Information Repository Requirements of the CORDS Multidatabase Service

Neil Coburn and Per-Åke Larson[*]
University of Waterloo
Waterloo, Canada

Report CS-93-15
(Version 1.0)

March 23, 1993

Abstract

A multidatabase is a *virtual database*. As such, it requires support for the storage and retrieval of its operational data. That is, is requires the equivalent of the data dictionary or catalogue found in traditional relational databases.

The CORDS project is a collaborative effort between several IBM laboratories and several North American universities. The goal of the CORDS project is to investigate and prototype a distributed environment which supports the development and operation of distributed applications. The CORDS Multidatabase project is one component within the CORDS environment. A second component of this environment is an Information Repository.

We are investigating the viability of using the Information Repository as a means of implementing the Multidatabase catalogue. This report states the (current) Multidatabase data requirements on the Information Repository. We have attempted to outline our future requirements as clearly and broadly as possible. However, since we are creating a research prototype the requirements may change as we gain experience and understanding.

1 Introduction

The Multidatabase Service (MDBS) and the Information Repository Service are two components within the CORDS environment. This document specifies the current and expected Information Repository Service requirements of the functional units within the Multidatabase component. This is not a definitive specification: as the functional units are completed and we gain experience with them the requirements may change.

This section contains a general discussion of information required by clients and servers. It also lists the functional units within the Multidatabase Service; including a description of the current packaging of functional units into servers. Each of the remaining sections discusses the repository requirements of one of the functional units.

1.1 Clients and Servers

The functional units of the Multidatabase Service are implemented as a set of interacting servers; more than one functional unit may be packaged in a single server. Thus units may be a client in one interaction and

[*]This research project is supported by IBM Canada Ltd., Toronto and by the Natural Sciences and Engineering Research Council of Canada.

a server in another interaction. The Multidatabase Service requires the Information Repository to act as a name server: maintaining location information so that these servers are available to the appropriate clients.

For each server the repository must maintain enough information for a client to be able to communicate with it. In a general client-server environment a client specifies a desired server by a type (i.e. a generic name) and requires the location of (at least) one instance of a server of that type and the protocol required to communicate with it. In our case, we assume that communication is performed by remote procedure calls (Sun RPC). Thus, we can ignore the protocol information. Also, this means that a server's type is specified by it's program number and version number. Therefore, for each server, the minimal amount of information required is:

- server program number
- server version number
- server address

The Monitor is another component of the CORDS environment. In order to monitor system performance other server data must also be maintained; for example, server availability, up/down time. We feel that this data should be maintained in the information repository. However, as we gain experience with the MDBS the performance demands may require that the data be stored by the monitor. For the time being we will include this type of data in the repository requirements.

Each server must be able to register and unregister itself with the Information Repository. The Sun RPC portmapper maintains a directory service for programs, procedures, and versions; other RPC environments have similar capabilities. It is not yet settled whether we will continue to use these RPC facilities or maintain all of this information in the Information Repository instead.

1.2 Functional Units

The following lists the functional units within the Multidatabase Service:

- Schema Integration (constructs MDBS tables from component tables)
- MDBS Client (the client library used by applications)
- Coordinator (coordinates query processing and execution)
- Catalogue (the MDBS "data dictionary")
- Parser (includes semantic checking)
- View Integration (translates MDBS queries to component queries)
- Optimizer (produces an "optimized" access plan)
- Plan Manager (stores access plans)
- Execution Engine (performs "global" data processing operations)
- MDBS Agent (MDBS unit associated with a component data source)
- Security Manager
- Transaction Manager
- Recovery Manager

- Log Manager
- Monitor (monitors system performance)
- System Manager (maintains capabilities and sets parameters)

The MDBS Catalogue is a set of cover routines for the Information Repository. All MDBS access to the Information Repository is performed via the Catalogue. Thus, the Catalogue has no data requirements of its own on the repository; it provides routines to service the data requirements of the other functional units. However, it may have some management or control requirements on the repository.

