Introduction to Database Applications for DB2

The relational database of choice for modern distributed applications is IBM’s widely popular DB2 Universal Database (UDB). DB2 UDB V7.1 provides high-performance data access, reliability, and an abundance of advanced features. Whether your application is an e-commerce site on the Web or a directory service such as a Lightweight Directory Access Protocol (LDAP) server, the overall concepts for all database applications remain the same. Database applications mostly perform two major tasks, and are categorized as such. Applications that modify the contents of databases and perform transaction processing are known as Online Transaction Processing (OLTP) applications. Applications that perform ad hoc queries to analyze the contents of databases are known as Decision Support Systems (DSS), or Online Analytical Processing (OLAP) applications.

All applications communicate with DB2 in a client/server environment using the Structured Query Language (SQL) within the context of one or more programming interfaces such as embedded SQL, JDBC, and the Perl DBI (Database Interface). Different interfaces also provide additional features that you may find useful. This is why it is important to understand the options available to you as an application developer.

In this chapter, we introduce fundamental DB2 concepts and describe how applications connect to DB2 and use SQL statements to access and modify data in a DB2 UDB database. We assume that you already know about database objects such as tables and indexes, and that you are also familiar with SQL from the companion to this guide, The DB2 V7.1 Administration Certification Guide. We still cover each of these topics briefly within this chapter to ensure you have an understanding of them for the purposes of the certification exam.

You can find a complete reference for SQL and DB2 database objects in the SQL Reference, which is included among the DB2 online books. The SQL Getting Started guide is another online book that includes a good tutorial on SQL.
1.1 Objectives
In each chapter, we will begin with a set of goals for the reader. These goals will describe the concepts you should be familiar with after reading the chapter and completing the exercises. After reading the first chapter, you should have a general understanding of:

- The DB2 UDB client server environment.
- Database objects that will be relevant to you as an application developer.
- How SQL is used to interact with DB2 databases.
- The common programming structure of all database applications.

We begin by giving you an understanding of how DB2 UDB V7.1 is organized.

1.2 DB2 UDB Server Instances and Databases
A database is simply an organized collection of related data. High-end Relational Database Management Systems (RDBMS) such as DB2 UDB provide an encapsulated client server environment where a persistent database server process serves requests from external client applications. Every DB2 instance is an RDBMS engine process that is uniquely named on a particular system. A DB2 database is created within the context of a specific DB2 instance. To create a DB2 database, you need to install a DB2 UDB V7.1 server product. The DB2 UDB V7.1 product packages are as follows:

- **Satellite Edition (SE)** – A special RDBMS engine with a small footprint that is intended for occasionally connected systems. It will not accept incoming database requests from remote clients.
- **Personal Edition (PE)** – An RDBMS engine that will not accept incoming remote database requests. Available on Windows NT, Linux, and OS/2 only.
- **Workgroup Edition (WE)** – An RDBMS engine that will accept incoming database requests from remote clients. Available on Windows NT, Linux, and OS/2 only.
- **Enterprise Edition (EE)** – Similar to Workgroup Edition, but also allows remote and local clients to access data on a host database such as DB2/390 or DB2/400.

Using any of these products, you can create one or more DB2 server instances. Every instance has a name unique to the system on which it is installed, and this name is specified when creating the instance. Each DB2 server instance is a separate DBMS (Database Manage-
ment System) process within whose context you can create and manage one or more DB2 databases. Every instance can also be configured with regard to memory usage and other parameters in the database manager configuration and DB2 profile registry.

If your server instance has remote client functionality, you can also configure one or more network protocol listeners for the instance to accept incoming database requests from remote clients. Each server instance on a system must have a unique protocol listener. For example, if you have two server instances on your Windows NT system and you want them to be accessible to remote TCP/IP clients, you must configure each to listen on a unique TCP/IP service or port.

