

import

Import tuples from a file into a relation. This is equivalent to a *read_file* operation.

Format: import <file_name> into <rel_name>

Rules:

1. <file_name> is the name of the input file, a *.dat* suffix is assumed;
2. the input file contains the values of the tuple attributes with each on a separate line;
3. <rel_name> is the name of a relation;
4. tuples are appended to the named relation.

export

Export tuples from a relation into a file. This is equivalent to a *write_file* operation.

Format: export <rel_name> [into <file_name>] ;

Rules:

1. <rel_name> is the name of a relation;
2. <file_name> is the name of the output file, to which a *.dat* suffix will be appended;
3. if the output file name is omitted, output is to the terminal;
4. tuples are written to the output file with one attribute value per line.

Example:

```
export films into films ;
```

will write the file *films.dat*. Note that the (*.dat*) suffix is not specified on the command line, and that if *films.dat* already exists it will be overwritten, not appended.

extract

Extract the definition of a relation into a file.

Format: extract <rel_name> [into <file_name>] ;

Rules:

1. <rel_name> is the name of a relation;
2. <file_name> is the name of the output file, to which a *.def* suffix will be appended;
3. if the output file name is omitted, output is to the terminal;
4. the definition of the relation is written to the output file.

compress

Compress a relation file.

Format: compress <rel_name>

Rules:

1. <rel_name> is the name of a relation file;
2. tuples are copied toward the front of the relation file such that any space freed by previously deleted tuples becomes adjacent to the free space at the end of the file, thus becoming available for use in inserting new tuples.

rename

Rename a relation or an attribute within a relation.

Format: rename <r_name1> into <r_name2>
 rename <rel_name>.<a_name1> into <a_name2>

Rules:

1. <r_name1> and <rel_name> are the names of existing relations. The relation named <r_name1> is renamed to <r_name2>.
2. <a_name1> is the name of an attribute in relation with name <r_name>. <a_name1> is changed to <a_name2> inside <rel_name>.
- 3.

Names of more than 8 characters cannot be renamed.

Example:

```
rename films into flix
rename parts.name into star
```

remove

Remove an entire relation file from SDB.

Format: remove <rel_name>

Rules:

1. <rel_name> is the name of a relation file;
2. Normally, <rel_name> denotes a relation (and corresponding *.sdb* file) in the current directory, but it can be an arbitrary path name.

6.2. Display Functions

SDB has several features designed to provide a "nice" display of relational information. The *sort* operator can be used to put tuples into a more readable sort order. The basic operator is *print* which will display the tuples of a relation in table form to either the terminal or to a (.txt) file.

The *print* operator can also employ a *form definition* file (.frm) to format the data with interspersed literal

text. A *form definition* file contains a template into which attribute values are substituted during a print operation. There are two types of information that can be included in a form definition:

1. Literal text
2. Attribute references

Attribute references are indicated by placing the name of the attribute being referenced between a pair of angle brackets. Literal text is anything that is not enclosed in angle brackets.

Example:

```
print using starred title, role, name, year from parts ;
```

where *starred.frm* contains:

```
In <year>, <name> starred
as <role> in <title>.
```

This feature can be useful in developing 'form letters'; however, the entire declared length of *char* attributes are displayed yielding rather ugly output unless the attribute is carefully placed at the end of a line.

Alias attribute names can also be used in references to that attribute in the where clause as well as in a form definition file, as in:

```
print using starred title (a), role (b),
name (c), year (d) from parts ;
```

where *starred.frm* contains

```
In <d>, <c> starred
as <b> in <a>.
```

print

Print a table of values of selected attributes.

Format: `print [using <form_name>] { <attrs> | * } from <tse> [into <file_name>] ;`

Rules:

1. using <form_name> indicates output using a form definition file (.frm);
2. <attrs> is a list of attribute names to be printed;
3. * means all attributes;
4. <tse> is a tuple selection expression;
5. <file_name> is the name of an file to which the table will be output (.txt);
6. if the output file name is omitted, output is to the terminal;
7. for each set of selected tuples, a table entry is printed containing the selected attributes.