The MDBS Client runs as part of an application process; it will never be a separate process. Every other functional unit may be implemented as a separate process. As such we can consider them as potential servers. The current implementation packages functional units into servers as shown in Figure 1. Each functional

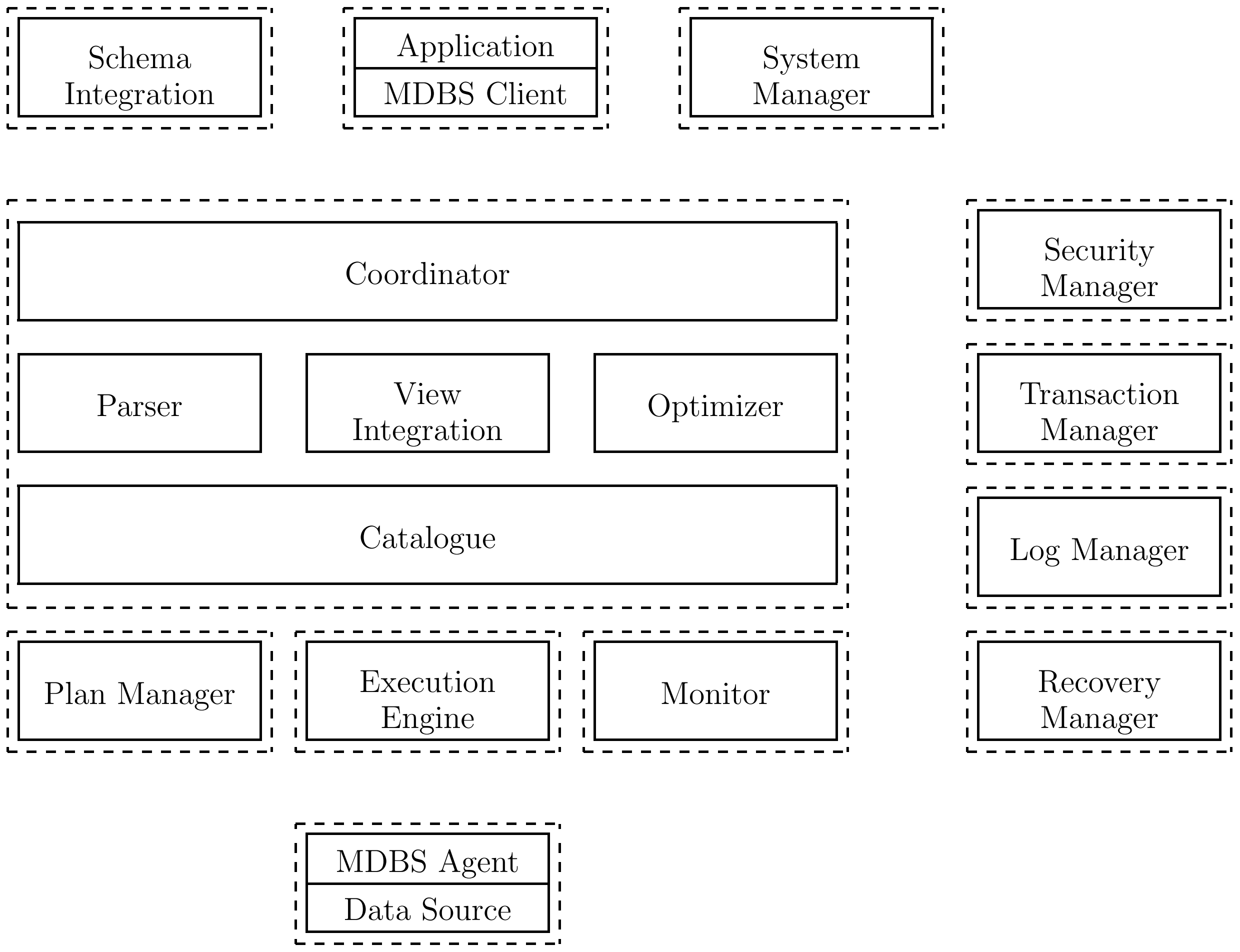

Figure 1: Packaging functional units into servers.

unit is represented as a solid rectangle. A group of functional units which are packaged into a single server is depicted by a dotted rectangle around the units. This grouping may change in future implementations.

In the remaining sections we discuss the information repository requirements of the functional units. Our implementation is based on (local and remote) procedure calls. We will denote the unit initiating an interaction as the *caller* and the unit being asked to perform some service as the *callee*. For each functional unit we list those functional units that correspond to callees in an interaction with this unit. If a callee is within the same server as the caller, then their interaction is achieved via a (local) procedure call. If a callee is not within the same server as the caller, then their interaction is achieved via a remote procedure call. In this latter case the caller acts as a client; it requires the repository to store the server information discussed earlier. Similarly, in each server one of the functional units must act as the server representative; registering and unregistering with the Information Repository Service.

