Predefined Stored Commands Version R3v5m4
Release/Installation Notes For Unix BSD 4.2/4.3

Product Format

The Predefined Stored Commands are supplied on 9-track tape at 1600 bpi in tar format. They consist of 79 files that contain the command scripts to be installed in any Britton Lee Shared Database system. The commands are described in the document *Predefined Stored Commands*, Britton Lee part number 205-1607-rev. This document also includes a listing of the individual stored command scripts. The command scripts come in two versions, one for IDL query language and one for SQL. Only one of these sets need be installed in a database to be useable from both query languages.

Installing from Tape

Create a directory called scmds in the desired path. Generally, these commands should be accessible to the DBA for the database machine. It may be most convenient to create the directory /a/host/scmds. On release 3.4 or later of the Britton Lee host software, the command script files will be loaded into a directory /a/host/scmds automatically. From inside the directory /scmds, type:

```
tar x
```

There should be 79 files loaded from tape into this directory.

Loading the Commands into a Database

The Commands may loaded into a databse by using the command script "loadcmds" found in the directory. Any number of database names may follow the command. For instance, to load the stored commands into databases "mydb" and "db2", type the following:

```
loadcmds mydb db2
```

To install the same commands in the SQL form into the same databases, type:

```
loadcmds -s mydb db2
```

To install the system database commands simply type:

```
loadcmds system
```

The installation using the loadcmds script may also reference a database port (IDMDEV) by using the -B flag with the name of the device.

```
loadcmds -Bt3%xns mydb
```

This command will load the stored commands in database "mydb" on the database found through device address "t3%xns". There must be no space between the "-B" and the device identifier.

The command script will run for about five minutes, with frequent messages of the variety "xxx not found" and "n tuples affected". These messages are normal for a first installation. Read the Predefined Stored Commands document Introduction to find out how to avoid name conflicts, or how to install only some of the commands in a database.

Editing Control Files

There are a set of files in the directory with the suffix ".iin". These files are control files used to load the commands scripts for IDL. The actual command scripts are suffixed ".idl". You may edit any file in this directory with any Unix editor. You may wish to edit the "perms.iin" file, since it contains the default execute permissions for the commands. Similarly, all files suffixed ".sin" will control the loading of SQL scripts (suffix: ".sql").

Britton Lee Host Software

PREDEFINED STORED COMMANDS

(r3v5)

March 1988
Part Number 205-1607-002

Printed February 1988.

This document supersedes all previous documents. This edition is intended for use with Britton Lee Host Software Release 3.5 and future software releases, until further notice.

The information contained within this document is subject to change without notice. Britton Lee assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under license and may only be used or copied by the terms of such license.

Table of Contents

PART I

Overview

Introduction

The stored commands described in this manual are delivered with Britton Lee Release 3.5 Host Software. They are designed to facilitate the use of the BL300 and BL700 system families through the interactive query languages IDL and SQL. The commands are contained in machine- and human-readable IDL and SQL scripts in a host directory. The DBA of a database may choose to install either the full set of commands or any subset.

The stored commands are described on-line by the informational stored commands **cmds**, **cmds_dba**, **cmds_date**, **cmds_permit**, **cmds_space**, and **cmds_system**. These interrogate the "descriptions" relation and print the results in a neat format. Thus, the user need only know the command **cmds** to list the stored commands that describe the database contents. Security considerations may cause the DBA either not to install these commands in a database or to permit their use to only certain groups of users.

The stored commands are contained in a directory or diskette directory called **scmds**. For the location (path name) of this directory on your system, see the associated document *Predefined Stored Commands — Release Notes and Installation Guide* or see your system administrator. The directory **scmds** contains a file for each stored command described in this document. Stored commands that must be loaded together because they share the use of objects are included in common files whose names end with a digit (e.g., **objects7.sql** contains seven stored command scripts for commands using the view, "objects").

There is a two similar sets of stored commands described by the two types of scripts, IDL and SQL. The same functionality is provided by both sets of commands, but the SQL version will use SQL terminolgy while the IDL version uses IDL terminology. Whichever set of commands is loaded, they may be invoked from either language. Two commands have different names. The SQL scripts (**.sql**) will provide a **cols** command that will give the SQL data types and column names for a table, while the IDL version (**.idl**) is called **atts** and will yield IDL data types. It is not necessary to load both sets of scripts into a database for the commands to accessible to both command parsers.

The directory also contains a set of command files labeled **.iin** for IDL and **.sin** for SQL. The use of these files is described below.

How to Load

On any host type, the stored commands may be loaded by entering the IDL or SQL parser, opening the database to be accessed, and typing

%input filename

If just a single command script is desired, the *filename* should be the file containing that script (e.g., **freespc.idl**). When installing all of the stored commands, the special file **input.iin** may be used from IDL and **input.sin** from SQL. By invoking the parser with the name of the desired command script, the script will be entered as if from a **%input** command. This invocation may be of the UNIX variety,

idl dbname **–f input.iin**

or

sql dbname **–f input.sin**

or it may be of the VMS, VM/CMS, or PC MS-DOS variety:

IDL dbname **/INPUT=INPUT.IIN**

or

SQL dbname **/INPUT=INPUT.SIN**

Each of the input command scripts will cause the individual commands to be loaded in the database. If this is the first time these commands are loaded, an error message will appear on the screen for each command, stating that the command name was not found. Each command script attempts to destroy any previously existing object of the same name in the database before definition. If there is no such object, this error message will appear. In addition, several of the helper data relations will be installed and loaded with data. The resulting operations will print quite a few "*n* tuples affected" messages. Since the installation takes several minutes, the DBA may wish to redirect standard output and standard error, or place the job in a batch stream, rather than tie up a terminal for the load time.

A possible side effect of loading these commands into an existing database is a conflict of object names. Objects of the same name as the stored commands will be destroyed by the loading script. To prevent loss of data, another pair of helper scripts, **conflict.iin** or **conflict.sin**, can be run to print any possible name conflicts that would arise from installing the commands in a database. The scripts **sconflct.iin** or **sconflct.sin** perform the same function for the stored commands specific to the system database. To run these scripts, simply invoke them in the database from the IDL or SQL command parsers. A report will be

printed to the screen that contains the names of existing objects that will be destroyed. A conflict can be resolved by either renaming the existing object or changing the script file to alter the name of the command. The file name containing the conflicting command is listed alongside the object name. Simply edit the named file and do a global replace on the name. In most cases this will solve the problem of name conflict.

On each host, there is a sample command script that will load the stored commands on the default shared database system (IDMDEV) into designated databases. This script will appear as the executable file **loadcmds** on UNIX systems, **loadcmds.bat** on PC MS-DOS, **loadcmds.com** on VMS, and **loadcmds exec** on VM/CMS. Each such file will take the names of one or more databases as arguments. In addition, the files on UNIX, VMS and PC-DOS will accept a flag to load the commands from the SQL parser rather than the IDL parser.

For the commands to be loaded in the "system" database, there is a file called **sinput.iin** for IDL and **sinput.sin** for SQL, or these may be loaded by opening the system database and typing

%input "sinput.iin"

or

%input "sinput.sin"

Alternatively, the **loadcmds** command file with the database name "system" added may be used to load the system database specific commands.

There are some special requirements for the system-database commands. First, the system database may need to be extended beyond its default allocation. This means the command

extend database system (IDL)
or
alter database system (SQL)

must be issued in the system database. Second, there must be an "M" tuple in the "configure" system relation for the **mon*zzz*** commands to function properly.

Permissions

A separate file contains the permissions for each command. This file may be modified to restrict the usage of each command. The **input.iin** and **sinput.iin** files also execute the permissions file, so this file should be edited appropriately before being loaded. Similarly, the file may be executed by itself via the **%input** facility in each parser. The file names are **perms.iin** and **sperms.iin** for the IDL parser, and **grants.sin** and **sgrants.sin** for the SQL parser.

There are also separate files that contain the commands to destroy all of the stored commands and related objects in a database or the system database. These files are called **destroy.iin** and **sdestroy.iin** for the IDL parser, and **drop.sin** and **sdrop.sin** for the SQL parser.

Executing the Stored Commands

The stored commands may be executed during an interactive or batch session through the IDL or SQL parsers. They may also be executed from application programs and used as background queries for screen-based application programs such as Freeform. In IDL, one invokes a stored command by either issuing an **execute** command or naming the command on the first line of a prompt. Commands that need parameters must be invoked with their parameters listed in order, separated by commas. Sometimes strings must be quoted.

IDL examples for the **atts** command:

```
atts relation;
execute atts relation;
execute atts "relation" go
```

All of the above invocations will yield the correct results. The first form (implicit execution) will only work after a **1)** prompt. All IDL commands must end with a **go** or a semicolon ";" unless a continuation character is specified.

SQL example for the **cols** command:

```
start cols ("table");
```

SQL does not allow implicit invocation. All strings must be enclosed in quotation marks, and the parameter list must be in parentheses.

In the examples, the implicit IDL form is given for most commands.

The Stored Commands

The commands are divided into roughly three groups:

(1) Commands that are useful to all users. These are generally informational in nature. None of them permit users to modify structures except the **rename** command.

(2) Commands that are useful primarily to the DBA. These are both informational and modifying.

(3) Commands that are useful only in the system database.

This manual is divided into parts according to these types. Part II describes the general user commands. Part III describes the DBA commands. Part IV describes the system database commands. An alphabetical listing of all the command scripts can be found in the appendices — Appendix B covers the IDL versions and Appendix C covers the SQL versions. In addition, a display of each of the standard data relations is given in Appendix D. These relations are used by the stored commands to decode various system relations. They are also available for user applications.

Each part contains an alphabetical listing of commands. For each command, there is a description of the purpose of the command, its usage, and an example.

Several of the stored commands make use of relations and views with data tables to aid in decoding system relations. To help distinguish these from the system relations and user relations or views, the names always end in "_I", and the types in the "relation" relation are changed to type "I" for relations. A listing of these relations can be found in Appendix D.

The stored commands operate best if certain standards are maintained. There should be only one user mapped to DBA for each database. There should be corresponding entries for all users in the "host_users" and "users" system relations. There should be only one user 0 named ALL in any database. To help maintain these standards, there is a stored command for DBAs, **usersright**, which will report any violations.

PART II

Stored Commands — All Users

Summary

These are stored commands that are useful to all users:

atts	Displays attributes and types for given ***relation*** (IDL only).
base1900	Changes base date for date conversion.
base1970	Changes base date for date conversion.
cmds	Lists predefined stored commands with one-line descriptions.
cmds_date	Lists stored commands dealing with date conversions.
cmds_permit	Lists stored commands dealing with protection of objects.
cmds_space	Lists stored commands dealing with space/size/storage.
cols	Displays the column names and types for a given ***table*** (SQL only).
date	Returns the current date and time from the relational system clock.
dateconv	Returns a readable date given an ***idmdate*** value.
depend	Displays objects depending on ***object***.
describe	Displays description of given ***relation*** or ***command***.
dir	Displays all objects of given ***name*** or fragment.
expire	Sets the expiration date for an ***object*** to current + ***ndays***.
expiredate	Sets the expiration date for an ***object*** to ***yyyymmdd***.
files	Displays files owned by you and the DBA.
freespace	Displays total space and free space in database.
gmt_date	Returns Greenwich mean time and date from shared database system.
mine	Displays objects owned by you only.
othercmds	Displays user stored commands (those not displayed by **cmds**).
otherviews	Displays user views (those not displayed by **views**).
permits	Shows the explicit protection for an ***object***.
permitsall	Shows all explicit permits and denies.
permitsgen	Shows the explicit tape and create permissions.
permitsme	Shows the explicit permissions for me.
permitsuser	Shows the explicit permissions for ***user_name***.
pgms	Displays stored programs owned by you or the DBA.
rels	Displays user relations owned by you or the DBA (IDL only).
rename	Allows users to rename their ***relation*** to ***new_name***.
size	Displays size of ***object***.
sizebyzone	For ***object***, lists number of blocks in each zone.
sizes	Displays size of database and larger relations.
spacebyuser	Lists disk space utilization by user.
tabs	Displays user tables owned by you or the DBA (SQL only).
views	Lists standard views with a one-line description.
uses	Lists relations or views used by a stored command.
ymd	Takes a ***yyyymmdd*** date string, returns ***idmdate***.

attname relation

DESCRIPTION

This is the stored command used by early releases of IDL to return attribute names for **.all** in an attribute list. It is not needed with release 25 or later IDM/RDBMS code. It may be used by IDL users to return a list of attribute names, but the **atts** command is preferable.

EXAMPLE

1) attname host_users;

name
s1
hid
huid
uid

4 tuples affected

atts relation

DESCRIPTION

This command returns both the attribute names and the attribute types for the named *relation*. It makes use of the view "columns" which contains all of the information from the system relation "attributes" in decoded form. The decoding information is contained in a relation called "dtype_I" which is used by the "columns" view.

This command is only installed by IDL. It is equivalent to the "cols" stored command installed by SQL. The data types are named differently in the two languages.

EXAMPLE

1) atts host_users;

attribute	type
s1	i1
hid	i2
huid	i4
uid	i2

4 tuples affected

base1970
base1900

DESCRIPTION

These commands convert the base date for other date-related stored commands from January 1, 1900 to January 1, 1970 and back. When the stored commands are loaded, the default base date 1900 is used in the table "calendar_l". The command **base1970** will convert this table and all date functions that depend on it to a system that assumes dates are stored as number of days since January 1, 1970. Similarly, **base1900** will convert the table back to 1900 base date. The text of these commands can be found in Appendices B and C under the **date** command script. If some other base date is needed, it may be useful to look at those scripts.

cmds

DESCRIPTION

This command is the **help** command replacement. It lists the highest level of predefined stored commands. Each of the commands has a description, type, and key in the "descriptions" system relation, allowing the **cmds** command to distinguish it from other objects. At the end of each of the scripts, you will find an **associate** command with the one-line description and keys necessary to place this command in the proper category.

EXAMPLE

1) cmds;

command	description
atts	Displays attributes and types for given RELATION
cmds	Lists standard stored commands with a one line description
cmds_date	Lists stored commands relating to date conversion
cmds_dba	Lists stored commands mostly used by the dba
cmds_permit	Lists stored commands dealing protection of objects
cmds_space	Lists stored commands dealing with space/size/storage
depend	Displays objects depending on OBJECT
describe	Displays description of given RELATION or COMMAND
dir	Displays all objects of given NAME or fragment
files	Displays files owned by you and the dba
mine	Displays all objects owned by this user
othercmds	Displays user stored commands (those not displayed by cmds)
otherviews	Displays user views (those not displayed by views)
pgms	Displays stored programs owned by you or the dba
rels	Displays user relations owned by you or the dba
rename	Allows users to rename their RELATION to NEW_NAME
uses	Lists objects that OBJECT depends on
views	Lists standard views with a one line description

17 tuples affected

cmds_date

This is a version of the **cmds** command for those predefined commands that relate to date conversion. The Britton Lee shared database system stores dates as 4-byte integers. There are several commands and views that convert between this format and various human-readable forms. Two commands are provided to allow the user to condition the "expire" attribute in the "relation" system relation with friendly dates.

EXAMPLE

1) cmds_date;

command	description
base1900	Change base for date conversion to Jan 1,1900 from 1970
base1970	Change base for date conversion to Jan 1,1970
date	Returns the current date and time from the database server clock
dateconv	Returns the month day, year given idmdate NUMBER
expire	Sets the expiration date for an OBJECT to getdate + NDAYS
expiredate	Sets the expiration date for an OBJECT to date YYYYMMDD
gmt_date	Returns Greenwich Mean Time and Date from the database server
ymd	Returns the idmdate given a date string: YYYYMMDD

8 tuples affected

cmds_permit

DESCRIPTION

This is a version of the **cmds** command for those predefined commands that show non-default permissions in the database. There are five such commands. It may be desirable to restrict some of these to the DBA and privileged users.

EXAMPLE

1) cmds_permit;

command	description
permits	Shows the explicit protection for an OBJECT
permitsall	Shows all explicit permits and denies
permitsgen	Shows the explicit tape and create permissions
permitsme	Shows the explicit permissions for me
permitsuser	Shows the explicit permissions for USER

5 tuples affected

cmds_space

DESCRIPTION

There are so many commands related to space usage that they have been given a menu of their own. These commands are of particular interest to the DBA, but they may also be useful to the general user.

EXAMPLE

1) cmds_space;

command	description
freelog	Displays space usage for hard allocated transact log
freespace	Displays total space and free space in this database
size	Displays size of RELATION
sizebyzone	For OBJECT, lists number of blocks in each zone
sizes	Displays size of this database and its larger relations
spacebyuser	Lists disk space utilization by user

5 tuples affected

cols table

DESCRIPTION

This command returns both the column names and the column types for the named *table*. It makes use of the view "columns" which contains all of the information from the system table "attributes" in decoded form. The decoding information is contained in a table called "dtype_SQL" which is used by the "columns" view.

This command is only installed by SQL. It is equivalent to the "atts" stored command installed by IDL. The data types are named differently in the two languages.

EXAMPLE

SQL version:

1) start cols("host_users");

col_name	type
s1	**tinyint (1)**
hid	**smallint (2)**
huid	**integer (4)**
uid	**smallint (2)**

4 rows affected

date

DESCRIPTION

The **date** command prints the date and time in a neatly formatted form. Information is derived from the **getdate** and **gettime** functions of Britton Lee's IDM/RDBMS.

Since the relational system hardware is not user-programmable, the tests normally used to convert from Julian date and ticks (1/60ths of a second) to human-readable dates are not available. Instead, relations are used to aid the conversions.

The **date** command and its close relative **gmt_date** make use of a number of such relations for conversions of day of the month, length of the year, and day of the week. The script itself is lengthy and well documented.

Britton Lee's IDM/RDBMS stores dates as 4-byte integers that usually represent the number of days since January 1, 1900. Some systems may choose other base dates (the EPOCHOFFSET parameter in host software). The conversion routines provided by **date** and other commands depend on the relation "calendar_I" to contain the proper conversions. Two additional stored commands are provided with the **date** script. One, **base1970**, will convert the "calendar_I" relation to conversions based on January 1, 1970. Another, **base1900**, will restore the conversion table from 1970 to 1900. The view "DateAndTime" contains a variety of representations of the underlying current date that can be used in any query. The relation "savings_I" should be adjusted to the local time zone before **date** is loaded. Beware, **date** will expire in 1995.

EXAMPLE

1) date;

Day	Date	Time
Tuesday	June 11, 1985	13:08:07

1 tuple affected

dateconv idmdate

DESCRIPTION

This command converts dates from Britton Lee's IDM/RDBMS form (4-byte integers) into human-readable form. This is convenient when trying to decode some date other than the current date. (Use the **date** command for the current IDM/RDBMS date.) It uses the relations "calendar_I" and "month_I" for date conversion.

EXAMPLE

1) dateconv 32214;

date_written
March 14, 1988

1 tuple affected

depend object

DESCRIPTION

When a view or stored command references another object (generally a relation or view), the view or stored command is said to be dependent on that relation or view. Among other things, this means that the underlying relation may not be destroyed without first destroying the dependent object. When you try to destroy such a relation, you will get back an error message that lists at least one dependent object. This command lists all the dependent objects on a named object.

EXAMPLE

1) depend indices;

object	dependents	downer
indices	indexes	DBA
indices	omni_full	DBA
indices	omni_xs	DBA
indices	omni_xsof	DBA

4 tuples affected

describe object

DESCRIPTION

The **describe** command is an easy way to extract information from the "descriptions" system relation. It will retrieve all of the description text for a named object and format it neatly on the screen. It only looks at the first 72 characters of each "description.text" line, so if you use the **associate** command, keep your descriptions down to 72 characters per line.

Along with the script of the **describe** command is a set of multi-line descriptions of the system relations for each database. These demonstrate how to use **associate** properly for this command to work. They also give on-line information on the system relations.

The key field used by **associate** is essentially a sequential line number for each description.

EXAMPLE

1) describe attribute;

description	
attribute:	Catalog of each attribute of each relation. Each tuple represents one attribute.

2 tuples affected

dir name

DESCRIPTIONS

This command is a simple query into the "objects" view that searches for all objects of the given **name**. By using wild-card characters, it can also identify objects of a given name fragment. It is meant to work similarly to a "directory" command on other operating systems. It is different from the other object-related commands (**rels**, **pgms**, **files**, **othercmds**, **otherviews**) in that it searches all objects in the database no matter who owns them.

It also incorporates a decoding table ("otype_I") for the known object types.

EXAMPLE

1) dir "*tup*";

relid	object	type	definition	owner	tups
416	onektup	U	User relation	eds	1000
20247	tenktup1	U	User relation	eds	10000
9198	tenktup2	U	User relation	eds	10000

3 tuples affected

expire object,ndays

DESCRIPTION

There is a user-definable attribute in the "relation" relation that permits users to list an expiration date for an **object**. Databases that permit users to create objects can thus accumulate objects and provide a maintenance problem for the DBA. If users are required to condition this attribute, the DBA may write stored commands or programs that regularly purge the database of expired data.

The attribute is a 4-byte integer, so it most conveniently accepts a valid IDM/RDBMS date (see the **date** command). This attribute is automatically set to the current date + 5 days when a relation is created. If the user wishes to use a different date, he can update through the **expire** command. The user need not have write permission on the "relation" relation to provide an update in this way.

The parameter *ndays* is added to the value returned by **getdate** to produce a new expiration date for the named *object*.

EXAMPLE

1) expire onektup,60;

1 tuple affected

expiredate object,yyyymmdd

DESCRIPTION

This command works in the same way as **expire**, except that it accepts a date parameter in the form of an absolute date written as an 8-character string ***yyyymmdd*** (year-month-day). It sets the expiration date of **object** to the **idmdate** equivalent value in the "relation" system relation. It is up to the DBA to enforce this expiration date, so its use is optional.

EXAMPLE

1) expiredate onektup,"19891018"

1 tuple affected

files

DESCRIPTION

This is one of the commands that depend on the view "objects". It is a friendly way to list all of the files owned by you and the DBA. The "objects" view includes the more readable information from the "relation" relation joined to the owner name and a relation for decoding logging information called "logged_I".

EXAMPLE

1) files;

files	owner	logging
fort.ex	DBA	Not Logged
johntest	DBA	Not Logged

2 tuples affected

freespace

DESCRIPTION

This command reports the amount of space allocated to this database, how many disk blocks are free, and what percent of the database is used.

EXAMPLE

1) freespace;

total_blks	free_blks	percent_used
15120	6571	56

1 tuple affected

gmt_date

DESCRIPTION

Like the **date** command, **gmt_date** gives the Greenwich date and time in a neatly formatted form. It is actually a little simpler than **date**, because Britton Lee's shared database system stores uses Greenwich Mean Time internally.

EXAMPLE

Interactive IDL version 3.5

1) gmt_date;

Day	Date	Time
Tuesday	June 11, 1985	23:16:18

1 tuple affected

mine

DESCRIPTION

This is one of the commands that depend on the view "objects". It is a friendly way to list all of the objects owned by you only. In a large database with many objects owned by the DBA and only a few owned by each user, this can be a very convenient command. The "objects" view includes the more readable information from the "relation" table joined to the owner name and a table for decoding logging information called "logged_I".

EXAMPLE

1) mine;

relid	object	type	definition	tups
17649	addauthor	C	Stored command	0
24980	addpubset	C	Stored command	0
4546	author	U	User relation	10
16590	authttl	U	User relation	8
31874	balance	U	User relation	3
15947	pubsmaster	U	User relation	101

6 tuples affected

othercmds

DESCRIPTION

This is one of the commands that depend on the view "objects". It is a friendly way to list all of the stored commands owned by you and the DBA that are not part of the predefined commands. The "objects" view includes the more readable information from the "relation" relation joined to the owner name and a relation for decoding logging information called

EXAMPLE

Interactive IDL version 3.5

1) othercmds;

command	owner	logging
distrib	DBA	Not Logged
fields_of	DBA	Not Logged
inc_rank	DBA	Not Logged
lcode	DBA	Not Logged
median	DBA	Not Logged
relq	DBA	Logged
scode	DBA	Not Logged
st_median	DBA	Not Logged
stateprof	DBA	Not Logged

9 tuples affected

otherviews

DESCRIPTION

This is one of the commands that depend on the view "objects". It is a friendly way to list all of the views owned by you and the DBA. The "objects" view includes the more readable information from the "relation" relation joined to the owner name and a relation for decoding logging information called "logged_I".

EXAMPLE

1) otherviews;

views	owner	logging
DateAndTime	DBA	Logged
GMT	DBA	Logged
field_view	DBA	Not Logged
tables_view	DBA	Not Logged

4 tuples affected

permits object

DESCRIPTION

This command extracts information encoded in the "protect" system relation and displays it in readable form. It describes all non-default protection placed on *object*. It does not describe the protection placed on individual attributes in a relation.

EXAMPLE

1) permits configure;

access	object	user
permit read	configure	ALL
permit write	configure	eds

2 tuples affected

permitsall

DESCRIPTION

This command describes all non-default protection for all objects in the database. It does not describe the protection placed on individual attributes in relations. It extracts information encoded in the "protect" relation and displays it in readable form.

EXAMPLE

1) permitsall;

access	object	user
permit execute	freespace	ALL
permit execute	permitsall	ALL
permit read	vintners	ALL
permit read	ptype_I	ALL
permit read	kindsq	ALL
permit read	atype_I	ALL
permit read	wines	ALL
permit read	pricings	ALL
permit read	stores	ALL
deny write	ptype_I	ALL
deny write	atype_I	ALL

11 tuples affected

permitsgen

DESCRIPTION

This command describes all non-default permissions regarding the database. This includes tape read and write permissions, create permissions, create index permissions, and create database permissions (system database only). These permissions apply to the database in general and to all objects in the database. This command extracts information encoded in the "protect" relation and displays it in readable form.

EXAMPLE

1) permitsgen;

access	user
permit create	ALL
permit create index	ALL
permit read tape	ALL
permit write tape	ALL

4 tuples affected

permitsme

DESCRIPTION

This command describes all non-default permissions regarding objects in the database for the user entering the command. It does not report protection on individual attributes in a relation. It extracts information encoded in the "protect" relation and displays it in readable form.