Examples:

```
print * from films ;
print title, year from films
      where studio = "MGM" ;
print * from parts
      where year < 1940 ;
```

The first command would display

title	year	studio	type	color
The African Queen	1951	UA	romance	COL
All About Eve	1950	Fox	drama	BW
The Awful Truth	1938	Columbia	comedy	BW
Ball of Fire	1941	RKO	comedy	BW
The Big Sleep	1946	Warner	mystery	BW
Born Yesterday	1950	Columbia	comedy	BW
Camille	1936	MGM	romance	BW
Casablanca	1942	Warner	romance	BW
Dinner at Eight	1933	MGM	drama	BW
You Can't Take It With You	1938	Columbia	comedy	BW
High Society	1956	MGM	musical	COL

The second command will display

title	year
Camille	1936
Dinner at Eight	1933
High Society	1956

While the last command would display

name	title	year	role
Irene Dunne	The Awful Truth	1938	Lucy Warriner
Cary Grant	The Awful Truth	1938	Jerry Warriner
Greta Garbo	Camille	1936	Marguerite Gautier
Robert Taylor	Camille	1936	Armand
Jean Arthur	You Can't Take It With You	1938	Alice Sycamore
James Stewart	You Can't Take It With You	1938	Tony Kirby

sort

Sort a relation file.

Format: sort <rel_name> by <s_attr> { , <s_attr> } ... [into <rel_name>] ;

Rules:

1. <rel_name> is the name of a relation file;
2. <s_attr> is the name of an attribute to sort on followed optionally by "ascending" or "descending";
3. if a sort order is not specified, ascending is assumed;
4. tuples within the relation are sorted in place using the attributes indicated;
5. if the optional 'into' clause is used, the result of the sort can be directed into a different relation file while leaving the original file undisturbed.

6.3. Relational Algebra Operators

This section describes the standard operators of the relational algebra that are defined in SDB. As noted before, SDB is not a relationally complete system. It provides the relational operations of *select*, *project*, *join*, *union*, and *difference*, but does not provide the set operation of *intersection*. The *intersection* may be derived, however, by performing two sequential *difference* operations. (I.e., the *intersection* of A and B can be computed by *diffing* A and B, and then *diffing* A with the result of the initial *diff*.)

Note that all algebraic operators have the basic structure

<operator> <rel_name_1>, <rel_name_2> [into <rel_name_3>] ;

where the result relation `<rel_name_3>` will be the SDB default relation `sdbcur` if the `into` clause is omitted. However, we do not recommend using `sdbcur`; because once defined, like any other relation, it can not be redefined.

select

Select attributes from one or more relations and put the resulting cross product into the default relation file `sdbcur`.

Format: `select { <attrs> | * } from <tse> [into <rel_name>] ;`

Rules:

1. `<attrs>` is a list of attribute names to be selected;
2. `*` means resulting file will contain all attributes;
3. `<tse>` is a tuple selection expression;
4. if more than one relation is specified and there is a common attribute then the attribute must be qualified with the relation name (see example 2) and given an alias;
5. if an attribute only exists in one of the relations it doesn't need to be qualified with a relation name;
6. if the optional 'into' clause is used, the result of the SELECT will appear in the specified relation; otherwise, the default relation `sdbcur` will be the relation that results.

Example:

```
select * from films where studio = "MGM"
      into MGM_films ;

select title, year from parts
      where year < 1940 & ~(role = "Armand")
      into early_films

select name, films.title, films.year, role
from   films, parts
where  films.title = parts.title &
      films.year  = parts.year  &
      studio = "MGM"
into   MGM_stars
```

select does not eliminate duplicate tuples, as *project* does. The second example will have many duplicate tuples.

The third example illustrates that SDB is a superset of the basic SQL language. Here we have a classic project-select-join query in which the join operator is implemented by the tuple selection expression which forces equality between common attributes.

Such multi-relation selection is relatively slow; one should first join the two relations, then select, and then project (if necessary to eliminate duplicate tuples).

Notice the use of `<rel>.<attr>` to qualify an attribute name with a relation name. It is necessary in multi-relation selects.

project

Project attributes from one relation and put the result into the named relation.

Format: `project { <attrs> | * } from <r_name1> [into <r_name2>] ;`

Rules:

1. <attrs> is a list of attribute names to be selected;
2. * means resulting file will contain all attributes;
3. <r_name1> is a relation (source or input relation);
4. <r_name2> is the name of a relation (result relation);
5. a tuple will not be appended if an equal tuple has already been appended (duplication is avoided);
6. if the into clause is omitted, *sdbcure* will be the relation in which the result occurs.

Example:

```
project names, year from parts ;
```

Note that the *project* guarantees there will be no duplicate tuples in its result, whereas *select* does not.

join

Join tuples from two relations into a third relation. This is the natural join. The resulting relation will have the attributes which appear in either of the relations being joined.

Format: `join <r_name1>, <r_name2> into <r_name3> ;`

Rules:

1. <r_name1>, <r_name2>, and <r_name3> are the names of relations;
2. only two input relations may be specified, and should have at least one common attribute;
3. tuples are appended to the named relation, if the values for all common attributes in one relation are equal to the values in the other relation.