2 Schema Integration

The Schema Integration unit requires the Information Repository to maintain schema information for the export schemas and application schemas (see Martin et al. [1]). Since the schemas may change, the Schema Integration unit must have insert, update, delete, and retrieve capabilities on this information.

- Application Schema
  - names of application level objects
  - definitions of application level objects
  - semantic constraints on application level objects
  - MDBS tables
    - name (local and global)
    - number of columns
    - table definition
    - export tables used to define this table
  - MDBS columns
    - domain
    - export columns used to populate this column
    - properties of the column
- Export Schema
  - names of export level objects
  - definitions of export level objects
  - semantic constraints on export level objects
  - Export tables
    - name (global)
    - number of columns
  - Export columns
    - domain information
    - properties of the column

The Schema Integration unit calls the Coordinator unit to submit requests to the Catalogue. This unit will probably be packaged with a windowing interface (i.e. an application) to provide a DBA tool for the MDBS.

3 MDBS Client

The MDBS client library calls the Coordinator unit to submit Open Database Connectivity (ODBC) requests to the MDBS. This unit is always packaged with an application.

4 Coordinator

A Coordinator acts as an MDBS server. It calls the following units:

- Catalogue
- Parser
- View Integration
- Optimizer
- Plan Manager
- Execution Engine
- Transaction Manager
- Recovery Manager
- Security Manager
- Monitor

As previously mentioned, in the current implementation the first four units are packaged with the Coordinator in the MDBS server. However, in future implementations they may be distinct servers.

5 Parser

The Parser requires information about application schema in order to perform parsing and semantic checking on the MDBS requests. The type of information it requires is the same as that outlined in the Schema Integration section. The only unit called by the Parser is the Catalogue.

6 View Integration

The View Integration unit takes a parsed MDBS request expressed in terms of an application schema and transforms it into a semantically equivalent set of requests where each request is expressed in terms of one of the export schemas. Thus, this unit requires the same information about application schemas and export schemas as that outlined in the Schema Integration section. The only unit called by the View Integration unit is the Catalogue.

7 Query Optimizer

The only unit called by the Query Optimizer is the Catalogue. The Optimizer requires data for two purposes: making optimization decisions and storing intermediate decisions during optimization.

In the latter case the Optimizer requires a “black box” field in which it can place any type of intermediate result it requires. It needs the ability to store, retrieve, and delete according to result identifiers. The identifiers will probably be generated by the Repository. For standard query optimization the retrieve and delete operations possibly may be combined into a single operation. If the Optimizer supports multiple query optimization then an intermediate result may be retrieved several times before being deleted.

In order to make optimization decisions the Optimizer needs data about component data sources and about the network links. This information is of two types: structural and performance. As discussed, the performance information will be stored in the Repository by a Monitor. The structural information will be stored in the Repository by the System Manager unit. Although the performance and the components of the distributed environment will change over time, the Optimizer requires only a retrieve capability for this information. The following is a list of the structural information required:

- Component Characteristics - Data Sources
  - source type - e.g. relational database, network database, file system
  - source product - e.g. Oracle version 5.0
  - storage capabilities
    - define temporary relations
    - build indices
  - processing capabilities
    - perform remote join
    - perform semi-join
    - perform max aggregation
    - perform min aggregation
    - perform sum aggregation
    - perform avg aggregation
    - perform count aggregation
    - perform sort
    - perform groupby
    - list of available indices
      - index name
      - index type
  - list of semantic constraints
    - schema
    - table
    - constraint type
    - constraint description
- Component Characteristics - Processors
  - number of processors
  - processor product - e.g. Intel 486, R/S 6000

  - processor capabilities/parameters
    - processor speed
    - cache size
  - operating system product - e.g. AIX version 3.2
  - operating system capabilities/parameters
    - description of optional software installed

- Network Characteristics - Protocols
  - protocol product - e.g. TCP/IP
  - protocol capabilities/parameters
    - connection-based

- Network Characteristics - Links
  - link product - e.g. Ethernet
  - link capabilities
    - specified transmission rate
  - communication parameters
    - usage fee rates
    - connection costs

These are the performance characteristics required:

