CHAPTER 2

Database Design and Implementation

The concepts in database design and implementation are some of the most important in a DBA's role. Twenty-six percent of the 312 exam revolves around a DBA's ability to design, create, alter, and remove objects in a DB2 database. This requires first understanding the creation of a database to support the business needs. The DBA must work from a logical model and be able to create a proper physical database to support this. The creation of a database involves many components such as table spaces, tables, and indexes. This chapter examines how to create those objects, some of the specifics for various options for each object, and how to change objects after creation, if necessary.

Database Design

Before diving into the physical creation of DB2 database objects, you must have a well-thought-out design. To design a database, the DBA performs two general tasks:

- Logical data modeling lets you design a model of the data without paying attention to specific functions and capabilities of the DBMS that stores the data.
- Physical data modeling lets you begin moving toward physical implementation with the primary purpose of optimizing performance while ensuring data integrity.

Logical Data Modeling

Logical data modeling is the process of documenting the comprehensive business information requirements in an accurate and consistent format. Designing and implementing a successful database, one that satisfies an organization's needs, requires a logical data model. Analysts who do data modeling define the data items and the business rules that affect those data items. The process of data modeling acknowledges that business data is a vital asset that the organization must understand and carefully manage.

Building a logical model requires taking these general steps:

- Build critical user views: create a representation of critical information that the business activity requires, define the major entities (significant objects of interest), and determine the general relationships between these entities.
- Add key business rules to user views: key business rules affect INSERT, UPDATE, and DELETE operations on the data.
- Add detail to user views and validate them: define the key entities and relationship associated with these descriptive details, called attributes, to the entities.
- Determine additional business rules that affect attributes: identify data-driven business rules that are put on constraints on particular data values.

After you create a logical model, it is often recommended to create a data dictionary to supplement the logical data model diagrams. A data dictionary is a repository of information about an organization's application programs, databases, logical data models, users, and authorizations. A data dictionary can be manual or automated.

Entities and Attributes

Entities are the items about which the business wants to store information. Attributes are the values or characteristics associated with an entity. Every occurrence of an entity has associated attributes. Each attribute can be represented as a field.

When you identify attributes, consider characteristics such as data type, size, how used, and an attribute's related entities.

Entity Relationships

Many types of *relationships* can exist between entities:

- One-to-one: A bidirectional relationship that is single-valued in both directions
- One-to-many: One entity has a multivalued relationship with another entity
- Many-to-many: Multivalued in both directions

Normalization

Normalization is the process of nonloss decomposition of data relations. It is performed during logical database design and promotes the formalization of domains that contain a single value. Several forms of normalization are available; the most common form for database designs is Third Normal Form (3NF).

A normalized design's objectives are fairly straightforward:

- Eliminate all data anomalies
- Avoid data redundancy
- Avoid potential inconsistency among data
- Preserve all relevant information
- Maintain maximum flexibility
- Accommodate changes easily

One main issue that DBAs must often address is that of repeating groups existing in the data. In a relational design, repeating groups can become dependent tables and thereby normalized to avoid any anomalies or redundancies. A denormalized design can eliminate these benefits and introduce problems, even though sometimes this design appears to be easier to use (i.e., avoiding excessive joins by adding a column to the table).

Data Types for Attributes

You must specify a DB2 data type for each attribute of an entity. The attributes will eventually be represented in the column in the table. General guidelines exist for choosing a data type, but first here is a list of the available DB2 built-in data types (more about data types later in this chapter).

String data contains a combination of letters, numbers, and special characters. String data types are as follows:

- CHAR: fixed-length character strings
- VARCHAR: varying-length character strings
- CLOB: varying-length character large object strings
- GRAPHIC: fixed-length graphic strings that contain double-byte characters
- VARGRAPHIC: varying-length graphic strings that contain double-byte characters
- DBCLOB: varying-length strings of double-byte characters in a large object
- BINARY: a sequence of bytes that is not associated with a code page
- VARBINARY: varying-length binary strings

- BLOB: varying-length binary strings in a large object
- XML: varying-length string that is an internal representation of XML

Numeric data contains digits. These are the numeric data types:

- SMALLINT: for small integers
- INTEGER: for large integers
- BIGINT: for bigger values
- DECIMAL(*p*,*s*) or NUMERIC(*p*,*s*): where *p* is precision and *s* is scale
- DECFLOAT: for decimal floating-point numbers
- REAL: for single-precision floating-point numbers
- DOUBLE: for double-precision floating-point numbers

Datetime data values represent dates, times, or timestamps. Datetime data types are as follows:

- DATE: dates with a three-part value that represents a year, month, and day
- TIME: times with a three-part value that represents a time of day in hours, minutes, and seconds
- TIMESTAMP: timestamps with a seven-part value that represents a date and time by year, month, day, hour, minute, second, and microsecond

Many of these data types, and more, will be discussed throughout the chapter, as they often have implementation considerations.

Physical Data Model

The physical design of the database optimizes performance while ensuring data integrity by avoiding unnecessary data redundancies. During physical design, the entities are transformed into tables, the instances into rows, and the attributes into columns.

