Chapter 2

DB2 concepts

• Structured query language  38
• DB2 data structures    40
• Enforcing business rules  49
• DB2 system structures    52
• Application processes and transactions  54
• Packages and application plans    55
• Distributed data       57
This chapter introduces you to the concepts, structures, and processes that comprise a relational database. The concepts give you a basic understanding of what a relational database is. The structures and processes are the key elements of a DB2 database. Later chapters discuss these topics in more depth.

In a relational database, data is perceived to exist in one or more tables. Each table contains a specific number of columns and a number of unordered rows. Each column in a table is related in some way to the other columns. Thinking of the data as a collection of tables gives you an easy way to visualize the data that is stored in a DB2 database.

Tables are at the core of a DB2 database. However, a DB2 database involves more than just a collection of tables. You will read about other objects such as views and indexes and larger data containers such as table spaces. These objects are collectively referred to as DB2 structures.

**Structured query language**

The language you use to access the data in DB2 tables is the structured query language (SQL). SQL is a standardized language for defining and manipulating data in a relational database.

The language consists of SQL statements. Some statements let you define data objects, such as tables. Some statements let you retrieve, insert, update, or delete data in tables. And other statements let you authorize users to access specific resources, such as tables or views.

When you write an SQL statement, you specify what you want done, not how to do it. To access data, for example, you only need to name the tables and columns that contain the data. You do not need to describe how to get to the data.

In accordance with the relational model of data:

- The database is perceived as a set of tables.
- Relationships are represented by values in tables.
Structured query language

- Data is retrieved by using SQL to specify a result table that can be derived from one or more tables.

DB2 transforms each SQL statement, that is, the specification of a result table, into a sequence of internal operations that optimize data retrieval. This transformation occurs when the SQL statement is prepared. This transformation is also known as binding.

All executable SQL statements must be prepared before they can run. The result of preparation is the executable or operational form of the statement.

As the following example illustrates, SQL is generally intuitive.

Example: Assume that you are shopping for shoes and you want to know what shoe styles are available in size 8. The SQL query you need to write is very similar to the question you would ask a salesperson, “What shoe styles are available in size 8?” Just as the salesperson checks the shoe inventory and returns with an answer, DB2 retrieves information from a table (SHOES) and returns a result table. The query looks like this:

```
SELECT STYLE, SIZE
FROM SHOES
WHERE SIZE = '8';
```

Assume that the answer to your question is that two shoes styles are available in a size 8: loafers and sandals. The result table looks like this:

```
+-------+-----+
| STYLE | SIZE|
|-------+-----|
| LOAFERS | 8 |
| SANDALS | 8 |
```

You can send an SQL statement to DB2 in several ways. One way is interactively, with a user entering SQL statements at a keyboard. Another way is through an application program. The program can contain SQL statements that are statically embedded in the application. Alternatively, the program can create its SQL statements dynamically, for example, in response to information that a user provides by filling in a form. Later in this book, you will read about each of these methods.
DB2 data structures

The elements that DB2 manages can be divided into two broad categories:

- Data structures, which are accessed under the user’s direction and used to organize user data (and some system data).
- System structures, which are controlled and accessed by DB2. “DB2 system structures” on page 52 describes the DB2 system structures.

This section describes the DB2 data structures, beginning with the smaller data structures and progressing to larger data structures. They include:

- “Tables”
- “Indexes” on page 42
- “Keys” on page 43
- “Views” on page 44
- “Table spaces” on page 47
- “Index spaces” on page 47
- “Databases” on page 47

Tables

Tables are logical structures that DB2 maintains. As you already learned, tables are made up of columns and rows. The rows of a relational table have no fixed order. The order of the columns, however, is always the order in which you specified them when defining the table.

At the intersection of every column and row is a specific data item called a value. A column is a set of values of the same type. A row is a sequence of values such that the nth value is a value of the nth column of the table. Every table must have one or more columns, but the number of rows can be zero.

DB2 accesses data by referring to its content instead of its location or organization in storage.
DB2 supports several different types of tables, some of which are listed here:

**base table**
A table that is created with the SQL statement CREATE TABLE and that holds persistent user data.

**temporary table**
A table that is defined by the SQL statement CREATE GLOBAL TEMPORARY TABLE or DECLARE GLOBAL TEMPORARY TABLE to hold data temporarily. You can use temporary tables to store intermediate SQL results. For example, a temporary table is useful for storing a large result set that you need to sort. A temporary table exists only as long as the application process that uses it, and it is not shared outside of that application process. There are two types of temporary tables: created and declared.