1.2.1 DB2 UDB Client Instances
Each DB2 UDB V7.1 product package installation also includes three common client components:

- **DB2 Runtime Client** – Includes DB2 runtime libraries that allow applications to access local and remote DB2 databases.
- **DB2 Administration Client** – Similar to the Runtime Client, but also provides graphical DB2 UDB administration tools, as well as additional utilities and DB2 bind files.
- **DB2 Application Development (AD) Client** – Similar to the Administration Client, but provides DB2 developer tools such as the Stored Procedure Builder (SPB), program pre-compilers, static libraries, and numerous samples.

Figure 1-1 shows the various DB2 client and server components and their hierarchical nature. By viewing this diagram, you should be aware that DB2 client components are also present in each DB2 server instance. Thus, when we use the term “DB2 client,” we may be referring to either a client-only install or the client component on a DB2 server. You can determine which we are referring to based on the following definitions. All DB2 clients are freely downloadable from the official DB2 Web site.

If an application connects to a database within the same instance from which it runs, the application is known as a *local client*. In this case, there is no need to install an additional DB2 Runtime Client since this component is also included within the libraries of the DB2 server instance. On the other hand, if the application is installed on a system other than the DB2 UDB server, or run on a different instance on the same system, the application is known as a *remote client*. The DB2 Runtime Client component must be installed on each remote workstation or system executing the application.

In some cases, you may have installed a DB2 server product on one platform and will be running an application on a different platform. For example, you may have DB2 UDB V7.1 EE installed on AIX, but your SQLj applications will run on Windows NT. In this case, you will need to obtain a DB2 UDB AD client for Windows NT. When you install any DB2 UDB client on a Windows 95/98/NT workstation, a client instance called “DB2” will be automatically cre-
ated. On UNIX platforms, you can create multiple client instances using the `db2icrt` command. Figure 1-2 illustrates how a DB2 UDB Runtime Client connects to a DB2 server instance.

### 1.2.2 DB2 UDB Environment

It is important to understand the DB2 client/server environment to set up local and remote applications to access DB2. There are a number of instance-specific environment considerations that we will discuss in this section. Every DB2 client and server instance consists of the following components:

![DB2 UDB V7.1 software components](image1)

**Figure 1-1** DB2 UDB V7.1 software components.

![DB2 UDB Runtime Client](image2)

**Figure 1-2** DB2 UDB Runtime Client.
• DB2 node directory.
• DB2 database directory.
• DB2 database manager configuration.
• DB2 profile registry entries.
• DB2 environment variables.

We approach these one by one, and then discuss some interesting nuances. In many of the sections below, we will discuss DB2 commands. To run these commands, you will need to use the DB2 Command Line Processor (CLP) or the Command Center. The CLP is a tool included with every DB2 installation. It allows you to administer DB2 databases and instances and run dynamic SQL. We will use this tool throughout the book in many examples. To start the CLP shell on Windows platforms, select **Start->Programs->IBM DB2->Command Line Processor**. On UNIX platforms, simply type `db2` at the command line.

The Command Center is a Graphical User Interface (GUI) utility provided with the Administration Client that lets you perform all of the same tasks as the CLP as well as create scripts from the commands you run. You can start the Command Center from the DB2 Control Center, or on Windows platforms, you can open it from the IBM DB2 folder from the Start menu.

**Tip** You can also enter DB2 commands directly from the operating system prompt by prepending them with “db2”. This is useful when you want to pipe the output of various commands to utilities such as `more` or `grep`. In Windows environments, you will need to open a DB2 command window from the IBM DB2 folder, or you can issue the “db2cmd” command from a regular MS-DOS command prompt.

**Troubleshooting**

If you have trouble getting the DB2 command line to work, ensure that the `DB2INSTANCE` environment variable has been set. See details in the section below on environment variables.

