CHAPTER 1

Overview of Databases

1.1 WHAT'S A "DATABASE"?

Today, generally everyone uses a computer in one form or another.

- Home-based computers are frequently used for managing a personal business, update spreadsheets, or complete school assignments. Others use them for email, social interaction with friends and family members, monitoring the Internet for news, or for entertainment.
- Owners of small businesses use spreadsheets and/or software products such as QuickBooks to keep track of personal or business expenses.
- Office environments must gather and store and manage information for a wide range of topics or subjects, such as customers or clients, appointments, or customer orders.
- Business environments must manage a much wider scope of data regarding the information and data needed to run or manage the business.
- Users using computers in government offices need computers to manage their jobs. For those working as analysts in the Department of Defense (DOD) or in the Intelligence Community, the *nature of the job* is continually expanding, requiring analysts to monitor or track new information or data as it becomes available. Analytical teams continually face the responsibility of analyzing new and evolving forms of information to identify and extract information of relevance using software tools available to them. Often, that means having not much more than the desktop Microsoft Office products ranging from Excel to Microsoft Access.

As the data needed by the user or customer community grow in size, complexity, and importance, the *care and feeding* of that data requires the use of a database management system (DBMS) to store, manage, and protect it.

A DBMS[1] is a special software package that is designed to define and manage data within one or more databases. Individual databases, in turn, manage the definition of data objects/tables in a given subject area and provide controlled user access to that data.

Examples of DBMSs include Structured Query Language (SQL) Server, Oracle, and Microsoft Access. An SQL Server or Oracle instance would then serve as host to, for example, a personnel database.

1.2 GUARANTEED ACCURACY AND AVAILABILITY OF DATA

A DBMS is, by its very nature, built to guarantee the accuracy and availability of data as updates occur. Updates are bundled as application *transactions*[2] that apply all data updates within a logical *unit of work*[3] associated with that application. These updates must be made on an *all or nothing* basis; either *all* the updates are applied, or, if a logical or database error occurs, *none* of the updates are applied, leaving all of the data in a clean consistent state from the user and application perspective.

The application software updating the database issues commands to the database to start a unit of work. If all updates complete successfully, a *commit* call is issued to make those updates permanent. If in the process of making those updates some condition is found that prevents the update from occurring, a *rollback* call is made to reverse any updates and put the data back in a logical state representing the data at the beginning of the transaction.

For example, a user might log on to their banking system and start an update to move funds from their savings to checking accounts.

- After logging in and starting the transfer, the software performing the updates first issues a database update to debit the savings account for the specified amount.
- If that update is successful, it issues an update to credit the checking account by that amount.
- Upon successful completion, a *commit* call is issued to commit the changes and release database locks on the rows being updated. An appropriate message would be sent to the user confirming that the funds transfer was completed.
- If, however, the update to the checking account failed (e.g., the user entered the wrong savings account number), a *rollback* call would be made to reverse all updates made, and an appropriate error message would be sent to the user. As a result, the database and the underlying data are left in a clean, consistent state.

The ACID[4] properties (atomicity, consistency, isolation, and durability) of database systems and transactions guarantee the accuracy and availability of data.

1.2.1 Atomicity

The *atomicity* is the *all or nothing* requirement when making updates. Either all updates made during the unit or work succeed or no updates are made. This protection includes updates in a unit of work or transaction, device input/output errors, network errors, and power failures.

1.2.2 Consistency

Consistency requires that transactions take the database from one valid state to another. Any and all updates must conform and enforce any referential integrity[5] constraints defined. (Referential integrity constraints define and control any *one-to-many* relationships between tables in a database.)

1.2.3 Isolation

Isolation of database updates involves mechanisms that enable multiple concurrent users to simultaneously access and update the same data elements within a database.

As database updates occur, locks are transparently placed on updated rows that prevent subsequent users to access or update those rows until the updating process commits those updates and the locks are released. Any processes requesting access to rows being updated are *held*/delayed until the updater's commit point is made.