EXAMPLE

1) permitsme;

access	object
permit write	configure

1 tuple affected

permitsuser user_name

DESCRIPTION

This command describes all non-default permissions regarding objects in the database for the stated user. It does not report protection on individual attributes in a relation. Notice that asking for permissions for user ALL will return permissions that apply to everyone. This command extracts information encoded in the "protect" system relation and displays it in readable form.

EXAMPLE

1) permitsuser helen;

access	object
deny read	tenktup1

1 tuple affected

pgms

DESCRIPTION

This is one of the commands that depend on the view "objects". It is a friendly way to list all of the stored programs owned by you and the DBA. The "objects" view includes the more readable information from the "relation" relation joined to the owner name and a relation for decoding logging information called "logged_I".

EXAMPLE

1) pgms;

pgms	owner	logging
dar_updatep	root	Logged
dartypesp	garry	Logged
dhelpp	root	Logged
freearnum	root	Not Logged
gar_retrp	root	Logged
gartermp	garry	Logged
gartypesp	root	Logged
ghelpp	root	Logged
narqp	root	Logged
sattdetail	root	Logged

10 tuples affected

rels

DESCRIPTION

This is one of the commands that depend on the view "objects". It is a friendly way to list all of the relations owned by you and the DBA. The "objects" view includes the more readable information from the "relation" relation joined to the owner name and a relation for decoding logging information called "logged_I".

It is equivalent to the SQL command "tabs".

EXAMPLE

1) rels;

relation	owner	tups	logging
Bprime1	DBA	1000	Not Logged
dict	DBA	24001	Not Logged
eds10	DBA	20004	Not Logged
eds5	DBA	10002	Logged
onektup	DBA	1000	Not Logged
tenktup1	DBA	0	Not Logged

6 tuples affected

rename object,objectname

DESCRIPTION

This command allows a user to change the name of an object owned by that user without needing write permission on the "relation" relation. Only the name recorded in "relation.name" is changed. In this way, dependencies are not altered. For instance, if a stored command was defined on a relation named "old", and the name of the relation was changed to "new", the stored command would still function in the same manner. Similarly, if a different relation were given the name "old", it would not be recognized by the stored command.

EXAMPLE

1) rename onektup,uniquetup;

1 tuple affected

size object

DESCRIPTION

This command gives two kinds of information: (1) the number of tuples and the number of disk pages for a given object according to the "relation" relation; (2) the actual number of disk pages (blocks) allocated to that object from the "blockalloc" system relation. The command does a count of the "blockalloc" tuples with the "relid" of the named relation.

The value stored in the "relation.pages" attribute is also given as "rel_pages", so that it may be checked for accuracy. There is no guarantee that this attribute or "relation.tups" is correct.

EXAMPLE

1) size geography;

relid	rel_tups	rel_pages	num_blocks
9053	8282	145	236

1 tuple affected

sizebyzone object

DESCRIPTION

The best indication of how fragmented a relation may be is the distribution of the data among the zones of the disk. Scans of the data will perform better if all of the zones are close together and if they are largely filled. When a new relation becomes fragmented, it may indicate that the database is too full. Extending the database and then recopying the relation may help. **Sizebysone** lists all of the disk zones allocated to the named relation and how many blocks within each zone are allocated. The zone addresses are given in hexadecimal, and the number of blocks in decimal. A zone contains up to 255 disk blocks, depending on how the disk is formatted.

EXAMPLE

1) sizebysone tenktup1;

zone	blocks
001D89	6
002059	12
00210D	86
0021C1	172
002275	147
002329	2

6 tuples affected

sizes

DESCRIPTION

This command provides quite a bit of useful information about space allocation in the database. It first lists the number of blocks allocated for the database, the number of free blocks, and the percentage of allocated blocks used by data. This is the same display one gets from the **freespace** command. It then prints a table of all of the objects larger than 10 disk blocks, showing their types and their respective sizes. The table is in descending order of size.

The command takes a while to execute, since it must do an aggregate count of "blockalloc" by "relid".

EXAMPLE

1) sizes;

total_blks	free_blks	percent_used
13680	4832	64

1 tuple affected

name	owner	type	rel_pages	num_of_blks
work	1	Q	24	952
school_vet	1	Q	45	582
tenktup1	1	U	120	425
eds5	1	U	67	287
dict	1	U	16	168
onektup	1	U	1	106
Bprime1	1	U	200	100
query	1	S	4	80
blockalloc	1	S	189	76
transact	1	T	27	37
attribute	1	S	12	18

11 tuples affected

spacebyuser

DESCRIPTION

This command gives a listing of the space allocated to all the objects owned by each user in the database. For each user, the number of blocks allocated to each owned object is added up and tabulated. The display lists these sums with the user name. The entries are listed in decreasing order of space used.

EXAMPLE

1) spacebyuser;

user	user_id	pages
eds	145	8012
DBA	1	267
toms	144	200
susan	125	35
miscguest	100	23
loren	34	6
student2	156	5
john	55	4
helen	165	4
garry	167	3
miket	169	2
pitta	133	2
sharon	123	2
berlind	175	2
garvey	185	1
steven	184	1
melinda	177	1

17 tuples affected

tabs

DESCRIPTION

This is one of the commands that depend on the view "objects". It is a friendly way to list all of the tables owned by you and the DBA. The "objects" view includes the more readable information from the "relation" table joined to the owner name and a table for decoding logging information called "logged_I".

It is equivalent to the IDL command "rels".

EXAMPLE

1) start tabs;

table_name	owner	rows	logging
Bprime1	DBA	1000	Not Logged
dict	DBA	24001	Not Logged
eds10	DBA	20004	Not Logged
eds5	DBA	10002	Logged
onektup	DBA	1000	Not Logged
tenktup1	DBA	0	Not Logged

6 rows affected

uses

DESCRIPTION

Views or stored commands use underlying objects (relations or views) and are dependent on them. The **depend** command takes and underlying object and lists the objects dependent on it. The **uses** takes a dependent object and lists the relations or views it uses.

EXAMPLE

1) uses omni_full;

relationused	owner
indices	DBA
relation	DBA

2 tuples affected

views

DESCRIPTION

This command gives a listing of the views used by the predefined stored commands with a brief description of each. These views may be useful for making queries on the data dictionary without having to worry about the encoding used by the system relations.

EXAMPLE

1) views;

views	description
DateAndTime	Gives current date and time in various formats
GMT	Gives Greenwich date and time in various formats
columns	Friendly view of attributes relation
objects	Objects, their owners, type, size, and if logged

4 tuples affected

ymd yyyymmdd

DESCRIPTION

This stored command takes a standard date string and returns Britton Lee's IDM/RDBMS date (4-byte integer). It uses the "calendar_I" and "month_I" relations to decode the date string ***yyyymmdd***. The same relations and algorithms are used in all IDM/RDBMS date-conversion functions.

EXAMPLE

1) ymd "19851225";

idmdate
31404

1 tuple affected

PART III

Stored Commands — Database Administrators

Summary

The stored commands described in this part are largely for the use of the DBA in managing the shared database system:

allindexes	Lists all the indices in the database (132 columns)
cmds_dba	Lists stored commands mostly used by the DBA.
freelog	Shows the space usage in a hard-allocated transaction log.
group_off	Turns off all access to database for group (DBA only).
group_on	Turns group database access back on (DBA only).
groups	Lists the groups and membership for this database.
indexes	Displays keys and type of indices for given *relation*.
notowned	Lists objects not owned by people in "users".
rmuser	Removes a user from database (DBA only).
setcard	Sets cardinality for *relation*, *index_id*, *cardinality*.
system	Displays system relation names and sizes.
usersright	Checks consistency of "users" with "host_users".
who	Lists users who can use this database.
whois	Returns "user_id", "host_id", and "huid" for given *user_name*.
whoisid	Returns "user_name", "host_id", and "huid" for given *user_id*.

allindexes

DESCRIPTION

This command returns information on all indices on all tables in the database. It includes the names and the first five concatenated keys for each index. There may be more keys in the concatenation sequence, but you will have to look into the "indices" relation to find out.

The "indices.card" attribute is also listed, so you may see that this accurately reflects the cardinality of each index. A unique index has a cardinality of 1. A zero means there is no information. This attribute is important for assuring proper performance of joins and other queries. It may be set with the **setcard** command.

Unlike other stored commands, this one does not fit in an 80-column screen. It uses a 132-column display, so it may be easier to route its results to a printer.

EXAMPLE

1) allindexes;

relation	type	key1	key2	key3	key4	key5	indid	card
summary	clustered	rec_type	state	seqno			0	0
summary	nonclustered	area_name					1	1
summary	nonclustered	seqno					2	0
onektup	clustered	unique2				0	1	
onektup nonclustered	unique1				1	1		

3 tuples affected

cmds_dba

DESCRIPTION

This is a version of the **cmds** command for those predefined commands that are in the DBA category. This stored command is in the user command section, although the commands it describes may be permitted only to the DBA.

EXAMPLE

1) cmds_dba;

command	description
allindexes	Displays keys and type of indices for all user relations (132 columns)
group_off	Turn off all access to database for GROUP - for DBA only
group_on	Turn GROUP database access back on - for DBA only
groups	Lists the groups and membership for this database
indexes	Displays keys and type of indices for given RELATION
notowned	List objects which are not not owned by people in 'users'
rmuser	Removes a USER from database - for DBA only
setcard	Set cardinality for RELATION, INDEXID, CARD
system	Displays system relation names and sizes
usersright	Checks consistency of 'users' with 'host_users'
who	Lists users who can use this database
whois	Returns userid, host_id and huid for a given username
whoisid	Returns user name, host_id and huid for a given user_id

13 tuples affected

freelog

DESCRIPTION

This command allows the DBA to see how much space is allocated and how much is free in the "transact" system relation, where that relation has been explicitly allocated space at database creation or extend time. The **freespace** command shows the total space allocated to a database including the transaction log. If the database is created with a "with logblocks" option, the transaction log may be placed on a separated disk from the rest of the data. Free space in this log is not seen as free space in the database, since it cannot be used by other objects. Since it is undesirable to have a transaction log overflow the allocated space on a separate disk, this command gives the DBA an idea of how much space remains.

EXAMPLE

1) freelog;

log_blocks	free_blks
25212	12374

1 tuple affected

group_off group_name

DESCRIPTION

This command allows the DBA to delete a group from access to this database. The members of the group are stored in a temporary relation, so that they can be restored by a **group_on** command subsequently. This command is useful when the DBA wishes to grant temporary access to a database to a group of guests or students. Likewise, a group of users may be denied access temporarily while some changes are being made to the data.

The command works by actually deleting all the tuples for the users in the named group from the "host_users" system relation and the "users" system relation. These tuples are added to temporary relations called "host_users_I" and "users_I". The response should be four statements of "tuples affected", two for the appends and two for the deletes.

EXAMPLE

1) group_off centfolk;

6 tuples affected
7 tuples affected
7 tuples affected
6 tuples affected

group_on group_name

DESCRIPTION

This command restores access to the users that have been temporarily removed by the **group_off** command. It has no effect if the specified group is not currently a group that is off access. Its function is just the reverse of the **group_off** command, since it first appends the users from the temporary relations "users_I" and "host_users_I" to the real system relations, then deletes the same entries from the temporary relations.

EXAMPLE

1) group_on centfolk;

6 tuples affected
7 tuples affected
7 tuples affected
6 tuples affected

groups

DESCRIPTION

This is an easy way to see the groups in the "users" relation and the number of members in each. Use the **who** command to find out the members' names.

EXAMPLE

1) groups;

id	gid	name	members
1000	1000	centfolk	4

1 tuple affected

indexes relation

DESCRIPTION

This command is designed to give friendly information from the "indices" system relation, and because we don't want to confuse the command with the relation, we use an alternative spelling. The indices on a named relation are listed, along with the first three concatenated keys for each index. There may be more keys in the concatenation sequence, but you will have to look into the "indices" relation to find out.

The "indices.card" attribute is also listed, so you may see that this accurately reflects the cardinality of each index. A unique index has a cardinality of 1. A zero means there is no information. This attribute is important for assuring proper performance of joins and other queries. It may be set with the **setcard** command.

EXAMPLE

1) indexes geography;

type	key1	key2	key3	indid	card
clustered	rec_type	state	seqno	0	0
nonclustered	area_name			1	1
nonclustered	seqno			2	0

3 tuples affected

notowned

DESCRIPTION

This command provides a utility for determining whether there are any objects that are not owned by current users (members of the "users" system relation) in the database.

EXAMPLE

1) notowned;

name	relid	owner_num
myparts	10610	5001
myvendors	29569	5001
part_list	16632	5001

3 tuples affected

rmuser user_name

DESCRIPTION

This command allows the DBA to remove a user from the database. It deletes the "users" and "host_users" tuples associated with *user_name* from their respective relations. The command may not be used to remove the DBA tuples from these relations.

EXAMPLE

1) rmuser pcguest;

1 tuple affected
1 tuple affected

setcard relation,index_id,cardinality

DESCRIPTION

This command allows the DBA to set the **cardinality** of an index in the "indices" relation. You must know the *relation* name and the *index_id*, which may be obtained from the **indexes** stored command. The correct cardinality is important for the proper functioning of the cost algorithms in query optimization. The default is zero. For a unique index, 1 is the value.

EXAMPLE

1) setcard geography, 2, 1;

1 tuple affected

1) indexes geography;

type	key1	key2	key3	indid	card
clustered	rec_type	state	seqno	0	0
nonclustered	area_name			1	1
nonclustered	seqno			2	1

3 tuples affected

system

DESCRIPTION

This commands displays the system relations and their sizes in number of tuples and number of blocks.

EXAMPLE

1) system;

relation	type	tuples	pages
attribute	S	715	18
batch	T	0	1
blockalloc	S	180	1
crossref	S	161	1
descriptions	S	73	5
disk_usage	S	5	1
host_users	S	6	1
indices	S	41	3
protect	S	94	1
query	S	304	80
relation	S	129	7
transact	T	8310	44
users	S	6	1

13 tuples affected

usersright

DESCRIPTION

This command helps the DBA check for errors in the system relations that keep track of users. The exception conditions that are caught by this command include:

(1) "Users" tuples for which there are no corresponding "host_users" tuples.

(2) "Host_users" tuples for which there are no corresponding "users" tuples.

(3) Groups which have no members.

(4) Users with no groups.

(5) Duplicate user ids in either relation.

(6) User id 0 for a user other than ALL, or any entry in "host_users" for user id 0.

(7) List of all those users who are DBA for this database if there are more than one.

EXAMPLE

1) usersright;

hid	huid	uid	comment

0 tuples affected

name	id	gid	comment

0 tuples affected

name	id	gid	comment
ALL	0	0	Duplicate ids in 'users'
default	0	0	Duplicate ids in 'users'

2 tuples affected

name	id	gid	comment
default	0	0	Inappropriate use of uid = 0

1 tuple affected

hid	huid	uid	comment

0 tuples affected

hid	huid	uid	comment
1	167	1	DBA for this database
4	4390924	1	DBA for this database
4	4390935	1	DBA for this database
5	167	1	DBA for this database

4 tuples affected

who

DESCRIPTION

This command does a join of the "users" and "host_users" relations to give a neatly formatted listing of the users in this database. It will miss any users who aren't in both, so you might want to do a **userright** command first.

EXAMPLE

1) who;

user	group	hst_id	host_user_id	id
DBA	centfolk	4	4390915	1
DBA	centfolk	5	167	1
DBA	centfolk	5	244	1
guest	centfolk	-1	0	100
helen	centfolk	5	176	176
miket	centfolk	5	114	114

6 tuples affected

whois user_name

DESCRIPTION

This command will look up a *user_name* in the "users" relation and report back all information about this user, mainly the "user_id" in this database and the "host_id" and "host_user_id" information.

EXAMPLE

1) whois eds;

id	group	hst_id	host_user_id
244	BL users	4	4390915
244	BL users	5	244
244	BL users	7	267
244	BL users	11	18

4 tuples affected

whoisid user_id

DESCRIPTION

This command will look up a *user_id* and return the "user_name" along with the "host_ids" and "host_user_ids" that map to this user in this database.

EXAMPLE

1) whoisid 244;

name	group	hst_id	host_user_id
eds	BL users	4	4390915
eds	BL users	5	244
eds	BL users	7	267
eds	BL users	11	18

4 tuples affected

PART IV

Stored Commands — System Database

Summary

The stored commands listed here are for use in the system database:

baudrate	Reports the speed set for each serial line.
channel	Decodes the bits set in "configure.value".
cmds_system	Lists stored commands dealing with protection of objects.
config	Lists the tuples from the "configure" relation.
dbs	Displays databases on this relational database system.
diskio	Lists the on-line disks and their I/O activity.
mon	Monitors CPU and DAC usage for last *n* monitor intervals.
mondisk	Monitors disk usage for last *n* monitor intervals.
monfail	Monitors suspended processing for lack of memory.
monlock	Monitors current locks and history for last *n* monitor intervals.
monun	Monitors unused memory buffers for last *n* monitor intervals.
monwait	Summarizes wait queues for last *n* monitor intervals.
ps	Lists the active processes and status for known users.

These commands are designed to give configuration and performance information available in the system-database data dictionary. They all depend on the special system relations, and generally make the information available in these relations more easy to read. Some of the commands (**ps**, **mon***) give information on the current or recent state of process and hardware activity. Other commands (**config**, **diskio**, **baudrate**, **channel**, **dbs**) supply and decode information about the relational system hardware's configuration and databases.

There are five separate monitor commands, each of which provides certain types of information from the "monitor" relation during the last *n* intervals. These commands all take an argument *n* between 1 and 54.

baudrate

DESCRIPTION

The **baudrate** command decodes the baud rate set by each of the S tuples in the "configure" system relation. This command uses the Invisible relation: **cbaud_I**, to decode the last 5 bits of the "S" tuple. See the *System Administrator's Manual* for more information.

EXAMPLE

1) baudrate;

channel	baudrate
2	No communication
4	9600
5	9600
6	9600

4 tuples affected

channel

DESCRIPTION

The **channel** command decodes some of the bits set in the "configure" system relation. These bits determine host trustworthiness, modem control, packet size, timeout protocol, byte order, and character type. This command uses the Invisible relations: **channel_I**, and **mask_I** to mask out the binary bits and interpret their meaning. The decoding is different for each tuple type. See the *System Administrator's Manual* for more information.

EXAMPLE

1) channel;

type	channel	bit_set	meaning
E	0	12	Packet size 512-byte, (256 if bit 12 set)
S	3	11	Nontrustworthy host_userids

2 tuples affected

cmds_system

DESCRIPTION

This lists all of the stored commands unique to the system database. Its description will appear in the output of the **cmds** command for the system database only.

EXAMPLE

1) cmds_system;

command	description
baudrate	Gives the serial channel bits per second rate
channel	Decodes the bits set in 'configure.value'
config	Identify the tuples from the configure relation
dbs	Displays databases on this shared database system
diskio	Lists the on-line disks and their I/O activity
mon	Monitor CPU and DAC usage for last N monitor intervals
mondisk	Monitor disk usage for last N monitor intervals
monfail	Suspended processing for lack of memory from monitor
monlock	Current locks and history for last N monitor intervals
monun	Unused memory buffers for last N monitor intervals
monwait	Summary of wait queues for last N monitor intervals
ps	Lists the active processes and status--known users

12 tuples affected

config

DESCRIPTION

The **config** command simply lists the tuples in the "configure" system relation along with a brief explanation of each tuple type.

EXAMPLE

1) config;

type	number	value	name
A	0	16	Accelerator, value=WCS level
C	0	5	Checkpoint interval
D	0	0	Default char set 0=ASCII 1=EBCDIC
E	0	4096	Ethernet host interface
E	1	10	Ethernet host interface
E	4	0	Ethernet host interface
E	5	4	Ethernet host interface
M	0	1	Monitor interval(minutes)
P	1	8257	IEEE-488 host interface
P	9	8194	IEEE-488 host interface
R	0	43	IDM/RDBMS software release

11 tuples affected

dbs

DESCRIPTION

The **dbs** command lists the databases from the "databases" system relation, along with their owners and status descriptions. The decoding of descriptions is defined in the relation "dbs0_I" that is established by installing the stored command. This list is current through release 3.5 of IDM/RDBMS software. Succeeding releases of software add new codes. These new codes may be added to the relation "dbs0_I" by the DBA.

EXAMPLE

1) dbs;

name	dbid	owner_name	status
ar_system	4	DBA	On-line (ASCII)
att_test	15	DBA	On-line (ASCII)
census	18	DBA	On-line (ASCII)
census1	8	DBA	Unused since boot, recovery had to run (ASCII)
demos	2	DBA	On-line (ASCII)
error	6	DBA	On-line (ASCII)
jim	5	DBA	On-line (ASCII)
lucy	13	DBA	On-line (ASCII)
pitta	10	pitta	Unused since boot, recovery had to run (ASCII)
sales	3	DBA	Unused since boot, recovery had to run (ASCII)
sandbox	9	DBA	On-line (ASCII)
sara	7	DBA	Unused since boot, recovery had to run (ASCII)
system	1	DBA	On-line (ASCII)
templt	11	pitta	Unused since boot, recovery had to run (ASCII)
vino	14	DBA	On-line (ASCII)

15 tuples affected

diskio

DESCRIPTION

This command lists the disks currently known in the "disks" relation and in "devmonitor." It also summarizes their activity for the last 54 monitor intervals. The number of disk accesses and the amount of time each disk has been active are given. The quotient of these two values gives the average access time for that disk as well.

If a disk was not used during the previous 54 monitor intervals, that disk does not appear in the output of this command.

EXAMPLE

1) diskio;

name	slot	accesses	time_ticks	avg_ms
1fuji160mb	4	3793	4338	19.061
2cdc160mb	5	16	16	16.667

2 tuples affected

mon n

DESCRIPTION

This is one of the commands to view the "monitor" relation. It permits you to view the last *n* intervals of "monitor." The command consists of two separate retrieves. The first tells you the average monitor interval in minutes over the period you have selected. The second lists the entries from the last intervals recorded in chronological order. The most recent interval will always be the last one listed.

The **mon** command gives a summary of CPU activity, the number of commands executed during each interval, and the average completion time in seconds for commands terminating during that interval. The DAC, CPU, and IDLE times are given as a percentage of the interval duration. The meaning of the DAC column depends on whether or not a DAC is present in the relational system hardware. If it is not present, this column represents the time that the CPU spent emulating DAC routines. The CPU time includes the DAC time, so the CPU value will always be larger than the DAC value.

EXAMPLE

1) mon 5;

CPU_activity	time
avg monitoring interval (minutes):	0.99000

1 tuple affected

SEQNO	CPU_percent	DAC_percent	IDLE_percent	COMMANDS	AVG_seconds
15	0.25000	0.083333	0.	1	0.10000
16	0.083333	0.	0.	0	0.
17	27.425	2.1172	19.055	0	0.
18	0.36526	0.040584	99.188	1	0.18333
19	2.0556	0.19444	97.278	2	0.56667

5 tuples affected

mondisk n

DESCRIPTION

This is one of the commands to view the "monitor" relation. It permits you to view the last *n* intervals of "monitor." The command consists of two separate retrieves. The first tells you the average monitor interval in minutes over the period you have selected. The second lists the entries from the last intervals recorded in chronological order. The most recent interval will always be the last one listed.

The **mondisk** command summarizes disk activity over the last *n* intervals. The number of disk reads and writes is given, as well as the number of cache hits (the number of times a disk block was found to already exist in cache memory, thus eliminating a read). The "disk_wait" is the cumulative time processes spent waiting for disk I/O during that interval. The "disk_wait" may be greater than the length of the interval.

EXAMPLE

1) mondisk 5;

DISK_active	time
avg monitoring interval (minutes):	0.99000

1 tuple affected

SEQNO	READS	WRITES	HITS	DISK_WT_sec
15	0	0	104	0.
16	0	0	38	0.
17	896	62	206	10.250
18	8	0	51	0.21667
19	11	1	115	0.40000

5 tuples affected

monfail n

DESCRIPTION

This is one of the commands to view the "monitor" relation. It permits you to view the last *n* intervals of "monitor." The command consists of two separate retrieves. The first tells you the average monitor interval in minutes over the period you have selected. The second lists the entries from the last intervals recorded in chronological order. The most recent interval will always be the last one listed.

The **monfail** command reports on situations where the shared database system has had to suspend processing due to lack of memory space. The INDELAY and OUTDELAY indicate what percent of the interval no processing could occur for lack of buffer space in either the input or output buffers. Any appreciable number of non-zero entries in these columns would indicate a need for increasing the size of the affected buffer through the "configure" system relation.

The MEMDELAY column indicates the percent of the interval that the CPU was idle with processes waiting to run due to lack of available memory space. Any non-zero entry in this column may indicate the need for more memory on the relational system hardware. The DBINFAILS column represents the number of "dbins" that had to be rejected due to lack of available entries in the "dbins" table. See the *Database Administrator's Manual.*

These values can be used in conjunction with their corresponding values given in the **monun** command.

EXAMPLE

1) monfail 5;

SUSPENDED	time
avg monitoring interval (minutes):	1.0000

1 tuple affected

SEQNO	INDELAY_pct	OUTDELAY_pct	MEMDELAY_pct	DBINFAILS
18	0.	0.	0.	0
19	0.	0.	0.	0
20	0.	0.	0.	0
21	0.	0.	0.	0
22	0.	0.	0.	0

5 tuples affected

monlock n

DESCRIPTION

This is one of the commands to view the "monitor" relation, but it also gives a view of the "lock" system relation. The command consists of three separate retrieves. The first tells you the average monitor interval in minutes over the period you selected. The second lists the entries from the last intervals recorded in chronological order, most recent last. The third lists the current locks on the shared database system. The list of current locks from the "lock" relation will change rapidly as locks are acquired and released by query processes. It can help to identify locks that have not been released.