### 1.2.3 DB2 Node and Database Directory

Each DB2 client has a node directory containing entries for all remote DB2 server instances the client can access. The term “remote instances” in this case also includes other instances on the same machine. Each instance on the same machine is cataloged separately based on a unique network protocol listener. Directory entries are added using the `CATALOG <protocol> NODE` command. For example, to catalog a node called *steve* on the system *sanyal1*, which
has a TCP/IP listener for the instance on port 50000, we would catalog the instance from a remote client as:

```
CATALOG TCPIP NODE steve REMOTE sanyall SERVER 50000
```

If a server instance is being used as the client, then there is no node cataloged for the local instance. To list all of the nodes in the DB2 node directory, use the following DB2 command:

```
LIST NODE DIRECTORY
```

For each node in the DB2 node directory, you can catalog one or more databases. To catalog the DB2MALL database at the node named steve, use the following command:

```
CATALOG DB DB2MALL AT NODE steve
```

If you are cataloging the database on the local instance, omit the “AT NODE” portion of the command. You may often want to access multiple databases on different instances that have the same name. For example, you may have two online mall databases, both called DB2MALL, one at the node called steve and another at a node called prod. You cannot refer to both of them as DB2MALL since it will be impossible to tell them apart from their names. Instead, you can create a database alias for one of them or for both.

In our example, we assume that our production database is on the node called prod and our development database is on the node called steve. We will catalog the production database using the name prodmall and the development database using the name devmall as follows:

```
CATALOG DB DB2MALL AS prodmall at NODE prod
CATALOG DB DB2MALL AS devmall at NODE steve
```

We can now refer to each of them uniquely in our application as prodmall and devmall, respectively. If no alias is specified when you catalog a database, the database alias is automatically the same as the database name.

### Note

When you catalog a database alias, it must refer to a local database name or a remote database alias. You cannot create an alias of a local database alias.

#### 1.2.4 Database Manager Configuration

Each client and server instance has a database manager configuration file. Client instances have fewer parameters than server instances. You can look at the database manager configuration from the CLP using the following command:

```
GET DBM CFG
```
Each parameter name is placed in brackets and is preceded by a text description. All of the parameters are described in detail in the Administration Guide online book. To update a database configuration parameter, use the syntax:

```
UPDATE DBM CFG USING <parameter> <value>
```

To learn more about these parameters, refer to the Administration Guide.

### 1.2.5 DB2 Profile Registry

Some of the DB2 UDB environment is controlled by entries stored in the DB2 profile registry. The objective of this registry is to consolidate and manage DB2 environment settings at different level scopes. Registry settings can apply to an entire instance, or globally to all instances on a system. The `db2set` command is used to set and query entries in the DB2 profile registry. To view a list of all the profile registry entries that can be set, use the following command:

```
db2set -lr
```

To view the entries that are applicable to your current environment, use the command:

```
db2set -all
```

To set a parameter value, use the syntax:

```
db2set parameter=value [-i instance_name | -g ]
```

If neither optional clause is used, the parameter is set for the current instance. The `-i` option indicates that the parameter will be set for a particular instance (including all physical nodes in a DB2 UDB EEE environment), while the `-g` option sets the parameter globally for the physical system. Further usage of this command is described in the Command Reference, and all of the environment and registry variables that can be set are described in Appendix D of the Administration Guide.

### 1.2.6 Environment Variables

The most important environment variable that must be set is the `DB2INSTANCE` variable. This variable must be set to the name of the instance that is being accessed by the common set of DB2 client libraries. For example, on an AIX system, there may be two DB2 server instances, `db2inst1` and `db2inst2`, and two DB2 client instances, `db2cli1`, and `db2cli2`. Each of these instances will have their own DB2 database and node directories, along with database manager configuration and instance-level DB2 profile registry variables. The `DB2INSTANCE` variable must be set to distinguish which set of instance-specific settings to use.