Example:

```
join films, parts into filmparts ;
```

Be careful when joining over string attributes. SDB properly handles string attributes of different lengths; but it is case-sensitive.

Since this is a *natural* join, its behavior can be altered by renaming attributes in either of the argument relations.

diff

Diff two relations into a third. This is the relative complement operator, that puts those tuples of the first relation which do not exist in a second relation, into a third relation.

Format: `diff <r_name1>, <r_name2> [into <r_name3>] ;`

Rules:

1. <r_name1>, <r_name2>, and <r_name3> are the names of relations;
2. tuples are appended to the named relation;

3. only two input relations may be specified, and the schema of each must consist of attributes with identical domains (type and length), but need not have the same names;
4. if the 'into' clause is omitted, *sdbcur* will be the relation, resulting from the *diff*.

Example:

```
diff parts, MGM_stars into other_parts
```

union

Union one relation with another. Put the tuples of one relation and the tuples of a second relation, into a third relation.

Format: union <r_name1>, <r_name2> [into <r_name3>]

Rules:

1. <r_name1>, <r_name2>, and <r_name3> are the names of relations;
2. tuples are appended to the named relation;
3. only two input relations may be specified, and the schema of each must consist of attributes with identical domains (type and length), but need not have the same names;
4. if the same tuple appears in both input relations, it appears only once in the output relation (duplication is avoided);
5. if the optional 'into' clause is omitted, *sdbcur* will be the relation, resulting from the union.

Example:

```
union MGM_stars, other_stars into all_stars ;
```

6.4. Aggregation Operators

Many relational systems include a class of aggregation operators which perform simple statistical operations on relations or their attributes. SDB supports *sum*, *max*, *min*, and *avg* over attributes and *count* over relations.

sum

Sum the values of an attribute.

Format: sum <aname> from <rel_name> [where clause] ;

Rules:

1. <aname> is the name of an attribute in the <rel_name> relation;
2. only attributes of types real or int may be summed;
3. the optional where clause can use attributes different from <aname> for selection criteria;
4. output is to the screen.

Example:

```
sum attr_1 from rel_1 ;
sum attr_1 from rel_1 where attr_2 > 27 ;
```

max

max: return the maximum value entered for an attribute.

Format: max <aname> from <rel_name> [where clause] ;

Rules:

1. <aname> is the name of an attribute in the <rel_name> relation;
2. only attributes of types real or int may have the max operation applied to them;
3. the optional where clause can use attributes different from <aname> for selection criteria;
4. output is to the screen.

Example:

```
max year from films ;
max attr_1 from rel_1 where attr_2 > 2.7 ;
```

min

min: return the minimum value entered for an attribute.

Format: min <aname> from <rel_name> [where clause] ;

Rules:

1. <aname> is the name of an attribute in the <rel_name> relation;
2. only attributes of types real or int may have the min operation applied to them;
3. the optional where clause can use attributes different from <aname> for selection criteria;
4. output is to the screen.

Example:

```
min year from films ;
min year from films where studio = "MGM" ;
```

avg

avg: return the average value entered for an attribute.

Format: avg <aname> from <rel_name> [where clause] ;

Rules:

1. <aname> is the name of an attribute in the <rel_name> relation;
2. only attributes of types real or int may have the avg operation applied to them;
3. the optional where clause can use attributes different from <aname> for selection criteria;
4. output is to the screen.

Example:

```
avg attr_1 from rel_1 ;
avg attr_1 from rel_1 where attr_2 <= 27.23 ;
```

count

count: return count of number of active tuples.

Format: count * from <rel_name> [where clause] ;

Rules:

1. * required to account for all tuples;
2. the optional where clause can use attributes from <rel_name> for selection criteria;

- output is to the screen.

Example:

```
count * from films ;
count * from parts where year >= 1950 ;
```

6.5. Macro Operators and Command Files

Interesting relational queries typically require a sequence of commands. Entering these can become tedious — particularly, if one makes a mistake in the middle. SDB provides a macro/command file facility, in which such a sequence of commands can be defined and then executed by a single macro command. Unfortunately, SDB does not provide for macro parameters which severely limits the utility of this concept. Moreover, macro's can not be edited within SDB, and exist only for the duration of the current execution unless written to a file by a *save* command.

Nevertheless, this facility is an extremely useful one. Long sequences can be defined and tested. If successful they can then be *saved* for reuse. Alternatively they can be coded off-line in (.cmd) files, then tested and run. The *command file* concept effectively provides a programming capability in an interactive environment. We will use this mechanism to submit assignments.