The third retrieve requires some interpretation. The "block" column is the disk block id for the lock. If it is 0, the lock is a relation lock. The RELID is the "relid" of the relation in its database. It is not possible to decode that into a name in the system database.

EXAMPLE

1) monlock 5;

LOCKS_BLOCKS	time
avg monitoring interval (minutes):	1.0000

1 tuple affected

SEQNO	DEADLOCKS	LOCK_sec
21	0	0.
22	0	0.
23	0	0.
24	0	0.
25	0	21.167

5 tuples affected

DBIN	DATABASE	RELID	block	lock_type
7	john	3031	000000	write lock
2	lucy	23900	002FF1	read lock
2	lucy	23900	0027C2	read lock
2	lucy	23900	000DDB	read lock
2	lucy	13	000000	write lock
2	lucy	23900	0027C2	write lock
2	lucy	23900	000000	intend to set block read lock
2	lucy	23900	000000	intend to set block write lock

8 tuples affected

monun n

DESCRIPTION

This is one of the commands to view the "monitor" relation. It permits you to view the last *n* intervals of "monitor." The command consists of two separate retrieves. The first tells you the average monitor interval in minutes over the period you have selected. The second lists the entries from the last intervals recorded in chronological order. The most recent interval will always be the last one listed.

The **monun** command records the amount of memory that was not used during the monitor interval. The UNIN and UNOUT columns record the unused input and output buffer space. Should these values fall to zero during an interval, one should check the **monwait** command to see if the CPU had to wait for these buffers to clear. The UNDBIN column records the minimum number of unused "dbins" during each interval. The UNMEM records the minimum number of pages for query buffers that were not used.

Should any of these values fall to zero during a monitor interval, there is a need for more memory in that category. This can be obtained through reconfiguration or adding memory to the relational system hardware.

EXAMPLE

1) monun 5;

BUFFER_SPACE	time
avg monitoring interval (minutes):	0.99000

1 tuple affected

SEQNO	UNUSED_INPUT	UNUSED_OUT	UNUSED_MEM	UNUSED_DBIN
16	67584	135168	184	253
18	67552	135136	178	253
19	67568	134656	177	253
20	67552	134656	177	253
21	67500	134912	177	253

5 tuples affected

monwait n

DESCRIPTION

This is one of the commands to view the "monitor" relation. It permits you to view the last *n* intervals of "monitor." The command consists of two separate retrieves. The first tells you the average monitor interval in minutes over the period you have selected. The second lists the entries from the last intervals recorded in chronological order. The most recent interval will always be the last one listed.

The **monwait** command summarizes the various wait queues that processes occupy. The total number of seconds that processes waited in that queue during the monitor interval is recorded. This number may be greater than the length of the interval when multiple processes are running.

The INWAIT and OUTWAIT values and the TAPE value represent processes scheduled out while waiting for I/O processes to complete with the host or shared database system tape. The DISK wait for disk I/O is reported from the **mondisk** command. The CPU wait queue is for processes that are waiting to be run while another process has the CPU. In multi-process conditions, this value will normally be quite high as jobs are scheduled. The MEMORY wait time represents processes waiting for available buffer space to process their query.

EXAMPLE

1) monwait 5;

WAIT_QUEUES	time
avg monitoring interval (minutes):	0.99000

1 tuple affected

SEQNO	INPUT_sec	OUTPUT_sec	CPU_sec	MEMORY_sec	TAPE_sec
15	0.	0.	0.	0.	0.
16	0.	0.	0.	0.	0.
18	0.	0.	0.	0.	0.
19	0.	0.	0.033333	0.	0.
20	0.	0.	0.033333	0.	0.

5 tuples affected

ps

DESCRIPTION

This is the process status command (name borrowed from UNIX) that tells you what queries are active on Britton Lee's shared database system at the time of the command. The information is largely derived from the "dbinstat" system relation. It is joined to the relation "ps_data_I" to decode the status field of that relation. The list of status meanings is current as of release 3.5 of the IDM/RDBMS code. Subsequent releases may include new values that may be updated by the DBA. This command does not preserve the CANCEL information available from that field.

The second query is a list of the users attached to the "dbins" where those users are known in the system database.

EXAMPLE

1) ps;

dbin	status	block	time	data_base
0	wait--input in a transaction	-1	11	system
2	runnable	-1	6	system

2 tuples affected

dbin	user	hid	huid
0	eds	5	244
2	eds	5	244

2 tuples affected

APPENDICES

Appendix A: Summary of Commands

allindexes
atts relation
base1970
base1900
baudrate
channel
cmds
cmds_date
cmds_dba
cmds_permit
cmds_space
cmds_system
cols
config
date
dateconv idmdate
dbs
depend object
describe object
dir name
diskio
expire object,ndays
expiredate object,yyyymmdd
files
freelog
freespace
gmt_date
group_off group_name
group_on group_name
groups
indexes relation
mon n
mine
mondisk n
monfail n
monlock n
monun n
monwait n
notowned
othercmds
otherviews
permits object
permitsall
permitsgen
permitsme
permitsuser user_name
pgms
ps

rels
rename object,objectname
rmuser user_name
setcard relation,index_id,cardinality
size object
sizebysone object
sizes
spacebyuser
system
usersright
views
who
whois user_name
whoisid user_id
ymd yyyymmdd

Appendix B: IDL Definitions

This appendix lists the IDL stored-command scripts in alphabetical order. Since some of the commands are included in files by other names, the file name is given in parentheses after the command name. The first two sections define two views, "columns" and "objects", that are used to decode attribute and object lists.

columns (atts.idl)

```
destroy columns go
destroy dtype_I go

/* Build "dtype_I" for "atts" command, used to generate mnemonic */

create dtype_I (
                mne=c7,                 /* mnemonic */
                code=i2,        /* internal rep */
                uncomp=i1,      /* 0 = fixed, +1=uncompressed, -1=compressed */
                dpb=i1,                 /* digits per byte; bcd=2, all others=1 */
                adj=i1          /* size adjustment; bcd=3, all others=0 */
              )
go
append to dtype_I (mne="bcdflt", code=35, uncomp= -1, dpb=2, adj=3) go
append to dtype_I (mne="bin",code=45, uncomp= -1, dpb=1, adj=0) go
append to dtype_I (mne="bcd",code=46, uncomp= -1, dpb=2, adj=3) go
append to dtype_I (mne="c",code=47, uncomp= -1, dpb=1, adj=0) go
append to dtype_I (mne="ubcdflt",code=35, uncomp= 1, dpb=2, adj=3) go
append to dtype_I (mne="ubin",code=45, uncomp= 1, dpb=1, adj=0) go
append to dtype_I (mne="ubcd",code=46, uncomp= 1, dpb=2, adj=3) go
append to dtype_I (mne="uc",code=47, uncomp= 1, dpb=1, adj=0) go
append to dtype_I (mne="i",code=48, dpb=1, adj=0) go
append to dtype_I (mne="i",code=52, dpb=1, adj=0) go
append to dtype_I (mne="i",code=56, dpb=1, adj=0) go
append to dtype_I (mne="f",code=57, dpb=1, adj=0) go
append to dtype_I (mne="f",code=60, dpb=1, adj=0) go

range of r is relation
replace r (type="I") where r.relid=rel_id("dtype_I") go
/*
** define "columns" view
*/
range of a is attribute
range of dt is dtype_I
range of r is relation
create view columns (attribute = a.name,
                     relid = a.relid,
                     relation = r.name,
                     type = concat(dt.mne,string(4,
                                          mod(256+((a.len * dt.dpb) - dt.adj),256)
                                            )
```

```
                                )
                        /* mod used to force length postive when > 127 */
            )
        where a.type = dt.code
        and ((dt.uncomp < 0 and a.offset < 0)
                        or
            (dt.uncomp >= 0 and a.offset >= 0)
            )
        and r.relid = a.relid
go

associate columns with "Friendly view of attributes relation","V1" go
```

objects (objects7.idl)

```
/*
** The view objects is used by the stored commands:
** rels, dir, mine, otherviews, pgms, files, othercmds
**
*/

destroy objects go  /* view upon which all of the above are based  */

destroy logged_I go

/*
** logged_I -- used to decode stat field to find if object is logged
*/

create logged_I (value=i2, text=c10) go

append to logged_I (value=0, text="Not Logged") go
append to logged_I (value=64, text="Logged") go

range of r is relation
replace r (type="I") where r.relid=rel_id("logged_I") go

/*
** create "objects" view
*/

range of r is relation
range of u is users
range of log is logged_I

create view objects (object = r.name,
                    relid = r.relid,
                    owner = u.name,
                    ownerid = r.owner,
                    tups = r.tups,
```

```
                type = r.type,
                logging = log.text
               )
        where r.owner *= u.id
        and log.value = mod(int2(r.stat)+int4(65536),128)
                      - mod(int2(r.stat)+int4(65536),64)
go
associate objects with
"Objects, their owners, type, tups, and if logged","V1" go
```

allindexes (indexes.idl)

```
/*
** Define "allindexes" command
*/
destroy allindexes;

define allindexes
retrieve (relation= r.name,
        type = it.desc,
        key1 = att_name(i.relid,int1(substring(4,1,i.keys))),
        key2 = att_name(i.relid,int1(substring(14,1,i.keys))),
        key3 = att_name(i.relid,int1(substring(24,1,i.keys))),
        key4 = att_name(i.relid,int1(substring(34,1,i.keys))),
        key5 = att_name(i.relid,int1(substring(44,1,i.keys))),
        i.indid,
        i.card
        )
        order by relation, indid
        where i.relid = r.relid
        and mod(i.stat,4) = it.type
        and r.type="U"
end define;

associate allindexes with
"Displays keys and type of indices for all user relations","1D";
```

attname (attname.idl)

```
/*
 * IDL to define "attname" command for ".all"
 * pseudo-attribute call from the parser
 */

destroy attname go
range of a is attribute
```

```
define attname
        retrieve (a.name) where a.relid = rel_id($relation)
end define
go
```

atts (atts.idl)

```
/* atts command */

destroy atts go

define atts

        range of col is columns
        retrieve (col.attribute,col.type) where col.relid = rel_id($relation)

end define
go

associate atts with
        "Displays attributes and types for given RELATION","1L" go
```

baudrate (channel.idl)

```
/*
*  channel command to determine serial channel baud rate
*/

create cbaud_I (value=i1,baud=c20);
append to cbaud_I(value=0,baud="9600");
append to cbaud_I(value=1,baud="Illegal");
append to cbaud_I(value=2,baud="150");
append to cbaud_I(value=3,baud="300");
append to cbaud_I(value=4,baud="600");
append to cbaud_I(value=5,baud="1200");
append to cbaud_I(value=6,baud="1800");
append to cbaud_I(value=7,baud="2400");
append to cbaud_I(value=8,baud="4800");
append to cbaud_I(value=9,baud="9600");
append to cbaud_I(value=10,baud="19200");
append to cbaud_I(value=11,baud="No communication");
append to cbaud_I(value=12,baud="Illegal");
append to cbaud_I(value=13,baud="Illegal");
append to cbaud_I(value=14,baud="Illegal");
append to cbaud_I(value=15,baud="Illegal");

replace r (type="I") where r.relid=rel_id("cbaud_I");
```

```
range of cb is cbaud_I
destroy baudrate go

define baudrate
retrieve (channel=c.number, baud=cb.baud)
        order by channel
        where c.type="S"
        and mod(c.value,16) = cb.value
end define;

associate baudrate with "Gives the serial channel bit/second rates","1M";
```

channel (channel.idl)

```
/*
*  channel command to decode the configure tuple bit strings
*/

destroy channel;
destroy channel_I;
destroy mask_I;
destroy cbaud_I;

range of r is relation

/* table of powers of two to 16 */

create mask_I(bit=i1,value=i4);

append to mask_I (bit=0,value=1);
range of mI is mask_I

append to mask_I (bit=max(mI.bit)+1,value=max(mI.value)*2);
append to mask_I (bit=max(mI.bit)+1,value=max(mI.value)*2);
append to mask_I (bit=max(mI.bit)+1,value=max(mI.value)*2);
append to mask_I (bit=max(mI.bit)+1,value=max(mI.value)*2);
append to mask_I (bit=max(mI.bit)+1,value=max(mI.value)*2);
append to mask_I (bit=max(mI.bit)+1,value=max(mI.value)*2);
append to mask_I (bit=max(mI.bit)+1,value=max(mI.value)*2);
append to mask_I (bit=max(mI.bit)+1,value=max(mI.value)*2);
append to mask_I (bit=max(mI.bit)+1,value=max(mI.value)*2);
append to mask_I (bit=max(mI.bit)+1,value=max(mI.value)*2);
append to mask_I (bit=max(mI.bit)+1,value=max(mI.value)*2);
append to mask_I (bit=max(mI.bit)+1,value=max(mI.value)*2);
append to mask_I (bit=max(mI.bit)+1,value=max(mI.value)*2);
append to mask_I (bit=max(mI.bit)+1,value=max(mI.value)*2);
append to mask_I (bit=max(mI.bit)+1,value=max(mI.value)*2);
append to mask_I (bit=max(mI.bit)+1,value=max(mI.value)*2);
```

```
append to mask_I (bit=max(mI.bit)+1,value=max(mI.value)*2);

replace r (type="I") where r.relid=rel_id("channel_I");

create channel_I (type=uc1,number=i1,bit=i1,meaning=c50);

append to channel_I(type="S",bit=4,meaning="DCD (Drop Carrier Detect)");
append to channel_I(type="S",bit=5,meaning="CTS (Clear to Send)");
append to channel_I(type="P",bit=5,meaning="No Timeout");
append to channel_I(type="P",bit=6,meaning="20 Second Timeout");
append to channel_I(type="S",bit=6,meaning="Cancel host");
append to channel_I(type="S",bit=10,meaning="Trustworthy hunames");
append to channel_I(type="S",bit=11,meaning="Nontrustworthy huids");
append to channel_I(type="P",bit=10,meaning="Trustworthy hunames");
append to channel_I(type="P",bit=11,meaning="Nontrustworthy huids");
append to channel_I(type="P",bit=12,meaning="1024 byte packet size");
append to channel_I(type="P",bit=13,meaning="512 byte packet size (256 if bit 12 is set)");
append to channel_I(type="S",bit=14,meaning="Cancel user output in 1 min");
append to channel_I(type="P",bit=14,meaning="Cancel user output in 1 min");
append to channel_I(type="P",bit=15,meaning="Cancel user output in 5 min 20 min if bit 14 set");
append to channel_I(type="S",bit=15,meaning="Cancel user output in 5 min 20 min if bit 14 set");
append to channel_I(type="E",bit=0,number=1,meaning="TCP Protocol");
append to channel_I(type="E",bit=10,number=1,meaning="Trustworthy hunames");
append to channel_I(type="E",bit=11,number=1,meaning="Nontrustworthy huids");
append to channel_I(type="E",bit=12,number=1,meaning="1024 byte packet size");
append to channel_I(type="E",bit=13,number=1,meaning="512 byte packet size (256 if bit 12 is set)"

replace r (type="I") where r.relid=rel_id("channel_I");

range of c is configure
range of ci is channel_I

define channel
retrieve (c.type, channel=c.number,bit_set=mI.bit,ci.meaning)
        order by type,channel
          where c.type=ci.type
          and (ci.number=0 or mod(c.number,8)=0)
          and mI.bit=ci.bit
          and mod(c.value/mI.value,2) = 1
end define;

associate channel with "Decodes the bits set in 'configure.value' "," 1M";
```

cmds (cmds5.idl)

```
destroy cmds go

/*
** Define "cmds" command.
*/

range of r is relation
range of d is descriptions

define cmds
retrieve unique(command = r.name,
                description = substring(1,60,d.text)
               )
        order by command, d.key
        where r.relid = d.relid
        and (r.owner = dba or r.owner = userid)
                and r.type = "C"
        and d.key = "1L"
end define
go

associate cmds with
        "Lists standard stored commands with a one line description","1L" go
```

cmds_date (cmds5.idl)

```
/*
** Define "cmds_date" command.
*/

destroy cmds_date go

range of r is relation
range of d is descriptions

define cmds_date
retrieve unique(command = r.name,
                description = substring(1,60,d.text)
               )
        order by command, d.key
        where r.relid = d.relid
        and (r.owner = dba or r.owner = userid)
        and r.type = "C"
        and d.key = "1T"
end define go

associate cmds_date with
```

```
"Lists stored commands relating to date conversion","1L" go
```

cmds_dba (cmds5.idl)

```
destroy cmds_dba go

/*
** Define "cmds_dba" command.
*/

range of r is relation
range of d is descriptions

define cmds_dba
retrieve unique(command = r.name,
                description = substring(1,60,d.text)
                )
        order by command, d.key
        where r.relid = d.relid
        and (r.owner = dba or r.owner = userid)
        and r.type = "C"
        and d.key = "1D"
end define go

associate cmds_dba with
        "Lists stored commands mostly used by the dba","1L" go
permit execute of cmds_dba go
```

cmds_permit (cmds5.idl)

```
destroy cmds_permit go

/*
** Define "cmds_permit" command.
*/

range of r is relation
range of d is descriptions

define cmds_permit
retrieve unique (command = r.name,
                 description = substring(1,60,d.text)
                )
        order by command, d.key
        where r.relid = d.relid
        and (r.owner = dba or r.owner = userid)
        and r.type = "C"
```

```
        and d.key = "1P"
end define go

associate cmds_permit with
        "Lists stored commands dealing protection of objects","1L" go
```

cmds_space (cmds5.idl)

```
destroy cmds_space go

/*
** Define "cmds_space" command.
*/

range of r is relation
range of d is descriptions

define cmds_space
retrieve unique (command = r.name,
        description = substring(1,60,d.text))
        order by command, d.key
        where r.relid = d.relid
        and (r.owner = dba or r.owner = userid)
        and r.type = "C"
        and d.key = "1S"
end define go

associate cmds_space with
        "Lists stored commands dealing with space/size/storage","1L" go
```

cmds_system (cmds_sys.idl)

```
/*
*  Define "cmds_system" command.
*/

destroy cmds_system go

define cmds_system
range of r is relation
range of d is descriptions
retrieve unique (command = r.name,
                 description = substring(1,60,d.text)
                )
        order by command, d.key
        where r.relid = d.relid
        and (r.owner = dba or r.owner = userid)
```

```
        and r.type = "C"
        and d.key = "1M"
end define

associate cmds_system with
        "Lists stored commands for monitoring performance","1L" go
```

config (config.idl)

```
/*
** "config" stored command:
**
** dumps the "configure" relation from the "system" database.
*/

destroy config go

/* "config_I" -- constant relation for "config" command. */

destroy config_I go

create config_I
                (type = uc1,
                 name = c40
                )
go

append config_I (type = "A", name = "Accelerator, value=WCS level") go
append config_I (type = "B", name = "Block multiplexer host interface") go
append config_I (type = "C", name = "Checkpoint interval") go
append config_I (type = "D", name = "Default char set 0=ASCII 1=EBCDIC") go
append config_I (type = "E", name = "Ethernet host interface") go
append config_I (type = "F", name = "Pages used for Track Buffer") go
append config_I (type = "I", name = "IDM I.D. for IDENTIFY response") go
append config_I (type = "K", name = "Memory configuration") go
append config_I (type = "L", name = "Passes for on-line dump") go
append config_I (type = "M", name = "Monitor interval(minutes)") go
append config_I (type = "P", name = "IEEE-488 host interface") go
append config_I (type = "R", name = "IDM software release") go
append config_I (type = "S", name = "RS-232 host interface") go
append config_I (type = "T", name = "Mag tape interface, value=4 for h/p") go
append config_I (type = "V", name = "Pages to promote locks") go

range of r is relation
replace r (type="I") where r.relid=rel_id("config_I") go

define config
range of c is configure
range of cx is config_I
retrieve (c.all,
```

```
        name = cx.name)
        order by c.type
        where c.type *= cx.type
end define go

associate config with
"Identify the tuples from the configure relation","1M" go
permit execute of config go
```

date (date8.idl)

