

5. DII COE Data Concepts

In today's environment, warriors and supporting DOD personnel often:

- Cannot find the data they need;
- Cannot access it if they find it;
- Cannot use it within their local applications;
- Cannot send data to others when required;
- Are creating redundant, inconsistent “stovepipe” data stores and streams; and
- Have no proven, efficient data access and permissions security approach.

SHARED Data Engineering (SHADE) is the component within the DII COE whose purpose is to provide a comprehensive strategy for mitigating these problems. It includes data sharing approaches, data storage and access architectures, reusable software and data components, development guidelines, and standards for data service developers. SHADE's overall objective is to enable migration from many redundant, dissimilar, but overlapping, data stores to standardized COE-compliant data services built from “plug-and-play” components that blend multiple data technologies. To do this, the DII COE Data Engineering organization provides engineering support services for system developers and administrators that are intended to reduce barriers to interoperability, development costs, and schedules. These services include the organization and publication of existing components to encourage reuse. These services also encourage migration away from application-centric data stores to data servers built from common components and extended to meet application-specific requirements.

The DII COE provides guidance herein for transforming information systems software to be more open, portable, multi-tier, and interoperable. One of the minimal COE objectives (compliance level 5) is the separation of data from applications software to allow them to be managed and used independently. Application developers should use data related services provided by the COE rather than re-implementing them for each application. The DII COE uses COTS products, primarily relational database management systems (RDBMS), to provide mainline data services, while emphasizing coordinated management of the sharable data structures and semantics employed within RDBMS product frameworks.

5.1 The SHADE Vision

Key SHADE objectives are to

1. leverage investments in existing databases, data structures, and data values;
2. promote interoperability through their reuse; and
3. provide a foundation for data fusion.

A prerequisite to achieving these objectives is a common representation of battlefield data. The common representation provides “to be” migration objectives, a common understanding of warrior data, plus agreement on core objects, their identifiers, and valid domain values. Additionally, it constitutes the core set of battlefield data which mission applications extend as required. The common representation is maintained as a logical model, but it is manifested in multiple physical forms (e.g., Informix, Sybase or Oracle databases, XML documents, flat files, OODBMS, etc.). The common representation is being evolved by the COE Chief Engineer’s Data Engineering team from the existing C2 Core Data Model, a subset of the Defense Data Model (DDM), and from data structures/semantics used by key C3I systems. It is being made available as COE component database segments, XML tags/metadata, reference set code values, and other forms as required. These and other COE data products can be located via the COE’s Data Emporium (<http://diides.ncr.disa.mil/shade/index.cfm>).

The present DII COE supports a multi-tier architecture, which also applies to database services. Developers must preserve the independence of their applications, functioning as DBMS clients, from the data servers. Specifically, applications that access databases must not be built so that they have to reside on the data server in order to work correctly. It cannot be assumed that all operational sites will have a local data server. Further, where sites have a local data server, it may be on a separate machine that is dedicated to the DBMS, or the server may be collocated with the application on a single machine acting as the application server and the data server. Therefore, to maintain independence and support the client/server architecture, applications cannot assume they reside on the data server.

From the SHADE perspective, all data servers are shared assets, whether they are common or not, because they are accessed by multiple concurrent users. They are also dynamic because their data changes even if their structure remains static. Databases within the data server may be interdependent (see Figure 5-1). Databases can be accessed by applications other than those written by the database developer. The function of data servers, regardless of the specific set of COTS DBMS and database segments, remain the same:

- Support independent, evolutionary implementation of databases and applications accessing databases
- Manage concurrent access to multiple, independent, and autonomous databases
- Maintain integrity of data stored in the data server
- Provide discretionary access to multiple databases

- Sustain client/server connections independent of the client application's and data server's hosts
- Support distribution of databases across multiple hosts with replicated data and with distributed updates
- Provide maintainability of users' access rights and permissions
- Support backup and recovery of data in the databases.

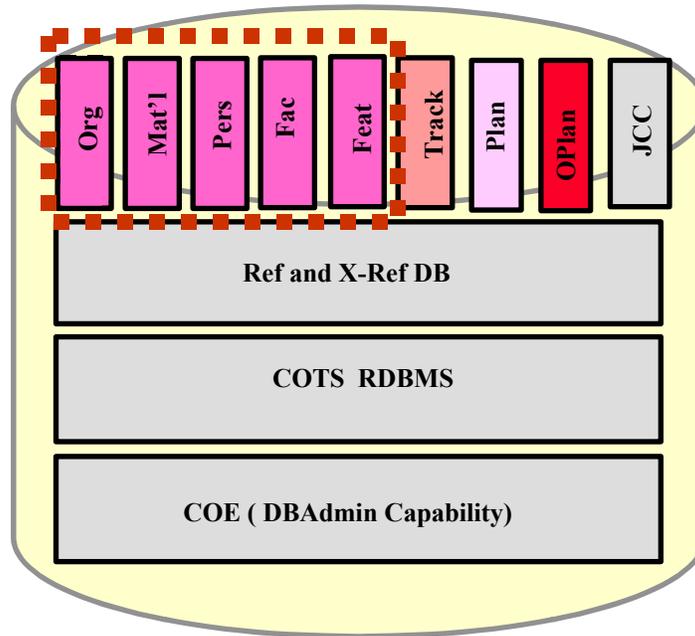


Figure 5-1: Shared Data Server Architecture

5.2 SHADE Terminology

Multi-tier architectures allow for flexible distribution of data service providers. A client application may interact with an application server (including web servers), which in turn, interact with a COE data server or the client may interact directly with a data server. A *data server* contains a DBMS and a collection of database and data segments. A data server and an application server may be co-resident on the same physical platform or they may each be implemented as a cluster of networked machines. The common availability of machines with multiple processors and software support for parallel operations makes these distinctions less important. Some data access services may be provided by software segments such as mediators or object request brokers (ORBs) that look like a DBMS to a client application and that look like a client to a DBMS or other data source.