**result table**
A table that contains a set of rows, called a result set, that DB2 returns when you use an SQL statement to query the tables in the database. Unlike a base table or a temporary table, a result table is not an object that you define using a CREATE statement.

**Example tables**

The examples in this chapter are based on two example tables: a department (DEPT) table and an employee (EMP) table. The tables represent information about the employees of a computer company.

Figure 2.1 shows the DEPT table. Each row in the DEPT table contains data for a single department: its number, its name, the employee number of its manager, and the administrative department. Notice that depart-

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>DEPTNAME</th>
<th>MGRNO</th>
<th>ADMRDEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A00</td>
<td>CHAIRMANS OFFICE</td>
<td>000010</td>
<td>A00</td>
</tr>
<tr>
<td>B01</td>
<td>PLANNING</td>
<td>000020</td>
<td>A00</td>
</tr>
<tr>
<td>C01</td>
<td>INFORMATION CENTER</td>
<td>000030</td>
<td>A00</td>
</tr>
<tr>
<td>D11</td>
<td>MANUFACTURING SYSTEMS</td>
<td>000060</td>
<td>D11</td>
</tr>
<tr>
<td>E21</td>
<td>SOFTWARE SUPPORT</td>
<td>-----</td>
<td>D11</td>
</tr>
</tbody>
</table>

*Figure 2.1*  
Example DEPT table.
ment E21 has no manager. In this case, the dashes represent a null value, a special value that indicates the absence of information.

Figure 2.2 shows the EMP table. Each row in the EMP table contains data for a single employee: employee number, first name, last name, the department the employee reports to, the employee’s hire date, job title, education level, salary, and commission.

Indexes

An index is an ordered set of pointers to rows of a table. Conceptually, you can think of an index to the rows of a DB2 table like you think of an index to the pages of a book. Each index is based on the values of data in one or more columns.

DB2 can use indexes to improve performance and ensure uniqueness. In most cases, access to data is faster with an index than with a scan of the data. For example, you can create an index on the DEPTNO column of the DEPT table to easily locate a specific department and avoid reading through each row, or scanning, the table.
An index is an object that is separate from the data in the table. When you define an index using the CREATE INDEX statement, DB2 builds this structure and maintains it automatically.

**Keys**

A *key* is one or more columns that are identified as such in the description of a table, an index, or a referential constraint. (Referential constraints are described in “Referential integrity and referential constraints” on page 49.) The same column can be part of more than one key.

A key that is composed of more than one column is called a *composite key*. The ordering of the columns is not constrained by their actual order within the table. The term *value*, when used with respect to a composite key, denotes a *composite value*.

**Unique keys**

A *unique key* is a key that is constrained so that no two of its values are equal. DB2 enforces the constraint whenever you add or modify data. DB2 uses a *unique index* to enforce the constraint. Every unique key is a key of a unique index. Such an index is also said to have the UNIQUE attribute. You define a unique key by using the UNIQUE clause of the CREATE TABLE or the ALTER TABLE statement. A table can have any number of unique keys.

**Primary keys**

A *primary key* is a special type of unique key, one without null values. For example, the DEPTNO column in the DEPT table is a primary key.

A table can have no more than one primary key. Primary keys are optional and can be defined in CREATE TABLE or ALTER TABLE statements.

The unique index on a primary key is called a *primary index*. When a primary key is defined in a CREATE TABLE statement, the table is marked unavailable until a primary index exists.
When a primary key is defined in an ALTER TABLE statement, a unique index must already exist on the columns of that primary key. This unique index is designated as the primary index.

Parent keys

A parent key is either a primary key or a unique key in the parent table of a referential constraint. The values of a parent key determine the valid values of the foreign key in the constraint.

Foreign keys

A foreign key is a key that is specified in the definition of a referential constraint using the CREATE or ALTER TABLE statement. A foreign key refers to or is related to a specific parent key. Unlike other types of keys, a foreign key does not require an index on its underlying column or columns.

A table can have zero or more foreign keys. The value of a composite foreign key is null if any component of the value is null.

Figure 2.3 shows the relationship between some columns in the DEPT table and the EMP table.

Views

A view provides an alternative way of looking at the data in one or more tables. A view is a named specification of a result table.

Conceptually, creating a view is like using binoculars. You might look through binoculars to see an entire landscape or to look at a specific image within the landscape, like a tree. Similarly, you can create a view that combines data from different base tables or create a limited view of a table that omits data. In fact, these are common reasons to use a view. Combining information from base tables simplifies retrieving data for an end user, and limiting the data a user can see is useful for security purposes.
You use the CREATE VIEW statement to define a view. Specifying the view in other SQL statements is effectively like running an SQL SELECT statement. At any time, the view consists of the rows that would result from the SELECT statement it contains. You can think of a view as having columns and rows just like the base table on which the view is defined.