All of the structures used by the **date** command and "DateAndTime" view are included to improve readability. The **date** command uses several data relations and views defined below. In addition, there are several stored command examples embedded in the scripts.

```
/* make sure that the values in the savings_I relation below are correct
  for the local time zone    */

destroy date, gmt_date go /* the commands for date and time  */

destroy DateAndTime go    /* the view that holds the current date and time */

destroy GMT go            /* the view that holds GMT current date and time */

destroy ymd go           /*  stored command returns idmdate given YYYYMMDD */

destroy dateconv go      /*  stored command to  convert idmdate to readable
                             form
                             */

destroy expire go       /* Updates relation.expire for given number of days */

destroy expiredate go    /*  Updates relation.expire to YYYYMMDD as idmdate  */

destroy number_I go      /* a relation containing tuples for
                            the numbers from 0 to 60, along with their two-
                            digit string representations.  The numbers from
                            0 to 23 also contain the day correction
                           */

destroy mon_I go         /* a view of month_I with the values corrected for
                            leap years                          */

destroy month_I go       /* a relation containing the first and last days,
                            name and number of the 12 months          */

destroy local_I go      /* a view of getdate to correct for the date
                               in the local time zone, based on julian_I */

destroy julian_I go     /* a view of the calendar relation containing the
```

```
                        information for the current year          */

destroy savings_I go    /* a relation that corrects for daylight savings time
                              and time zone                       */

destroy base1970 go     /*  set the base for calendar_I to Jan 1, 1970  */

destroy base1900 go     /*  set the base for calendar_I to Jan 1, 1900  */

destroy calendar_I go    /* a relation containing information needed to
                            convert to julian date and allow for leap years
                            for the years 1985 - 1992             */

destroy day_I go         /* a relation containing the 7 days of the week */

/*****************************************************************************/

range of r is relation   /* needed to change types of relations and cmds  */

        /* savings_I is the                              */
        /* displacement of hours from GMT and the change */
        /* julian dates for daylight savings time        */
        /* Check these tuples for local time zone        */
create savings_I (
                 first=i4,
                 last=i4,
                 displacement = i4
                ) go

replace r (type="I") where r.relid=rel_id("savings_I") go

append to savings_I ( first = 0, last = 75, displacement = 8) go
append to savings_I ( first = 76, last= 300, displacement = 7) go
append to savings_I ( first = 301, last= 366, displacement = 8) go

create calendar_I(
                 start=i4,          /* first day year since epoch */
                 shortyear=uc2,      /* YY form of year ie. 85 */
                 longyear=uc4,       /* YYYY form of year ie. 1984 */
                 leapyear=i4        /* 0 if not leap year, 1 if leap year */
                ) go

replace r (type="I") where r.relid=rel_id("calendar_I") go


append to calendar_I ( start=31045,shortyear="85",longyear="1985",leapyear=0) go
append to calendar_I ( start=31410,shortyear="86",longyear="1986",leapyear=0) go
append to calendar_I ( start=31775,shortyear="87",longyear="1987",leapyear=0) go
append to calendar_I ( start=32140,shortyear="88",longyear="1988",leapyear=1) go
append to calendar_I ( start=32506,shortyear="89",longyear="1989",leapyear=0) go
append to calendar_I ( start=32871,shortyear="90",longyear="1990",leapyear=0) go
append to calendar_I ( start=33236,shortyear="91",longyear="1991",leapyear=0) go
```

```
append to calendar_I ( start=33601,shortyear="92",longyear="1992",leapyear=1) go
append to calendar_I ( start=33967,shortyear="93",longyear="1993",leapyear=0) go
append to calendar_I ( start=34332,shortyear="94",longyear="1994",leapyear=0) go
append to calendar_I ( start=34697,shortyear="95",longyear="1995",leapyear=0) go

/*
** In case your IDM requires a different base date than 1900
** this command will reset the base date for the above conversion
** table (calendar_I) to Jan 1, 1970. Other dates may be used
** by the same method. Note there are 25,566 days from 1900 to 1970.
*/

range of cal is calendar_I

define base1970
        replace cal (start = cal.start -25566)
                    where cal.start > 25566
end define go

associate base1970 with
        "Change base for date conversion to Jan 1,1970","1T" go

/* This will reset from 1970 to 1900  */

define base1900
        replace cal (start = cal.start + 25566)
                    where cal.start < 25566
end define go

associate base1900 with
        "Change base for date conversion to Jan 1,1900 from 1970","1T" go

/* set up current year */
range of cal is calendar_I
range of sav is savings_I

create view julian_I(
                    start = cal.start,
                    cal.shortyear,
                    cal.longyear,
                    cal.leapyear,
                    sav.displacement
                    )
        where cal.start <= getdate
        and cal.start + 365 + cal.leapyear > getdate
        and sav.first <= getdate - cal.start
        and sav.last > getdate - cal.start
go

associate julian_I with
        "View of the current year in calendar_I", "V1" go
```

```
range of jul is julian_I

/*
** day_I -- used to give the day of the week.
*/

create day_I(number=i4,day=c9,shortday=uc3) go

replace r (type="I") where r.relid=rel_id("day_I") go

append to day_I(number=0,day="Monday",shortday="Mon") go
append to day_I(number=1,day="Tuesday",shortday="Tue") go
append to day_I(number=2,day="Wednesday",shortday="Wed") go
append to day_I(number=3,day="Thursday",shortday="Thu") go
append to day_I(number=4,day="Friday",shortday="Fri") go
append to day_I(number=5,day="Saturday",shortday="Sat") go
append to day_I(number=6,day="Sunday",shortday="Sun") go

/*
** month_I -- used to give the month
*/

create month_I( first=i4,
                last=i4,
                month=c9,
                month_num=uc2,
                leapfirst=i4,
                leaplast=i4
              ) go

replace r (type="I") where r.relid=rel_id("month_I") go

append to month_I(first=1,last=31,month="January",month_num="01",
                leapfirst =0,leaplast = 0) go
append to month_I(first=32,last=59,month="February",month_num="02",
                leapfirst =0,leaplast = 1) go
append to month_I(first=60, last=90, month="March",month_num="03",
                leapfirst =1,leaplast = 1) go
append to month_I(first=91, last=120, month="April",month_num="04",
                leapfirst =1,leaplast = 1) go
append to month_I(first=121, last=151, month="May",month_num="05",
                leapfirst =1,leaplast = 1) go
append to month_I(first=152, last=181, month="June",month_num="06",
                leapfirst =1,leaplast = 1) go
append to month_I(first=182, last=212, month="July",month_num="07",
                leapfirst =1,leaplast = 1) go
append to month_I(first=213, last=243, month="August",month_num="08",
                leapfirst =1,leaplast = 1) go
append to month_I(first=244, last=273, month="September",month_num="09",
                leapfirst =1,leaplast = 1) go
append to month_I(first=274, last=304, month="October",month_num="10",
                leapfirst =1,leaplast = 1) go
```

```
append to month_I(first=305, last=334, month="November",month_num="11",
                leapfirst =1,leaplast = 1) go
append to month_I(first=335, last=365, month="December",month_num="12",
                leapfirst =1,leaplast = 1) go

/*  view of month that corrects for leapyear on leapyears  */

range of month is month_I
create view mon_I (first = month.first + month.leapfirst * jul.leapyear,
                   last = month.last + month.leaplast * jul.leapyear,
                   month = month.month,
                   month_num = month.month_num
                  ) go

associate mon_I with
        "View of the months from the current year", "V1" go

/*
**   number_I relation --
**  useful for numeric conversions
**  especially date, hour, minute, and second conversions
**            also
**  Used to compensate for GMT date standard.
**  Basically this is an if statement.  For a
**  given hour of the day what do we have to
**  add (subtract) to the GMT date to get the
**  local date
*/

create number_I(number=i4,str=uc2) go

replace r (type="I") where r.relid=rel_id("number_I") go

append to number_I(number=0,str="00") go
append to number_I(number=1,str="01") go
append to number_I(number=2,str="02") go
append to number_I(number=3,str="03") go
append to number_I(number=4,str="04") go
append to number_I(number=5,str="05") go
append to number_I(number=6,str="06") go
append to number_I(number=7,str="07") go
append to number_I(number=8,str="08") go
append to number_I(number=9,str="09") go
append to number_I(number=10,str="10") go
append to number_I(number=11,str="11") go
append to number_I(number=12,str="12") go
append to number_I(number=13,str="13") go
append to number_I(number=14,str="14") go
append to number_I(number=15,str="15") go
append to number_I(number=16,str="16") go
append to number_I(number=17,str="17") go
append to number_I(number=18,str="18") go
```

```
append to number_I(number=19,str="19") go
append to number_I(number=20,str="20") go
append to number_I(number=21,str="21") go
append to number_I(number=22,str="22") go
append to number_I(number=23,str="23") go
append to number_I(number=24,str="24") go
append to number_I(number=25,str="25") go
append to number_I(number=26,str="26") go
append to number_I(number=27,str="27") go
append to number_I(number=28,str="28") go
append to number_I(number=29,str="29") go
append to number_I(number=30,str="30") go
append to number_I(number=31,str="31") go
append to number_I(number=32,str="32") go
append to number_I(number=33,str="33") go
append to number_I(number=34,str="34") go
append to number_I(number=35,str="35") go
append to number_I(number=36,str="36") go
append to number_I(number=37,str="37") go
append to number_I(number=38,str="38") go
append to number_I(number=39,str="39") go
append to number_I(number=40,str="40") go
append to number_I(number=41,str="41") go
append to number_I(number=42,str="42") go
append to number_I(number=43,str="43") go
append to number_I(number=44,str="44") go
append to number_I(number=45,str="45") go
append to number_I(number=46,str="46") go
append to number_I(number=47,str="47") go
append to number_I(number=48,str="48") go
append to number_I(number=49,str="49") go
append to number_I(number=50,str="50") go
append to number_I(number=51,str="51") go
append to number_I(number=52,str="52") go
append to number_I(number=53,str="53") go
append to number_I(number=54,str="54") go
append to number_I(number=55,str="55") go
append to number_I(number=56,str="56") go
append to number_I(number=57,str="57") go
append to number_I(number=58,str="58") go
append to number_I(number=59,str="59") go
append to number_I(number=60,str="60") go

/* This view corrects getdate to the local date based on the
** current local displacement from GMT
*/

range of jul is julian_I

create view local_I (
                    lgetdate = getdate
                        + (((gettime - jul.displacement*216000)
```

```
                        / abs(gettime - jul.displacement*216000)
                       ) - 1
                        ) / 2
                ) go

associate local_I with
        "View of getdate corrected for time zone", "V1"  go

/*
**  For string conversion of numbers whose preferred format is with a
**  preceding zero (as in the ninth of the month being YYMM09 or
**  minutes in HH:03:SS) the number_I relation is used rather than the
**  string function (which won't give preceding zeroes)
*/

range of mon is mon_I
range of day is day_I
range of local is local_I
range of jul is julian_I
range of da is number_I
range of n_day is number_I
range of n_min is number_I
range of n_secs is number_I

/*
**  define "DateAndTime" view.
*/
create view DateAndTime (
                        day = day.day,
                date_numeric = concat(jul.shortyear,concat(mon.month_num,n_day.str)),
                date_slashed = concat(jul.shortyear,
                                        concat("/",
                                            concat(mon.month_num,
                                              concat("/",n_day.str)
                                                )
                                             )
                                       ),
                date_written = concat(mon.month,
                                        concat(" ",
                                            concat(string(0, n_day.number),
                                              concat(",",
                                                concat(" ",jul.longyear)
                                                  )
                                                  )
                                              )
                                      ),
                Time = substring (1,10,
                                concat( string(0,
                                      mod((((gettime/60)/60)/60 + (24 - jul.displacement)),24)
                                       ),
                                            concat(":",
                                              concat(n_min.str,
```

```
                                                    concat(":",n_secs.str)
                                                      )
                                                )
                                            )
                                        )
                                ) /* end attribute list */

    /* jul.start is  year's first day */

    where mon.first <= local.lgetdate - jul.start
    and mon.last >= local.lgetdate - jul.start
    and n_day.number = local.lgetdate - jul.start - mon.first + 1
    and da.number = ((gettime/3600)/60)
    and n_min.number = mod(gettime/3600,60)
    and n_secs.number = mod(gettime/60,60)
    and day.number = mod((local.lgetdate),7) /* gets day of week */
go

associate DateAndTime with
        "Gives current date and time in various formats", "V1" go

/*
** date command
*/

define date
range of Date is DateAndTime
retrieve (
          Day = Date.day,
          Date = Date.date_written,
          Time = Date.Time
         )
end define
go

associate date with
        "Returns the current date and time from the database server","1T" go

create view GMT(
                day = day.day,
                date_numeric = concat(jul.shortyear,concat(mon.month_num,n_day.str)),
                date_slashed = concat(jul.shortyear,
                                concat("/",
                                    concat(mon.month_num,
                                      concat("/",n_day.str)
                                        )
                                )
                                ),
                date_written = concat(mon.month,
                                concat(" ",
                                    concat(string(0, n_day.number),
                                      concat(",",
```

```
                                        concat(" ",jul.longyear)
                                            )
                                        )
                                    )
                                ),
                    Time = substring (1,10,
                                concat( string(0,
                                            mod((((gettime/60)/60)/60 + 24),24)
                                            ),
                                    concat(":",
                                        concat(n_min.str,
                                            concat(":",n_secs.str)
                                            )
                                    )
                                )
                            )
                ) /* end attribute list */

    /* jul.start is magic for year's first day */

    where mon.first <= getdate-jul.start
    and mon.last >= getdate -jul.start
    and n_day.number = getdate - jul.start - mon.first + 1
    and n_min.number = mod(gettime/3600,60)
    and n_secs.number = mod(gettime/60,60)
    and day.number = mod(getdate,7) /* gets day of week */
go

associate GMT with
        "gives Greenwich date and time in various formats", "V1" go

/*
** gmt_date command
*/

define gmt_date
range of GMT is GMT
retrieve (
    Day = GMT.day,
    Date = GMT.date_written,
    Time = GMT.Time)
end define go

associate gmt_date "Returns Greenwich Mean Time and Date from the database server","1T" go

/*
**  dateconv command converst date from idmdate to readable form
*/

range of cal is calendar_I
range of n_day is number_I
range of month is month_I
```

```
define dateconv
retrieve(
                date_written = concat(month.month,
                                      concat(" ",
                                          concat(string(0, n_day.number),
                                            concat(",",
                                              concat(" ",cal.longyear)
                                                  )
                                            )
                                          )
                                      )
        )


  where month.first + month.leapfirst * cal.leapyear <= int4($idmdate) - cal.start
  and month.last + month.leaplast * cal.leapyear >= int4($idmdate) - cal.start
  and n_day.number = int4($idmdate) - cal.start - month.first + 1
                        - month.leapfirst * cal.leapyear
end define go

associate dateconv with
        "returns the month day, year given idmdate NUMBER", "1T" go

/*
**  Define ymd command to convert a string of the form "YYYYMMDD" to
**  an idmdate
*/

range of cal is calendar_I
range of month is month_I

define ymd
retrieve (idmdate = cal.start + month.first + int4(substring(7,2,$YYYYMMDD))
                  + cal.leapyear * month.leapfirst - 1
         )
          where cal.longyear = substring(1,4,$YYYYMMDD)
          and month.month_num = substring(5,2,$YYYYMMDD)
end define go

associate ymd with
        "Returns the idmdate given a date string: YYYYMMDD","1T" go
```

dbs (dbs.idl)

```
/*
**  The "dbs" command list databases, along with owner and status
*/


destroy dbs;
destroy dbs0_I;
create dbs0_I (value=i2, text=c46);

/*
**  For ASCII databases
*/

/*  'text can only be this long!!                    ' */
append to dbs0_I (value=1, text=
   "Database/ disks not on-line");
append to dbs0_I (value=5, text=
   "Database is being created or destroyed (ASCII)");
append to dbs0_I (value=9, text=
   "Locked: database marked unrecoverable (ASCII)");
append to dbs0_I (value=17, text=
   "Unused since boot-recovery not needed (ASCII)");
append to dbs0_I (value=33, text=
   "On-line (ASCII)");
append to dbs0_I (value=49, text=
   "Unused since boot, recovery had to run (ASCII)");
append to dbs0_I (value=65, text=
   "being loaded or rolled forward (ASCII)");
append to dbs0_I (value=69, text=
   "being created or destroyed (ASCII)");
append to dbs0_I (value=145, text=
   "LOCKED: unused since boot-no recovery need (A)");
append to dbs0_I (value=161, text=
   "LOCKED for dump/load/rollforward (on-line) (A)");
append to dbs0_I (value=177, text=
   "LOCKED: Unused since boot, recovery run(ASCII)");
append to dbs0_I (value=193, text=
   "LOCKED: being loaded or rolled forward (ASCII)");
append to dbs0_I (value=197, text=
   "LOCKED: being created or destroyed (ASCII)");
/*
**  for EBCDIC databases
*/
append to dbs0_I (value=3, text=
   "Database/ disks not on-line (EBCDIC)");
append to dbs0_I (value=7, text=
   "Database is being created or destroyed (EBCDIC)");
append to dbs0_I (value=11, text=
   "Database marked unrecoverable (EBCDIC)");
append to dbs0_I (value=19, text=
```

```
    "Unused since boot, no recovery needed (EBCDIC)");
append to dbs0_I (value=35, text=
    "On-line (EBCDIC)");
append to dbs0_I (value=51, text=
    "Unused since boot, recovery had to run (E)");
append to dbs0_I (value=67, text=
    "being loaded or rolled forward (EBCDIC)");
append to dbs0_I (value=71, text=
    "being created or destroyed (EBCDIC)");
append to dbs0_I (value=147, text=
    "LOCKED: unused since boot-no recovery need(E)");
append to dbs0_I (value=163, text=
    "LOCKED for dump/load/rollforward (on-line) (E)");
append to dbs0_I (value=179, text=
    "LOCKED: Unused since boot, recovery run (E)");
append to dbs0_I (value=195, text=
    "LOCKED: being loaded or rolled forward (E)");
append to dbs0_I (value=199, text=
    "LOCKED: being created or destroyed (EBCDIC)");

create clustered index on dbs0_I(value);

range of r is relation
replace r (type="I") where r.relid=rel_id("dbs0_I");

define dbs
range of db is databases
range of u is users
range of dbs0 is dbs0_I
retrieve unique (db.name,dbid=string(3,db.id), owner_name = u.name,
         status = dbs0.text)
         order by name
        where u.id =* db.owner
        and dbs0.value =* mod(int4(db.stat) + 65536 ,256)
end define
go
associate dbs with
"Displays databases on this shared database system","1M";
```

depend (depend.idl)

```
/*
** Define "depend" command.
*/

destroy depend go

range of cross is crossref
range of u is users
range of r is relation
```

```
define depend
retrieve unique (object = rel_name(cross.relid),
                 dependents = rel_name(cross.drelid),
                 downer = u.name)
        order by object
        where cross.relid = rel_id($object)
        and r.owner *= u.id
        and r.relid = cross.drelid
end define
go

associate depend with
        "Displays objects depending on OBJECT","1L" go
```

describe (describe.idl)

```
/*
** Define "describe" command
*/

destroy describe go

range of r is relation
range of d is descriptions

define describe
retrieve(description = substring(1,72,d.text))
        order by d.key
        where rel_id($object) = r.relid
        and d.relid = r.relid
        and d.attid = 0
end define
go

associate describe with
        "Displays description of given RELATION or COMMAND","1L" go

/*
  associate commands for descriptions of system relations
*/
associate relation
"relation:    Catalog of all objects in database. An object is a","91" go
associate relation
"             relation, view, file, stored command or stored pro-","92" go
associate relation
"             gram. Each tuple represents a single object.      ","93" go

associate attribute
"attribute:   Catalog of each attribute of each relation. Each  ","91" go
associate attribute
```

```
"           tuple represents one attribute.                    ","92" go

associate indices
"indices:   Catalog of indices that exist in database. Each    ","91" go
associate indices
"           tuple represents one index.                        ","92" go

associate protect
"protect:   Catalog of protection information for the database.","91" go
associate protect
"           Each tuple represents one type of access (e.g.,    ","92" go
associate protect
"           read, write, create index) for one user or group   ","93" go
associate protect
"           for all attributes of one view, relation, file, or ","94" go
associate protect
"           stored command.                                    ","95" go

associate query
"query:     Text of stored commands                            ","91" go

associate crossref
"crossref:  Catalog of dependencies among relations, views and ","91" go
associate crossref
"           stored commands.                                   ","92" go

associate transact
"transact:  Transaction logging relation                       ","93" go

associate batch
"batch:     Temporary transaction logging relation, used for   ","91" go
associate batch
"           transaction management so that even transactions   ","92" go
associate batch
"           against non-logged relations may be cancelled if   ","93" go
associate batch
"           needed.                                            ","94" go

associate descriptions
"descriptions: User definable descriptions, keyed to relation id's","91" go

associate users
"users:     Mapping of user and group names to user id         ","91" go
associate host_users
"host_users: Mapping from host id and user's id to IDM user id. ","91" go

associate blockalloc
"blockalloc: Catalog of disk blocks, showing relations assigned ","91" go
associate blockalloc
"           to blocks. Each tuple represents a block.          ","92" go

associate disk_usage
```

```
"disk_usage:  Shows relation and database allocation.        ","91" go
```

dir (objects7.idl)

```
/*
**  Define "dir" command to find an object by name
*/

destroy otype_I go

create otype_I (type=uc1,definition=c17) go
append to otype_I(type="S",definition = "System relation") go
append to otype_I(type="T",definition = "Transaction log") go
append to otype_I(type="U",definition = "User relation") go
append to otype_I(type="C",definition = "Stored command") go
append to otype_I(type="P",definition = "Stored program") go
append to otype_I(type="F",definition = "File") go
append to otype_I(type="I",definition = "Standard relation") go
append to otype_I(type="V",definition = "User view") go

replace r (type="I") where r.relid= rel_id ("otype_I") go


range of o is objects
range of ot is otype_I

destroy dir go

define dir
retrieve (o.relid,
          o.object,
          o.type,
          ot.definition,
          o.owner,
          o.tups
         )
        order by object,owner
        where o.type *= ot.type
        and o.object = $name
end define go

associate dir with
"Displays all objects of given NAME or fragment","1L" go
```

diskio (diskio.idl)

```
/*
** Find average disk access time from devmonitor
*/

destroy diskio go

range of dev is devmonitor
range of disk is disks

define diskio
     retrieve unique(disk.name,dev.slot,
                accesses= sum(dev.d2 by dev.d3,dev.slot where dev.type = "D" and dev.d2>0),
                time_ticks= sum(dev.d1 by dev.d3,dev.slot where dev.type ="D" and dev.d2>0),

        /* avg = time(msec) / accesses  */

                avg_ms= bcdflt(5,
                                bcdflt(5,sum(dev.d1 by dev.d3,dev.slot where dev.type="D" and dev.d2
                                        * 1000 / 60 /
                                bcdflt(5,sum(dev.d2 by dev.d3,dev.slot where dev.type="D" and dev.d2
                              )
                /* prevent division by zero */
            ) where dev.type="D" and dev.d2>0
                /* devmonitor identifies disks by low block*/
               and dev.d3 = disk.low
                /* only meaningful for physical disks */
                and (disk.type= "P"   or disk.type="M")
end define
go

associate diskio with "lists the on-line disks and their I/O activity","1M" go
```

expire (date8.idl)

```
/*  Set expiration date on an object to current date plus <N> days */

destroy expire go

range of r is relation

define expire
replace r (expire = getdate + $n_days)
        where r.relid = rel_id ($1object)
        and r.owner = userid
end define go

associate expire with
```

```
"sets the expiration date for an OBJECT to getdate + NDAYS","1T" go
```

expiredate (date8.idl)

```
/*
**  Define "expiredate" stored command to update relation.expire to
**      a given  YYYYMMDD
*/

destroy expiredate go

range of r is relation
range of cal is calendar_I
range of month is month_I

define expiredate
replace r (expire = cal.start + month.first + int4(substring(7,2,$YYYYMMDD))
                + cal.leapyear * month.leapfirst -1
          )
           where cal.longyear = substring(1,4,$YYYYMMDD)
           and month.month_num = substring(5,2,$YYYYMMDD)
           and r.relid= rel_id( $OBJECT)
           and r.owner = userid
end define go

associate expiredate with
        "sets the expiration date for an OBJECT to date YYYYMMDD","1T" go
```

files (objects7.idl)

```
/*
** define "files" command.
*/

range of o is objects
destroy files go

define files
retrieve unique (files = o.object,
                 o.owner,
                 o.logging
                )
        order by owner,files
        where o.type = "F"
        and ((o.ownerid = userid or o.ownerid = dba)
            or userid =dba /* show all relations for dba  */
          )
```

```
end define
go

associate files with "Displays files owned by you and the dba","1L"
go
```

freelog (freespc.idl)

```
/*
** Displays space usage for hard allocated log
*/
destroy freelog go

define freelog
retrieve (log_blocks = sum ( du.high - du.low + 1 where du.relid= 7),
          free_blks  = count(b.relid where b.relid = 7 and b.mode=32)
         )
end define
go

associate freelog with
"Displays space usage for hard allocated transact log","1S" go
```

freespace (freespc.idl)

```
/*
** Report freespace in database in blocks
*/

destroy freespace go

range of r is relation
range of b is blockalloc
range of du is disk_usage

define freespace
retrieve (total_blks = sum ( du.high - du.low + 1 ),
          free_blks  = count(b.relid where b.relid = 0),
          percent_used =
                      100 - ((count(b.relid where b.relid = 0) * 100) /
                   sum ( du.high - du.low + 1 ))
         )
end define
go

associate freespace with
"Displays total space and free space in this database","1S" go
```

groups (groups3.idl)

```
destroy groups go

range of u is users

/*  define 'groups' stored command.  Lists names and membership of groups. */

define groups
retrieve (u.id,
          u.gid,
          u.name,
          members = count(u.id by u.gid) - 1
        )
          where u.id = u.gid
          and u.id != 0
end define
go

associate groups with
"Lists the groups and membership for this database","1D" go
```

group_off (groups3.idl)

```
/*
**  Create temporary relations to stored removed users
**  Allows groups to be temporarily removed from host_users
**    and users, and restored later.
**  These are logged so we don't lose them.  They are not destroyed first
**  lest there already be some relations like these with members.  The
**  Create will therefore fail and give an error message when re-installed
*/

create host_users_I (
                     hid = i2,
                     huid = i4,
                     uid = i2
                    ) with logging go

create users_I (
                   id = i2,
                   gid = i2,
                   name = c12
                 ) with logging go

range of r is relation
replace r (type="I") where
r.relid=rel_id("host_users_I") or r.relid=rel_id("users_I") go
```

```
destroy group_off go

range of hu is host_users
range of u is users
range of u1 is users

/* Remove database access to group */
define group_off

        /* First move group member tuples to safe place */

        append to users_I (id=u.id,
                           gid=u.gid,
                           name=u.name
                          )
                where u1.name = $group
                and u1.id = u1.gid
                and u1.gid = u.gid

        append to host_users_I (hid=hu.hid,
                                huid=hu.huid,
                                uid=hu.uid
                               )
                where u1.name = $group
                and u1.id = u1.gid
                and u1.gid = u.gid
                and u.id = hu.uid

        /* Now delete group tuples from host_users and users */

        delete hu where u1.name = $group
                and u1.id = u1.gid
                and u1.gid = u.gid
                and u.id = hu.uid

        delete u where u1.name = $group
                and u1.id = u1.gid
                and u1.gid = u.gid

end define
go

associate group_off with
        "Turn off all access to database for GROUP - for DBA only", "1D" go
```

group_on (groups3.idl)

```
destroy group_on go

/* Restore database access to group */

range of huI is host_users_I
range of uI is users_I
range of u1I is users_I

define group_on

        /* First restore the group members to users and host_users */

        append to users (id=uI.id,
                         gid=uI.gid,
                         name=uI.name
                        )
                where u1I.name = $group
                and u1I.id = u1I.gid
                and u1I.gid = uI.gid

        append to host_users (hid=huI.hid,
                              huid=huI.huid,
                              uid=huI.uid
                             )
                where u1I.name = $group
                  and u1I.id = u1I.gid
                  and u1I.gid = uI.gid
                  and uI.id = huI.uid

        /* Then delete group member tuples from host_users_I and users_I */

        delete huI where u1I.name = $group
                and u1I.id = u1I.gid
                and u1I.gid = uI.gid
                and uI.id = huI.uid
        delete uI where u1I.name = $group
                and u1I.id = u1I.gid
                and u1I.gid = uI.gid
end define
go

associate group_on with
        "Turn GROUP database access back on - for DBA only", "1D" go
```

indexes (indexes.idl)

```
/*
** Install Indexes stored command to decode indices relation
**
*/

destroy indexes go
destroy itype_I go

/*
** Build "itype_I" for "indexes" command
*/

create itype_I(type=i1, desc=c17)  go

append itype_I(type = 1, desc="unique noncluster") go
append itype_I(type = -3, desc="unique noncluster") go
append itype_I(type = 0, desc="nonclustered") go
append itype_I(type = 2, desc="clustered") go
append itype_I(type = -2, desc="clustered") go
append itype_I(type = -1, desc ="unique clustered") go
append itype_I(type = 3, desc ="unique clustered") go

range of r is relation
replace r (type="I") where r.relid=rel_id("itype_I") go

/*
** Define "indexes" command
*/
range of it is itype_I
range of i is indices
range of r is relation

define indexes
retrieve (type = it.desc,
          key1 = att_name(i.relid,int1(substring(4,1,i.keys))),
          key2 = att_name(i.relid,int1(substring(14,1,i.keys))),
          key3 = att_name(i.relid,int1(substring(24,1,i.keys))),
          i.indid,
          i.card
         )
        order i.indid
        where i.relid = rel_id($relation)
        and mod(i.stat,4) = it.type
end define
go

associate indexes with
"Displays keys and type of indices for given RELATION","1D" go
```

mine (objects7.idl)

```
/*
**  Define "mine" command to find an object owned by me
*/

destroy mine go

define mine
        retrieve (o.relid,
                  o.object,
                  o.type,
                  ot.definition,
                  o.tups
                  )
                 order by type,object
                 where o.type *= ot.type
                 and o.ownerid = userid
end define;

associate mine with
"Displays all objects owned by this user","1L";
```

mon (mon5.idl)

```
/*
** "mon" command--cpu usage for the last N monitor intervals.
*/

destroy mon go

range of m is monitor

define mon
retrieve

        ( CPU_activity    = "avg monitoring interval (minutes):",
        time = bcdfixed(5,2,avg(bcdflt(0,m.length))/3600)
        )

retrieve
        ( SEQNO     = m.seqno,
          CPU_percent     = bcdflt(5,100 * bcdflt(5,m.cpu)/bcdflt(5,m.length)),
          DAC_percent     = bcdflt(5,100 * bcdflt(5,m.dac)/bcdflt(5,m.length)),
          IDLE_percent    = bcdflt(5,100 * bcdflt(5,m.idle)/bcdflt(5,m.length)),
          COMMANDS  = int2(m.cmnds),
          AVG_seconds =bcdflt(5,bcdflt(5,m.avgcmnd)/60)
        )
order by m.date,m.time
```

```
where (m.date=getdate  and m.time >gettime - m.length * int4($INTERVALS))
  or (m.date=getdate-1 and m.time> gettime - m.length * int4($INTERVALS) +
                                  (int4(24) * int4(60) * int4(3600))
    )

end define go

associate mon with "Monitor CPU and DAC usage for last N monitor intervals","1M"
go
```