In the context of the DII COE, a *DBMS* is a set of software segments that provide services such as data storage and indexing, query processing which returns both metadata and data, and data manipulation (e.g., the Oracle, Sybase, or Informix sets of segments). It is possible to install multiple DBMSs on a single physical machine although it is not common. It is more common to use a copy of a DBMS to manage multiple database instances.

A COE *database segment* contains the data definition language (DDL) needed to create all or a part of a single database instance, including data stores and data objects. A *database instance* contains a set of data, data structures, and other data objects that are managed as a single configuration item, although the objects may have been created by one or more database segments and be owned by one or more database accounts. A database instance includes the means of accessing a particular database, including the background processes, memory and so on. Once the database instance is created, the applications must set their environment appropriately to use it. The number of database instances that can be managed effectively by a single copy of a DBMS is dependent on the size of the database instances, the capabilities (storage and processing capacity) of the host machine, and the processing demand.

A database instance contains one or more *data stores*, which are the physical representation of the functional sets that have been grouped in a database segment. The physical implementation of the data stores varies depending on the DBMS being used.

Note: Data stores are often referred to in Oracle as tablespaces and in Sybase and Informix as databases.

Many types of *data objects* may be created and stored in database segments. These include:

- Tables and columns
- Views
- Triggers
- Stored procedures
- Constraints
- Indexes
- Users and Roles.

A *data segment* contains data to initialize, modify, or augment the data objects provided by a database segment. Databases that use reference tables to enforce database integrity often use an ordered set of dependent segments to define the tables, populate the reference tables, and then impose constraints on other tables based on the contents of the reference tables.

Data models are created using a number of syntactic formats: Entity-Relationship, IDEF1x, and UML. One can reverse engineer a database instance into a modeling tool as well as generating database segments from a data model. Unfortunately, reverse engineered data models tend to be insufficiently documented for interoperability. Models of physical database instances can be related to more abstract logical/conceptual models to address this problem.

5.3 Database Developer Guidelines

This section provides some general guidance to database developers on proper database usage within the context of the DII COE.

5.3.1 DBMS Usage

The developers of databases and applications accessing databases must conform to the COE data server environment so that they do not bypass its features. Conformance also limits the likelihood of data corruption. The combination of the data server configuration and the developers' implementations must ensure two things. First, each connection of a user to a database through an application must function in the proper context for that application and database. Second, each user's connection to a database must not interfere with any other user's connection to the same or any other database.

The principal consideration for developers is that their applications and databases no longer have exclusive use of the database management system. Instead of being an application-specific data management tool, the DBMS is a central, shared service that supports all applications' databases. As a result, developers cannot customize or tune the DBMS to the particular behavior of any single application. Any such modifications to the DBMS will inevitably affect other applications and databases. Similarly, the individual component databases are no longer the sole occupants of the DBMS. Developers shall implement their applications, constraints, and component databases so that they do not interfere with others sharing the same DBMS. Further, because there are multiple databases in the DBMS, applications can connect improperly to other databases. Developers shall ensure that their applications connect only to the database they intended to use. They shall also design their databases to maintain their own integrity without reference to external applications.

Database services within the COE are not restricted to a single vendor's DBMS. Database segments are dependent on the DBMS used for the data server. The specific commands used for their implementation within the DBMS and the environment it provides are both defined by the DBMS vendor. Database developers must be careful in their use of vendor-specific features so they do not create unintended dependencies on specific database management systems or, more importantly, particular versions of the DBMS, while still taking advantage of the database server's capabilities.

The same constraints on databases also apply to applications accessing those databases. Application developers shall ensure that applications connect through regular, documented APIs and shall not assume the use of particular DBMS versions. This does not prohibit developers from designing to the current version of a COE-compliant DBMS, using vendor-supplied tools that are part of the COE, or from accessing objects in other database segments. It prohibits developers from embedding DBMS vendor's runtime libraries or environment variables in the application segment. For example, developers should not provide their own `coraenv` script in the application segment because it creates an implicit version dependency on that version of the Oracle RDBMS.

In addition, this example interferes with the Database Administrator's (DBA) management functions.

The key to managing the evolution of databases and the applications that use them is documenting their interrelationships. Applications' dependencies on databases (and specific segments if appropriate) shall be documented so that database-segment version changes can be tested with the applications. The database's dependency on the DBMS shall also be documented for the same reason. If developers use DBMS vendor-supplied tools to implement applications, the dependency on the tools shall be documented. When applications or databases access data objects belonging to other databases, the dependency among the databases shall be documented as well. These dependencies are documented under the Database and Requires descriptors of the segment's SegInfo file. See Chapter 6 for more information.

5.3.2 DBMS Tuning and Customization

The core data server segments are configured and tuned by the organization responsible for it (e.g., DISA, GCCS, GCSS) based on the combined requirements¹ of all developers' databases (within the program or DOD wide) taken together.