**Example:** Figure 2.4 shows a view of the EMP table that omits sensitive employee information and renames some of the columns.
The following CREATE VIEW statement defines the EMPINFO view in Figure 2.4:

```
CREATE VIEW EMPINFO (EMPLOYEE, FIRSTNAME, LASTNAME, TEAM, JOBTITLE)
    AS SELECT EMPNO, FIRSTNAME, LASTNAME, TEAM, JOB
        FROM EMP;
```

You can use views for a number of different purposes. A view can:

- Control access to a table and make data easier to use
- Simplify authorization by granting access to a view without granting access to the table
- Show only portions of data in the table
- Show summary data for a given table
- Combine two or more tables in meaningful ways
- Show only the selected rows that are pertinent to the process using the view

In general, a view inherits the attributes of the object from which it is derived. Columns that are added to the tables after the view is defined on those tables do not appear in the view. An index cannot be created for a view.
For retrieval, you can use views like base tables. Whether a view can be used in an insert, update, or delete operation depends on its definition.

Table spaces

All tables are kept in table spaces. Table spaces, DB2 storage structures, are one or more data sets that store one or more tables.

The two primary types of table spaces are:

**Segmented**
A table space that can contain more than one table. The space is composed of groups of pages called *segments*. Each segment is dedicated to holding rows of a single table.

**Partitioned**
A table space that can contain only a single table. The space is divided into separate units of storage called *partitions*. Each partition belongs to a range of key values. And each partition can be processed concurrently by utilities* and SQL. A partitioned table space is a good choice for tables that are larger than 1 GB in size. A partitioned index determines what data goes into each partition.

You define a table space by using the CREATE TABLESPACE statement.

Index spaces

An *index space*, another DB2 storage structure, contains a single index. When you create an index using the CREATE INDEX statement, an index space is automatically defined in the same database as the table.

Databases

In DB2, a *database* is a set of table spaces and index spaces. These index spaces contain indexes on the tables in the table spaces of the same data-

---

*In Version 7 of DB2 for OS/390, some utility functions are available as optional products. Customers must separately order and purchase a license to such utilities, and discussion of those utility functions in this publication is not intended to otherwise imply that all DB2 for OS/390 Version 7 customers are licensed to use them.
base. You define databases by using the CREATE DATABASE statement. Whenever a table space is created, it is explicitly or implicitly assigned to an existing database.

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 allows you to start or stop access to all the data in one operation. You can also grant authorization for access to all the data as a single unit. Assuming that you are authorized to do so, you can access data that is stored in different databases.

**Recommendation:** Avoid using a single database for a large number of tables. Defining a maximum of twenty to fifty tables in a database improves performance.

Figure 2.5 shows how the main DB2 data structures fit together.

![Figure 2.5: Data structures in a DB2 database](image)
Enforcing business rules

The discussion in “Keys” on page 43 raises the topic of referential integrity. This section describes how referential integrity ensures the integrity of your data by enforcing rules with referential constraints, check constraints, and triggers. You can put the database to work by using constraints and triggers. You can rely on these mechanisms to ensure the integrity and validity of your data, rather than relying on individual applications to do that work.

Referential integrity and referential constraints

Referential integrity is the state in which all values of all foreign keys are valid.

DB2 ensures referential integrity between your tables when you define referential constraints. A referential constraint is the rule that the nonnull values of a foreign key are valid only if they also appear as values of a parent key. The table that contains the parent key is called the parent table of the referential constraint, and the table that contains the foreign key is a dependent of that table.

The relationship between some rows of the DEPT and EMP tables, shown in Figure 2.6, illustrates referential integrity concepts and terminology. For example, referential integrity ensures that every foreign key value in the DEPT column of the EMP table matches a primary key value in the DEPTNO column of the DEPT table.

Two parent and dependent relationships exist between the DEPT and EMP tables.

- The foreign key on DEPT establishes a parent and dependent relationship. The DEPT column in the EMP table depends on the DEPTNO in the DEPT table. Through this foreign key relationship, the DEPT table is the parent of the EMP table. You can assign an employee to no department, but you cannot assign an employee to a department that does not exist.
The foreign key on MGRNO also establishes a parent and dependent relationship. Because MGRNO depends on EMPNO, EMP is the parent table of the relationship and DEPT is the dependent table. A table can be a dependent of itself; this is called a self-referencing table. For example, the DEPT table is self-referencing because the value of the administrative department (ADMREPT) must be a department ID. To enforce the self-referencing constraint, a foreign key needs to be defined. Similar terminology applies to the rows of a parent-and-child relationship. A row in a dependent table, called a dependent row, refers to a row in a parent table, called a parent row. But a row of a parent table is not always a parent row—perhaps nothing refers to it. Likewise, a row of a dependent table is not always a dependent row—the foreign key can allow null values, which refer to no other rows.