mondisk (mon5.idl)

```
/*
** "mondisk" command--disk usage for the last N monitor intervals.
*/

destroy mondisk go

range of m is monitor
define mondisk
retrieve
        ( DISK_active   = "avg monitoring interval (minutes):",
          time = bcdfixed(5,2,avg(bcdflt(0,m.length))/3600)
        )
retrieve(
          SEQNO          = m.seqno,
          READS          = m.reads,
          WRITES         = m.writes,
          HITS           = m.hits,
          DISK_WT_sec    = bcdflt(5,bcdflt(5,m.diskwait) / 60)
        )
order by m.date,m.time
where (m.date=getdate  and m.time >gettime - m.length * int4($INTERVALS))
  or (m.date=getdate-1 and m.time> gettime - m.length * int4($INTERVALS) +
                                  (int4(24) * int4(60) * int4(3600))
    )

end define go

associate mondisk with "Monitor disk usage for last N monitor intervals","1M"
go
```

monfail (mon5.idl)

```
/*
** "monfail" command--Time processing or processes are suspended for lack
**   of available memory resources
*/

destroy monfail go

range of m is monitor
define monfail
retrieve(
         SUSPENDED    = "avg monitoring interval (minutes):",
         time = bcdfixed(5,2,avg(bcdflt(0,m.length))/3600)
        )
retrieve
        (SEQNO        = m.seqno,
         INDELAY_pct  = bcdflt(5,100 * bcdflt(5,m.indelay)/bcdflt(5,m.length)),
         OUTDELAY_pct = bcdflt(5,100 * bcdflt(5,m.outdelay)/bcdflt(5,m.length)),
         MEMDELAY_pct = bcdflt(5,100 * bcdflt(5,m.memloss)/bcdflt(5,m.length)),
         DBINFAILS    = m.dbinfails
        )
        order by m.date,m.time
        where (m.date=getdate and m.time >gettime - m.length * int4($INTERVALS))
         or (m.date=getdate-1 and m.time> gettime - m.length * int4($INTERVALS)
            + (int4(24) * int4(60) * int4(3600))
          )
end define go

associate monfail with
      "Suspended processing for lack of memory from monitor","1M" go
```

monlock (mon5.idl)

```
/*
** "monlock" command--blocked wait, deadlocks, current locks
*/

destroy monlock go
destroy lockdef_I go

/*  interpretation of lock types  */

create lockdef_I(code = ubin1, meaning = c36) go

append to lockdef_I(code=binary(1), meaning ="read lock") go
append to lockdef_I(code=binary(2), meaning ="write lock") go
append to lockdef_I(code=binary(3), meaning ="read, intend to set block write lock") go
append to lockdef_I(code=binary(5), meaning ="intend to set block read lock") go
```

```
append to lockdef_I(code=binary(6), meaning ="intend to set block write lock") go
deny write of lockdef_I go

range of r is relation
replace r(type="I") where r.relid=rel_id("lockdef_I") go

range of lock is lock
range of lockdef is lockdef_I
range of db is databases
range of ds is dbinstat
range of m is monitor

define monlock

/*  first define the interval length  */

retrieve(
        LOCKS_BLOCKS    = "avg monitoring interval (minutes):",
       time = bcdfixed(5,2,avg(bcdflt(0,m.length))/3600)
       )
/* history of locks and deadlocks over last N intervals  */

retrieve(
        SEQNO     = m.seqno,
       DEADLOCKS       = m.deadlocks,
       LOCK_sec        = bcdflt(5,bcdflt(5,m.blockwait) / 60)
       )
       order by m.date,m.time
       where (m.date=getdate and m.time >gettime - m.length * int4($INTERVALS))
       or (m.date=getdate-1 and m.time> gettime - m.length * int4($INTERVALS)
            + (int4(24) * int4(60) * int4(3600))
        )

/* list current locks */

retrieve (
         DBIN = ds.dbin,
         DATABASE=db.name,
                /* this is the relid in the database, not system */
         RELID=lock.dnum,
         block=lock.pid,
         lock_type=lockdef.meaning
        )
        where ds.xactid=lock.tnum
        and ds.dbid=db.id
        and lockdef.code =* lock.type

end define go

associate monlock with
       "Current locks and history for last N monitor intervals","1M" go
```

monun (mon5.idl)

```
/*
** "monun" command--unused memory resources for the last N intervals.
*/

destroy monun go

range of m is monitor
define monun
retrieve(
        BUFFER_SPACE    = "avg monitoring interval (minutes):",
        time = bcdfixed(5,2,avg(bcdflt(0,m.length))/3600)
       )
retrieve
       ( SEQNO     = m.seqno,
        UNUSED_INPUT    = m.unin,
        UNUSED_OUT      = m.unout,
        UNUSED_MEM      = m.unmem,
        UNUSED_DBIN     = m.undbin
       )
       order by m.date,m.time
       where (m.date=getdate and m.time >gettime - m.length * int4($INTERVALS))
       or (m.date=getdate-1 and m.time> gettime - m.length * int4($INTERVALS)
           + (int4(24) * int4(60) * int4(3600))
        )
end define go

associate monun with
"Unused memory buffers for last N monitor intervals","1M" go
```

monwait (mon5.idl)

```
/*
** "monwait" command--wait queues from monitor for the last N intervals.
*/

destroy monwait go

range of m is monitor
define monwait
retrieve(
        WAIT_QUEUES     = "avg monitoring interval (minutes):",
        time = bcdfixed(5,2,avg(bcdflt(0,m.length))/3600)
       )
retrieve(
        SEQNO     = m.seqno,
        INPUT_sec       = bcdflt(5,bcdflt(5,m.inwait) / 60),
        OUTPUT_sec      = bcdflt(5,bcdflt(5,m.outwait) / 60),
```

```
        CPU_sec         = bcdflt(5,bcdflt(5,m.cpuwait) / 60),
        MEMORY_sec      = bcdflt(5,bcdflt(5,m.memwait) / 60),
        TAPE_sec        = bcdflt(5,bcdflt(5,m.tapewait) / 60)
      )
      order by m.date,m.time
      where (m.date=getdate and m.time >gettime - m.length * int4($INTERVALS))
      or (m.date=getdate-1 and m.time> gettime - m.length * int4($INTERVALS)
        + (int4(24) * int4(60) * int4(3600))
        )
end define go

associate monwait with
      "Summary of wait queues for last N monitor intervals","1M" go
```

notowned (notowned.idl)

```
/*
** Define "notowned" command.
**      Finds objects without owners in users relation
*/

destroy notowned go

range of r is relation
range of u is users

define notowned
      retrieve unique (r.name,
                       r.relid,
                       owner_num = r.owner
                      )
            order by owner_num, name
            where 0 = count (u.id by r.relid where u.id = r.owner)
end define
go

associate notowned with
      "List objects which are not not owned by people in 'users'","1D" go
```

othercmds (objects7.idl)

```
/*
** define "othercmds" command.
*/

range of o is objects
range of d is descriptions
```

```
destroy othercmds go

define othercmds
retrieve unique (command = o.object,
                 o.owner,
                 o.logging
                )
        order by owner, command
        where o.type = "C"
        and ((o.ownerid = userid or o.ownerid = dba)
            or userid =dba /* show all relations for dba  */
         )
                /* exclude all standard commands  */
        and 0 = any(d.key by o.relid
                        where o.relid = d.relid
                        and (     d.key = "1M"
                                or d.key = "1L"
                                or d.key = "1P"
                                or d.key = "1D"
                                or d.key = "1S"
                                or d.key = "1T"

                             )
                    )
end define
go

associate othercmds with
"Displays user stored commands (those not displayed by cmds)","1L" go
```

otherviews (objects7.idl)

```
/*
** define "otherviews" command.
*/

range of o is objects
range of d is descriptions
destroy otherviews go

define otherviews
retrieve unique (views = o.object,
                 o.owner,
                 o.logging)
        order by owner,views
        where o.type = "V"
        and ((o.ownerid = userid or o.ownerid = dba)
            or userid =dba /* show all relations for dba  */
         )
                /* exclude any standard views     */
```

```
        and 0 = any(d.key by o.relid
                        where o.relid = d.relid
                        and d.key = "V1"
                    )
end define
go

associate otherviews with
"Displays user views (those not displayed by views)","1L" go
```

permits (protect5.idl)

```
destroy permits go
destroy ptype_I go

/*  If there is no user tuple for "all" with id=0, put one there */

range of u is users
append to users (id = 0,name = "ALL")
        where any(u.id where u.id = 0) = 0 go

/*
** ptype_I relation contains the decoding of the access attribute
**   of the protect system relation
*/
create ptype_I(access = i1, desc = c20) go

append to ptype_I(access = 1, desc = " read") go
append to ptype_I(access = 2, desc = " write") go
append to ptype_I(access = 3, desc = " all") go
append to ptype_I(access = -32, desc = " execute") go
append to ptype_I(access = -53, desc = " create database") go
append to ptype_I(access = -58, desc = " create") go
append to ptype_I(access = -56, desc = " create index") go
append to ptype_I(access = 4, desc = " read tape") go
append to ptype_I(access = 8, desc = " write tape") go
append to ptype_I(access = 12, desc = " all tape") go

create unique clustered index on ptype_I (access) go

range of r is relation
replace r (type="I") where r.relid=rel_id("ptype_I") go


/*
**   atype_I decodes the bit pair for permit/deny in protect.attmap
*/

destroy atype_I go
```

```
create atype_I(access = i1, desc = c8) go

append to atype_I(access = 1, desc = "permit") go
append to atype_I(access = 2, desc = "deny") go
append to atype_I(access = 3, desc = "BOTH!") go

range of r is relation
replace r (type="I") where r.relid=rel_id("atype_I") go

/*
** Permits command
** Returns explicit permissions on a given OBJECT
*/

range of p is protect
range of at is atype_I
range of pt is ptype_I
range of u is users

define permits

retrieve (access = concat(at.desc, pt.desc),
          object = $object,
          user = u.name
         )
        where at.access = mod(int1(substring(1,1,p.attmap)), 4)
        and pt.access = p.access
        and rel_id($object) = p.relid
        and u.id = p.user
        and rel_id($object) > 0
end define go

associate permits with "shows the explicit protection for an OBJECT","1P" go
```

permitsall (protect5.idl)

```
/*
** Permitsall command
** Returns all explicit permissions from the protect relation
*/

destroy permitsall go

define permitsall
retrieve(access = concat(at.desc, pt.desc),
         object = r.name,
         user = u.name
        )
        where at.access = mod(int1(substring(1,1,p.attmap)), 4)
        and pt.access = p.access
```

```
        and p.relid = r.relid
        and u.id = p.user
end define
associate permitsall with "shows all explicit permits and denies","1P" go
```

permitsgen (protect5.idl)

```
/*
** Permitsgen command
** Returns explicit tape and create permissions
*/

destroy permitsgen go

define permitsgen
retrieve(access = concat(at.desc, pt.desc),
         user = u.name
        )
        where at.access = mod(int1(substring(1,1,p.attmap)), 4)
        and pt.access = p.access
        and p.relid = 0
        and u.id = p.user
end define
go

associate permitsgen with "shows the explicit tape and create permissions","1P"
go
```

permitsme (permits5.idl)

```
/*
** Permitsme
** Returns explicit permissions on all objects for this user
*/

destroy permitsme go

define permitsme
retrieve(access = concat(at.desc, pt.desc),
         object = r.name
        )
        where at.access = mod(int1(substring(1,1,p.attmap)), 4)
        and pt.access = p.access
        and p.relid = r.relid
        and u.id = p.user
        and u.id = userid
end define go
```

```
associate permitsme with "shows the explicit permissions for me","1P" go
```

permitsuser (protect5.idl)

```
/*
** Permitsuser command
** Returns explicit permissions on all objects for a user
*/

destroy permitsuser go

define permitsuser
retrieve(access = concat(at.desc, pt.desc),
         object = r.name
        )
        where at.access = mod(int1(substring(1,1,p.attmap)), 4)
        and pt.access = p.access
        and p.relid = r.relid
        and u.id = p.user
        and u.name= $username
end define

associate permitsuser with "shows the explicit permissions for  USER","1P" go
```

pgms (objects7.idl)

```
/*
** define "pgms" command.
*/

range of o is objects

destroy pgms go

define pgms
retrieve unique (pgms = o.object,
                 o.owner,
                 o.logging
                )
        order by owner,pgms
        where o.type = "P"
        and ((o.ownerid = userid or o.ownerid = dba)
            or userid =dba /* show all relations for dba  */
           )
end define
go
```

```
associate pgms with "Displays stored programs owned by you or the dba","1L" go
```

ps (ps.idl)

```
/*
**  "ps" command:
**   Without an outer join, the maintenance port listener process
**   will not be reported.
*/

destroy ps go              /* Destroy stored command...   */
destroy ps_data_I go       /* ... & constant relation.    */

create ps_data_I           /* Create constant relation,   */
                           /* for translating codes from  */
                           /* "dbinstat.status" into      */
                           /* strings.                    */

        (status = i1,      /* status code, matching that  */
                           /* from "dbinstat.status"      */

         meaning= c29 /* string description of status.*/
        )
        go

range of r is relation
replace r (type ="I") where r.relid=rel_id("ps_data_I") go

/* Now populate the constant relation. */
range of ps is ps_data_I
append to ps_data_I (status = 1, meaning = "UNDEFINED") go
append to ps_data_I(status = ps.status + max(ps.status), meaning= "UNDEFINED") go
append to ps_data_I(status = ps.status + max(ps.status), meaning= "UNDEFINED") go
append to ps_data_I(status = ps.status + max(ps.status), meaning= "UNDEFINED") go
append to ps_data_I(status = ps.status + max(ps.status), meaning= "UNDEFINED") go
append to ps_data_I(status = ps.status + max(ps.status), meaning= "UNDEFINED") go
append to ps_data_I(status = ps.status + max(ps.status), meaning= "UNDEFINED") go
delete ps where ps.status > 50 go

replace ps (meaning= "runnable")  where ps.status = 1 go
replace ps (meaning= "wait--input; no transaction")  where ps.status = 2 go
replace ps (meaning= "wait--input in a transaction")  where ps.status = 3 go
replace ps (meaning= "DEBUGGING (timed out)")  where ps.status = 4 go
replace ps (meaning= "DEBUGGING")  where ps.status = 5 go
replace ps (meaning= "DEBUGGING")  where ps.status = 6 go
replace ps (meaning= "wait--output to drain")  where ps.status = 7 go
replace ps (meaning= "DEBUGGING")  where ps.status = 8 go
replace ps (meaning= "suspended--checkpoint")  where ps.status = 9 go
replace ps (meaning= "terminated abnormally")  where ps.status = 10 go
replace ps (meaning= "wait--tape controller")  where ps.status = 11 go
```

```
replace ps (meaning= "wait--child to terminate") where ps.status = 12 go
replace ps (meaning= "wait--special disk command") where ps.status = 13 go
replace ps (meaning= "DEBUGGING") where ps.status = 14 go
replace ps (meaning= "wait--maint port input") where ps.status = 18 go
replace ps (meaning= "wait--transaction lock") where ps.status = 30 go
replace ps (meaning= "wait-interruptible quick lock") where ps.status = 31 go
replace ps (meaning= "wait-non-interrupt quick lock") where ps.status = 32 go
replace ps (meaning= "wait--one disk I/O") where ps.status = 35 go
replace ps (meaning= "wait--multiple disk I/O's") where ps.status = 36 go
replace ps (meaning= "wait--process memory") where ps.status = 40 go
replace ps (meaning= "wait-process memory has input") where ps.status = 41 go
replace ps (meaning= "normal exit, draining output") where ps.status = 42 go

define ps
   range of ds is dbinstat
   range of db is databases
   range of u is users
   range of hu is host_users
   retrieve unique
                  (ds.dbin,
                   STATUS= ps.meaning,
                   block=int4(ds.block),
                   ds.time,
                   DATABASE = db.name
                  )
       order by dbin
       where ds.dbid *= db.id
       and mod(ds.status,50) = ps.status  /* ignore CANCEL STATUS */

   retrieve unique(ds.dbin, USER = u.name, ds.hid, ds.huid)
      order by USER
      where u.id    = hu.uid
        and hu.hid  =* ds.hid
        and hu.huid =* ds.huid
end define  go

associate ps with "Lists the active processes and status--known users","1M" go
```

rels (objects7.idl)

```
/*
** define "rels" command.
*/

define rels
range of o is objects
retrieve unique (relation = o.object,
                 o.owner,
                 o.tups,
                 o.logging
```

```
            )
        order by owner,relation
        where o.type = "U"
        and ((o.ownerid = userid or o.ownerid = dba)
            or userid =dba /* show all relations for dba  */
           )
end define
go

associate rels with
"Displays user relations owned by you or the dba","1L" go
```

rename (rename.idl)

```
/*
** Rename permits users to change
** the name of relations they own
*/

destroy rename go

range of r is relation

define rename
replace r (name=$newname)
      where r.name=$curr_name
     and r.owner = userid
end define
go

associate rename with
      "Allows users to rename their RELATION to NEW_NAME","1L" go
```

rmuser (rmuser.idl)

```
/*
**  Define "rmuser" command - remove user
*/

destroy rmuser go

range of hu is host_users
range of u is users

define rmuser
      delete hu where u.name = $username
      and hu.uid = u.id
```

```
        and u.id != dba

        delete u where u.name = $username and u.id != dba
end define
go

associate rmuser with
        "Removes a USER from database - for DBA only", "1D" go
```

setcard (setcard.idl)

```
/*
**      setcard command for set cardinality of an index
*/

destroy setcard go

range of i is indices

define setcard
         replace i (card= $3card)
                 where i.relid = rel_id($1relation)
                   and i.indid = $2indexid
end define
go

associate setcard with
        "Set cardinality for RELATION, INDEXID, CARD","1D" go
```

size (size.idl)

```
/*
**  Report sizes of objects in blocks
*/

destroy size go

range of r is relation
range of b is blockalloc

define size
retrieve (r.relid,
          rel_tups=r.tups,
          rel_pages=r.pages,
          num_blocks  = count(b.relid where b.relid = rel_id($object))
         )
        where r.relid=rel_id($object)
```

```
        and rel_id($object) > 0
end define
go

associate size with
        "Displays size of RELATION","1S" go
```

sizebyzone (szbyzone.idl)

```
/*
** Define "sizebyzone" command.
**      Lists the fragmentation of any relation
*/

destroy sizebyzone go

range of b is blockalloc

define sizebyzone
retrieve (zone=substring(1,3,b.tid),
          blocks=count(b.relid by substring(1,3,b.tid)
                       where b.relid = rel_id($object)
                      )
         )
        where 0!=count(b.relid by substring(1,3,(b.tid))
                           where b.relid = rel_id($object)
                      )
       and rel_id($object) > 0
end define
go

associate sizebyzone with
        "For OBJECT, lists number of blocks in each zone","1S" go
```

sizes (sizes.idl)

```
/*
** Report sizes of objects in blocks
*/

destroy sizes go

range of r is relation
range of b is blockalloc

define sizes
retrieve (total_blks = count(b.relid),
```

```
            free_blks  = count(b.relid where b.relid = 0),
            percent_used =
                    (count(b.relid where b.relid != 0) * 100) / count(b.relid)
           )

retrieve (r.name,
            r.owner,
            r.type,
            rel_pages =r.pages,
            num_of_blks = count(b.relid by b.relid)
           )
         order by num_of_blks:d
         where b.relid = r.relid
         and count(b.relid by b.relid) > 10
end define
go

associate sizes with
"Displays size of this database and its larger relations","1S" go
```

spacebyuser (spbyuser.idl)

```
/*
** Define "spacebyuser" command.
** Lists space allocated to each user in the database
*/

destroy spacebyuser go

range of b is blockalloc
range of u is users
range of r is relation

define spacebyuser

retrieve (user = u.name,
          user_id = u.id,
            pages = sum(count(b.mode by b.relid) by r.owner
                        where b.relid=r.relid
                       )
         )
          order by pages:d
          where r.owner = u.id
end define
go

associate spacebyuser with
          "Lists disk space utilization by user","1S" go
```

system (system.idl)

```
/*
** Define "system" command.
**        Retrieves info about the system relations
*/

destroy system go

range of r is relation

define system
retrieve unique (relation = r.name,
                 r.type,
                 tuples = r.tups,
                 r.pages
                )
        order by relation
        where r.type = "S" or r.type = "T"
                /* S = system, T = transaction logs */
end define
go

associate system with
        "Displays system relation names and sizes","1D" go
```

usersright (uright.idl)

```
/*
** Define "usersright" command.
** Checks for consistency in users and host_users
*/

destroy usersright go

range of hu is host_users
range of u is users
range of u1 is users

define usersright
        retrieve unique (hu.hid,
                         hu.huid,
                         hu.uid,
                         comment= "In 'host_users' not in 'users'"
                        )
                where 0 = any (u.id by hu.uid where u.id = hu.uid)

        retrieve unique (u.name,
                         u.id,
```

```
                    u.gid,
                    comment= "In 'users' not in 'host_users'"
                   )
            where 0 = any (hu.uid by u.id where u.id = hu.uid)
            and u.gid != u.id /* exclude groups */

        retrieve unique (u.name,
                    u.id,
                    u.gid,
                    comment= "Group--no members or user--no group"
                   )
            where 1 >= count (u1.gid by u.gid where u.id = u1.id)
            and u.name != "ALL"

        retrieve unique (u.name,
                    u.id,
                    u.gid,
                    comment= "Duplicate id's in 'users'"
                   )
            where  count (u.id by u.id) > 1

        retrieve unique (u.name,
                    u.id,
                    u.gid,
                    comment= "Inappropriate use of uid = 0"
                   )
            where u.id = 0
            and u.name != "ALL"

        retrieve unique (hu.hid,
                    hu.huid,
                    hu.uid,
                    comment="Inappropriate use of uid = 0"
                   )
            where hu.uid = 0

        retrieve unique (hu.hid,
                    hu.huid,
                    hu.uid,
                    comment= "DBA for this database"
                   )
            where hu.uid = dba and count(hu.uid where hu.uid=dba) > 1
end define
go

associate usersright with
        "Checks consistency of 'users' with 'host_users'","1D" go
```

uses (uses.idl)

```
/*
** Stored command to give the list of objects a named object depends on
** Usage is "uses <objectname>"
*/

destroy uses go

range of cross is crossref
range of r is relation
range of u is users

define uses
        retrieve (relationused = rel_name(cross.relid),
                  owner = u.name
                 )
        where cross.drelid = r.relid
        and r.relid = rel_id ($object)
        and u.id =* r.owner
end define go

associate uses "list objects that OBJECT depends on","1L" go
```

views (views.idl)

```
/*
** Define "views" command.
**        This lists the standard views just like cmds lists standard cmds
*/

destroy views go

range of r is relation
range of d is descriptions

define views
retrieve(views = r.name,
         description = substring(1,60,d.text)
        )
        order by views, d.key
        where r.relid = d.relid
        and (r.owner = dba or r.owner = userid)
        and r.type = "V"
        and d.key = "V1"
end define
go
associate views with
        "Lists standard views with a one line description","1L" go
```

who (who3.idl)

```
/*
** Define "who" command
*/

destroy who go

range of hu is host_users
range of ul is users
range of u is users

define who
retrieve(user = u.name,
         group = ul.name,
         hst_id = hu.hid,
         host_user_id = hu.huid,
         id = u.id
        )
        order by group,user
        where hu.uid = u.id
        and ul.id =* u.gid /* outer join to get groupless users */
        and u.id != u.gid
end define
go

associate who with
      "Lists users who can use this database", "1D" go
```

whois (who3.idl)

```
/* Define whois command  to find userid from name  */

destroy whois go

define whois
retrieve(
         id = u.id,
         group = ul.name,
         hst_id = hu.hid,
         host_user_id = hu.huid
        )
        where hu.uid = u.id
        and ul.id =* u.gid /* outer join to get groupless users */
        and u.name = $username
end define
go

associate whois with
```

```
        "Returns userid, host_id and huid for a given username", "1D" go
```

whoisid (who3.idl)

```
/*  Define whoisid command to find username from userid */

destroy whoisid go

define whoisid
retrieve(
         u.name,
         group = u1.name,
         hst_id = hu.hid,
         host_user_id = hu.huid
        )
        where hu.uid = u.id
        and u1.id =* u.gid /* outer join to get groupless users */
        and u.id = $user_id
end define
go

associate whoisid with
        "Returns user name, host_id and huid for a given user_id", "1D" go
```

Appendix C: SQL Definitions

This appendix lists the SQL stored-command scripts in alphabetical order. Since some of the commands are included in files by other names, the file name is given in parentheses after the command name.

None of the stored commands scripts uses the continuation character. Because users are free to turn the continuation character on, every input file shown below turns it off before defining any stored commands. Britton Lee strongly discourages use of the continuation character.