If a user named `joe` also wanted to run applications that accessed the DB2 Runtime Client, he must also include the DB2 libraries in his environment settings. On UNIX, this means including the `sqllib/lib` directory in his `LIBPATH`. On Windows and OS/2, this involves adding `sqlib\bin` to the `PATH` variable.
1.2.7 Local vs. Remote Clients

The major difference between local clients and remote clients is the manner in which the DB2 client libraries communicate with the DB2 database engine. As we discussed above, all client applications access the DB2 libraries by referencing them in their environment and setting the `DB2INSTANCE` variable. When a database is accessed from a local client, the DB2 shared libraries communicate with a DB2 server agent process using shared memory segments for Inter-Process Communication (IPC). Remote clients use network protocol-specific buffers to communicate with a remote DB2 server agent. Generally speaking, IPC over a shared memory segment is faster than over a network connection.

It is possible for applications running on the same local instance to behave like remote clients by cataloging a TCP/IP loopback connection. In some cases, this may be a necessity, while in others it may provide a useful way to test an application that is developed locally but may run on a variety of remote clients.

The most applicable example is on AIX systems, which by default can have a maximum of 11 shared memory segment attachments per process. Since each database connection requires one shared memory segment attachment, an application can have at most 10 concurrent local connections (the eleventh segment is reserved). To get around this limitation, you can configure a TCP/IP loopback connection to the local instance.

Let’s assume that the local instance has a TCP/IP listener corresponding to the service named `db2tcp`, and we want to access the `DB2MALL` database. TCP/IP loopback can be configured as follows:

```
UNCATALOG DB DB2MALL
CATALOG DB DB2MALL AS mall
CATALOG TCPIP NODE local AT 127.0.0.1 SERVER db2tcp
CATALOG DB2 mall AS DB2MALL AT NODE local
```

As we mentioned earlier, this can be a useful way to test your applications by eliminating network transfer lag since a local TCP/IP connection should be almost as fast as IPC. It is useful to test remote applications this way if possible, since applications may follow a slightly different code path within the DB2 libraries depending on whether a database is local or remote.

1.3 Database Objects

Before we explore how to access DB2 databases using SQL, we begin by describing application-related database objects. Even if you are already familiar with relational databases, you may find the following sections useful, because different vendors often vary in their terminology even though the overall concepts remain the same. This section discusses how to structure your understanding of fundamental relational database concepts toward the DB2 family of products. In general, database objects are created using the syntax:

```
CREATE <object-type> <object-name> ...
```
Similarly, you can delete a database object using the syntax:

   DROP <object-type> <object-name> ...

Some database objects can also be modified after they are created using the syntax:

   ALTER <object-type> <object-name> ...

Each DB2 database contains a set of system catalog tables that contain information about
all database objects within the database, as well as statistics associated with those objects. By
default, all object names are created in uppercase, even if you specify the name in lowercase. For
case-sensitive names, you should put double quotes (") around the name of each object.

   Tip: Database statistics are key for obtaining better query performance. Statistics should be maintained and updated using the
   RUNSTATS command. See the Administration Guide and Command Reference for details.

1.3.1 Tables
A table is an unordered set of data records consisting of rows and columns. Each row is known
as a record, and each column within a row is a field within that record. Permanent (base) tables
are created using the CREATE TABLE statement, which defines a logical view of how the data
is stored on disk. DB2 also has derived temporary tables, which we discuss in Chapter 3.

1.3.2 Data Types
Each column within a table is based on a data type. The data type of a column indicates the
length of the values in it and the kind of data that is valid for it. There are two major categories
of data types in DB2 UDB V7.1:

   • Built-in data types.
   • User-Defined Types (UDTs).

Built-in data types are defined by DB2 UDB. UDTs are categorized as follows:

   • User-defined distinct type – Enables you to create a new data type that has its own semantics based on existing built-in types.
   • User-defined structured type – Lets you create a structure that contains a sequence of named attributes, each of which has a data type.
   • User-defined reference type – Is used to reference a row in another table that uses a user-defined structured type.