1.2.4 Durability

This feature/requirement ensures that any updates made by a transaction (i.e., a unit of work completed and updates committed) will survive a subsequent system error or problem, for example, a system failure or a power or disk failure.

Database systems have mechanisms/features that support a full database backup. In addition, database systems log updates to nonvolatile devices (a database log file) as updates are made to the database. If/When necessary, a database can be rebuilt/recovered totally by first using the database backup to recover all data to the point the backup was made, then using the database log to reapply all updates made to the database after that point in time. This subject is covered in more detail in Section 1.6.

1.3 DYNAMIC ALTERATION OF DESIGN

Relational database management system (RDBMS) represent the *third generation* of DBMS products. As one of their key features, these products give the user the ability to dynamically add or drop columns to data or make other changes *live* while the database is online and being updated by users. That provides a significant change over the second-generation *hierarchical* systems that had to be taken down and modified off-line to apply changes. Third-generation systems include products such as SQL Server, Oracle, and Microsoft Access.

Note that MySQL is *touted* as an RDBMS and it has many relational-like features. However, it has significant limitations that, in my opinion, prevent it from being classified as a true RDBMS.

> For example, each table in MySQL is implemented as a *flat file* with indexes as needed to support data retrieval. If/When any changes are required, for example, a column is to be added, MySQL creates a new temporary table with the new column, copies all records from the original file to the new, and then deletes and renames the old and new files accordingly.

> In a former role, I prototyped a MySQL implementation for a data collection application running a UNIX (Solaris) server. As the prototype progressed, it was no surprise to find that I needed to add new columns to the MySQL table to help track information about what was being collected. I found that the time requirements to make changes to a MySQL table with a million rows were *anything* but *transparent*.
>
> As a work around, I then made what I hoped was a one-time modification to the table adding *spare* columns (Spare1, Spare2, Spare3, etc.) with the plan of renaming these columns if/when needed to reflect application-specific, meaningful names. That helped, but even then I found that MySQL required/used too much overhead for managing large tables.

The ability to dynamically change table definitions can, in most products, be made using that product's database administrator (DBA) graphical user interface, or by working at the command line by issuing commands using the product's data definition language (DDL). The DBA user interface is much easier and quicker to use, but when supporting mission-critical applications, change management procedures are used to control updates across multiple environments and platforms, each with their own copy and version of the application database.

- A Development platform is used to design, develop, and test individual software components and tables within a database.
- Incremental changes are made by manually running DDL changes at the command prompt.
- All incremental changes are accumulated as they are applied, creating a change package with all modifications needed for that release.
- When all changes have been made and tested for a software release, a Test platform is used.
- With the test system database configured for the older software release, the change package is applied and the software release is tested to ensure all updates have been correctly applied and the software works as intended.
- If errors are found, the change package must be revised as necessary and the entire update process repeated.
- After changes have been successfully applied and tested on the Test system, the change package is ready to be applied to the Production platform.

The careful application and use of change packages on the Test platform allows the scheduling of downtime of the Production system with the expectation of *no surprises* when the updates are applied.

1.4 DYNAMIC QUERIES—ANY DATA, ANY TIME

Although RDBMS products made a huge improvement by allowing dynamic changes, an even more significant enhancement was through the implementation of the SQL language. This gives the user the ability to ask for any data, at any time. For example, any authorized user can connect to the database and, using one of several query managers typically available, issue a command to retrieve data using the command syntax, where {} indicates optional parameters:

```
"SELECT <column name(s)> FROM <table name> {WHERE <column name> =
<value>};"
```

For example,

```
"SELECT First_Name, Last_Name FROM EMPLOYEE WHERE DEPT-NUMBER =
12;"
```

would retrieve the first and last names from the Employee table where that table's DEPT-NUMBER column has a value of 12.