The first two sections define two views, "columns" and "objects", that are used to decode attribute and object lists. "Columns" depends on the "dtype_SQL" table, which supplies mnemonic names for internal data types. "Objects" depends on the table "logged_I", which decodes status fields to log messages. Both of these tables are defined before the tables that depend on them.

columns (cols.sql)

```
/*
** Define columns view and dtype_SQL table
*/


/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation


/* Build "dtype_SQL" for "cols" command, used to generate mnemonic */

drop columns;
drop dtype_SQL;
create table dtype_SQL(
        mne char(13),           /* mnemonic */
        code smallint,        /* internal rep */
        uncomp tinyint,   /* 0 = fixed, +1 = uncompressed , -1 = compressed */
        dpb tinyint,            /* digits per byte; bcd=2, all others=1 */
        adj tinyint);           /* size adjustment; bcd=3, all others=0 */

insert into dtype_SQL (mne,code,uncomp,dpb,adj) values ("bcdflt(",35,-1,2,3);
insert into dtype_SQL (mne,code,uncomp,dpb,adj) values ("binary(",45,-1,1,0);
insert into dtype_SQL (mne,code,uncomp,dpb,adj) values ("bcd(",46,  -1, 2, 3);
insert into dtype_SQL (mne,code,uncomp,dpb,adj) values ("char(",47,  -1, 1, 0);
insert into dtype_SQL (mne,code,uncomp,dpb,adj) values ("fixed bcdflt(",35,1,2,3);
```

```
insert into dtype_SQL (mne,code,uncomp,dpb,adj) values ("fixed binary(",45,1,1,0);
insert into dtype_SQL (mne,code,uncomp,dpb,adj) values ("fixed bcd(",46,1,2,3);
insert into dtype_SQL (mne,code,uncomp,dpb,adj) values ("fixed char(",47,1,1,0);
insert into dtype_SQL (mne,code,uncomp,dpb,adj) values ("tinyint (",48, 0,1, 0);
insert into dtype_SQL (mne,code,uncomp,dpb,adj) values ("smallint (",52, 0,1, 0);
insert into dtype_SQL (mne,code,uncomp,dpb,adj) values ("integer (",56, 0,1, 0);
insert into dtype_SQL (mne,code,uncomp,dpb,adj) values ("smallfloat (",57,0,1,0);
insert into dtype_SQL (mne,code,uncomp,dpb,adj) values ("float (",60, 0,1, 0);

update relation set type="I" where relid=table_id("dtype_SQL");

/*
** Define "columns" view
*/
create view columns (column_name, relid,tab_name,type)
      as select a.name,
                a.relid,
                r.name,
                type = concat(concat(d.mne,string(4,
                        mod(256+((a.len * d.dpb) - d.adj),256)      )),")")
                        /* mod used to force length postive when > 127 */
        from attribute a, relation r, dtype_SQL d
        where a.type = d.code
        and ((d.uncomp < 0 and a.offset < 0) or
            (d.uncomp >= 0 and a.offset >= 0))
        and r.relid = a.relid;

comment on columns is "Friendly view of attribute table","V1";
```

objects (objects7.sql)

```
/*
** Define view objects and the tables used by the commands that depend on the
** view: dirs, tabs, mine, otherviews, pgms, files and othercmds
*/

/*
**  Stored command scripts do not use continuation characters.
**  We turn continuation off here in case the user or system
**  profile files turned them on.
*/

%continuation

drop objects;   /* view upon which other commands are based  */

drop logged_I;

/*
** logged_I -- used to decode stat field to find if object is logged
*/
```

```
create table logged_I (value smallint, text char(10));

insert into logged_I (value,text) values(0,"Not Logged");
insert into logged_I (value,text) values(64,"Logged");

update relation set type="I" where relid =table_id("logged_I");

/*
** create "objects" view
*/

create view objects (object,
                     relid,
                     owner,
                     ownerid,
                     rows,
                     type,
                     logging
                    )
          as select r.name,
                     r.relid,
                     u.name,
                     r.owner,
                     r.tups,
                     r.type,
                     t.text
            from relation r, users u, logged_I t
            where r.owner *= u.id
            and t.value = mod(smallint(r.stat)+integer(65536),128)
                        - mod(smallint(r.stat)+integer(65536),64);

comment on objects is
"Objects, their owners, type, rows and if logged","V1";

drop otype_I;

create table otype_I (type fixed char(1),
                      definition  char(17)
                     );

insert into otype_I (type,definition) values("S", "System relation");
insert into otype_I (type,definition) values("T", "Transaction log");
insert into otype_I (type,definition) values("U", "User relation");
insert into otype_I (type,definition) values("C", "Stored command");
insert into otype_I (type,definition) values("P", "Stored program");
insert into otype_I (type,definition) values("F", "Files");
insert into otype_I (type,definition) values("I", "Standard relation");
insert into otype_I (type,definition) values("V", "User view");

update relation set type="I" where relid= table_id ("otype_I");

drop logged_I;
```

```
/*
** logged_I -- used to decode stat field to find if object is logged
*/

create table logged_I (value smallint, text char(10));

insert into logged_I (value,text) values(0,"Not Logged");
insert into logged_I (value,text) values(64,"Logged");

update relation set type="I" where relid =table_id("logged_I");
```

allindexes (indexes.sql)

```
/*
** Store "allindexes" command
*/

drop allindexes;

store allindexes
select  tab_name= r.name,
        type = it.desc,
        key1 = col_name(i.relid,tinyint(substring(4,1,i.keys))),
        key2 = col_name(i.relid,tinyint(substring(14,1,i.keys))),
        key3 = col_name(i.relid,tinyint(substring(24,1,i.keys))),
        key4 = col_name(i.relid,tinyint(substring(34,1,i.keys))),
        key5 = col_name(i.relid,tinyint(substring(44,1,i.keys))),
        i.indid,
        i.card
        from itype_I it, indices i, relation r
        where i.relid = r.relid
        and mod(i.stat,4) = it.type
        and r.type="U"
        order by 1,8
end store;

comment on allindexes is
"Displays keys and type of indices for all user tables","1D";
```

attname (attname.sql)

```
/*
** SQL to define "attname" command for ".all"
** pseudo-attribute call from the parser
*/
```

```
/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop attname;
store attname
        select name
        from attribute
        where  relid = table_id(&table)
end store;
```

baudrate (channel.sql)

```
/*
** Channel command to determine the serial baud rate and the
** table used by the command (cbaud_I).
*/

create table cbaud_I (value tinyint, baud char(20));
insert into cbaud_I(value, baud) values(0,"9600");
insert into cbaud_I(value, baud) values(1,"Illegal");
insert into cbaud_I(value, baud) values(2,"150");
insert into cbaud_I(value, baud) values(3,"300");
insert into cbaud_I(value, baud) values(4,"600");
insert into cbaud_I(value, baud) values(5,"1200");
insert into cbaud_I(value, baud) values(6,"1800");
insert into cbaud_I(value, baud) values(7,"2400");
insert into cbaud_I(value, baud) values(8,"4800");
insert into cbaud_I(value, baud) values(9,"9600");
insert into cbaud_I(value, baud) values(10,"19200");
insert into cbaud_I(value, baud) values(11,"No communication");
insert into cbaud_I(value, baud) values(12,"Illegal");
insert into cbaud_I(value, baud) values(13,"Illegal");
insert into cbaud_I(value, baud) values(14,"Illegal");
insert into cbaud_I(value, baud) values(15,"Illegal");

update relation set type="I" where relid=table_id("cbaud_I");

drop baudrate;
store baudrate
select channel=c.number, baudrate=cb.baud
        from configure c, cbaud_I cb
        where c.type="S"
        and mod(c.value,16) = cb.value
        order by channel
end store;
```

```
comment on baudrate is "Gives the serial channel bit/second rates","1M";
```

channel (channel.sql)

```
/*
** Channel command to decode the configure tuple bit strings
*/


/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop channel;
drop channel_I;
drop mask_I;

drop cbaud_I;

create table mask_I(bit tinyint,value integer); /*table of powers of two to 16*/

insert into mask_I (bit,value) values (0,1);
insert into mask_I select bit=max(bit)+1,value=max(value)*2 from mask_I;
insert into mask_I select bit=max(bit)+1,value=max(value)*2 from mask_I;
insert into mask_I select bit=max(bit)+1,value=max(value)*2 from mask_I;
insert into mask_I select bit=max(bit)+1,value=max(value)*2 from mask_I;
insert into mask_I select bit=max(bit)+1,value=max(value)*2 from mask_I;
insert into mask_I select bit=max(bit)+1,value=max(value)*2 from mask_I;
insert into mask_I select bit=max(bit)+1,value=max(value)*2 from mask_I;
insert into mask_I select bit=max(bit)+1,value=max(value)*2 from mask_I;
insert into mask_I select bit=max(bit)+1,value=max(value)*2 from mask_I;
insert into mask_I select bit=max(bit)+1,value=max(value)*2 from mask_I;
insert into mask_I select bit=max(bit)+1,value=max(value)*2 from mask_I;
insert into mask_I select bit=max(bit)+1,value=max(value)*2 from mask_I;
insert into mask_I select bit=max(bit)+1,value=max(value)*2 from mask_I;
insert into mask_I select bit=max(bit)+1,value=max(value)*2 from mask_I;
insert into mask_I select bit=max(bit)+1,value=max(value)*2 from mask_I;
insert into mask_I select bit=max(bit)+1,value=max(value)*2 from mask_I;
insert into mask_I select bit=max(bit)+1,value=max(value)*2 from mask_I;
insert into mask_I select bit=max(bit)+1,value=max(value)*2 from mask_I;

update relation set type="I" where relid=table_id("mask_I");

create table channel_I (type fixed char(1),
                        number tinyint,
                        bit tinyint,
```

```
                        meaning char(50)
                        );

insert into channel_I(type, bit, meaning) values ("S",4,"DCD (Drop Carrier Detect)");
insert into channel_I(type, bit, meaning) values ("S",5,"CTS (Clear to Send)");
insert into channel_I(type, bit, meaning) values ("P",5,"No Timeout");
insert into channel_I(type, bit, meaning) values ("P",6,"20 Second Timeout");
insert into channel_I(type, bit, meaning) values ("S",6,"Cancel host");
insert into channel_I(type, bit, meaning) values ("S",10,"Trustworthy hunames");
insert into channel_I(type, bit, meaning) values ("S",11,"Nontrustworthy huids");
insert into channel_I(type, bit, meaning) values ("P",10,"Trustworthy hunames");
insert into channel_I(type, bit, meaning) values ("P",11,"Nontrustworthy huids");
insert into channel_I(type, bit, meaning) values ("P",12,"1024 byte packet size");
insert into channel_I(type, bit, meaning) values ("P",13,"512 byte packet size (256 if bit 12 is set)");
insert into channel_I(type, bit, meaning) values ("S",14,"Cancel user output in 1 min");
insert into channel_I(type, bit, meaning) values ("P",14,"Cancel user output in 1 min");
insert into channel_I(type, bit, meaning) values ("P",15,"Cancel user output in 5 min 20 min if bit 14 set");
insert into channel_I(type, bit, meaning) values ("S",15,"Cancel user output in 5 min 20 min if bit 14 set");
insert into channel_I(type, bit, number,meaning) values ("E",0,1,"TCP Protocol");
insert into channel_I(type, bit, number,meaning) values ("E",10,1,"Trustworthy hunames");
insert into channel_I(type, bit, number,meaning) values ("E",11,1,"Nontrustworthy huids");
insert into channel_I(type, bit, number,meaning) values ("E",12,1,"1024 byte packet size");
insert into channel_I(type, bit, number,meaning) values ("E",13,1,"512 byte packet size (256 if bit 12 is set)

update relation set type="I" where relid=table_id("channel_I");


store channel
select c.type, channel=c.number,bit_set=mI.bit,ci.meaning
        from configure c, channel_I ci, mask_I mI
          where c.type=ci.type
          and (ci.number=0 or mod(c.number,8)=0)
          and mI.bit=ci.bit
          and mod(c.value/mI.value,2) = 1
        order by type,channel
end store;

comment on channel is "Decodes the bits set in 'configure.value' ","1M";
```

cmds (cmds5.sql)

```
/*
** Description for stored commands from the descriptions table.
*/


/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
```

```
*/

%continuation

drop cmds;

store cmds
select distinct command = r.name,
                description = substring(1,60,d.text)
        from relation r, descriptions d
        where r.relid = d.relid
        and (r.owner = dba or r.owner = userid)
        and r.type = "C"
        and d.key = "1L"
        order by command
end store;

comment on cmds is
"Lists standard stored commands is - a one line description","1L";
```

cmds_date (cmds5.sql)

```
/*
** store "cmds_date" command.
*/

drop cmds_date;

store cmds_date
select distinct command = r.name,
                description = substring(1,60,d.text)
        from relation r, descriptions d
        where r.relid = d.relid
        and (r.owner = dba or r.owner = userid)
        and r.type = "C"
        and d.key = "1T"
        order by command
end store;

comment on cmds_date is
"Lists stored commands relating to date and time","1L";
```

cmds_dba (cmds5.sql)

```
/*
** store "cmds_dba" command.
*/

drop cmds_dba;

store cmds_dba
select distinct command = r.name,
                description = substring(1,60,d.text)
        from relation r, descriptions d
        where r.relid = d.relid
        and (r.owner = dba or r.owner = userid)
        and r.type = "C"
        and d.key = "1D"
        order by command
end store;

comment on cmds_dba is
"Lists stored commands mostly used by the dba","1L";
```

cmds_permit (cmds5.sql)

```
/*
** Store "cmds_permit" command.
*/

drop cmds_permit;

store cmds_permit
select distinct command = r.name,
                description = substring(1,60,d.text)
        from relation r, descriptions d
        where r.relid = d.relid
        and (r.owner = dba or r.owner = userid)
        and r.type = "C"
        and d.key = "1P"
        order by command
end store;

comment on cmds_permit is
"Lists stored commands dealing protection of objects","1L";
```

cmds_space (cmds5.sql)

```
/*
** Store "cmds_space" command.
*/
drop cmds_space;

store cmds_space
select distinct command = r.name,
                description = substring(1,60,d.text)
        from relation r, descriptions d
        where r.relid = d.relid
        and (r.owner = dba or r.owner = userid)
        and r.type = "C"
        and d.key = "1S"
        order by command
end store;

comment on cmds_space is
"Lists stored commands dealing is - space/size/storage","1L";
```

cmds_system (cmds_sys.sql)

```
/*
** Define "cmds_system" command.
*/


/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop cmds_system;

store cmds_system
select distinct command = r.name,
                description = substring(1,60,d.text)
        from relation r, descriptions d
        where r.relid = d.relid
        and (r.owner = dba or r.owner = userid)
        and r.type = "C"
        and d.key = "1M"
        order by command, d.key
end store;
```

```
comment on cmds_system is
"Lists stored commands for monitoring performance","1L";
```

config (config.sql)

```
/* cols command */

drop cols;

store cols
  select column_name, type from columns where relid = table_id(&table)
end store;

comment on cols is
"Displays column names and types for given TABLE","1L";
```

config (config.sql)

```
/*
** The "config" stored command dumps the "configure" table from the
** "system" database.
*/

/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

/* "config_I" - constant relation for "config" command. */

drop config;
drop config_I;

create table config_I
   (type fixed char(1), name char(40));

insert into config_I (type,name) values( "A", "Accelerator, value=WCS level");
insert into config_I (type,name) values( "B","Block multiplexer host interface");
insert into config_I (type,name) values( "C", "Checkpoint interval");
insert into config_I (type,name) values( "D","Default charset 0=ASCII 1=EBCDIC");
insert into config_I (type,name) values( "E", "Ethernet host interface");
insert into config_I (type,name) values( "I", "Database server id for IDENTIFY response");
insert into config_I (type,name) values( "K", "Memory configuration");
insert into config_I (type,name) values( "M", "Monitor interval(minutes)");
```

```
insert into config_I (type,name) values( "P", "IEEE-488 host interface");
insert into config_I (type,name) values( "R", "IDM/RDBMS software release");
insert into config_I (type,name) values( "S", "RS-232 host interface");
insert into config_I (type,name) values( "T",
                                         "Mag tape interface, value=4 for h/p"
                                       );

update relation set type="I" where relid=table_id("config_I");

store  config
select  c.*,
         meaning =cx.name
        from configure c, config_I cx
         where c.type = cx.type
         order by c.type
end store;

comment on config is "Identify the tuples from the configure relation","1M";
```

date (date8.sql)

All of the objects used by the **date** command and the "DateAndTime" view are included to improve readability. A number of stored commands for date conversion are also found in this file.

```
/* This installs the date command and the gmt_date command along
   with their needed relations and views    */

/* make sure that the values in the savings_I relation below are correct
   for the local time zone    */


/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop date, gmt_date;  /* the commands for date and time  */

drop DateAndTime;     /* the view that holds the current date and time */

drop GMT;             /* the view that holds GMT current date and time */

drop ymd;             /*  stored command returns idmdate given YYYYMMDD */

drop dateconv;        /*  stored command to  convert idmdate to readable
                                 form */
```

```
drop base1970;      /*  set the base for calendar_I to Jan 1, 1970  */

drop base1900;      /*  set the base for calendar_I to Jan 1, 1900  */

drop local_I;       /*  a view of getdate to correct for GMT displacement */

drop mon_I;          /* a view of month_I with the values corrected for
                            leap years                              */

drop month_I;        /* a relation containing the first and last days,
                            name and number of the 12 months        */

drop julian_I;       /* a view of the calendar relation containing the
                            information for the current year        */

drop calendar_I;     /* a relation containing information needed to
                            convert to julian date and allow for leap years
                            for the years 1985 - 1992               */

drop day_I;          /* a relation containing the 7 days of the week */

drop number_I;       /*  relation containing the days of the month   */

drop savings_I;      /* a relation for daylight savings correction   */

/*****************************************************************************/

create table savings_I (   /* displacement of hours from GMT and the change*/
     first integer,      /* julian dates for daylight savings time      */
     last integer,       /* Check these tuples for local time zone      */
     displacement integer   );

update relation set type="I" where relid=table_id("savings_I");

insert into savings_I ( first, last,displacement)
                values (0,120, 8);
insert into savings_I ( first, last,displacement)
                values (121,240, 7);
insert into savings_I ( first, last,displacement)
                values (241,365, 8);

create table calendar_I(
                day_one integer,      /* first day year since epoch */
                shortyear fixed char(2),  /* YY form of year ie. 85 */
                longyear fixed char(4),   /* YYYY form of year ie. 1984 */
                leapyear integer  /* 0 if not leap year, 1 if leap year */
                );

update relation set type="I" where relid=table_id("calendar_I");

insert into calendar_I (day_one,shortyear, longyear,leapyear)
          values (31045,"85","1985",0);
```

```
insert into calendar_I (day_one,shortyear, longyear,leapyear)
        values (31410,"86","1986",0);

insert into calendar_I (day_one,shortyear, longyear,leapyear)
        values (31775,"87","1987",0);

insert into calendar_I (day_one,shortyear, longyear,leapyear)
        values (32140,"88","1988",1);

insert into calendar_I (day_one,shortyear, longyear,leapyear)
        values (32506,"89","1989",0);

insert into calendar_I (day_one,shortyear, longyear,leapyear)
        values (32871,"90","1990",0);

insert into calendar_I (day_one,shortyear, longyear,leapyear)
        values (33236,"91","1991",0);

insert into calendar_I (day_one,shortyear, longyear,leapyear)
        values (33601,"92","1992",1);

insert into calendar_I (day_one,shortyear, longyear,leapyear)
        values (33967,"93","1993",0);

insert into calendar_I (day_one,shortyear, longyear,leapyear)
        values (34332,"94","1994",0);

insert into calendar_I (day_one,shortyear, longyear,leapyear)
        values (34697,"95","1995",0);

/*  This will reset calendar_I to work from Jan 1,1970 base date  */

store base1970
        update calendar_I
                set day_one = day_one - 25566
                where day_one > 25566
end store;

comment on base1970 is
"Change base for date conversion to Jan 1,1970";"1T";

/*  This will reset from 1970 to 1900  */

store base1900
        update calendar_I
                set day_one = day_one + 25566
                where day_one < 25566
end store;

comment on base1900 is
"Change base for date conversion to Jan 1,1900 from 1970","1T";
```

```
/* set up current year */

create view julian_I
  as select
        day_one = cal.day_one,
        cal.shortyear,
        cal.longyear,
        cal.leapyear,
        s.displacement
        from calendar_I cal,savings_I s
        where cal.day_one <= getdate
        and cal.day_one + 365 + cal.leapyear > getdate
        and s.first <= getdate - cal.day_one
        and s.last > getdate - cal.day_one;

comment on julian_I is
"View of the current year in calendar_I", "V1";

/*
** day_I
** used to give the day of the week.
*/

create table day_I(number integer,day char(9),shortday fixed char(3));

update relation set type="I" where relid=table_id("day_I");

insert into day_I(number,day,shortday) values(0,"Monday","Mon");
insert into day_I(number,day,shortday) values(1,"Tuesday","Tue");
insert into day_I(number,day,shortday) values(2,"Wednesday","Wed");
insert into day_I(number,day,shortday) values(3,"Thursday","Thu");
insert into day_I(number,day,shortday) values(4,"Friday","Fri");
insert into day_I(number,day,shortday) values(5,"Saturday","Sat");
insert into day_I(number,day,shortday) values(6,"Sunday","Sun");

/*
** month_I
** used to give the month
*/

create table month_I(first integer,
                    last integer,
                    month char(9),
                    month_num fixed char(2),
                    leapfirst integer,
                    leaplast integer);

update relation set type="I" where relid=table_id("month_I");

insert into month_I(first, last, month,month_num, leapfirst, leaplast)
                values (1,31,"January","01", 0, 0);
insert into month_I(first, last, month,month_num, leapfirst, leaplast)
```

```
                values (32,59,"February","02", 0, 1);
insert into month_I(first, last, month,month_num, leapfirst, leaplast)
                values (60, 90, "March","03", 1, 1);
insert into month_I(first, last, month,month_num, leapfirst, leaplast)
                values (91, 120, "April","04", 1, 1);
insert into month_I(first, last, month,month_num, leapfirst, leaplast)
                values (121, 151, "May","05", 1, 1);
insert into month_I(first, last, month,month_num, leapfirst, leaplast)
                values (152, 181, "June","06", 1, 1);
insert into month_I(first, last, month,month_num, leapfirst, leaplast)
                values (182, 212, "July","07", 1, 1);
insert into month_I(first, last, month,month_num, leapfirst, leaplast)
                values (213, 243, "August","08", 1, 1);
insert into month_I(first, last, month,month_num, leapfirst, leaplast)
                values (244, 273, "September","09", 1, 1);
insert into month_I(first, last, month,month_num, leapfirst, leaplast)
                values (274, 304, "October","10", 1, 1);
insert into month_I(first, last, month,month_num, leapfirst, leaplast)
                values (305, 334, "November","11", 1, 1);
insert into month_I(first, last, month,month_num, leapfirst, leaplast)
                values (335, 365, "December","12", 1, 1);

/*  view of month that corrects for leapyear on leapyears  */

create view mon_I
        as select first = m.first + m.leapfirst * jul.leapyear,
                  last = m.last + m.leaplast * jul.leapyear,
                  month = m.month,
                  month_num = m.month_num
                from julian_I jul, month_I m;

comment on mon_I is
"View of the months from the current year", "V1";

/*
**  Used to compensate for GMT date standard.
**  Basically this is an if statement.  For a
**  given hour of the day what do we have to
**  add (subtract) to the GMT date to get the
**  local date
**    also useful for numeric conversions
**    especially date, hour, minute, and second conversions
*/

create table number_I(number integer,
                     str fixed char(2)
                    );

update relation set type="I" where relid=table_id("number_I");

insert into number_I(number,str) values (0,"00");
insert into number_I(number,str) values (1,"01");
```

```
insert into number_I(number,str) values (2,"02");
insert into number_I(number,str) values (3,"03");
insert into number_I(number,str) values (4,"04");
insert into number_I(number,str) values (5,"05");
insert into number_I(number,str) values (6,"06");
insert into number_I(number,str) values (7,"07");
insert into number_I(number,str) values (8,"08");
insert into number_I(number,str) values (9,"09");
insert into number_I(number,str) values (10,"10");
insert into number_I(number,str) values (11,"11");
insert into number_I(number,str) values (12,"12");
insert into number_I(number,str) values (13,"13");
insert into number_I(number,str) values (14,"14");
insert into number_I(number,str) values (15,"15");
insert into number_I(number,str) values (16,"16");
insert into number_I(number,str) values (17,"17");
insert into number_I(number,str) values (18,"18");
insert into number_I(number,str) values (19,"19");
insert into number_I(number,str) values (20,"20");
insert into number_I(number,str) values (21,"21");
insert into number_I(number,str) values (22,"22");
insert into number_I(number,str) values (23,"23");
insert into number_I(number,str) values (24,"24");
insert into number_I(number,str) values (25,"25");
insert into number_I(number,str) values (26,"26");
insert into number_I(number,str) values (27,"27");
insert into number_I(number,str) values (28,"28");
insert into number_I(number,str) values (29,"29");
insert into number_I(number,str) values (30,"30");
insert into number_I(number,str) values (31,"31");
insert into number_I(number,str) values (32,"32");
insert into number_I(number,str) values (33,"33");
insert into number_I(number,str) values (34,"34");
insert into number_I(number,str) values (35,"35");
insert into number_I(number,str) values (36,"36");
insert into number_I(number,str) values (37,"37");
insert into number_I(number,str) values (38,"38");
insert into number_I(number,str) values (39,"39");
insert into number_I(number,str) values (40,"40");
insert into number_I(number,str) values (41,"41");
insert into number_I(number,str) values (42,"42");
insert into number_I(number,str) values (43,"43");
insert into number_I(number,str) values (44,"44");
insert into number_I(number,str) values (45,"45");
insert into number_I(number,str) values (46,"46");
insert into number_I(number,str) values (47,"47");
insert into number_I(number,str) values (48,"48");
insert into number_I(number,str) values (49,"49");
insert into number_I(number,str) values (50,"50");
insert into number_I(number,str) values (51,"51");
insert into number_I(number,str) values (52,"52");
insert into number_I(number,str) values (53,"53");
```

```
insert into number_I(number,str) values (54,"54");
insert into number_I(number,str) values (55,"55");
insert into number_I(number,str) values (56,"56");
insert into number_I(number,str) values (57,"57");
insert into number_I(number,str) values (58,"58");
insert into number_I(number,str) values (59,"59");
insert into number_I(number,str) values (60,"60");

/*  This view corrects getdate to the local date based on the
**  current local displacement from GMT
*/

create view local_I as
        select lgetdate = getdate
                + (((gettime - displacement * 216000)
                   / abs(gettime - displacement * 216000)
                  ) - 1) / 2
        from julian_I;

comment on local_I is
"View of getdate corrected for time zone", "V1";

/*
**  Store "DateAndTime" view.
** For string conversion of numbers whose preferred format is with a
** preceding zero (as in the ninth of the month being YYMM09 or
** minutes in HH:03:SS) the number_I relation is used rather than the
** string function (which won't give preceding zeroes)
*/

create view DateAndTime as
  select day = day.day,
     date_numeric = concat(jul.shortyear,concat(month.month_num,n_day.str)),
      date_slashed =
          concat(jul.shortyear,
            concat("/",
                concat(month.month_num,
                 concat("/",n_day.str)
                   )
                 )
                ),
        date_written =
           concat(month.month,
             concat(" ",
                 concat(string(0, n_day.number),
                  concat(",",
                      concat(" ",jul.longyear)
                        )
                   )
                )
                ),
         Time =
```

```
            substring (1,10,       /* used only to narrow column width */
              concat( string(0,
                    mod((((gettime/60)/60)/60 + (24 - jul.displacement)),24)
                        ),
                concat(":",
                  concat(n_min.str,
                    concat(":",n_secs.str)
                      )
                    )
                  )
                  )
  /* end attribute list */
    from mon_I month,
    day_I day,
    julian_I jul,
    number_I da,
    number_I n_day,
    number_I n_min,
    number_I n_secs,
    local_I local
    /* jul.day_one is  year's first day */
    where month.first <= lgetdate-jul.day_one
    and month.last >= lgetdate -jul.day_one
    and n_day.number = lgetdate - jul.day_one - month.first + 1
    and da.number=(((gettime/60)/60)/60)
    and n_min.number = mod(gettime/3600,60)
    and n_secs.number = mod(gettime/60,60)
    and day.number = mod((lgetdate),7); /* gets day of week */

comment on DateAndTime is
"Gives current date and time in various formats", "V1";

/*
** date command
*/

store date
select
   Day = day,
   Date = date_written,
   Time = Time
   from DateAndTime
end store;

comment on date is "Returns the current date and time from the database server clock","1T";

create view GMT as
   select day = day.day,
       date_numeric = concat(jul.shortyear,concat(month.month_num,n_day.str)),
        date_slashed =
            concat(jul.shortyear,
              concat("/",
```

```
                concat(month.month_num,
                 concat("/",n_day.str)
                   )
                 )
               ),
        date_written =
          concat(month.month,
            concat(" ",
                concat(string(0, n_day.number),
                 concat(",",
                     concat(" ",jul.longyear)
                       )
                  )
              )
              ),
        Time =
          substring (1,10,        /* used only to narrow column width */
            concat( string(0,
                  mod((((gettime/60)/60)/60 + 24),24)
                        ),
             concat(":",
               concat(n_min.str,
                 concat(":",n_secs.str)
                   )
                 )
               )
                )
 /* end attribute list */
  from mon_I month,
  day_I day,
  julian_I jul,
  number_I da,
  number_I n_day,
  number_I n_min,
  number_I n_secs
  /* jul.day_one is magic for year's first day */
  where month.first <= getdate-jul.day_one
  and month.last >= getdate -jul.day_one
  and n_day.number = getdate - jul.day_one - month.first + 1
  and n_min.number = mod(gettime/3600,60)
  and n_secs.number = mod(gettime/60,60)
  and day.number = mod(getdate,7); /* gets day of week */

comment on GMT is
"gives Greenwich date and time in various formats", "V1";


/*
** Store gmt_date command
*/

store gmt_date
```

```
select
  Day = day,
  Date = date_written,
  Time = Time
from GMT
end store;

comment on gmt_date  is "Returns Greenwich Mean Time and Date from database server","1T";

/*
**  Store dateconv to convert from idmdate to readable date format
*/

store dateconv
select date_written = concat(month,
                             concat(" ",
                                 concat(string(0, number),
                                   concat(",",
                                     concat(" ",longyear)
                                       )
                                   )
                                 )
                             )
      from calendar_I, month_I, number_I
      where first + leapfirst * leapyear <= integer(&idmdate) - day_one
      and last + leaplast * leapyear >= integer(&idmdate) - day_one
      and number = integer(&idmdate) - day_one - first + 1
               - leapfirst * leapyear
end store;

comment on dateconv is
"returns the month day, year given idmdate NUMBER", "1T";

/*
**  Define ymd command to convert a string of the form "YYYYMMDD" to
**  an idmdate
*/

store ymd
select idmdate = day_one + first + integer(substring(7,2,$YYYYMMDD))
                       + leapyear * leapfirst - 1
        from calendar_I, month_I
          where longyear = substring(1,4,$YYYYMMDD)
          and month_num = substring(5,2,$YYYYMMDD)
end store;

comment on ymd is
"Returns the idmdate given a date string: YYYYMMDD","1T";
```