Sizing of system recovery logs, log archiving directories, and users temporary workspace is based on the combination of the requirements of the various applications that use DBMS services. Developers must communicate their minimum requirements for these so that the core DBMS is not set to be too small. Most of the application tools provided by DBMS vendors are incorporated in the DBMS segment in the functional category of Server Tools. When such tools are used, the developer must identify the dependency under the database application segment's Requires descriptor.

Developers shall not modify the core DBMS instance's configuration. Extensions or modifications of that configuration require the specific approval of the DII COE Chief Engineer. If developers modify any of the executable tools (e.g., add User Exits to Oracle*Forms), then the modified version of the tool does not reside with the core database services, but becomes a part of the application's client segment. This prevents conflicts among different modified versions of a core function. The maintenance of that modified tool also becomes the responsibility of the developers.

5.3.3 Access Management

Access management requires the coordination of system and database administrators (DBA) to provide the users the permissions they need to use applications, databases, and applications connected to databases. They must also be able to revoke or modify those permissions as users transfer or assume different responsibilities. The large number of applications and databases available within COE-based systems could make the administrators' tasks unmanageable if access management is not supported with the

¹ An implication of this statement is that the combined requirements may lead to the need to develop a multiple instance database server segment.

proper tools. This section discusses the developers' responsibilities for supporting access management.

Traditionally, discretionary access control (DAC) has been used to manage users' permissions to employ applications to access or modify data managed by the data server. DAC had a broader scope than information security. Security focused on whether users are permitted to know about and allowed to view certain information. DAC dealt not only with users' permissions to change information but also the context in which they were permitted to make changes. Database access was discretionary because not all users had the same permissions to use applications.

To avoid problems with constraint enforcement in the DII environment, developers shall place their business rules and constraints in their databases rather than their applications. (It is sometimes necessary to include business rules in the application to improve performance.) The reason for placing constraints in the database is shown in Figure 5-2. Application 1 and its associated component database were implemented with business rules and constraints in the application. Application 2 placed those constraints in the database. When a third application (Browser) accesses both databases, it is unaware of Database One's business rules because they are inaccessible. If this application modifies Database One, it could unintentionally corrupt the database.

In the current DII COE environment, users will have access to multiple databases through many different applications. These applications include both traditional GOTS software, and COTS packages and browsers that provide ad hoc access directly to DBMS services. Therefore, a user's overall set of database permissions is the union of their individual accounts' database permission sets and the permission sets of the individual applications they have the right to use.

As data is shared more widely through the department, read-only access to data is less likely to be provided by a small set of locally managed applications. As Figure 5-2 illustrates, if critical business rules are only implemented in the applications, a wide variety of users may retrieve incomplete or incorrect data. This suggests a requirement for the developers to provide views of the data for external users designed to minimize the chances of incorrect interpretation of the data.

It is the responsibility of both database and application developers to provide discretionary access controls, including scripts for the DBA's use to add, modify and remove user privileges. The objective is to ensure that users' database connections operate in the proper context for the applications. Users must be able to operate several different applications at the same time. This means permission to access the specific tables and the mode (read or write) of that access. The operational sites' administrators are responsible for using those controls when assigning database and application privileges to users. The necessary access controls will be defined in the database segment design via grants and roles, as discussed in sections below. Users shall have unique accounts within the DBMS. Those accounts shall have only the database permissions needed for their work.