After completing the database's logical design, you next move to the physical design. You will now make several decisions that affect the physical design:

- How to translate entities into physical tables
- What attributes to use for the physical tables' columns
- Which columns of the tables to define as keys
- What indexes and views to define on the tables
- How and whether to denormalize the tables

- How to resolve many-to-many relationships
- What designs can take advantage of hash access

As far as the translation from logical goes, you can follow some basic general guidelines for transforming a logical model to a physical model. They include the following:

- A physical table for each entity and a column for each attribute
- A unique index for each primary key
- A non-unique index for each foreign key

While it might seem that some decisions you will make based on the guidelines are clear-cut, it is still a good practice to document the processing needs to consider before actual implementation. Processing requirements can have a profound impact on actual physical implementation. In other words, physical implementation can look a bit different from the original logical model, and that is OK as long as the integrity and needs of the logical model are represented.

Creating DB2 Structures

After completing the physical design, you are ready to physically implement the database. You will create several DB2 structures and must follow a hierarchy to the objects during the creation of the object. Next is a discussion about the DB2 structures that follows the general flow of that hierarchy (storage groups, databases, table spaces, tables, indexes).

DB2 structures are created using Data Definition Language (DDL). To create objects, use the DDL `CREATE` statement. This statement also allows DBAs to specify specifics about the structure they are creating. The following sections look at how to create the DB2 structures, choose correct options, and consider various design aspects.

Storage Groups

DB2 storage groups are a set of volumes on disks that hold the data sets in which tables and indexes are stored. The description of a storage group names the group and identifies its volumes and the Virtual Storage Access Method (VSAM) catalog that records the data sets. The default storage group, `SYSDEFLT`, is created when DB2 is installed.

Within the storage group, DB2 does the following actions:

- Allocates storage for table spaces and indexes
- Defines the necessary VSAM data sets

- Extends and deletes VSAM data sets
- Alters VSAM data sets

All volumes of a given storage group must have the same device type. However, parts of a single database can be stored in different storage groups.

DB2 can manage a database's auxiliary storage requirements by using DB2 storage groups. Data sets in these DB2 storage groups are called *DB2-managed data sets*. These DB2 storage groups are not the same as storage groups that the DFSMS storage management subsystem (DFSMSsms) defines.

Several options are available for managing DB2 data sets:

- Let DB2 manage the data sets.
- Let SMS manage some or all data sets, either when using DB2 storage groups or when using user-defined data sets.
- Define and manage data sets independently using VSAM Access Method Services.

It is recommended to use DB2 storage groups, whenever possible, either specifically or by default. Also use SMS-managed DB2 storage groups whenever possible.

You use the CREATE STOGROUP statement to create the storage group. The following syntax shows all available options:

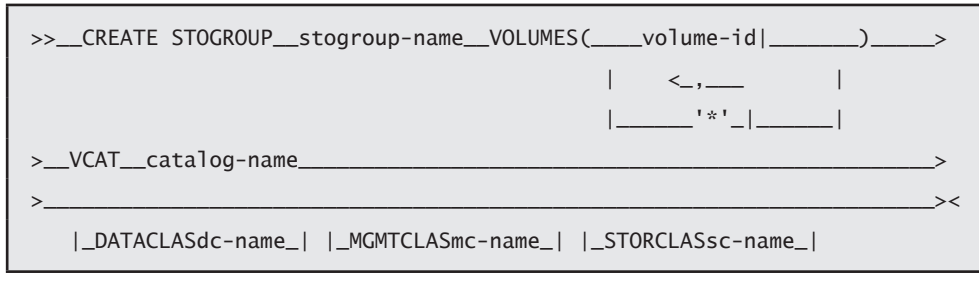

Table spaces and index spaces that are defined using a storage group are considered to be DB2-managed, and DB2 will create the data set for them; if the PRIQTY and SECQTY are specified for the table or index space, DB2 will use this for the space allocation. Objects not defined using a storage group are considered user-managed and must be defined explicitly in the Integrated Catalog Facility (ICF).

Here is an example of the creation of a storage group:

```
CREATE STOGROUP MYSTG
  VOLUMES (*)
  VCAT DBA111;
```

Using the asterisk (*) indicates that SMS will manage the volumes to be used. You can also define the specific SMS data class, management class, or storage class.

Databases

DB2 databases are a set of DB2 structures that include a collection of tables, their associated indexes, and the table spaces in which they reside. To define a database, use the CREATE DATABASE statement.

Whenever a table space is created, it is explicitly or implicitly assigned to an existing database. If a table space is created and a database name is not specified, the table space is created in the default database, DSNDB04. In this case, DB2 implicitly creates a database or uses an existing implicitly created database for the table. All users who have the authority to create table spaces or tables in database DSNDB04 have authority to create tables and table spaces in an implicitly created database. If the table space is implicitly created, and the IN clause in the CREATE TABLE statement is not specified; DB2 implicitly creates the database to which the table space is assigned.