Note that when SDB is first run, it automatically attempts to read and process commands from a file named "*sdb.ini*". This file, if it exists, can contain macro definitions and/or relation definition (creation) operations, or any other valid SDB commands.

define

Define a run-time macro. If the macro exists as a (.cmd) file it will be read in from that file, otherwise SDB will interactively prompt the user for the lines constituting the macro.

Format: define <macro_name>

Rules:

- <macro_name> is the name of the macro being defined;
- if a macro with the specified name already exists, it is replaced;
- after entering the define command, definition mode is entered;
- definition mode is indicated by the prompt "SDB-DEF>";
- all lines typed in definition mode are added to the macro definition;
- a blank line terminates definition mode;
- a macro can be deleted by entering a blank line as the only line in the definition;
- after a macro is defined, every occurrence of the macro name is replaced by the macro definition.

Example:

```
define MGM_parts
  select title, year from films
     where studio = "MGM"
     into temp1 ;
  join temp1, parts into temp2 ;
  sort temp2 by year, title ;
  print year, title, name, role from temp2 ;
```

save

Save a run-time macro by writing it to a (.cmd) file. Also, show, or display, the macro.

Format: save <macro_name>

Rules:

1. <macro_name> is the name of a macro whose definition is to be saved;
2. a file '<macro_name>.cmd' is written in the current directory; if the file already exists, it is overwritten.

Example:

```
save  MGM_parts
```

command_file

Execute the named command file

Format: @<command_file_name>

Example:

```
@MGM_parts.cmd
```

will generate the output

title	year	name	role
Dinner At Eight	1933	Marie Dressler	Carlotta Vance
Dinner At Eight	1933	John Barrymore	Larry Renault
Dinner At Eight	1933	Jean Harlow	Kitty Packer
Camille	1936	Greta Garbo	Marguerite Gautier
Camille	1936	Robert Taylor	Armand
High Society	1956	Bing Crosby	C. K. Dexter Haven
High Society	1956	Grace Kelly	Tracy Lord
High Society	1956	Frank Sinatra	Mike Connor

Note, if the macro has been defined as a run-time macro, one can omit the leading @ and .cmd suffix. Just the <macro_name> is sufficient, as in

```
MGM_parts
```

6.6. Miscellaneous Commands**exit**

Exit from SDB.

Format: exit

schema

Display the schema of all relations on the terminal, in the format:

```
<rel_name> ( <attr1>, <attr2>, ... <attrn> ) <max_size>
```

Format: schema

Rules:

1. Only relation files with the (.sdb) suffix are displayed.

help

Print a short help message. Note that the file 'sdb.hlp' must be in the current directory. Even if it is, its not much help!

Format: help

comments

Interpret the remainder of the line as a comment

Format: comment <text of comment>

Rules:

1. The <text of comment> is displayed at the terminal, preceded by
2. Every comment line must be preceded by the 'comment' key word, much in the style of Fortran comments.

This option is primarily of value only in *command* files and *macro* definitions to inform the user of what is taking place.

escape

Escape from SDB into UNIX.

Format: ! "<shell command>"

A single <shell command> can be given; but it must be enclosed in double quotations. This can be useful to list available relations, using

```
! "ls *sdb".
```

or to clean up temporary relations with

```
! "rm temp*sdb"
```

set

Set various internal parameters.

Format: set page <lines_per_page>
 set fold
 set no fold

Only these three set commands are currently operative. The first establishes the (integer) number of screen display lines per page. Default is 16. For certain string comparisons, upper case are folded onto lower case. Default is "fold".

7. Important System Limits

The following are a few of the known system limits:

relation names	≤ 10
attribute names	≤ 10
number of attributes	≤ 31
string length	≤ 132
tuple length	≤ 256

Others may be uncovered through use.

8. Summary of File Suffixes

SDB interacts with a variety of files in the user's directory. Different suffixes have different meanings to SDB; they are

<code>.sdb</code>	relation (together with its schema);
<code>.dat</code>	relation data in import/export form (1 value per line);
<code>.txt</code>	relation data in tabular (display) form;
<code>.def</code>	relation schema in "create" form;
<code>.cmd</code>	macro definition and/or command file.

While it is customary to suffix command files with `.cmd`, it is not strictly necessary. All other suffixes are mandatory.