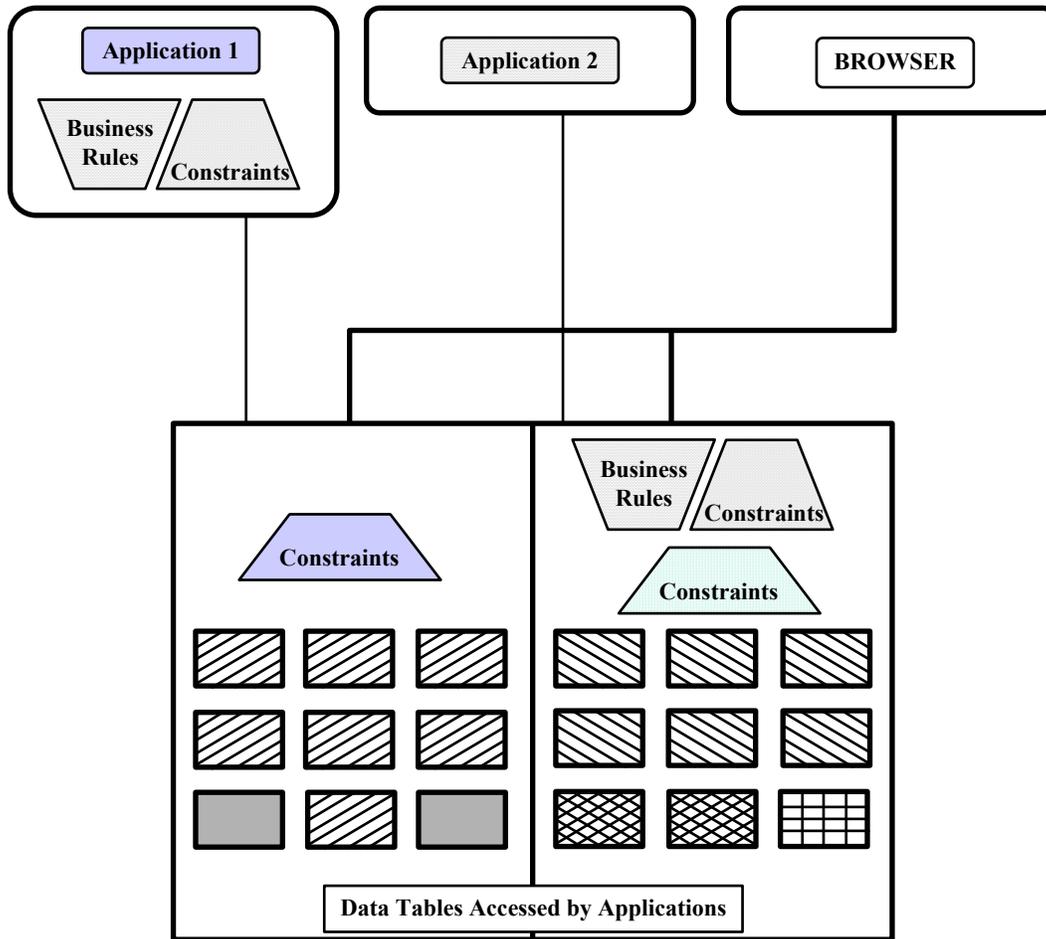


Figure 5-2: Business Rules and Constraints

The System and Database Administrators at each site are responsible for creating, modifying, and removing users' DBMS and UNIX accounts using COE Tools. This means, as required by the system Security Policy, that database actions can be traced to the individual user. Security auditing is the responsibility of the sites' DBAs. They are implemented as each site needs using the audit features provided by the DBMS. This means that users cannot use DBMS accounts defined by developers and that developers cannot assume the existence of any particular user accounts except for accounts created by the developer to support DBMS services.

The act of adding an application to a user's access list, menu, etc. entails adding associated database permissions to that user's DBMS account. Similarly, revoking access to an application requires that corresponding privileges be revoked within the DBMS. Users must have the proper permissions on both the application and the database, so the two system activities are interdependent. Access to applications will often be granted in

logical sets or groups of related applications. One application could have multiple permission sets if the same executable is used for both read-only and read/write accesses.

5.3.4 Database Accounts

Three categories of database accounts have been defined within the COE: DBAs, Owners, and Users. They have different functions and levels of access to the DBMS based on those functions.

5.3.4.1 Database Administrators

The Database Administrator (DBA) accounts have access to all parts of the DBMS. They are to be used only for system administration. Their use by database segments is prohibited except during the installation process as discussed in Chapter 6.

5.3.4.2 Database Owners

The Database Owner (DBO) accounts are the creators and owners of the data objects that make up an application's database segment. The name must be unique within the COE community; therefore developers should use the segment prefix or a variation of it as the owner account name. The segment prefix will also be used as the database schema name and will be incorporated in database file names as discussed below. Users shall not use the owner accounts to connect to databases. Developers shall not grant the DBA privilege to owner accounts.

Note: The COE Database administration tools (DBAdmSCreatedDS) will assign the segment prefix as the default owner and password for initial installation of the database segment.

All of the database segment's installation, except the definition of physical storage, the creation of the DBO, and the creation of database roles, must execute using the DBO account and password. After a successful installation of the data store segment, the DBO account's connect capability must be disabled.

5.3.4.3 Users

User accounts belong to the individuals accessing COE databases. Each individual must have a unique user account. User account naming conventions are defined by the individual COE program office (e.g., GCCS Chief Engineer) and will usually be the same as the user's operating system account. The user's account name must be unique within a specific COE database server and may be required to be unique within a COE program. Creation and maintenance of user accounts are a site-DBA responsibility within the rules provided by the specific COE program office. Developers shall not assume the existence of particular users and shall not create user accounts.

The creation of accounts that perform database services is an exception to the rule that developers not create user accounts. Such accounts support autonomous processes, such

as message parsers, that access a database on their own. These processes cannot connect to a database using the DBO account for reasons of security and data integrity, but their identity must be known to developers for their specialized database permissions to be set up correctly. Such accounts will be defined by the developer and created as a part of the segment installation.

5.3.5 Database Roles

Database roles implement the discretionary access controls and simplify the management of user privileges within the DBMS. They are created by the database segment developers or the developers of applications accessing databases to define sets of access privileges that can be given to users by their sites' DBA. Developers may create generic roles (or groups in Sybase terminology) that have of a set of privileges on the entire database or a subset of it. The permissions required by application's database roles are subject to review by the Chief Engineer and by the associated database segment's sponsor. These roles will be granted the privileges required by the application. When developers associate roles and their privileges with applications accessing the database, each role should be granted only the privileges needed by the application it supports. Grants are the permissions on database objects that allow users to access data they do not own. When a database object is first created, the only account that can access its contents is the owner of that object. Users must be explicitly granted permission to access an object. Privileges that can be granted include delete, insert, select, update (for tables and views); and execute (for procedures, functions, and packages). Privileges that should not be granted include index and alter (for tables). In order for the privileges on objects to be assigned to a role, the grantor must have permission to do so, and those database objects must exist.

Database roles shall not be granted to a DBA account because their administrative privileges already allow them to grant roles to users without owning the roles. The DBA account has all necessary privileges to assign grants on any object to any role. Developers shall not alter any database role defined by the COTS DBMS. A database segment creates the specific database roles for the application and connects to the DBA account (see the COEPromptPsswd API in Appendix C) to assign the grants on the required objects to the newly created role.

Granting data access to DBMS 'PUBLIC' users is prohibited. Granting data-access privileges to user accounts with the 'GRANT OPTION' or granting administration privileges on database roles is prohibited. Developers shall not make grants of application-level permissions to DBA accounts or to database roles used for DBMS administration. Where segments' applications or databases need special permissions on DBMS objects (e.g., query Oracle's 'v\$' tables), the developer must request them from the DII COE Chief Engineer.