dbs (dbs.sql)

```
/*
** Define "dbs" command.
*/


/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop dbs;
drop dbs0_I;

create table dbs0_I (value smallint, text char(46));

/*
** for ASCII databases
*/

/*  'text can only be this long!!                    ' */
insert into dbs0_I (value, text) values (1,
    "Database/ disks not on-line (ASCII)");
insert into dbs0_I (value, text) values (9,
    "Locked: database marked unrecoverable (ASCII)");
insert into dbs0_I (value, text) values (17,
    "Unused since boot, recovery not needed (ASCII)");
insert into dbs0_I (value, text) values (145,
    "Locked: unused since boot-no recovery need (A)");
insert into dbs0_I (value, text) values (33,
    "On-line (ASCII)");
insert into dbs0_I (value, text) values (161,
    "LOCKED for dump/load/rollforward (on-line) (A)");
insert into dbs0_I (value, text) values (49,
    "Unused since boot, recovery had to run (ASCII)");
insert into dbs0_I (value, text) values (177,
    "LOCKED: Unused since boot-recovery run (ASCII)");
insert into dbs0_I (value, text) values (65,
    "being loaded or rolled forward (ASCII)");
insert into dbs0_I (value, text) values (193,
    "LOCKED: being loaded or rolled forward (ASCII)");
insert into dbs0_I (value, text) values (69,
    "being created or destroyed (ASCII)");
insert into dbs0_I (value, text) values (197,
    "LOCKED: being created or destroyed (ASCII)");
/*
** for EBCDIC databases
*/
```

```
insert into dbs0_I (value, text) values (3,
   "Database/ disks not on-line");
insert into dbs0_I (value, text) values (11,
   "Locked: database marked unrecoverable (EBCDIC)");
insert into dbs0_I (value, text) values (19,
   "Unused since boot, no recovery needed (EBCDIC)");
insert into dbs0_I (value, text) values (147,
   "Locked: unused since boot-no recovery need(E)");
insert into dbs0_I (value, text) values (35,
   "On-line (EBCDIC)");
insert into dbs0_I (value, text) values (163,
   "LOCKED for dump/load/rollforward (on-line) (E)");
insert into dbs0_I (value, text) values (51,
   "Unused since boot, recovery had to run (E)");
insert into dbs0_I (value, text) values (179,
   "LOCKED: Unused since boot, recovery run (E)");
insert into dbs0_I (value, text) values (67,
   "being loaded or rolled forward (EBCDIC)");
insert into dbs0_I (value, text) values (195,
   "LOCKED: being loaded or rolled forward (E)");
insert into dbs0_I (value, text) values (71,
   "being created or destroyed (EBCDIC)");
insert into dbs0_I (value, text) values (199,
   "LOCKED: being created or destroyed (EBCDIC)");

update relation set type="I" where relid=table_id("dbs0_I");

store dbs
select distinct d.name,
                dbid=string(3,d.id),
                owner_name = u.name,
                status = t0.text
                from databases d, users u,dbs0_I t0
                where u.id =* d.owner
                and t0.value =* mod(integer(d.stat) + 65536 ,256)
                order by name
end store;

comment on dbs is "Displays databases on this shared database system","1M";
```

depend (depend.sql)

```
/*
** Store "depend" command.
*/

/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
```

```
*/

%continuation

drop depend;

store depend
select distinct object = table_name(c.relid),
                dependents = table_name(c.drelid),
                downer = u.name
                from crossref c, users u, relation r
        where c.relid = table_id(&object)
        and r.owner = u.id
        and r.relid = c.drelid
        order by object
end store;

comment on depend is
"Displays objects depending on OBJECT","1L";
```

describe (describe.sql)

```
/*
** Define "describe" command
*/


/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop describe;
store describe
select description = substring(1,72,d.text)
        from relation r, descriptions d
        where table_id(&object) = r.relid
        and d.relid = r.relid
        and d.attid = 0
        order by d.key
end store;

comment on describe is
"Displays description of given RELATION or COMMAND","1L";

/*
**  comment on commands for descriptions of system relations
```

```
*/

comment on relation is
"relation:    Catalog of all objects in database.  An object is a","91";

comment on relation is
"            relation, view, file, stored command or stored pro-","92";

comment on relation is
"            gram.  Each tuple represents a single object.      ","93";

comment on attribute is
"attribute:   Catalog of each attribute of each relation.  Each  ","91";
comment on attribute is
"            tuple represents one attribute.                     ","92";

comment on indices is
"indices:     Catalog of indices that exist in database.  Each   ","91";
comment on indices is
"            tuple represents one index.                         ","92";

comment on protect is
"protect:     Catalog of protection information for the database.","91";
comment on protect is
"            Each tuple represents one type of access (e.g.,     ","92";
comment on protect is
"            read, write, create index) for one user or group    ","93";
comment on protect is
"            for all attributes of one view, relation, file, or ","94";
comment on protect is
"            stored command.                                     ","95";

comment on query is
"query:       Text of stored commands                             ","91";

comment on crossref is
"crossref:    Catalog of dependencies among relations, views and ","91";
comment on crossref is
"            stored commands.                                    ","92";

comment on transact is
"transact:    Transaction logging relation                        ","93";

comment on batch is
"batch:       Temporary transaction logging relation, used for   ","91";
comment on batch is
"            transaction management so that even transactions    ","92";
comment on batch is
"            against non-logged relations may be cancelled if    ","93";
comment on batch is
"            needed.                                             ","94";
```

```
comment on descriptions is
"descriptions: User definable descriptions, keyed to relation id's","91";

comment on users is
"users:        Mapping of user and group names to user id        ","91";
comment on host_users is
"host_users:   Mapping from host id and user's id to database server user id. ","91";

comment on blockalloc is
"blockalloc:   Catalog of disk blocks, showing relations assigned ","91";
comment on blockalloc is
"              to blocks. Each tuple represents a block.         ","92";

comment on disk_usage is
"disk_usage:   Shows relation and database allocation.           ","91";
```

dir (objects7.sql)

```
/*
**  Define "dir" command to find an object by name
*/


store dir
        select o.relid,
               o.object,
               o.type,
               ot.definition,
               o.owner,
               o.rows
              from objects o, otype_I ot
              where o.type *= ot.type
              and o.object = &name
              order by object,owner
end store;

comment on dir is
"Displays all objects of given NAME or fragment","1L";
```

diskio (diskio.sql)

```
/*
**  Find average disk access time from devmonitor
*/

/*
**  Stored command scripts do not use continuation characters.
```

```
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop diskio;

store diskio
select distinct d.name,
                  s.slot,
                accesses = sum(s.d2),
                time_ticks = sum(s.d1),
               avg_ms = bcdflt(5,bcdflt(5,sum(s.d1) * 1000 / 60 / bcdflt(5,sum(s.d2))))
    from disks d, devmonitor s
    where s.type ="D" and s.d2 > 0  /* prevent division by zero */
    and s.d3 = d.low       /* devmonitor identifies disks by low block*/
    and (d.type = "P" or d.type="M") /* only meaningful for physical disks */
    group by s.d3, s.slot
end store;

comment on diskio is "lists the on-line disks and their I/O activity","1M";
```

expire (date8.sql)

```
drop expire;

/* Set expiration date on an object to current date plus <N> days */

store expire
update relation
        set expire = getdate + &n_days
        where relid = table_id (&1object)
        and owner = userid
end store;

comment on expire is
"sets the expiration date for an OBJECT to getdate + NDAYS","1T";
```

expiredate (date8.sql)

```
/*
**  Define "expiredate" stored command to update relation.expire to
**      a given  YYYYMMDD
*/

drop expiredate;
```

```
store expiredate
update relation
        from calendar_I, month_I
        set expire = day_one + first + integer(substring(7,2,&YYYYMMDD))
                        + leapyear * leapfirst - 1
          where longyear = substring(1,4,&YYYYMMDD)
          and month_num = substring(5,2,&YYYYMMDD)
          and relid= table_id( &OBJECT)
          and owner = userid
end store;

comment on expiredate is
"sets the expiration date for an OBJECT to date YYYYMMDD","1T";
```

files (objects7.sql)

```
/*
** Store "files" command.
*/
store files
select distinct  files = object,
                 owner,
                 logging
       from objects
         where type = "F"
         and ((ownerid = userid or ownerid = dba)
             or userid =dba  /* show all relations for dba  */
           )
         order by owner, object
end store;

comment on files is "Displays files owned by you and the dba","1L";
```

freelog (freespc.sql)

```
drop freelog;

store freelog
select log_blocks = sum ( du.high -du.low + 1),
      free_blks = count(b.relid)
      from disk_usage du, blockalloc b
      where b.relid = 7
      and b.mode=32
      and du.relid = 7
end store;

comment on freelog is
```

```
"Displays space usage in hard allocated transact log","1S";
```

freespace (freespc.sql)

```
/*
** Report freespace in database in blocks
*/

/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop freespace;

store freespace
select total_blks = sum ( du.high - du.low + 1),
      free_blks = count(b.relid),
      percent_used = 100 -
                  (count(bl.relid) * 100) /
               sum ( du.high - du.low + 1)
      from disk_usage du, blockalloc b, blockalloc bl
      where du.relid = -32768
      and b.relid = 0
      and bl.relid != 0
end store;

comment on freespace is
"Displays total space and free space in this database","1S";
```

groups (groups3.sql)

```
/*
** Define 'groups' stored command.  Lists names and membership of groups.
*/

/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation
```

```
drop groups;

store groups
select distinct gid,
                name,
                members = count(id) - 1
        from users
          group by gid
          having gid=id
          and id != 0
end store;

comment on groups is "Lists the groups and membership for this database","1D";
```

group_off (groups3.sql)

```
/*
** Allows groups to be temporarily removed from host_users
** and users, and restored later.
*/

drop group_off;

/* A place to keep temporary users */

create table host_users_I (hid smallint,
                           huid integer,
                           uid smallint
                          )
                with logging;

create table users_I ( id smallint,
                       gid smallint,
                       name char(12)
                     )
                with logging;

update relation set type="I"
        where relid=table_id("host_users_I")
        or relid=table_id("users_I");

/* save group members in a safe place, */
store group_off
insert into users_I (id, gid, name)
        select u.id, u.gid, u.name
        from users u, users u1
        where u1.name = &group
        and u1.id = u1.gid
        and u1.gid = u.gid
insert into host_users_I (hid, huid, uid)
```

```
        select hid,huid,uid
        from host_users, users u, users u1
        where u1.name = &group
        and u1.id = u1.gid
        and u1.gid = u.gid
        and u.id = uid
/*  Now delete the group members  */
delete from host_users
      where uid =
                any (select u.id
                     from users u, users u1
                     where u1.name = &group
                     and u1.id = u1.gid
                     and u1.gid = u.gid
                 )
delete from users
      where gid =
                 any (select id
                     from users
                     where name = &group
                     and id = gid
                   )
end store;

comment on group_off is
"Turn off all access to database for GROUP - for DBA only", "1D";
```

group_on (groups3.sql)

```
/*   Group-on command to restore group members from holding  */

drop group_on;

store group_on
insert into users (id, gid, name)
        select u.id, u.gid, u.name
        from users_I u, users_I u1
        where u1.name = &group
        and u1.id = u1.gid
        and u1.gid = u.gid
insert into host_users (hid, huid, uid)
        select hid,huid,uid
        from host_users_I, users_I u, users_I u1
        where u1.name = &group
        and u1.id = u1.gid
        and u1.gid = u.gid
        and u.id = uid
delete from host_users_I
      where uid =
                any (select u.id
```

```
                    from users_I u, users_I u1
                    where u1.name = &group
                    and u1.id = u1.gid
                    and u1.gid = u.gid
                )
delete from users_I
    where gid =
            any (select id
                    from users_I
                    where name = &group
                    and id = gid
                )
end store;

comment on group_on is
"Turn GROUP database access back on - for DBA only", "1D";
```

help (cmds5.sql)

```
/* left here to get rid of the old "help" in old  databases  */
drop help;

/*
** Define "help" command.
*/

store help
select    note="USE THE 'cmds' COMMAND TO GET A LIST OF COMMANDS"
end store;
```

indexes (indexes.sql)

```
/*
** Install Indexes stored command to decode indices relation
**
*/

/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop indexes;
```

```
/* Build "itype_I" for "indexes" command */

drop itype_I;

create table itype_I(type tinyint, desc char(17));

insert into itype_I(type,desc) values (1,"unique noncluster");
insert into itype_I(type,desc) values (-3, "unique noncluster");
insert into itype_I(type,desc) values (0, "nonclustered");
insert into itype_I(type,desc) values (2, "clustered");
insert into itype_I(type,desc) values (-2, "clustered");
insert into itype_I(type,desc) values (-1, "unique clustered");
insert into itype_I(type,desc) values (3, "unique clustered");

update relation set type="I" where relid=table_id("itype_I");

/*
** Store "indexes" command
*/

store indexes
select  type = it.desc,
        key1 = col_name(i.relid,tinyint(substring(4,1,i.keys))),
        key2 = col_name(i.relid,tinyint(substring(14,1,i.keys))),
        key3 = col_name(i.relid,tinyint(substring(24,1,i.keys))),
        i.indid,
        i.card
        from itype_I it, indices i, relation r
        where i.relid = table_id(&table)
        and mod(i.stat,4) = it.type
        order by i.indid
end store;

comment on indexes is
"Displays keys and type of indices for given RELATION","1D";
```

mine (objects7.sql)

```
/*
**  Define "mine" command to find an object owned by me
*/
drop mine;

store mine
          select o.relid,
                 o.object,
                 o.type,
                 ot.definition,
                 o.rows
                from objects o, otype_I ot
```

```
                where o.type *= ot.type
                and o.ownerid = userid
                order by type,object
end store;

comment on mine is
"Displays all objects owned by this user","1L";
```

mon (mon5.sql)

```
/*
** "mon" commands--cpu usage for the last N monitor intervals.
**
*/

/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop mon;

store mon
select
  CPU_activity    = "avg monitoring interval (minutes):",
  time = bcdfixed(5,2,avg(bcdflt(0,length))/3600)
  from monitor

select
        SEQNO          = seqno,
        CPU_percent    = bcdflt(5,100 * bcdflt(5,cpu)/bcdflt(5,length)),
        DAC_percent    = bcdflt(5,100 * bcdflt(5,dac)/bcdflt(5,length)),
        IDLE_percent   = bcdflt(5,100 * bcdflt(5,idle)/bcdflt(5,length)),
        COMMANDS       = smallint(cmnds),
        AVG_seconds    = bcdflt(5,bcdflt(5,avgcmnd)/60)
        from monitor
        where (date=getdate and time >gettime - length * integer(&INTERVALS))
        or (date=getdate-1 and time> gettime - length * integer(&INTERVALS)
                        + integer(24) * integer(60) * integer(3600)
          )
        order by date,time
end store;

comment on mon is "Monitor CPU and DAC usage for last N monitor intervals","1M";
```

mondisk (mon5.sql)

```
/*
** "mondisk" command--disk usage for the last N monitor intervals.
*/

drop mondisk;

store mondisk
select
        DISK_active   = "avg monitoring interval (minutes):",
        time = bcdfixed(5,2,avg(bcdflt(0,length))/3600)
        from monitor

select
        SEQNO          = seqno,
        READS          = reads,
        WRITES         = writes,
        HITS           = hits,
        DISK_WT_sec    = bcdflt(5,bcdflt(5,diskwait) / 60)
        from monitor
        where (date=getdate and time >gettime - length * integer(&INTERVALS))
        or (date=getdate-1 and time> gettime - length * integer(&INTERVALS)
                        + integer(24) * integer(60) * integer(3600)
          )
        order by date,time
end store;

comment on mondisk is "Monitor disk usage for last N monitor intervals","1M";
```

monfail (mon5.sql)

```
/*
** "monfail" command--Time processing or processes are suspended for lack
**   of available memory resources
*/

drop monfail;

store monfail
select
        SUSPENDED    = "avg monitoring interval (minutes):",
        time = bcdfixed(5,2,avg(bcdflt(0,length))/3600)
        from monitor
select
        SEQNO        = seqno,
        INDELAY_pct  = bcdflt(5,100 * bcdflt(5,indelay)/bcdflt(5,length)),
        OUTDELAY_pct = bcdflt(5,100 * bcdflt(5,outdelay)/bcdflt(5,length)),
        MEMDELAY_pct = bcdflt(5,100 * bcdflt(5,memloss)/bcdflt(5,length)),
```

```
        DBINFAILS     = dbinfails
        from monitor
        where (date=getdate and time >gettime - length * integer(&INTERVALS))
        or (date=getdate-1 and time> gettime - length * integer(&INTERVALS)
                                + integer(24) * integer(60) * integer(3600)
        )
        order by date,time
end store;

comment on monfail is
"Suspended processing for lack of memory from monitor","1M";
```

monlock (mon5.sql)

```
/*
** "monlock" command--blocked wait, deadlocks.
**          current performance
*/

drop monlock;
drop lockdef_I;

create table lockdef_I(code  fixed binary(1), meaning char(36));

/*  interpretation of lock types  */
insert into lockdef_I(code, meaning) values(binary(1), "read lock");
insert into lockdef_I(code, meaning) values(binary(2), "write lock");
insert into lockdef_I(code, meaning) values(binary(3), "read, intend to set block write lock");
insert into lockdef_I(code, meaning) values(binary(5), "intend to set block read lock");
insert into lockdef_I(code, meaning) values(binary(6), "intend to set block write lock");

update relation set type="I" where relid=table_id("lockdef_I");

store monlock
/*  first store the interval length  */
select
        LOCKS_BLOCKS    = "avg monitoring interval (minutes):",
       time          = bcdfixed(5,2,avg(bcdflt(0,length))/3600)
        from monitor
/* history of locks and deadlocks over last N intervals  */
select
         SEQNO          = seqno,
        DEADLOCKS        = deadlocks,
        LOCK_sec        = bcdflt(5,bcdflt(5,blockwait) / 60)
         from monitor
        where (date=getdate  and time >gettime - length * integer(&INTERVALS))
      or (date=getdate-1 and time> gettime - length * integer(&INTERVALS)
                               + (integer(24) * integer(60) * integer(3600))
        )
        order by date,time
```

```
/* list current locks */
select
        DBIN = d.dbin,
        DATABASE = b.name,
        RELID = l.dnum, /* this is the relid in the database, not system */
        block = l.pid,
        lock_type = t.meaning
        from lock l, lockdef_I t, dbinstat d, databases b
    where d.xactid=l.tnum and d.dbid=b.id and t.code = l.type
end store;

comment on monlock is
"Current locks and history for last N monitor intervals","1M";
```

monun (mon5.sql)

```
/*
** "monun" command--unused memory resources for the last N intervals.
*/

drop monun;

store monun
select
        WAIT_QUEUES     = "avg monitoring interval (minutes):",
        time = bcdfixed(5,2,avg(bcdflt(0,length))/3600)
        from monitor
select
        SEQNO           = seqno,
        UNUSED_INPUT    = unin,
        UNUSED_OUT      = unout,
        UNUSED_MEM      = unmem,
        UNUSED_DBIN     = undbin
        from monitor
        where (date=getdate and time >gettime - length * integer(&INTERVALS))
        or (date=getdate-1 and time> gettime - length * integer(&INTERVALS)
                        + integer(24) * integer(60) * integer(3600)
        )
        order by date,time
end store;

comment on monun is "Unused memory buffers for last N monitor intervals","1M";
```

monwait (mon5.sql)

```
/*
** "monwait" command--wait queues from monitor for the last N intervals.
*/

drop monwait;

store monwait
select
        WAIT_QUEUES     = "avg monitoring interval (minutes):",
        time = bcdfixed(5,2,avg(bcdflt(0,length))/3600)
        from monitor
select
        SEQNO     = seqno,
        INPUT_sec       = bcdflt(5,bcdflt(5,inwait) / 60),
        OUTPUT_sec      = bcdflt(5,bcdflt(5,outwait) / 60),
        CPU_sec         = bcdflt(5,bcdflt(5,cpuwait) / 60),
        MEMORY_sec      = bcdflt(5,bcdflt(5,memwait) / 60),
        TAPE_sec        = bcdflt(5,bcdflt(5,tapewait) / 60)
        from monitor
        where (date=getdate and time >gettime - length * integer(&INTERVALS))
        or (date=getdate-1 and time> gettime - length * integer(&INTERVALS)
                        + integer(24) * integer(60) * integer(3600)
        )
        order by date,time
end store;

comment on monwait is "Summary of wait queues for last N monitor intervals","1M";
```

notowned (notowned.sql)

```
/*
** Define "notowned" command.
*/

/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop notowned;

store notowned
select distinct name,
                relid,
```

```
                owner_num = owner
                from relation r
                where not exists
                     (select any(id) from users
                                  where id = r.owner)
                order by owner, name
end store;

comment on notowned is
"List objects which are not not owned by people in 'users'","1D";
```

othercmds (objects7.sql)

```
/*
** store "othercmds" command.
*/

store othercmds
select distinct command = object,
                owner,
                logging
        from objects
        where type = "C"
        and ((ownerid = userid or ownerid = dba)
            or userid =dba  /* show all relations for dba  */
         )
        /* exclude any predefined stored commands */
        and relid not in (select relid from descriptions
                                  where key like "1_")
        order by owner, object
end store;

comment on othercmds is
"Displays user stored commands (those not displayed by cmds)","1L";
```

otherviews (objects7.sql)

```
/*
** store "otherviews" command.
*/

store otherviews
select distinct  views = object,
                 owner,
                 logging
       from objects o
        where type = "V"
```

```
        and ((ownerid = userid or ownerid = dba)
            or userid =dba  /* show all relations for dba  */
          )
        /* exclude any cmds in comments */
        and "V1" != all(select key from descriptions d
                        where relid = o.relid)
        order by owner, object
end store;

comment on otherviews is
"Displays user views (those not displayed by views)","1L";
```