The latter two types are part of DB2’s object relational features, and we will not delve into them in this book. We will discuss how to make use of UDTs in general in Chapter 3.
1.3.3 Indexes

Indexes are physical objects that are associated with a single permanent table (see Figure 1-3). You can define multiple indexes on the same table. Indexes are used to ensure uniqueness of data values and/or improve SQL query performance. Using indexes allows data to be sorted and accessed more quickly and avoids the time-consuming task of sorting data in temporary storage. DB2 UDB updates indexes automatically when data is inserted, updated, and deleted from a table.

**Note**

The maintenance overhead of indexes will negatively impact the performance of INSERT, UPDATE, and DELETE statements.
Indexes can be defined in ascending or descending order (dependent on the database code page), they can be unique or nonunique, and they can involve a single column’s data values or be compound indexes defined on multiple columns. They can also be defined to support both forward and reverse scans.

**Tip** The DB2 Explain facility can be used to provide index usage information for every explainable SQL statement. Refer to the Administration Guide for instructions.

### 1.3.4 Schemas

Schemas are database objects used in DB2 UDB to logically group a set of database objects. Most database objects are named using a two-part naming convention (schema_name.object_name). The first part of the name is referred to as the schema, and the second part is the name of the object. When the schema is not specified, objects are referenced in SQL statements using an implicit schema, which by default is your authorization ID. The CURRENT_SCHEMA special register within DB2 contains this qualifier and can be modified by your application at bind time, or by issuing the SET CURRENT_SCHEMA command at runtime.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>When accessing DB2/390, the CURRENT_SCHEMA is referred to as the CURRENT_SQLID.</td>
</tr>
</tbody>
</table>

### 1.3.5 Transactions

A transaction is a set of one or more SQL statements that execute as a single atomic operation. The term “unit of work (UOW)” is synonymous with the term “transaction.” A transaction either succeeds or fails, and is started implicitly with the first executable SQL statement in a program. A transaction is completed when either an explicit COMMIT or ROLLBACK is encountered. An implicit COMMIT or ROLLBACK can occur when a DB2 UDB application terminates.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>It is best to COMMIT or ROLLBACK any outstanding transactions explicitly before terminating your application, because the nature of an implicit rollback varies in different operating environments.</td>
</tr>
</tbody>
</table>

### 1.3.6 Locks

DB2 UDB is a multiuser database product designed for concurrent access. As users request data, DB2 UDB uses locking strategies to avoid resource conflicts and maintain data integrity. For
example, as SQL statements are processed in a transaction, an application connection may obtain locks. Locks are released when the resource is no longer required at the end of the transaction. The locks are stored in memory on the database server in a structure known as a locklist. DB2 UDB supports two types of locks, table locks and row locks.

The locking strategy used by DB2 UDB during transaction processing is specified using an isolation level. The isolation level is defined when binding an application. In addition, for JDBC and CLI (Call Level Interface) applications, the isolation level can be set as a connection property or attribute, respectively.

### 1.3.7 Packages

Packages are database objects that contain executable forms of SQL statements. Each statement is contained within a section of a package and references a specific statement within a single program source module. Packages are not referred to directly in an SQL query and remain largely transparent to the application developer.

**Note**

Only the corresponding program source module can invoke the contents of a package. Otherwise, an SQL0818 timestamp conflict error will occur.

Packages are stored in the database system catalog tables and contain the DB2 UDB access plan that was selected by the DB2 optimizer during BIND or PREP. This is known as static binding since it is performed prior to statement execution.

### 1.4 SQL Concepts

SQL is the industry-standard language for defining and querying relational databases. It is largely based on the relational algebraic operators select, project, and join. SQL also includes additional operators for inserting, updating, and deleting data, as well as creating database objects and administering access to them. These qualities separate SQL into three categories:

- **Data Manipulation Language (DML)**—Allows you to query and modify data within database tables.
- **Data Definition Language (DDL)**—Allows you to create database objects.
- **Data Control Language (DCL)**—Allows you to administer access rights to database objects.