The scripts that grant and revoke database roles are executed by the DBA when they are managing access to the applications. Such scripts should be installed on and removed from the server with the database segment(s) that need them.

Developers may need to create corresponding roles in multiple databases to satisfy an application's information needs. Since "database" roles are created at the database instance level, which segment they belong to is irrelevant. Since all objects must exist before the role may receive its grants, such roles shall be part of a dependent database segment as discussed in Chapter 6. That segment is dependent on every segment whose objects it references. It must list all of the segments under its `Requires` descriptor. See Chapter 6 for more discussion of the `Requires` descriptor.

5.4 Guidelines for Creating Database Objects

This section provides guidelines for developers in creating their database segments. Its objective is to support consistency across different databases and improve the mutual interoperability of the accessible databases. The guidelines strive to make sure database segments do not inadvertently affect each other directly or via changes to DBMS settings.

5.4.1 Database Object Naming Conventions

Developers should strive to make all object names meaningful. Names shall start with a letter of the alphabet and may include letters, numbers, and underscores. Names shall be 1 to 30 characters in length and shall not be a DBMS reserved word. If the database segment is a community or enterprise standard, then it must be viable for all COE RDBMS products (e.g. in the current COE versions of Informix, tablename are limited to 18 characters or less). Case does matter with names in the DII COE environment. While a specific RDBMS (such as Oracle) may not be case sensitive in naming objects, there are some (such as Sybase) which are case sensitive. To ensure consistency and portability of database objects and their elements, database object and element names shall be uppercase.

Role names shall be meaningful (the database or application name should be part of the group or role name) without using reserved words. They shall be a maximum of 30 characters (uppercase letters, numbers, and underscore).

Names of objects (including rules) created in other schemas must identify the inter-database linkage. Otherwise they are subject to the naming restrictions of their object type. Developers are responsible for ensuring that their object's names do not conflict with those already in the schema.

Data store names must also be associated with the segment and function. Most applications will have either two or three data stores: Data, Indexes, and (if needed) Static data. The following naming convention:

```
<segment prefix>_DATA,  
<segment prefix>_INDEX, and  
<segment prefix>_STATIC
```

shall be used for the three storage areas respectively. The Logs within a Sybase database are treated as data stores in a Sybase implementation.

To facilitate correct sharing of data, developers shall provide the definitions of their schema components for inclusion in the DBMS data dictionary. Definitions for data stores, tables, elements, stored procedures, and views are stored in the system's data dictionary tables as comments. 255 characters is the maximum length allowed for the description. In addition, narrative information on all these databases should be provided during Segment Registration so developers can access their definitions.

Note: Although segment registration may constrain the description of the database schema, these limitations do not apply to a data model that follows the recommendations that can be found at the COE Data Emporium.

5.4.2 Separation of Data and Applications

To sustain the independence between DBMS clients and the data server, developers shall not mix extensions to the DBMS with their databases and shall separate the database from the applications that use it. If specialized data management services are needed by particular applications and are not part of the COE database services, the provision of such services shall be approved by and coordinated with the DII COE Chief Engineer.

5.4.3 Database Packaging via Segments

The definition of a database schema - the set of data objects, their interrelationships, constraints, and rules for access or update - and the packaging of the schema to work with the COE installer as segment(s) is the responsibility of the developers. If a database is only supporting one application, (perhaps a migration from a legacy data source) then it might make sense to implement a single monolithic database segment.

However, if the database supports more than the data specific to one application then the developers should determine if the database should be developed as an aggregate segment with multiple database segments. Allocation of data objects to segments may be based on a variety of factors including but not limited to:

- Reuse
- Data objects that can be conveniently managed as a unit
- Data objects that are needed together to support a functional area
- Common sources or providers of data
- Data object interdependencies
- Frequency of update (static data vs. dynamic data)
- Mixed security classifications (of schema or data).

The advantage of multiple database segments is that the segments are more granular and a data server can be configured to support the data requirements of mission applications without having to carry superfluous data objects. A disadvantage of multiple database segments is the management of inter-segment dependencies between database object dependencies (such as foreign key constraints) that complicate the installation and deinstallation of database segments.

Such dependencies also affect the modularity and scalability of the data server. Dependent segments must be installed after the database segment they reference. Further, as is the case with other segment types, dependencies can easily propagate when placed on segments that are, in turn, dependent on other segments. Furthermore, when inter-

segment dependencies are defined, circular dependencies can be created. A circular dependency exists when two segments depend on each other. In such cases, neither segment can be installed because both require the other to be installed first. If a circular dependency cannot be resolved, then the two segments may have to be merged into a single, larger segment or the dependent code can be moved to a third segment.

Developers should consider the frequency of updates against data tables because separating static reference tables from those that are dynamic allows more flexible system and database administration. Static tables are likely to have fill segments and are also likely to be updated via a configuration management process rather than normal end-user processing. This may introduce the need for additional specific roles. The separation may be accomplished by defining the static objects in their own database segment, and/or by creating static objects in a separate storage area for read-only tables (e.g., an Oracle tablespace) or even a separate database instance.

When a change is to be made to a database segment that has dependencies on it, all developers of segments that access that segment must be notified and given time to evaluate the impact on and/or modify their segments. The revised database segment should provide legacy views to support the dependent segments until they can be modified.