permits (protect5.sql)

```
/*
** "permits" commands
*/

/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop permits;
drop ptype_I;
drop atype_I;

/* If there is no user tuple for "all" is id=0, put one there */

insert into users(id,name) values(0,"ALL");

/*
**  ptype_I relation contains the decoding of the access attribute
**  of the protect system relation
*/

create table ptype_I (access tinyint, desc char(20));

insert into ptype_I(access, desc) values( 1, " read");
insert into ptype_I(access, desc) values( 2, " write");
insert into ptype_I(access, desc) values( 3, " all");
insert into ptype_I(access, desc) values( -32, " execute");
insert into ptype_I(access, desc) values( -53, " create database");
insert into ptype_I(access, desc) values( -58, " create");
insert into ptype_I(access, desc) values( -56, " create index");
insert into ptype_I(access, desc) values( 4, " read tape");
insert into ptype_I(access, desc) values( 8, " write tape");
```

```
insert into ptype_I(access, desc) values( 12, " all tape");

create unique clustered index on ptype_I (access);

update relation set type="I" where relid=table_id("ptype_I");

/*
**  Atype_I decodes the bit pair for permit/deny in protect.attmap
*/

create table atype_I(access tinyint, meaning char(8));
insert into atype_I(access,meaning) values(1, "permit");
insert into atype_I(access,meaning) values(2, "deny");
insert into atype_I(access,meaning) values(3, "BOTH!");

create unique clustered index on atype_I (access);

update relation set type="I" where relid=table_id("atype_I");

/* permits   returns explicit permissions on a given OBJECT  */

store permits
        select access = concat(a.meaning, p.desc),
                object = &object,
                user = u.name
                  from users u, atype_I a, ptype_I p, protect t
                  where a.access = mod(tinyint(substring(1,1,t.attmap)), 4)
                  and p.access = t.access
                  and table_id(&object) = t.relid
                  and u.id = t.user
                  and table_id(&object) > 0
end store;

comment on permits is "shows the explicit protection for an OBJECT","1P";
```

permitsall (protect5.sql)

```
/* permitsall returns all explicit permissions from the protect relation */

drop permitsall;

store permitsall
 select access = concat(a.meaning, p.desc),
         object = r.name,
         user = u.name
         from users u, atype_I a, ptype_I p, protect t, relation r
         where a.access = mod(tinyint(substring(1,1,t.attmap)), 4)
         and p.access = t.access
         and t.relid = r.relid
         and u.id = t.user
```

```
end store;
comment on permitsall is "shows all explicit permits and denies","1P";
```

permitsgen (protect5.sql)

```
/* permitsgen   returns explicit tape and create permissions */

drop permitsgen;

store permitsgen
 select access = concat(a.meaning, p.desc),
       user = u.name
         from users u, atype_I a, ptype_I p, protect t
         where a.access = mod(tinyint(substring(1,1,t.attmap)), 4)
         and p.access = t.access
         and t.relid = 0
         and u.id = t.user
end store;

comment on permitsgen is "shows the explicit tape and create permissions","1P";
```

permitsme (protect5.sql)

```
/*  permitsme returns explicit permissions on all objects for this user*/

drop permitsme;

store permitsme
 select access = concat(a.meaning, p.desc),
         object = r.name
         from users u, atype_I a, ptype_I p, protect t, relation r
         where a.access = mod(tinyint(substring(1,1,t.attmap)), 4)
         and p.access = t.access
         and t.relid = r.relid
         and u.id = t.user
         and u.id = userid
end store;

comment on permitsme is "shows the explicit permissions for me","1P";
```

permitsuser (protect5.sql)

```
/*  permitsuser  returns explicit permissions on all objects for a user */

drop permitsuser;

store permitsuser
 select access = concat(a.meaning, p.desc),
        object = r.name
        from users u, atype_I a, ptype_I p, protect t, relation r
        where a.access = mod(tinyint(substring(1,1,t.attmap)), 4)
        and p.access = t.access
        and t.relid = r.relid
        and u.id = t.user
        and u.name= &user
end store;

comment on permitsuser is "shows the explicit permissions for  USER","1P";
```

pgms (objects7.sql)

```
/*
** store "pgms" command.
*/

store pgms
select distinct pgms = object,
                owner,
                logging
        from objects
        where type = "P"
        and ((ownerid = userid or ownerid = dba)
            or userid =dba   /* show all relations for dba  */
          )
        order by owner,object
end store;

comment on pgms is "Displays stored programs owned by you or the dba","1L";
```

ps (ps.sql)

```
/*
** "ps" command: Show current dbin status
*/

/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation


drop ps;                    /* Destroy stored command...   */
drop ps_data_I;     /* ... & constant relation.   */

/*
** Create constant relation, for translating codes from
** dbinstat relation
*/

create table ps_data_I
        (status tinyint,  /* status code, matching that    */
                                /* from "dbinstat.status"          */
         meaning char(30)       /* string description of status.*/
        );


update relation set type ="I" where relid=table_id("ps_data_I");

/* Now populate the constant relation. */

insert into ps_data_I (status, meaning) values (1, "UNDEFINED");
insert into ps_data_I
        select status  = status + max(status),
                meaning = "UNDEFINED" from ps_data_I;
insert into ps_data_I
        select status  = status + max(status),
                meaning = "UNDEFINED" from ps_data_I;
insert into ps_data_I
        select status  = status + max(status),
                meaning = "UNDEFINED" from ps_data_I;
insert into ps_data_I
        select status  = status + max(status),
                meaning = "UNDEFINED" from ps_data_I;
insert into ps_data_I
        select status  = status + max(status),
                meaning = "UNDEFINED" from ps_data_I;
insert into ps_data_I
        select status  = status + max(status),
```

```
                meaning = "UNDEFINED" from ps_data_I;
delete from ps_data_I where status > 50;

update ps_data_I set meaning= "runnable"  where status = 1;
update ps_data_I set meaning= "wait--input; no transaction"  where status = 2;
update ps_data_I set meaning= "wait--input in a transaction"  where status = 3;
update ps_data_I set meaning= "DEBUGGING (timed out)"  where status = 4;
update ps_data_I set meaning= "DEBUGGING"  where status = 5;
update ps_data_I set meaning= "DEBUGGING"  where status = 6;
update ps_data_I set meaning= "wait--output to drain"  where status = 7;
update ps_data_I set meaning= "DEBUGGING"  where status = 8;
update ps_data_I set meaning= "suspended--checkpoint"  where status = 9;
update ps_data_I set meaning= "terminated abnormally"  where status = 10;
update ps_data_I set meaning= "wait--tape controller"  where status = 11;
update ps_data_I set meaning= "wait--child to terminate"  where status = 12;
update ps_data_I set meaning= "wait--special disk command"  where status = 13;
update ps_data_I set meaning= "DEBUGGING"  where status = 14;
update ps_data_I set meaning= "wait--maint port input"  where status = 18;
update ps_data_I set meaning= "wait--transaction lock"  where status = 30;
update ps_data_I set meaning= "wait--interruptible quick lock"  where status = 31;
update ps_data_I set meaning= "wait--non-interrupt quick lock"  where status = 32;
update ps_data_I set meaning= "wait--one disk I/O"  where status = 35;
update ps_data_I set meaning= "wait--multiple disk I/O's"  where status = 36;
update ps_data_I set meaning= "wait--process memory"  where status = 40;
update ps_data_I set meaning= "wait--process memory-has input"  where status = 41;
update ps_data_I set meaning= "normal exit, draining output"  where status = 42;

store ps
   select distinct
                    ds.dbin,
                    STATUS = p.meaning,
                    block = integer(ds.block),
                    ds.time,
                    DATABASE = db.name
                from dbinstat ds, databases db , ps_data_I p
                where ds.dbid *= db.id
                and mod(ds.status,50) = p.status  /* ignore CANCEL STATUS */
                order by dbin

   select distinct
                    ds.dbin,
                    USER = u.name,
                    ds.hid,
                    ds.huid
              from dbinstat ds, users u, host_users hu
            where u.id  =* hu.uid
            and hu.hid  =* ds.hid
            and hu.huid =* ds.huid
            order by USER
end store;

comment on ps is  "Lists the active processes and status--known users","1M";
```

rename (rename.sql)

```
/*
** Rename permits users to change the name of tables they own
*/

/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop rename;

store  rename
update relation set name=&newname
        where name=&curr_name
       and owner = userid
end store;

comment on rename is
"Allows users to rename their RELATION to NEW_NAME","1L";
```

rmuser (rmuser.sql)

```
/*
**  Define "rmuser" command - remove user
*/

/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop rmuser;

store rmuser
delete from host_users
        where uid = (select id from users
                                where name = &name
                                and id != dba
                    )
delete from users
        where name = &name
```

```
        and id != dba
end store;

comment on rmuser is
"Removes a USER from database - for DBA only", "1D";
```

setcard (setcard.sql)

```
/*
**      Setcard command for set cardinality of an index
*/

/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop setcard;

store setcard
     update indices set card= &3card
                where relid = table_id(&1relation)
                  and indid = &2indexid
end store;

comment on setcard is "Set cardinality for TABLE, INDEXID, CARD","1D";
```

size (size.sql)

```
/*
** report sizes of objects in blocks
*/

/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop size;

store size
```

```
  select r.relid,
         rel_tups=tups,
         rel_pages=pages,
         num_blocks  = count(num)
        from relation r, blockalloc b
        where b.relid = table_id(&name)
        group by b.relid
        having b.relid = table_id(&name)
        and b.relid = r.relid
end store;

comment on size is
"Displays size of RELATION","1S";
```

sizebyzone (szbyzone.sql)

```
/*
** Define "sizebyzone" command.
*/

/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop sizebyzone;

store sizebyzone
  select zone = substring(1,3,b.tid),
         blocks = count(b.relid)
        from blockalloc b
        where b.relid = table_id(&object)
        and table_id(&object) > 0
        group by substring(1,3,b.tid)
        having b.relid = table_id(&object)
        and table_id(&object) > 0
end store;

comment on sizebyzone is
"For OBJECT, lists number of blocks in each zone","1S";
```

sizes (sizes.sql)

```
/*
** Report sizes of objects in blocks
*/

/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop sizes;

store sizes
select total_blks = sum ( du.high -du.low + 1),
      free_blks = count(b.relid),
      percent_used =
                (count(bl.relid) * 100) /
             sum ( du.high - du.low + 1)
      from disk_usage du, blockalloc b, blockalloc bl
      where du.relid = -32768
      and b.relid = 0
      and bl.relid != 0

select r.name,
      r.owner,
      r.type,
      rel_pages =r.pages,
      num_of_blks =  count(bl.relid )
    from relation r, blockalloc b, blockalloc bl
    where bl.relid = r.relid
    group by r.relid
    having 10 < (select count(b.relid)
                        where b.relid = r.relid
                        group by r.relid
                )
    order by num_of_blks d
end store;

comment on sizes is
"Displays size of this database and its larger relations","1S";
```

spacebyuser (spbyuser.sql)

```
/*
** Define "spacebyuser" command.
*/

/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop spacebyuser;

store spacebyuser
        select user = u.name,
               user_id = u.id,
               pages = count(b.mode)
            from users u, relation r, blockalloc b
            where r.owner = u.id
            and b.relid=r.relid
            group by u.name
            having 0< ( select count (b.mode)
                                where b.relid =r.relid
                                and u.id = r.owner
                                group by u.id
                      )
   order by pages d
end store;

comment on spacebyuser is
"Lists disk space utilization by user","1S";
```

system (system.sql)

```
/*
** Define "system" command.
*/

/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation
```

```
drop system;

store system
select distinct tab_name = name,
                type,
                tuples = tups,
                pages
        from relation
        where type = "S" or type = "T"
                /* S = system, T = transaction logs */
        order by tab_name
end store;

comment on system is
"Displays system relation names and sizes","1D";
```

tabs (object7.sql)

```
/*
** Store "tabs" command.
*/
drop tabs;

store tabs
select distinct table_name = object,
                owner,
                rows,
                logging
        from objects
          where type = "U"
          and ((ownerid = userid or ownerid = dba)
              or userid =dba     /* show all relations for dba  */
             )
          order by owner, object
end store;

comment on tabs is
"Displays user tables owned by you or the dba","1L";
```

usersright (uright.sql)

```
/*
** Define "usersright" command.
*/

/*
** Stored command scripts do not use continuation characters.
```

```
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop usersright;

store usersright
select distinct hid,
                huid,
                uid,
                problem= "In 'host_users' not in 'users'"
        from host_users
        where uid != all (select id from users)

select distinct name,
                id,
                gid,
                problem= "In 'users' not in 'host_users'"
        from users
        where id != all (select uid from host_users)
        and gid != id           /* exclude groups */

select distinct name,
                id,
                gid,
                problem= "Group--no members or user--no group"
        from users u
        where 1 =(select count(gid) from users
                                group by gid having id = u.id
                 )

select distinct name,
                id,
                gid,
                problem= "Duplicate id's in 'users'"
        from users u
        where  1< (select count(id) from users where id= u.id)

select distinct name,
                id,
                gid,
                problem= "Inappropriate use of uid = 0"
        from users
        where id = 0
        and name != "ALL"

select distinct hid,
                huid,
                uid,
                problem="Inappropriate use of uid = 0"
```

```
        from host_users
        where uid = 0

select distinct hid,
                huid,
                uid,
                status = "DBA(s) for this database"
        from host_users
        where uid = dba
end store;

comment on usersright is
"Checks consistency of 'users' with 'host_users'","1D";
```

uses (uses.sql)

```
/*
** Stored command to give the list of objects a named object depends on
** Usage is "uses <objectname>"
*/

/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop uses;

store uses
        select table_used = table_name(c.relid),
             owner = u.name
            from crossref c, users u, relation r
            where c.drelid = r.relid
            and r.relid = table_id (&table)
            and u.id =* r.owner
end store;

comment on uses is "list objects that OBJECT depends on","1L";
```

views (views.sql)

```
/*
** define "views" command.
*/

/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop views;

store  views
select views = r.name,
              description = substring(1,60,d.text)
        from relation r, descriptions d
        where r.relid = d.relid
        and (r.owner = dba or r.owner = userid)
        and r.type = "V"
        and d.key = "V1"
        order by r.name, d.key
end store;

comment on views is
"Lists standard views with a one line description","1L";
```

who (who3.sql)

```
/*
**  Define "who" command
*/

/*
** Stored command scripts do not use continuation characters.
** We turn continuation off here in case the user or system
** profile files turned them on.
*/

%continuation

drop who;
store who
        select user = u.name,
              group_name = t.name,
              hst_id = h.hid,
```

```
                host_user_id = h.huid,
                id=u.id
            from users u, users t, host_users h
            where h.uid = u.id
            and t.id =* u.gid     /* get users with no groups  */
            and u.id != u.gid     /* skip groups  */
            order by group_name,user
end store;

comment on who is
"Lists users who can use this database", "1D";
```

whois (who3.sql)

```
/*  Define whois command  to find userid from name  */

drop whois;

store whois
        select
                id = u.id,
                group_name = u1.name,
                hst_id = hu.hid,
                host_user_id = hu.huid
            from host_users hu, users u, users u1
            where hu.uid = u.id
            and u1.id =* u.gid /* outer join to get groupless users */
            and u.name = &username
end store;

comment on whois is
"Returns userid, host_id and huid for a given username", "1D";
```

whoisid (who3.sql)

```
/*  Define whoisid command to find username from userid */

drop whoisid;

store whoisid
        select
                u.name,
                group_name = u1.name,
                hst_id = hu.hid,
                host_user_id = hu.huid
            from host_users hu, users u, users u1
            where hu.uid = u.id
```

```
            and u1.id =* u.gid /* outer join to get groupless users */
            and u.id = $user_id
end store;

comment on whoisid is
"Returns user name, host_id and huid for a given user_id", "1D";
```

Appendix D: Data Tables

This appendix lists the data tables (views and relations) used by the predefined stored commands.

atype_I (protect5.*xxx*)

This is a decoder for protection bits in the "protect" relation.

1) select * from atype_I

access	desc
1	permit
2	deny
3	BOTH!

3 rows affected

calendar_I (date8.*xxx*)

This is the mapping of the years to days from January 1, 1900 used by most date conversions. It is modified by the **base1970** command to contain "start" values that are 25,566 days smaller.

2) select * from calendar_I

start	shortyear	longyear	leapyear
31045	85	1985	0
31410	86	1986	0
31775	87	1987	0
32140	88	1988	1
32506	89	1989	0
32871	90	1990	0
33236	91	1991	0
33601	92	1992	1
33967	93	1993	0
34332	94	1994	0
34697	95	1995	0

11 rows affected

config_I (config.*xxx*)

This decodes the "type" fields in the "configure" system relation.

1) select * from config_I

type	name
A	Accelerator, value=WCS level
B	Block multiplexer host interface
C	Checkpoint interval
D	Default char set 0=ASCII 1=EBCDIC
E	Ethernet host interface
I	IDM I.D. for IDENTIFY response
K	Memory configuration
M	Monitor interval(minutes)
P	IEEE-488 host interface
R	IDM software release
S	RS-232 host interface
T	Mag tape interface, value=4 for h/p

12 rows affected

day_I (date8.*xxx*)

The days of the week for date conversions.

3) select * from day_I

number	day	shortday
0	Monday	Mon
1	Tuesday	Tue
2	Wednesday	Wed
3	Thursday	Thu
4	Friday	Fri
5	Saturday	Sat
6	Sunday	Sun

7 rows affected

dbs0_I (dbs.*xxx*)

This table contains the status values currently defined from the "databases" system relation.

2) select * from dbs0_I

value	text
1	Database/ disks not on-line
3	Database/ disks not on-line
9	Locked: database marked unrecoverable (ASCII)
11	Locked: database marked unrecoverable (EBCDIC)
17	Unused since boot, recovery not needed (ASCII)
19	Unused since boot, no recovery needed (EBCDIC)
33	On-line (ASCII)
35	On-line (EBCDIC)
49	Unused since boot, recovery had to run (ASCII)
51	Unused since boot, recovery had to run (E)
65	being loaded or rolled forward (ASCII)
67	being loaded or rolled forward (EBCDIC)
69	being created or destroyed (ASCII)
71	being created or destroyed (EBCDIC)
145	Locked: unused since boot-no recovery needed (A)
147	Locked: unused since boot-no recovery needed (E)
161	LOCKED for dump/load/rollforward (on-line) (A)
163	LOCKED for dump/load/rollforward (on-line) (E)
177	LOCKED: Unused since boot, recovery run (ASCII)
179	LOCKED: Unused since boot, recovery run (E)
193	LOCKED: being loaded or rolled forward (ASCII)
value	text
195	LOCKED: being loaded or rolled forward (E)
197	LOCKED: being created or destroyed (ASCII)
199	LOCKED: being created or destroyed (EBCDIC)

24 rows affected

dtype_I (atts.idl)

This turns the codes from the "attributes" relation into the mnemonics that IDL uses.

4) select * from dtype_I

mne	code	uncomp	dpb	adj
bcdfit	35	-1	2	3
bin	45	-1	1	0
bcd	46	-1	2	3
c	47	-1	1	0
ubcdfit	35	1	2	3
ubin	45	1	1	0
ubcd	46	1	2	3
uc	47	1	1	0
i	48	0	1	0
i	52	0	1	0
i	56	0	1	0
f	57	0	1	0
f	60	0	1	0

13 rows affected

dtype_SQL (atts.sql)

This turns the codes from the "attributes" relation into the mnemonics that SQL uses.

1) select * from dtype_SQL

mne	code	uncomp	dpb	adj
bcdfit(	35	-1	2	3
binary(	45	-1	1	0
bcd(	46	-1	2	3
char(	47	-1	1	0
fixed bcdfit(	35	1	2	3
fixed binary(	45	1	1	0
fixed bcd(	46	1	2	3
fixed char(	47	1	1	0
tinyint (	48	0	1	0
smallint (	52	0	1	0
integer (	56	0	1	0
smallfloat (	57	0	1	0
float (	60	0	1	0

13 rows affected

host_users_I (groups3.*xxx*)

This is the copy of the "host_users" relation used by the **group_off** and **group_on** commands.

5) select * from host_users_I

hid	huid	uid

0 rows affected

itype_I (indexes.*xxx*)

This contains the decoding values for index types from the "indices" relation.

6) select * from itype_I

type	desc
1	unique noncluster
-3	unique noncluster
0	nonclustered
2	clustered
-2	clustered
-1	unique clustered
3	unique clustered

7 rows affected

julian_I (date8.*xxx*)

The current-year view of the "calendar_I" relation.

7) select * from julian_I

start	shortyear	longyear	leapyear	displacement
31045	85	1985	0	8

1 row affected

lockdef_I (mon5.*xxx*)

Locking is decoded from the "locks" system relation.

3) select * from lockdef_I

code	meaning
01	read lock
02	write lock
03	read, intend to set block write lock
05	intend to set block read lock
06	intend to set block write lock

5 rows affected

logged_I (objects6.*xxx*)

The "logged" flag from the "relation" relation.

8) select * from logged_I

value	text
0	Not Logged
64	Logged

2 rows affected

mask_I (channel.*xxx*)

This is a convenient way to find bit values when binary fields are encoded. It is used by the **channels** command to tell what bits are set in the "configure" system relation tuples.

4) select * from mask_I

bit	value
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768
16	65536
17	131072

18 rows affected

mon_I (date8.*xxx*)

A view of the months of the current year. This is a view of "month_I" that alters the "first" and "last" attributes.

9) select * from mon_I

first	last	month	month_num
1	31	January	01
32	59	February	02
60	90	March	03
91	120	April	04
121	151	May	05
152	181	June	06
182	212	July	07
213	243	August	08
244	273	September	09
274	304	October	10
305	334	November	11
335	365	December	12

12 rows affected

month_I (date8.*xxx*)

A tape containing the months of the year and their starting days. The "leapfirst" and "leaplast" help correct these starting dates for leap years when multiplied by "calendar_I.leapyear".

10) select * from month_I

first	last	month	month_num	leapfirst	leaplast
1	31	January	01	0	0
32	59	February	02	0	1
60	90	March	03	1	1
91	120	April	04	1	1
121	151	May	05	1	1
152	181	June	06	1	1
182	212	July	07	1	1
213	243	August	08	1	1
244	273	September	09	1	1
274	304	October	10	1	1
305	334	November	11	1	1
335	365	December	12	1	1

12 rows affected

number_I (date8.*xxx*)

This table does multiple jobs. It is used to get the "date" and "time" strings with preceding zeroes into a result. It is also used to adjust the day based on the hour displacement from Greenwich. The "adjust" column is modified by a join with "savings_I".

11) select * from number_I

number	str	adjust
0	00	-1
1	01	-1
2	02	-1
3	03	-1
4	04	-1
5	05	-1
6	06	-1
7	07	-1
8	08	-1
9	09	0
10	10	0
11	11	0
12	12	0
13	13	0
14	14	0
15	15	0
16	16	0
17	17	0
18	18	0
19	19	0
20	20	0

(continued next page)

number	str	adjust
21	21	0
22	22	0
23	23	0
24	24	0
25	25	0
26	26	0
27	27	0
28	28	0
29	29	0
30	30	0
31	31	0
32	32	0
33	33	0
34	34	0
35	35	0
36	36	0
37	37	0
38	38	0
39	39	0
40	40	0
41	41	0
number	str	adjust
42	42	0
43	43	0
44	44	0
45	45	0
46	46	0
47	47	0
48	48	0
49	49	0
50	50	0
51	51	0
52	52	0
53	53	0
54	54	0
55	55	0
56	56	0
57	57	0
58	58	0
59	59	0
60	60	0

61 rows affected

otype_I (objects6.*xxx*)

Used to identify the object type from "relation" with the **dir** command.

12) select * from otype_I

type	definition
S	System relation
T	Transaction log
U	User relation
C	Stored command
P	Stored program
F	File
I	Standard relation
V	User view

8 rows affected

ps_data_I (ps.*xxx*)

This table contains the interpretation of the status codes from the "dbinstat" system relation. It is used by the **ps** command.

5) select * from ps_data_I

status	meaning
1	runnable
2	wait--input; no transaction
3	wait--input in a transaction
4	DEBUGGING (timed out)
5	DEBUGGING
6	DEBUGGING
7	wait--output to drain
8	DEBUGGING
9	suspended--checkpoint
10	terminated abnormally
11	wait--tape controller
12	wait--child to terminate
13	wait--special disk command
14	DEBUGGING
15	UNDEFINED
16	UNDEFINED
17	UNDEFINED
18	wait--maint port input
19	UNDEFINED
20	UNDEFINED
21	UNDEFINED

(continued next page)

status	meaning
22	UNDEFINED
23	UNDEFINED
24	UNDEFINED
25	UNDEFINED
26	UNDEFINED
27	UNDEFINED
28	UNDEFINED
29	UNDEFINED
30	wait--transaction lock
31	wait--interruptible quick lock
32	wait--non-interrupt quick lock
33	UNDEFINED
34	UNDEFINED
35	wait--one disk I/O
36	wait--multiple disk I/O's
37	UNDEFINED
38	UNDEFINED
39	UNDEFINED
40	wait--process memory
41	wait--process memory-has input
42	normal exit, draining output
status	**meaning**
43	UNDEFINED
44	UNDEFINED
45	UNDEFINED
46	UNDEFINED
47	UNDEFINED
48	UNDEFINED
49	UNDEFINED
50	UNDEFINED

50 rows affected

ptype_I (protect5.*xxx*)

This table decodes the types of permissions from the "protection" relation.

14) select * from ptype_I

access	desc
-58	create
-56	create index
-53	create database
-32	execute
1	read
2	write
3	all
4	read tape
8	write tape
12	all tape

10 rows affected

savings_I (date8.*xxx*)

Two local values are stored in "savings_I" and they must be modified by the user for proper local date and time to work. "First" and "last" indicate the days on which daylight savings time shifts the clock. The displacement column contains the displacement from Greenwich Mean Time that separates local time between the days (Julian dates) indicated.

15) select * from savings_I

first	last	displacement
0	75	8
76	300	7
301	366	8

3 rows affected

users_I (groups3.*xxx*)

This is the temporary storage place for group members removed from database access by the **group_off** command. It serves the same function as "host_users_I".

16) select * from users_I

id	gid	name

0 rows affected

Index of Terms