A single database, for example, can contain all the data that is associated with one application or with a group of related applications. Collecting that data into one database provides the ability to start or stop access to all the data in one operation. You can also grant authorizations for access to all the data as a single unit.

It is recommended to keep only a minimal number of table spaces in each database, and a minimal number of tables in each table space (in most cases, only one is allowed). Excessive numbers of table spaces and tables in a database can cause decreases in performance and manageability issues. Reducing the number of table spaces and tables in a database can improve performance, minimize maintenance, increase concurrency, and decrease log volume.

Following is the syntax for creating a database:

```
>>__CREATE DATABASE__database-name_____________________________________>
   <_____________________________________________
                                                          Continued
```

```
>_____________________________________________|_____________________><
       |_BUFFERPOOL__bpname_____________________|
       |_INDEXBP__bpname________________________|
       |_AS____WORKFILE_________________________|
       |                    |_FOR__member-name_| |
       |            _SYSDEFLT______              |
       |_STOGROUP__|_stogroup-name_|_____________|
       |_CCSID____ASCII_________________________|
                  |_EBCDIC__|
                  |_UNICODE_|
```

You can also define the default buffer pool for table spaces or indexes in the database. The following is an example of the creation of the MYSTG database. The BUFFERPOOL parameter specifies that objects created in this database that do not explicitly assign a buffer pool will default to buffer pool BP10:

```
CREATE DATABASE MYDB
  STOGROUP MYSTG
  BUFFERPOOL BP10
  CCSID EBCDIC;
```

After creating the database, you can change the default buffer pools, encoding scheme, or storage group. The following is an example of the ALTER DATABASE:

```
ALTER DATABASE MYDB
BUFFERPOOL BP3;
```

If it is necessary to remove the database, you can do so with the DROP statement. When you drop a database, all dependent objects within the database are also dropped:

```
DROP DATABASE MYDB;
```

Storage Structures

In DB2, a storage structure is a set of one or more VSAM data sets that hold DB2 tables or indexes. A storage structure is also called a *page set*. The two primary types of storage structures in DB2 for z/OS are table spaces and index spaces.

Table Spaces

A table space is a set of volumes on disks that hold the VSAM data sets in which tables are actually stored. A table space can consist of numerous VSAM data sets. Data sets are VSAM linear data sets (LDSs). All tables are kept in table spaces. A table space can have one or more tables. Table spaces are divided into equally sized units, called *pages*, which are written to or read from disk in one operation. A table space can have various page sizes (4 KB [default], 8 KB, 16 KB, or 32 KB) for the data.

The number of tables in a table space depends on the tables' characteristics. Large tables are often in their own table space for better maintainability, performance, and availability. For smaller tables, multiple-table segmented table spaces are better. This design helps to reduce the number of data sets that a DBA must manage for backup and recovery, and the number of data sets that the database system must open and close during DB2 operations.

It is best to minimize the number of table spaces in each database. That is because DBD locks are taken during DDL execution, and the number of table spaces in a database can affect other operations. Data in most table spaces can be compressed, allowing more data on each data page.

To create table spaces, use the CREATE TABLESPACE statement. In this statement, you define the database for the table space as well as the storage group to use. DB2 can also implicitly create a table space if a CREATE TABLE statement is issued and an existing table space is not defined. In this case, DB2 assigns the table space to the default database and the default storage group. CREATE TABLESPACE allows an option to not define the underlying VSAM data sets. This is the DEFINE NO option, which will create the object but will not allocate data sets until it is time to put data into them.

The maximum number of partitions for a table space depends on the page size and on the data set size (DSSIZE). The size of the table space depends on how many partitions are in the table space and on the DSSIZE. The maximum number of partitions for a partition-by-growth table space depends also on the value of DSSIZE, the page size, and the value specified in the MAXPARTITIONS option on the CREATE TABLESPACE or ALTER TABLESPACE statement.

DB2 supports different types of table spaces:

Universal Table Space

A universal table space combines the characteristics of partitioned and segmented table spaces, and can hold one table. It provides better space management (for varying-length rows) and improved mass delete performance.

Partitioned (Classic) Table Space

A partitioned table space divides the available space into separate units of storage, or partitions. Each partition contains one data set of one table.

Segmented Table Space

A segmented table space divides the available space into groups of pages known as *segments*. Each segment is the same size and contains rows from only one table.

Large Object Table Space

A large object (LOB) table space holds large object data such as graphics, video, or very large text strings. A LOB is always associated with the table space that contains the logical LOB column values.

Simple Table Space

A simple table space can contain more than one table. The rows of different tables are not kept separate (unlike segmented table spaces). These types of table spaces can no longer be created.

XML Table Space

An XML table space holds XML data. It is always associated with the table space that contains the logical XML column value.

The following is the syntax (partial) for creating a table space:

```
>>__CREATE_____TABLESPACE__table-space name_____________________________>
       |_LOB_|            |     _DSNDB04_______  |
                          |_IN_|_database-name_|_|
   <__________________
>_____________________|_____________________________________________________>
   |_using-block_____|         |_DSSIZE__integer__G_|
                                                                 Continued
```