5.4.4 Inter-Database Dependencies and Reuse

As discussed in Chapter 3, where possible and appropriate, developers should reuse objects created by other database segments. Referencing existing database segments prevents unnecessary duplication of widely used or required database objects (such as reference tables), and procedures (such as validation or conversion routines). These segments should support interoperability at the data level by standardizing key cross-reference fields. The objects in a database segment must be accessible to many applications, regardless of which database they reside in, and may support other database segments that are then dependent on that database segment.

Note: Reuse of schema objects will tend to lessen interoperability problems, even if the dynamic data contents (fill + updates) of the existing database are not replicated to the new database.

If a database segment does not meet the full needs of the developer, changes should be proposed to the *cognizant DOD configuration authority* for the database segment that satisfies the most requirements. The developer may choose to develop a similar (subset or superset) database segment, pending resolution of the change request.

Inter-database dependencies occur whenever database objects in a segment are dependent upon objects in some other database segment. Database segments will have dependent data objects (tables or views) when their information needs can be partially satisfied from tables or views defined in other database segments. If an external table fully satisfies the information needs, it should be referenced directly. A table will be dependent on another database if its constraints reference objects in that other database or if it is populated or

maintained using a trigger based on an external object. Developers may use a dependent view to extract subsets of information from external tables or views or to change the presentation of information (e.g., change units of measure or combine columns). Developers may use views to combine internal tables with external objects to provide information supersets. Developers may also create a table that is a superset of an external object to avoid creating and maintaining partially redundant objects. That table would then be combined with a view that joins it with the external object. Developers must use a relatively complex mechanism such as an “outer join” when defining such a view/table combination unless appropriate triggers are created to prevent decoupling when either the internal or the external table gets updated.

When replicated objects are eliminated in favor of views, changes to those objects need not be propagated across multiple databases. At the same time, having only one copy of a widely referenced table is likely to increase data quality and currency. A view that references a table (or view) outside its own segment is dependent on the database containing the base table (or view). When a dependent object is created (installed in a database instance), all of its references to other objects must be resolved. If it has dependencies on non-existent objects, the dependent object may not already have been created or it may have to be validated when the objects it references come into existence. The creation of a dependent object may also fail if its owner does not have the appropriate access to all referenced objects. If the definition of any of the referenced objects is altered, the dependent object may not function properly or may become invalid. A view will become invalid and have to be recreated if its base table (or view) is modified, renamed, or dropped. Any privileges or synonyms on the invalid view also become invalid until it is recreated.

Database segments have dependent constraints or business rules when their integrity constraints or operations involve objects from other segments. Such rules may include foreign keys that reference another schema’s tables or triggers that propagate updates based on another schema’s transactions. Developers may create triggers and stored procedures or functions on objects in other schemas as long as they do not modify or update the other database’s information and do not change the constraints or business rules of the other database. It is permissible, for example, to use a ‘post-insert’ trigger to copy data from an external data object to one in the developer’s database segment. It is prohibited, by contrast, to use such a trigger to change data in that other segment’s table. Developers may not create or modify constraints on objects in other schemas. Such constraints could invalidate otherwise legal updates to the other database. Any rules - whether they are constraints, triggers, or procedures - shall be created in the segment of the object they are attached to unless approved by the Chief Engineer.

Excessive use of triggers can result in complex interdependencies that may be difficult to maintain. When implementing a specific function via triggers, developers must keep in mind that a database transaction will rollback if execution of the associated trigger(s) is not successful. Trigger developers must implement exceptions to handle errors or unexpected results that may occur during the execution of a trigger. These exception

handlers must ensure that a trigger fails ‘open’ and allows the owning segment’s database transactions to complete regardless of the processing of the dependent trigger.

Developers shall not create indexes on objects in other databases unless approved by the Chief Engineer. Indexes have significant impact on system performance. While they speed retrieval of records, indexes slow updates to tables. The effect of uncontrolled index proliferation could dramatically damage the overall functioning of a DII system. Indexes also require physical storage allocation as discussed in the next section.

The database descriptors, as discussed in Chapter 6, shall be used to describe database dependencies between segments. The database descriptors also provide a mechanism for handling issues involved with the installation and removal of inter-dependent segments. The following principles apply when inter-database dependencies exist:

- A segment cannot modify the table or column definitions of another segment’s database.
- The segment that owns the parent object creates the parent object.
- The segment that owns the child (dependent) object creates the child object.
- Inter-database dependencies will be in separate segments from the non-dependent portions of the schema whenever possible.
- The referencing object (child) owns referential dependencies (e.g., foreign keys). If the only dependencies are referential, separate segments are not needed.
- Schemas retain their autonomy. The developer of a dependency (including referential dependencies) is responsible for maintaining that dependency should other developers change their database schemas.

5.4.5 Physical Storage

Database management systems provide file management transparency across multiple host computer systems by hiding the details of file storage from the database’s data objects. At the same time, however, the placement of data objects on physical storage devices has an effect on system and database performance due to disk contention and other file system access issues.

Developers cannot assume that DII data servers have uniform hardware configurations. Some will have disk arrays, possibly mirrored, that appear as a single, large mount point; others will have multiple mount points representing separate disks or several mirrored arrays. Further, it cannot be assumed that existing hardware configurations will remain static or that current disk-mirroring technologies will remain in use. DII developers, therefore, shall not use ‘raw’ partitions, but shall place all files in their segment’s directory tree. When using the COE Database Administration tools, an Identify Storage GUI is supplied which allows an administrator to select the physical storage that will be used for a database installation (see the *DBAdmR Users Guide* for further details). DISA or the cognizant DOD program office is responsible for providing the core configuration for a COE data server. The site’s administrators are responsible for configuring installed servers for optimum performance.