For most of this book, we will be focusing on DML, since this is the main focus of OLAP and OLTP applications. In this section, we will describe the basic forms of DML queries. We will describe more advanced features in Chapter 3. Every RDBMS vendor has a slightly differ-
ent brand of SQL; however, if you are already familiar with SQL from using another vendor’s product, you should have no problem picking up SQL in DB2 UDB.

1.4.1 Querying Data

Data is queried from one or more tables using a `SELECT` statement. The data returned is known as a result set. A `SELECT` statement only specifies the criteria the result set must match, not how DB2 should return it. The DB2 optimizer makes this decision by constructing an access plan based on current database statistics from the system catalog tables and the types of plans it has been instructed to consider (using a setting called the query optimization level). The structure of a `SELECT` statement is explained using the following example, which returns all new inventory requests sorted by store in the past 10 days:

```
SELECT s.store_name, p.product_name, r.unit_cost, r.order_date
FROM product p, store_purchase r, store s
WHERE s.store_id=p.store_id AND p.product_id=r.product_id AND r.order_date > CURRENT DATE - 10 DAYS
ORDER BY s.store_name
```

The column list directly follows the `SELECT` keyword, and the corresponding table names are specified after the `FROM` keyword. Finally, restrictions on which rows to retrieve are placed after the `WHERE` clause using SQL-standard Boolean syntax. In addition, each column is referred to using a correlation name, which has been chosen as `p` for the `product` table, `r` for the `store_purchase` table, and `s` for the `store` table. Correlation names are mandatory whenever there is ambiguity, such as for the `product_id` column, which is in both the `product` and `store_purchase` tables. Finally, we use an `ORDER BY` to sort the rows in the result set. Sorting takes additional time and can considerably slow down query performance, so you should consider using unsorted queries if possible.

There are also more complex forms of the `SELECT` statement. We cover some of them in Chapter 3, as well as in the advanced SQL sections of *The DB2 UDB V7.1 Administration Certification Guide*.

1.4.2 Modifying Data

You can modify data in DB2 tables using `INSERT`, `UPDATE`, and `DELETE` statements. An `INSERT` statement allows you to add one or more rows to a table. There are two ways to perform an insert. First, you can specify the values for the columns in each row to be inserted using the `VALUES` clause:

```
INSERT INTO TABLE1 (COL1, COL2) VALUES(1, 2)
```

Here, we specify the names of all the columns we will include values for, prior to the `VALUES` clause. This is often useful if the order of columns is different from the table definition.
or if you are inserting the default value or a NULL value into columns that are not specified. If you are going to insert a value into every column, you do not need to include a column list. We recommend always using the column list, however, for style and clarity.

You can also insert rows into a table by selecting rows from another table as the second method. Here is an example:

```
INSERT INTO TABLE1(COL1, COL2) SELECT (C1, C2) FROM T1
```

In many cases, rows will already exist in the database, and we need to update them. For example, we may wish to update the amount of an item in stock after a purchase is made in our online mall database. We do this using an UPDATE statement:

```
UPDATE product SET units_in_stock=units_in_stock-10
WHERE product_id=123123
```

Our statement decreases the quantity on hand for product 123123. As you can see, an UPDATE statement allows you to specify a WHERE clause. Without specifying a WHERE clause in this statement, you would update every row in the table! This is the same for a DELETE statement. For example, if we wanted to delete the row we just updated, we would issue the statement:

```
DELETE FROM product WHERE productid=123123
```

You can make your WHERE clause as complicated as you like. You can use Boolean operators such as AND and OR, equality operators such as =, >, <, <=, and other comparison operators such as LIKE and BETWEEN. These are fully documented in the SQL Reference.