- Component Characteristics - Data Sources
  - system estimates
    - number of users
  - processing estimates
    - number of tuples per table
    - average tuple size per table
    - minimum value in a column
    - maximum value in a column
    - number of distinct values in a column
    - distribution of values in a column - e.g. uniform, normal
    - list of available indices
      - index name
      - number of entries
      - selectivity

- Component Characteristics - Processors
  - storage device capabilities/parameters
    - retrieval rate
    - device capacity

- Network Characteristics - Links

    - communication estimates
        * effective transmission rate
        * failure rate
        * load

Due to the autonomy of the Component Data Sources, some of this data may not be available for some sources. Thus, a null value may occur as a valid data value. Because the performance data is used as the basis for making optimization decisions, we need to know how up-to-date it is. Therefore, each piece of this data has an associated timestamp.

The performance data needs to be specified and collected on a per-query basis; we cannot a priori specify all the database/table statistics that we would like to monitor. Thus, the Query Optimizer will include with an access plan a specification of the statistics to be collected when the access plan is executed. It is not clear at this time how this collection will be done. One possibility is that the Coordinator or Execution Engine passes the data collection specification to the Monitor when it is about to execute an access plan.

8 Access Plan Manager

The Access Plan Manager unit requires information about access plans and export schemas. The export schema information has been discussed in previous sections. The access plan information is of two principal types; the plan itself and information regarding the dependence of the plan on particular export schemas. Both types of information may have supplementary information associated with it. In both cases the plan or dependencies (and supplementary information) will be accessed as a single unit. To make storage and retrieval simpler the plans and dependencies will each be store as a single encoded field. The following lists the additional data required by the Access Plan Manager:

- Access Plans
    - plan identifier
    - timestamp
    - schema dependencies
    - size of plan
    - access plan
- Export Schema
    - schema identifier
    - timestamp

The export schema timestamp must reflect the latest modification time of any change to the schema; including changes to columns, tables, and indexes.

The Plan Manager is implemented as a server. The only unit called by the Plan Manager is the Catalogue.

9 Execution Engine

The MDBS will probably contain multiple instances of an Execution Engine; each will be implemented as a server. An Execution Engine calls the following units:

- Plan Manager

- Transaction Manager
- another Execution Engine (server)
- MDBS Agent

10 MDBS Agent

An MDBS Agent is implemented as a server; it acts as the server for a component data source. The Agent calls the Recovery Manager unit to facilitate recovery procedures in the event of a system crash at the component site.

11 Security Manager

Multidatabase security has not been seriously considered to date and is not included in our research plans. Thus, we cannot state what the Information Repository Service requirements will be. It is anticipated that a Security Manager will be implemented as a server. The Security Manager may require the Catalogue to store some information but the type of data and the frequency of requests are completely unknown at the present.

12 Transaction Manager

The Transaction Manager will probably be implemented as a server. This unit calls the following MDBS units:

- MDBS Agent
- another Transaction Manager (server)
- Log Manager
- Monitor

The following is a list of the additional data required for transaction management:

- Log Manager
  - attributes
    * interface description
    * log capacity
- Monitor
  - monitor type - i.e. transaction processing environment

Before we know the requirements this places on the Information Repository Service, we need to determine what data can be stored at the Transaction Manager site and at the Monitor site.

13 Recovery Manager

The Recovery Manager will probably be implemented as a server. This unit calls the following MDBS units:

- MDBS Agent
- another Recovery Manager (server)
- Log Manager
- Monitor

The additional data required for recovery management is the same as that required for transaction management. As mentioned previously, the requirements this unit places on the Information Repository Services depends on what statistical, attribute, and location data is stored at the Recovery Manager site and at the Monitor site.

14 Log Manager

A Log Manager may be implemented as a server or it may be packaged with a Transaction or Recovery Manager. The Log Manager will probably not call any other MDBS units.

15 Monitor

We have had only preliminary discussions on the Monitor unit; it will probably be implemented as a server. As discussed previously, we assume that the Monitor will store data in the Information Repository Service. Thus it will call the Catalogue unit but is unlikely to call any other MDBS unit.