Database segments shall create their data stores through the segment's `PostInstall` descriptor. Database segments shall use the `DBAdmSCreateDS` API to implement physical data storage. This API allocates physical storage and creates the data store. For Sybase, this includes the storage area for logs. COE Database Administration tools hide the data server's implementation of physical storage. In this way the database segment is insulated from the physical server implementation, whether it uses raw devices or file system directories, has disk arrays, or uses other storage techniques. Figure 5-3 illustrates data store allocation.

Where `DBAdmSCreateDS` is not available, developers will provide the scripts to create their data stores and the operating system files associated with them. Data files will be created in the `DBS_files` subdirectory of the database segment using the API provided by the DBMS vendor. One or more data files may be created to support each storage area. The method for file creation varies with the DBMS being used.

```
CREATE_DS DBSORT < DBSORT_LIST
```

where `DBSORT_LIST` file contains:

```
DBSORT_DATA 1,000K
DBSORT_INDEX 1,000K
DBSORT_LOG 300K LOG
```

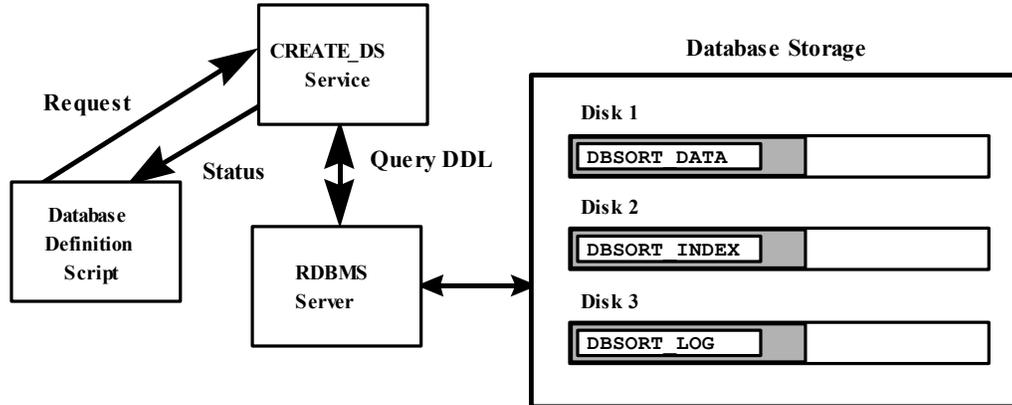


Figure 5-3: Data Allocation

5.4.6 Database Definition Scripts

A database definition script contains all database definition commands for a specific database object. These objects include tables, views, triggers, and stored procedures. The name of the script is the same as that of the database object it defines. Depending on the object type, multiple sections can be defined within one file to perform all the data definition functions required for that specific database object.

The scripts used to create data objects are also used by database administrators in the maintenance of the databases and the data server. DII database administrators have to manage thousands of data objects (tables, views, etc.) spread across multiple database owner accounts. Routine maintenance such as rebuilding corrupted indexes or views can become impossible because the DBA cannot locate the script file that contains the object's definition among the thousands of scripts on the data server. To avoid these problems, DII developers must organize their Data Definition Language (DDL) commands into a series of database definition scripts. These scripts must conform to a particular file naming convention and structure.

The database definition script is structured to execute various database definition commands based on the input argument given to it. This functionality is implemented with a case statement that executes on the input argument. Table 5-1 lists the input arguments to be used for database definition scripts. See the Database Installation section in Chapter 6 for the ordering of the scripts.

The `CREATE_DATA_STORE` argument for a database definition script should only be used when the COE tool `DBAdmSCreateDS` is not available.

5.4.7 Loading Data into Database Segments

After the objects belonging to a database segment have been created in `PostInstall`, they may need to be populated. Other objects, those containing dynamic data, may be initially empty. Where needed, a database segment can perform initial data fill in the Load Data phase of the `PostInstall`. Several methods are discussed below that can be used to accomplish data loads. Method selection should be based on the amount of data to be loaded.

If a small number of records are to be loaded into a table, the load can be accomplished with insert statements embedded in an SQL command script.

Argument	Purpose
CREATE_DATA_STORE	create a data store (use CREATE_USER to create the DBO account first)
DROP_DATA_STORE	remove a data store
CREATE_ROLE	create a database role
DROP_ROLE	drop a database role
CREATE_RULE	create a Sybase rule
DROP_RULE	drop a Sybase rule
CREATE_TABLE	create a database table
DROP_TABLE	drop a database table
CREATE_VIEW	create a database view
DROP_VIEW	drop a database view
CREATE_CONSTRAINT	create a database constraint (i.e., foreign key)
DROP_CONSTRAINT	drop a database constraint
CREATE_INDEX	create an index
DROP_INDEX	drop an index
UPDATE_INDEX	perform update statistics for Sybase indexes
CREATE_USER	create a database user account
DROP_USER	drop a database user account
DISABLE_LOGIN	revoke connection privileges (login) from a database account
ASSIGN_GRANTS	assign grants to a user or role/group
REVOKE_GRANTS	revoke grants from a user or role/group
LOAD_DATA	load a table with data (from within the command script)
CREATE_PROCEDURE	create a stored procedure or database package
DROP_PROCEDURE	drop a stored procedure or database package
CREATE_TRIGGER	create a database trigger
DROP_TRIGGER	drop a database trigger
CREATE_SEQUENCE	create an Oracle sequence
DROP_SEQUENCE	drop an Oracle sequence
REGISTER_DATA	load application data (i.e., configuration parameters)

Table 5-1: Definition Script Arguments

If a large amount of data is to be loaded into a database table, the use of a data loading utility furnished by the RDBMS is usually more efficient. In this case, the utility can be

invoked from the `LOAD_DATA` section of the database definition script. Examples of these data loading utilities are Oracle `SQL*Loader`, Informix `dbload`, Oracle or Informix `Import`, and Sybase `bcp`. These utilities require that the data to be loaded be stored in a file with a specific format.