1.5 DB2 Application Structure

Database applications issue SQL queries within the context of a programming interface such as embedded SQL, CLI/ODBC, JDBC, or SQLJ. All of these APIs follow the same series of steps when executing an SQL query, also known as a statement. We will try to describe the common structure of database applications in this section.

1.5.1 Connecting to a Database

Before an application can make requests to a DB2 database using SQL, it must connect to it. The general syntax for doing this in DB2 is demonstrated using a connection for the DB2MALL database:

```
CONNECT TO DB2MALL USER ssanyal USING mypass
```

The user’s ID follows the USER clause and the password follows the USING clause in the CONNECT statement. For implicit connections, both of these clauses can be omitted. Your application can use an implicit connection when CLIENT authentication is used or when it is connecting to a local database. For details on CLIENT authentication, consult the Administration Guide.
In many Application Programming Interfaces (APIs), you won’t need to specify a `CONNECT` statement using this syntax. Instead, you will use an API call, which will do this for you.

### 1.5.2 Dynamic and Static Statement Execution

Once connected, applications can execute SQL statements in two ways, dynamically and statically. To execute an SQL statement statically, the statement must be statically prepared and bound to the database. When a query is statically bound, the compiled instructions about how to execute it are stored in a database package. You can also generate an SQL statement at runtime and run it dynamically, in which case DB2 will have to prepare an access plan. A dynamic query’s access plan is never permanently stored in the database, so it may need to be generated each time it is executed. DB2 does have a package cache that stores the access plans for dynamic queries to improve performance, but this is not permanent storage. Beginning in the next chapter, we will continue this discussion so that you will gain better awareness about when to use static and when to use dynamic SQL.

### 1.5.3 Statement Life Cycle

Every SQL statement has a life cycle. The nature of a statement’s life cycle depends on whether it seeks to query the database or modify its contents, and whether it is dynamic or static. A dynamic statement must first be prepared. Here is a pseudocode example of a `PREPARE` for a `SELECT` statement issued against the `product` table of the `DB2MALL` database:

```sql
STMT = "SELECT * FROM product"
PREPARE S1 FROM :STMT
```

The statement is initially stored in the variable `STMT`. This variable is then referenced as a *host variable* in the `PREPARE` statement, `S1`, by preceding it with a colon (:). We will discuss host variables in detail in Chapter 4.

The `PREPARE` operation performs two tasks; first it checks the statement for syntax errors, and then it creates an access plan. The access plan contains a compiled set of instructions the database manager must perform to execute the query. A single query can have many access plans. The DB2 optimizer estimates the cost for a subset of possible plans and chooses the one with the lowest cost.

**Tip**

To accurately estimate the cost of an access plan, the database statistics must be up-to-date. The Database Administrator (DBA) should regularly ensure that the statistics are current by issuing the `RUNSTATS` command.
Once a statement has been prepared, it can be executed. Upon execution of a SELECT statement, the database manager retrieves all rows that match the criteria specified and places them into a temporary table, which is read using a cursor. This retrieved data is known as the result set for that statement. Statements that modify the database are not immediately written to disk. Instead, they are written to the database transaction logs and written to disk after the application issues an implicit or explicit COMMIT.

While there is an actual EXECUTE command for INSERT, UPDATE, and DELETE statements, a SELECT statement is executed by opening a database object known as a cursor. A cursor is used to navigate through the result set that was generated when the query was executed. The most basic type of cursor is used to scroll forward through a result set one row at a time. Some forms of cursors also allow you to update or delete data as it is being viewed, scroll backwards, or jump instantly to any row you desire in the result set. Updatable cursors also allow you to modify or delete the row where the cursor is currently positioned. A cursor is initially created for a prepared statement using a DECLARE operation, which associates it with a statement as the following example shows:

```
DECLARE C1 CURSOR FOR S1
```

Here, we have created a cursor called C1 for the statement we previously prepared, which we referred to as S1. If this had been a statically executed statement, we would have declared it as follows:

```
DECLARE C1 CURSOR FOR SELECT * FROM product
```

As you can see, here we have specified the SQL query directly in the cursor declaration rather than referencing it with a variable element as we do in dynamic SQL. This is important, because the variable STMT that we used earlier could have been assigned at runtime, and thus the same piece of code could have been used to prepare any SQL query. This is the true power of dynamic SQL, whereas in static SQL, a query must be hard-coded.