At present, the only information that the Monitor will be required to collect is the performance information required by the Query Optimizer. Since the system's performance will change over time, the Monitor requires insert, update, and delete capabilities for this information. The performance characteristics required by the Query Optimizer were specified previously. They are included here for convenience:

- Component Characteristics - Data Sources
  - system estimates
    - number of users
  - processing estimates
    - number of tuples per table (*)
    - average tuple size per table (*)
    - minimum value in a column (*)
    - maximum value in a column (*)
    - number of distinct values in a column (*)
    - distribution of values in a column (*) - e.g. uniform, normal
    - list of available indices
      - index name
      - number of entries (*)
      - selectivity (*)
- Component Characteristics - Processors

  - storage device capabilities/parameters
    * retrieval rate
    * device capacity

- Network Characteristics - Links
  - communication estimates
    * effective transmission rate
    * load

Some of this information is (relatively) static and some will probably be monitored continuously. The rest (marked with an *) needs to be collected on a per-query basis. It is not clear how this will be accomplished; the Coordinator or Execution Engine may pass a specification to the Monitor when it executes an access plan.

16 System Manager

The System Manager unit is a tool of the MDBS DBA. It is used by the DBA to specify characteristics of component systems and to set the parameter values for instances of MDBS functional units.

The structural information required by the Query Optimize will be stored in the Repository by the System Manager unit. Since the components of the distributed environment will change over time, the System Manager requires insert, delete, and update capabilities for this information. The following is a list of the structural information required:

- Component Characteristics - Data Sources
  - source type - e.g. relational database, network database, file system
  - source product - e.g. Oracle version 5.0
  - storage capabilities
    * define temporary relations
    * build indices
  - processing capabilities
    * perform remote join
    * perform semi-join
    * perform max aggregation
    * perform min aggregation
    * perform sum aggregation
    * perform avg aggregation
    * perform count aggregation
    * perform sort
    * perform groupby
    * list of available indices
      · index name
      · index type
- Component Characteristics - Processors

  - number of processors
  - processor product - e.g. Intel 486, R/S 6000
  - processor capabilities/parameters
    - processor speed
    - cache size
  - operating system product - e.g. AIX version 3.2
  - operating system capabilities/parameters
    - description of optional software installed
- Network Characteristics - Protocols
  - protocol product - e.g. TCP/IP
  - protocol capabilities/parameters
    - connection-based
- Network Characteristics - Links
  - link product - e.g. Ethernet
  - link capabilities/parameters
    - transmission rate
  - communication parameters
    - usage fee rates
    - connection costs

Due to the autonomy of the Component Data Sources, some of this data may not be available for some sources. Thus, a null value may occur as a valid data value.

The System Manager unit calls the Coordinator unit to submit requests to the Catalogue. It may interface with other functional units (or their servers) in order to control MDBS performance by setting parameters in these units. This unit will probably be packaged with a windowing interface (i.e. an application) to provide a DBA tool for the MDBS.