It can take a long time to fill a large database. Developers should indicate the approximate load time in their `ReleaseNotes`. The data load time can be reduced by loading the data before creating the database constraints and indexes. Estimating the load time should only be done with clean data that has been tested against the database constraints.

Files used for data fill belong in the data subdirectory of the database segment. The data directory within the segment can also be used as a ‘mount point’ for CDROM, tape drive, or other bulk storage devices. This is the preferred approach for large data loads. It allows the segment to be filled without occupying disk space during the data fill.

The security classification of the data to be loaded must be considered during the implementation of a database segment. When a classified data fill is part of the database segment, the entire segment becomes classified at the same level as the data. Therefore, developers must separate the data fill from the database segment when the database schema is not classified, but the contents are. The intent here is to keep database segments unclassified as much as possible so schemas can be reused.

If a separate data segment is provided to accompany a database segment, that data segment must have a `DEINSTALL` capability. This frees storage after the data fill is complete.

5.4.8 COE Data Emporium

The COE Engineering maintains a web-accessible store of reusable data components and SHADE technical information for developers (the SHADE URL is <http://diides.ncr.disa.mil/shade/index.cfm>). This data engineering support service is called the COE Data Emporium. Behind buttons on the Emporium’s homepage are galleries (registries) for chartered Communities of Interest, Reference Data Sets, Database Segments, and XML tags/metadata.

5.4.8.1 Communities of Interest (COI)

COIs are collections of people (services/agencies/activities/developers) whose information systems share a common digital language and common data definitions. This implies a common worldview as well as common abstractions, common data representations, and common metadata. COIs require a service that lets them “publish” their existence and their available information resources so that outsiders may discover them and assess whether or not they want to share. For the DII COE, that service is provided by the Data Emporium. To be published via the Emporium, a COI must have a sponsor; must be willing to make its communal definitions visible; must be willing to

finance/staff COI administration; must be chartered by the COE Chief Engineer (with CRCB oversight); and must involve specific COE system implementations.

5.4.8.2 Reference Data Sets

Reference Data Sets are packages of data structure and valid values that are standardized, relatively static, and widely used within DOD systems (e.g., Country Codes, Supply Codes). The scripts for hundreds of Reference Data Sets in the areas of C2, Intelligence, Supply, Logistics, and other domains are available for download from the Data Emporium.

5.4.8.3 Database Segments

Database segments, RDBMS-specific packages of data structure, are also made visible through the Data Emporium. Enterprise-level database segments for common battlespace objects such as Organization, Materiel, Person, Facility, and Feature - as well as their related reference sets - are currently available through the Data Emporium. In addition, a database segmentation tool, which can rapidly create first cut database segments from IDEF 1x models, is available for download.

5.4.8.4 XML Registry

XML (eXtensible Markup Language) is an emerging standard meta-markup language for creating semantic tags to describe data. It is more powerful and flexible than HTML because it allows documents to define their own set of tags and semantics. This makes XML suitable as a technique, among other approaches, for implementing shared data exchange between processes.

XML's flexibility also means that there is the potential for conflict if two developers should choose the same markup tag name for a data item but attach different semantics to the tag. Conflict may also occur if the same data item is referenced by different tag names, or if the attributes (such as precision) for the data item are different. To avoid conflict, the COE requires that segments register their XML tags.

Specifically, DII COE compliance requires that any segment that defines public interfaces based on XML must register and document the relevant XML structure and tags in the COE XML Registry. The requirements for the different levels of compliance are specified in Appendix B.

An XML Registry submission is considered "fully documented" if:

- Each XML element or attribute has an associated data type to describe its format.
- Valid domain values, where applicable, are specified by enumeration, reference (e.g., reference set, published documentation), or rule based generation.

- Relevant amplifying information has been submitted and associated with the appropriate XML elements or attributes (e.g., data models, full text descriptions and/or sample XML data documents).

The DII COE XML Registry contains further information, including how to submit XML tags and documents, and is available directly at the following URL: <http://diides.ncr.disa.mil/xmlreg/index.cfm>.

5.4.8.5 Database Segment Scope

A developer's evaluation of his database segment's scope and "shareability" is identified by the required \$SCOPE keyword (see Chapter 6). Data within the COE is implemented as a federation of application-owned (Unique) and common (Shared or Universal) database segments. Unique segments are limited in scope and therefore unlikely to be shared by many applications. Shared segments are typically mission-or-functionally-oriented, and are generally specific to a limited number of mission domains. Universal segments are very broad in scope and must be sharable across all applications in order to promote true interoperability. There is no physical difference in composition of the database segments, but the level of configuration management increases due to the wider impact changes would have on operational systems that use the database segments.

This page is intentionally blank.