Referential constraints are optional. You define referential constraints by using CREATE TABLE and ALTER TABLE statements.

To support referential integrity, DB2 enforces rules when users insert, load, update, or delete data.
Check constraints

A check constraint is a rule that specifies the values that are allowed in one or more columns of every row of a table. Like referential constraints, check constraints are optional and you define them by using the CREATE TABLE and ALTER TABLE statements. The definition of a check constraint restricts the values that a specific column of a base table can contain. For example, one restriction is that a column name in a check constraint on a table must identify a column of that table.

A table can have any number of check constraints. DB2 enforces a check constraint by applying the restriction to each row that is inserted, loaded, or updated.

Example: You can create a check constraint to ensure that all employees earn a salary of $30 000 dollars or more:

```sql
...SALARY INTEGER CHECK (SALARY >= 30000)
```

Triggers

A trigger defines a set of actions that are executed when an insert, update, or delete operation occurs on a specified table. When an insert, load, update, or delete is executed, the trigger is said to be activated.

You can use triggers along with referential constraints and check constraints to enforce data integrity rules. Triggers are more powerful than constraints because you can use them to:

- Update other tables
- Automatically generate or transform values for inserted or updated rows
- Invoke functions that perform operations both inside and outside of DB2

For example, assume that you need to prevent an update to a column when a new value exceeds a certain amount. Instead of preventing the update, you can use a trigger. The trigger can substitute a valid value and
invoke a procedure that sends a notice to an administrator about the invalid update.

You define triggers by using the CREATE TRIGGER statement.

DB2 system structures

DB2 has a comprehensive infrastructure that enables it to provide data integrity, performance, and the ability to recover user data. Unlike the DB2 data structures that users create and access, DB2 controls and accesses system structures. The system structures that this section describes are:

- “Catalog”
- “Active and archive logs” on page 53
- “Bootstrap data set” on page 53
- “Buffer pools” on page 53

Catalog

DB2 maintains a set of tables that contain information about the data that is under its control. These tables are collectively known as the catalog. The catalog tables contain information about DB2 objects such as tables, views, and indexes. When you create, alter, or drop an object, DB2 inserts, updates, or deletes rows of the catalog that describe the object.

To understand the role of the catalog, consider what happens when the EMP table is created. DB2 records:

- Table information
  To record the table name and the name of its owner, its creator, its type, the name of its table space, and the name of its database, DB2 inserts a row into the catalog.
- Column information
  To record information about each column of the table, DB2 inserts the name of the table to which the column belongs, its length, its
data type, and its sequence number by inserting rows into the cata-
log for each column of the table.

- Authorization information

To record that the owner of the table has authorization on the table,
DB2 inserts a row into the catalog.

Tables in the catalog are like any other database tables with respect to re-
trieval. If you have authorization, you can use SQL statements to look at
data in the catalog tables in the same way that you retrieve data from any
other table in the system. DB2 ensures that the catalog contains accurate
object descriptions.

Active and archive logs

DB2 records all data changes and other significant events in a log. By having
this record of changes, DB2 can re-create those changes for you in the event
of a failure. DB2 can even roll the changes back to a previous point in time.

DB2 writes each log record to a disk data set called the active log. When
the active log is full, DB2 copies the contents of the active log to a disk or
magnetic tape data set called the archive log.

Bootstrap data set

The bootstrap data set (BSDS) contains information that is critical to
DB2, such as the names of the logs. DB2 uses information in the BSDS
for system restarts and for any activity that requires reading the log.

Buffer pools

Buffer pools, also known as virtual buffer pools, are areas of virtual storage
in which DB2 temporarily stores pages of table spaces or indexes. When an
application program accesses a row of a table, DB2 retrieves the page con-
taining that row and places the page in a buffer. If the needed data is already
in a buffer, the application program avoids waiting for it to be retrieved from
disk, which significantly reduces the cost of retrieving the page.
DB2 lets you specify default buffer pools for user data and for indexes. A special type of buffer pool that is used only in Parallel Sysplex® data sharing is the group buffer pool.

Application processes and transactions

Many different types of programs access DB2 data: user-written applications, SQL statements that users enter dynamically, and even utilities. The single term that describes any type of access to DB2 data is called an application process. All SQL programs run as part of an application process. An application process involves running one or more programs. Different application processes might involve running different programs, or different runs of the same program.