17 Catalogue

The catalogue calls no other MDBS functional units. As discussed earlier, the catalogue provides a set of cover routines for the Information Repository Service. Therefore, it has no requirements for data storage in the Information Repository Service but rather it must provide facilities to meet the data storage requirements of all the other MDBS functional units. In brief, it must maintain information about MDBS schema, tables, and columns; export schema, tables, and columns; domains; users; applications; servers; component data sources; sites and communication links. All of these requirements appear in the section(s) for the relevant unit(s). The following relational schema for the catalogue is presented as a summary of the requirements we have discussed.

```
/*
 *      Relational Schema for MDBS Catalogue (revised version)
 *      Prepared by James W. Hong (Dec. 2, 1992)
 *        - after meeting Pat Martin and Mike Bauer at CAS on Dec. 1, 1992
 *      Modified by Neil Coburn (Mar. 3, 1993)
 *        - after discussions with Mike Bauer, James W. Hong and Pat Martin
 *
 */

table:  applicationUses                          /* key(applicationId, cdsId) */
        applicationId        integer  Not Null   /* foreign key for application.applicationId */
        cdsId                integer  Not Null   /* foreign key for componentDataSource.cdsId */

table:  column                                   /* key(columnId) */
        columnName           char (40,1)
        columnId             integer  Not Null
        columnType           char (10,1)         /* values: view, table, export */
        domainId             integer             /* foreign key for domain.domainId */
        nullValuesFlag       integer
        primaryKeyFlag       integer
        foreignKeyFlag       integer
        description          text (20,20,20,3)

table:  exportColumn                             /* key(columnId) */
/* This table contains a subset of the tuples in the column table.
 */
        columnName           char (40,1)
        columnId             integer  Not Null   /* foreign key for column.columnid */
        noOfIndexes          integer
        minValue             integer
        minValueTimestamp    date
        maxValue             integer
        maxValueTimestamp    date
        noDistinctValues     integer
        distinctValuesTimestamp date
        valueDistribution    char (40,1)
        description          text (20,20,20,3)

table:  exportColumnIndexes                      /* key(columnId, indexid) */
        columnId             integer  Not Null   /* foreign key for column.columnid */
        indexId              integer  Not Null   /* foreign key for index.indexId */

table:  columnUses                               /* key(resultColumnId, sourceColumnId) */
        resultColumnId       integer  Not Null   /* foreign key for column.columnid */
        sourceColumnId       integer  Not Null   /* foreign key for column.columnid */

table:  componentDataSource                      /* key(cdsId), key(cdsName) */
        cdsName              char (40,1)  Not Null
        cdsId                integer  Not Null
        siteId               integer  Not Null   /* foreign key for site.siteId */
        exportSchemaId       integer  Not Null   /* foreign key for schema.schemaId */
        description          text (20,20,20,3)
        type                 char (40,1)         /* values: relation, network, file, etc. */
        product              char (40,1)         /* values: Oracle 5.0, Vax DBMS, IMS, etc. */
        tempStorageFlag      integer
        buildIndexesFlag     integer
        remoteJoinFlag       integer
        semiJoinFlag         integer
        maxFlag              integer
        minFlag              integer
        sumFlag              integer
        avgFlag              integer
        countFlag            integer
        sortFlag             integer
        groupFlag            integer
        noOfUsers            integer
        noOfUsersTimestamp   date

table:  constraint                              /* key(constraintId) */
        constraintId         integer  Not Null
        description          text (20,20,20,3)

table:  domain                                  /* key(domainId), key(domainName) */
        domainName           char (40,1)  Not Null
        domainId             integer  Not Null
        domainType           char (10,1)
        domainUnits          char (10,1)
        domainMinValue       integer
        domainMaxValue       integer
        description          text (20,20,20,3)

table:  entityConstraint                        /* key(entityId, entityType, constraintId) */
        entityId             integer  Not Null   /* foreign key for column.columnId,
                                                  * exportTable.tableId, mdbsTable.tableId
                                                  * or schema.schemaId
                                                  */
        entityType           char (10,1) Not Null /* values: column, export, mdbs, schema */
        constraintId         integer  Not Null   /* foreign key for constraint.constraintId */

table:  exportTable                             /* key(tableId) */
        tableName            char (40,1)
        tableId              integer  Not Null
        noOfIndexes          integer
        noOfTuples           integer
        noOfTuplesTimestamp  date
        avgTupleSize         integer
        avgTupleSizeTimestamp date
        description          text (20,20,20,3)

table:  hasAccessTo                             /* key(userId, schemaId) */
        userId               integer  Not Null
        schemaId             integer  Not Null   /* foreign key for schema.schemaId */
        ownFlag              integer
        readFlag            integer
        updateFlag          integer
        insertFlag          integer
        deleteFlag          integer

table:  index                                   /* key(indexId) */
        indexId             integer  Not Null
        indexType           char (40,1)
        tableId             integer  Not Null   /* foreign key for exportTable.tableId */
        noOfColumns         integer
        indexSize           integer
        indexSizeTimestamp  date
        selectivity         char (10,1)
        selectivityTimestamp date

table:  linkedTo                                /* key(siteId1, siteId2) */
        siteId1             integer  Not Null   /* foreign key for site.siteId */
        siteId2             integer  Not Null   /* foreign key for site.siteId */
        linkProduct         char (40,1)