SQL has, of course, many variations to extend the power and flexibility of the commands issued. Some simpler examples are

```
"SELECT * FROM EMPLOYEE;"
```

would retrieve/display all columns in the Employee table;

```
"SELECT COUNT(*) FROM EMPLOYEE;"
```

would display the number of Employees in the Employee table; and

```
"SELECT MAX(Annual_Salary) FROM EMPLOYEE;"
```

would display the highest annual salary found in the Employee table.

The power of SQL is multiplied by creating a "View"[6] of two or more tables that create a *virtual* object to query against. For example, a View can be created named "Department_Employees" that combines the Department table with the Employee table matching the DepartmentID column in Department with the DepartmentID in Employee.

Using this *virtual table*,

```
"SELECT * FROM DEPARTMENT_EMPLOYEES WHERE DepartmentID = 12;"
```

will list all information for employees that are assigned to Department 12.

Developers and users, however, must be aware that SQL implementations *are not all equal*.[7] RDBMS vendors have worked together over the years to define and implement SQL within their products beginning in 1986. However, at the implementation level, SQL

commands will vary depending on the RDBMS product being used. The major vendors support the basic SQL standards (i.e., their product will provide some specific functionality), but the details of implementation will be different. For example, SQL provides for a *wildcard* character to allow/support either single character matches or to match any number of characters in a search string. If using SQL Server or Oracle:

```
"SELECT * FROM PRODUCTS WHERE Product_Name LIKE 'DELL%';"
```

will display all information from the *products* table for product names begin with "DELL."

If using Microsoft Access, this command would be

```
"SELECT * FROM PRODUCTS WHERE Product_Name LIKE 'DELL*';"
```

to match the wild card character used in Access.

In addition, each vendor will include their own SQL *enhancements* (features going beyond the standard) to support their products' competitive advantage.

Fortunately, from the developer's point of view, 90%–95% of their SQL *skills* will transfer/apply when working with another RDBMS product.

1.5 REFERENTIAL INTEGRITY ENFORCEMENT

A database will contain multiple tables, each containing information about one type of information to be stored in the database. For example, an employee database might have a table for department, one for employee, and another for employee-deduction. Each table contains detailed information about that particular item/object, each containing at a minimum the following information:

Department

- DepartmentID
- DepartmentName
- DepartmentManagerID

Employee

- EmployeeID
- EmployeeFirstName
- EmployeeMiddleName
- EmployeeLastName
- EmployeeWorkPhoneNumber
- EmployeeHomePhoneNumber

EmployeeStreetAddress

EmployeeCity

EmployeeState

EmployeeZipCode

DepartmentID

Employee-Deduction

EmployeeID

DeductionCode

DeductionAmount

When the tables are loaded, information for each occurrence is loaded as a row in the respective table and each data element/value is associated with its respective column in the table. For example, a row in the department table might have the following content:

DepartmentID	DepartmentName	DepartmentManagerID
SalesArea1	Northern Virginia Marketing	E1005

Looking at the employee table, the first row for an employee assigned to the above-mentioned department might appear as follows:

EmployeeID	EmployeeFirstName	EmployeeMiddleName	EmployeeLastName	…	DepartmentID
E2001	Albert	Alan	Smith		SalesArea1

Note that the relationship between the Department row for the Northern Virginia Marketing unit and the Employee row for Albert Smith is determined by the fact that the DepartmentID, "SalesArea1," is stored as the value for the DepartmentID column in the Employee table. This illustrates the *one-to-many* relationship between Department and Employee and is referred to as Referential Integrity.

Referential Integrity is fundamental to data integrity and is normally activated and enforced by the RDBMS as data are inserted or deleted from individual tables. For example

- Any DepartmentID value assigned to a new employee must match a DepartmentID value in the Department table.
- Any new Employee-Deduction rows must contain a valid/matching EmployeeID.
- More importantly, if/when an employee is terminated, all Employee-Deduction rows will be automatically deleted without the programmer having to remember to perform that function.