When an application interacts with a DB2 database, a transaction begins. A transaction is a sequence of actions between the application and the database that begins when data in the database is read or written. A transaction is also known as a unit of work.

Example: Consider what happens when you access funds in a bank account. A banking transaction might involve the transfer of funds from one account to another. During the transaction, an application program first subtracts the funds from the first account, and then adds the funds to the second account. Following the subtraction step, the data is inconsistent. Consistency is reestablished after the funds have been added to the second account.

DB2 uses techniques to ensure data consistency that include a commit operation, a rollback operation, and locking.

When the subtraction and addition steps of the banking transaction are complete, the application can use the commit operation to end the transaction, thereby making the changes available to other application processes. The commit operation makes the database changes permanent.

Consider what happens if more than one application process requests access to the same data at the same time. Or, under certain circumstances, an SQL statement might run concurrently with a utility on the same table.
space. DB2 uses locks to maintain data integrity under these conditions to prevent, for example, two application processes from updating the same row of data simultaneously.

DB2 acquires locks to prevent uncommitted changes that are made by one application process from being perceived by any other. DB2 automatically releases all locks it has acquired on behalf of an application process when that process ends, but an application process can also explicitly request that locks be released sooner. A commit operation releases locks that an application process has acquired and commits database changes that were made by the same process.

DB2 also provides a way to back out uncommitted changes that an application process makes. This might be necessary in the event of a failure on the part of an application process, or in a deadlock situation. Deadlock occurs when contention for the use of a resource, such as a table, cannot be resolved. An application process, however, can explicitly request that its database changes be backed out. This operation is called rollback. The interface used by an SQL program to explicitly specify these commit and rollback operations depends on the environment.

Packages and application plans

A *package* contains control structures that DB2 uses when it runs SQL statements. Packages are produced during program preparation. You can think of the control structures as the bound or operational form of SQL statements. All control structures in a package are derived from the SQL statements that are embedded in a single source program.

An *application plan* relates an application process to a local instance of DB2, specifies processing options, and contains one or both of the following elements:

- A list of package names
- The bound form of SQL statements
Every DB2 application requires an application plan. Packages make application programs more flexible and easier to maintain. For example, when you use packages, you do not need to bind the entire plan again when you change one SQL statement.

**Example:** Figure 2.7 shows an application plan that contains two packages. Suppose that you decide to change the SELECT statement in package AA to select data from a different table. In this case, you need to bind only package AA again and not package AB.

You create plans and packages by using the DB2 commands `BIND PLAN` and `BIND PACKAGE`.

![Figure 2.7](image)

*Figure 2.7*  
Application plan and packages.
Distributed data

Many businesses need to manage data from a wide variety of sources and locations. A distributed environment provides the flexibility that is required to allocate resources for data located at different sites, or database management systems (DBMSs), in a computer network.

*Distributed data* is data that resides on a DBMS other than your local system. Your *local* DBMS is the one on which you bind your application plan. All other DBMSs are *remote*.

Remote servers

When you request services from a remote DBMS, the remote DBMS is a *server* and your local system is a *requester* or *client*. Conceptually, a server is like a food server at a restaurant who takes food orders, delivers food, and provides other services to customers. The customer is like the requester, or client. The server’s purpose is to provide services to its clients.

A remote server can be truly remote in the physical sense—thousands of miles away—or a remote server can be part of the same operating system under which your local DBMS runs. This book generally assumes that your local DBMS is an instance of DB2 for OS/390. A remote server can be another instance of DB2 for OS/390 also, or an instance of one of many other products.

Figure 2.8 shows the client/server environment.

Connectivity

Connectivity in the client/server environment requires an architecture that can handle the stringent performance requirements of a transaction-based system or the flexibility of a decision-support-based system using ODBC or JDBC. The primary method that DB2 uses to provide connectivity to any number of DBMSs is Distributed Relational Database Architecture (DRDA), which is based on The Open Group technical standard. As “Distributing data and providing Web access” on page 7 explains,
DRDA is an open, published architecture that enables communication between applications and database systems on disparate operating systems.

Using standard communication protocols, DB2 can bind and rebind packages at other servers and run the statements in those packages. Communication protocols are rules for managing the flow of data across a computer network just as traffic lights and traffic rules manage the flow of car traffic. These protocols are invisible to DB2 applications. A system that uses DRDA, for example, can invoke DB2’s stored procedures or request that SQL statements run at any server that complies with the DRDA standard.

In a distributed environment, applications can connect to multiple databases on different servers and complete transactions, including commit and rollback operations, at the same time. This type of connectivity is known as a distributed unit of work.