        protocol            char (40,1)
        protocolType        char (40,1)
        linkLoad            integer
        linkLoadTimestamp   date
        startUpCost         integer
        unitCost            integer
        specTransmission    integer
        effTransmission     integer
        effTransmissionTimestamp date

table:  domainMapping                           /* key(domainMapId) */
        domainMapName       char (40,1)
        domainMapId         integer  Not Null
        inputdomainNameList  char (200,1)
        outputdomainNameList char (200,1)
        description         text (20,20,20,3)

table:  application                             /* key(applicationId) */
        applicationName     char (40,1)
        applicationId       integer  Not Null
        schemaName          char (40,1)         /* foreign key for schema.schemaName */
        description         text (20,20,20,3)

table:  mdbsTable                                /* key(tableId), key(tableName) */
        tableName           char (40,1)  Not Null
        tableId             integer  Not Null
        tableDefinition     char (40,1)
        parseTree           bulk (20,10000,10000,10)
        description         text (20,20,20,3)
```

```
table:  tableColumns                                 /* key(tableId, tableType, columnId) */
        tableId             integer  Not Null        /* foreign key for mdbsTable.tableId
                                                      * and exportTable.tableId
                                                      */
        tableType           char  (10,1)             /* values: mdbs, export */
        columnId            integer  Not Null        /* foreign key for column.columnId */

table:  schema                                       /* key(schemaId), key(schemaName) */
        schemaName          char (40,1)  Not Null
        schemaId            integer  Not Null
        schemaType          char (40,1)
        cdsId               integer                  /* foreign key for componentDataSource.cdsId
                                                      *   For an export schema this indicates the
                                                      *   corresponding component data source;
                                                      *   otherwise, Null.
                                                      */
        description         text (20,20,20,3)
        timestamp           date                     /* time of latest modification to this schema */

table:  schemaTables                                 /* key(schemaId, tableId, tableType) */
        schemaId            integer  Not Null        /* foreign key for schema.schemaId */
        tableId             integer  Not Null        /* foreign key for mdbsTable.tableId
                                                      * and exportTable.tableId
                                                      */
        tableType           char  (10,1)             /* values: mdbs, export */

table:  site                                         /* key(siteId), key(siteAddress) */
        siteAddress         char (40,1)
        siteId              integer  Not Null
        processorType       char (40,1)
        processorProduct    char (40,1)
        noOfProcessors      integer
        processorSpeed      integer
        memorySize          integer
        cacheSize           integer
        avgDeviceRetrievalRate integer
        avgDeviceRetrievalRateTimestamp date
        availableDeviceCapacity integer
        availableDeviceCapacityTimestamp date
        operatingSystem     char (40,1)
        OSDescription       text (20,20,20,3)        /* Optional software */
        login               char (40,1)              /* Require secure access to each site;
                                                      * needs further investigation.
                                                      */
        avgLoad             integer
        avgLoadTimestamp    date
        siteDescription     text (20,20,20,3)

table:  tableUses                                    /* key(mdbsTableId, exportTableId) */
```

```
        mdbsTableId          integer  Not Null    /* foreign key for mdbsTable.tableId */
        exportTableId        integer  Not Null    /* foreign key for exportTable.tableId */

table:  user                                      /* key(userId)  OR  key(userId, siteId) */
/* This table contains information maintained to provide authorized access to the MDBS.
 * The exact requirements are not known at this time.
 */
        userName             char (40,1)
        userId               integer  Not Null
        siteId               integer  Not Null    /* foreign key for site.siteId */
        accessPriviledges    integer
        description          text (20,20,20,3)

table:  accessPlan                                /* key(planId) */
        planId               integer  Not Null
        timestamp            date                 /* time of access plan creation */
        sizeAccessPlan       integer
        accessPlan           text (20,20,20,3)

table:  accessPlanSchemas                         /* key(planId, schemaId) */
        planId               integer  Not Null    /* foreign key for accessPlan.planId */
        schemaId             integer  Not Null    /* foreign key for schema.schemaId */

table:  intermediateResult                        /* key(resultId) */
        resultId             integer  Not Null
        resultDescription    text (20,20,20,3)

table:  rpcServer                                 /* key(serverName),
                                                   * key(programNumber, versionNumber, siteId)
                                                   * key(siteId, portNumber)
                                                   */
        serverName           char (40,1)  Not Null
        programNumber        integer  Not Null
        versionNumber        integer  Not Null
        siteId               integer  Not Null    /* foreign key for site.siteId */
        portNumber           integer

table:  serverInfo                                /* key(serverName) */
/* This table contains data for non-RPC servers.
 */
        serverName           char (40,1)  Not Null
        serverType           char (40,1)          /* values: log manager, monitor, database, etc. */
        interface            char (40,1)
        description          text (20,20,20,3)
```

References

[1] P. Martin, M. Bauer, N. Coburn, P. Larson, G. Neufeld, J. Pachl, and J. Slonim. Directory Requirements

for a Multidatabase Service. In *Proceedings of the 1992 CAS Conference*, volume II, pages 339–350, November 1992.