9. Program Interface

SDB provides a callable program interface to allow programs written in DECUS-C to access relation files. In order to use the call interface, the users program should be linked with the SDBUSR.OBJ object library. Also, additional stack space should be allocated at link time using the `/BOTTOM` qualifier on the link command. `/BOTTOM:3000` seems to work well, but it is probably possible to get away with

less. Example:

```
#include <stdio.h>
#include "sdbio.h"
#include "globals.h"

main ()
{
    SEL    *sptr;
    char   title[100], studio[100];

    /* setup retrieval */
    if ((sptr = (SEL *) db_retrieve ("films where year > 1940"))
        == NULL)
    {
        printf ("*** error: %s ***\n", db_ertxt(dbv_errcode));
        exit ();
    }

    /* bind user variables to attributes */
    db_bind (sptr, "films", "title", title);
    db_bind (sptr, "films", "studio", studio);

    /* loop through selection */
    while (db_fetch(sptr))
        printf ("%s  %s\n", title, studio);

    /* finish selection */
    db_done (sptr);
}
```

db_retrieve

Setup a tuple retrieval context.

Format: dbptr = db_retrieve (sexpr [,arg]...);

Rules:

1. sexpr is a pointer to a string containing an tse;
2. arg is a "printf" argument;
3. dbptr is a database context pointer;
4. db_retrieve returns NULL on errors;
5. on errors, the error code is in dbv_errcode.

db_fetch

Fetch the next set of tuples from a retrieval context.

Format: db_fetch (dbptr);

Rules:

1. dbptr is a database context pointer;
2. updates the values of all bound user variables;
3. db_fetch returns FALSE if no more tuples match or if an error occurs;
4. on errors, the error code is in dbv_errcode.

db_update

Update the current tuple within a retrieval context.

Format: db_update (dbptr);

Rules:

1. dbptr is a database context pointer;
2. db_update returns FALSE if an error occurs;
3. on errors, the error code is in dbv_errcode.

db_store

Store a new tuple within a retrieval context.

Format: db_store (dbptr);

Rules:

1. dbptr is a database context pointer;
2. db_store returns FALSE if an error occurs;
3. on errors, the error code is in dbv_errcode.

db_bind

Bind a user variable to the value of a tuple attribute within a retrieval context.

Format: db_bind (dbptr, rname, aname, value);

Rules:

1. dbptr is a database context pointer;
2. rname is a pointer to the relation name;
3. aname is a pointer to the attribute name;
4. value is a pointer to a character array to receive the attribute value;
5. db_bind returns FALSE if an error occurs;
6. on errors, the error code is in dbv_errcode.

db_get

Get the value of a tuple attribute within a retrieval context.

Format: db_get (dbptr, rname, aname, value);

Rules:

1. dbptr is a database context pointer;
2. rname is a pointer to the relation name;
3. aname is a pointer to the attribute name;
4. value is a pointer to a character array to receive the attribute value;
5. db_get returns FALSE if an error occurs;
6. on errors, the error code is in dbv_errcode.

db_put

Put the value of a tuple attribute within a retrieval context.

Format: db_put (dbptr, rname, aname, value);

Rules:

1. dbptr is a database context pointer;
2. rname is a pointer to the relation name;
3. aname is a pointer to the attribute name;
4. value is a pointer to the new value;
5. db_put returns FALSE if an error occurs;
6. on errors, the error code is in dbv_errcode.

db_done

Discontinue usage of a retrieval context.

Format: db_done (dbptr);

Rules:

1. dbptr is a database context pointer.

db_ertxt

Translate an error code to an error message text.

Format: db_ertxt (errcode);

Rules:

1. errcode is an SDB error code;
2. db_ertxt returns a pointer to the error message text.

1. Introduction	1
2. Relation File Formats	1
3. Selection Expressions	1
4. Attribute, Relation Names, and Aliases	2
5. Boolean Expressions	2
6. Interactive Commands	3
6.1. Basic Relation Management	3
create	3
insert	4
delete	4
update	5
import	5
export	5
extract	5
compress	6
rename	6
remove	6
6.2. Display Functions	6
print	7
sort	8
6.3. Relational Algebra Operators	8
select	9
project	10
join	10
diff	10
union	11
6.4. Aggregation Operators	11
sum	11
max	11
min	12
avg	12
count	12
6.5. Macro Operators and Command Files	13
define	13
save	14
command_file	14
6.6. Miscellaneous Commands	14
exit	14
schema	14
help	15
comments	15
escape	15
set	15

7. Important System Limits	15
8. Summary of File Suffixes	16
9. Program Interface	16
db_retrieve	17
db_fetch	17
db_update	18
db_store	18
db_bind	18
db_get	18
db_put	18
db_done	19
db_ertxt	19