In both cases, after having declared the cursor, we then execute the statement and open the cursor by simply issuing:

```
OPEN C1
```

When the statement is executed, a result set is built and stored inside a temporary table. Each row of data is then retrieved using a FETCH operation as follows:

```
FETCH C1
```
After all of the rows from the result set have been fetched in this manner, the database manager returns a +100 return code to indicate that no more data is available. At this point, the application should close the cursor using the CLOSE command on the cursor:

```
CLOSE C1
```

**Note**

Cursor names must be unique within a program source module. You should never declare another cursor with the same name in a source module.

Once the cursor is closed, the result set is discarded, but this is not necessarily the end of this statement. Once it has been prepared, we can execute it more than once. This saves the overhead of generating the access plan multiple times. This is possible because DB2 stores the prepared form of the statement in its package cache, thereby allowing you to reuse it.

You may never have to actually declare cursor names and fetch on them using the syntax noted above. Most interfaces such as CLI or JDBC will shield you from requiring this. However, you should be aware that no matter which programming interface you are using, the base operations described above are actually occurring.

### 1.5.4 Committing or Rolling Back Changes

Earlier, we discussed database objects known as transactions. An application run can have many transactions. Each ROLLBACK cancels changes made during the current atomic unit of work, whereas issuing a COMMIT applies these changes. For example, when purchasing something from the mall database, several tables need to be updated:

- The **product** table needs to have the quantity purchased subtracted from the quantity on-hand for each item purchased. This involves one or more UPDATE operations.
- The **invoice** table needs to have a record inserted.
- The **customer_order** table needs to have one or more rows inserted.

What if the application were to crash before all of these operations were complete? Or even worse, what if the database manager itself crashed due to a system outage or other reason? The database may end up in an inconsistent state since the entire unit of work was not completed.

To prevent this from occurring, the application needs to COMMIT or ROLLBACK each unit of work. This allows the database manager to permanently apply or cancel the changes made by the application, respectively. In the customer purchase example, a COMMIT would only be issued if all three of the steps were completed.
1.5.5 Disconnecting from a Database

After completing database operations, the application should disconnect from the database. This is done using the `CONNECT RESET` command. It is good design for your applications to explicitly clean up their connections and it is also wise to disconnect from the database if the application will not be performing database operations in the near future. This can also free up resources such as locks, which was discussed in the previous section.

1.6 Summary

In this chapter, we began by introducing you to the DB2 client server environment. We learned that each DB2 server instance is a separate database manager, and that DB2 client instances are remote clients that access databases managed by a DB2 server instance. DB2 commands and SQL can be run outside of an application using the CLP or Command Center.

Persistent data within a database is stored in base tables, and we learned about various database objects such as schemas, indexes, and packages. We also provided a quick overview of SQL and the four types of DML statements: `SELECT`, `INSERT`, `UPDATE`, and `DELETE`. Statements can be executed statically or dynamically. In the former case, an access plan is persistently stored in a database package. In the latter case, the access plan is generated at runtime.

Data can be retrieved using a `SELECT` statement, which produces a result set. The result set is a temporary table that is traversed using a cursor. Multiple SQL statements can comprise an atomic operation known as a transaction. The changes made during a transaction are either committed or rolled back at the end of the transaction.

Based on these concepts, application-database interaction can be summarized in a series of steps. First, the application must connect to the DB2 database it wishes to access. It can then perform as many queries as it likes, using as many transactions as required. Finally, it must disconnect from the database.