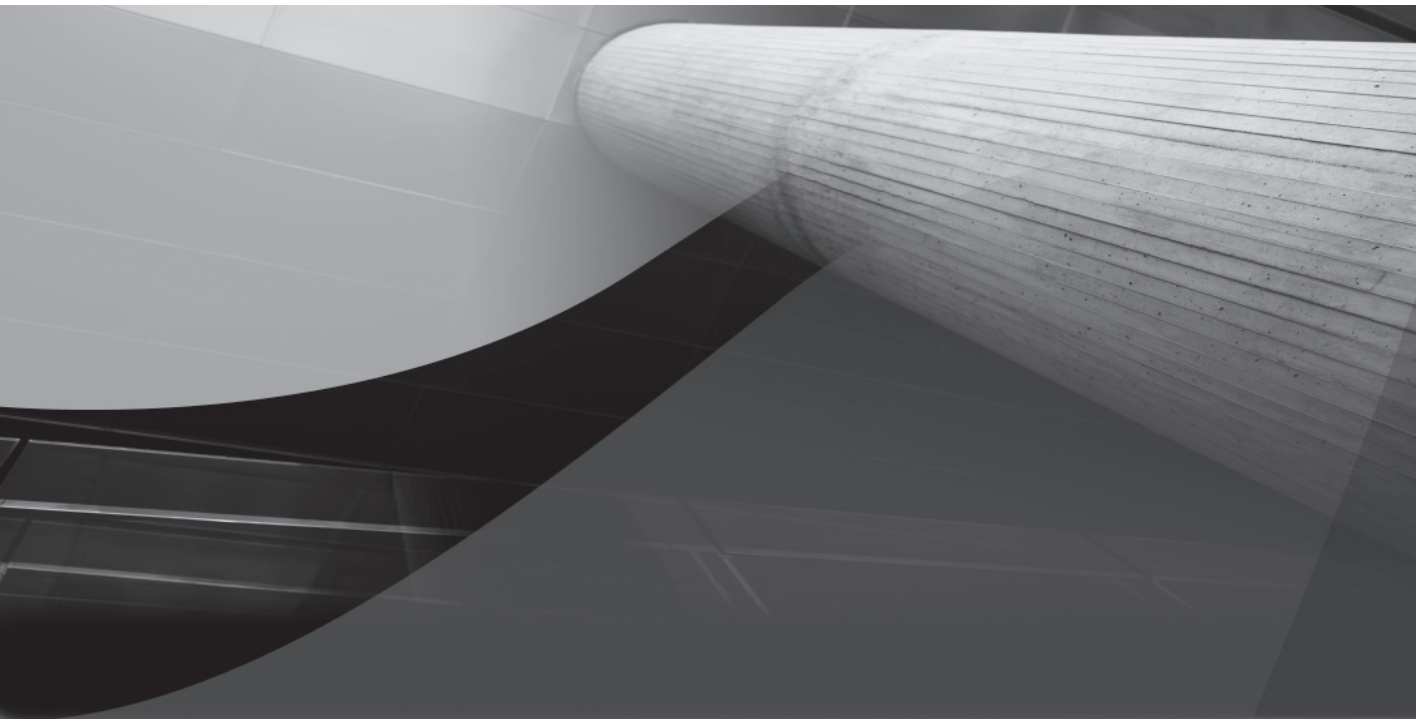


# PART I

## Oracle Database Security New Features





# CHAPTER 1

## Security Blueprints and New Thinking



Computer security is a field of study that continues to undergo significant changes at an extremely fast pace. As a result of research combined with increases in computing capacity, computer security has reached what many consider to be “early adulthood.” From advances in encryption and encryption devices to identity management and enterprise auditing, the computer security field is as vast and complex as it is sophisticated and powerful.

Database and application security form one end of the computer security field. These two areas are closely aligned because of the heavy and obvious relationship between applications and databases. Along with other areas of security, database and application security continues to evolve rapidly. Creating a sound and secure (database) application is challenging not just because it can be complex, but more so because of the many possible methods that can be used to accomplish it.

While working with the many customers of Oracle Corporation, the authors have noticed two important things with respect to application and database security. First, in the architecture and design phases, complexity exists due to the technology and the various options available to reach an optimal level of security. Second, in the development and deployment phases, most people don’t employ full and complete security implementations. Their reasons for this vary, but their obvious lack of knowledge and lack of a set of reference architectures from which to draw upon are key. This not only limits what can be crafted as a viable solution but can also have disastrous effects in cost overruns, project delays, inadequate results, and vulnerable systems. They have not mitigated the risks to an acceptable level.

## About This Book

In this book, we have brought together top experts in business intelligence (BI), application development, identity management, and the Oracle Database. We’ll show you how to use application security features, the application development environment, and the database itself to create secure database applications. As such, our focus is not simply on studying security features in a stand-alone context. Rather, our intent is to apply security technologies to the art of creating database applications.

We present several patterns for successful security implementations. Our objective is to provide successful security patterns, tips, and tricks that will help you understand what you should do to engineer a secure database application. We identify specific patterns that represent the predominant styles used by application-database interactions and then show you how to engage the features and capabilities in the database, the application server, the identity management platform, and the application itself to build secure and robust applications.

## Background Information

This book is purposefully designed as a follow-up to *Effective Oracle Database 10g Security By Design*, published by McGraw-Hill in 2004. In that book, author David Knox, who serves as lead author for this book, takes you from the basics of database security practices and designs to some advanced examples. Included are all the technologies available for the Oracle Database 10g Release 1, including secure application roles, virtual private database, Oracle label security, enterprise user security, proxy authentication, auditing, and database encryption using DBMS\_CRYPTO. If you are unfamiliar with those technologies or need a better understanding of them, you should read the first book before you read this one.

While an update to that text that added the new options—transparent data encryption, database vault, and audit vault—was certainly a possibility, we decided to create a new book for

two main reasons: First, identity and access management advances were needed to form any significant and complete description of security. Second, the goal of this book is to show how you can apply those base database technologies to secure database applications such as the business intelligence and Application Express (APEX) applications. Adding the new options, the identity management sections, and the application examples would have made the book too large for practical purposes.

In summary, the first book explains the basics of Oracle Database 10g R1 to help the reader understand what the available security technologies can do, and then it shows how to get them working. This book adds information about new technologies and applies all the security technologies to the task of building secure database applications.

Every attempt has been made to abstain from redundancy with material presented in the first book. Since we did not want to force the reader to refer back to that book too much and too often, parts of that text have been borrowed and placed in the appropriate places to enhance discussions. Our goal was to make this a complete and stand-alone guide that was not too voluminous.

## Organization

The book starts with a discussion of the new technologies that were added since the completion of the first book, such as transparent data encryption, audit vault, and database vault. From there, we move out, first to the application server infrastructure and identity and access management. This information is important to understanding what other things an application developer and security administrator need to know when constructing security-conscious applications. We next move out to APEX and finally the Oracle Business Intelligence Suite (OBI).

APEX is a popular platform for developing light-to-medium-weight Oracle Database applications. Its popularity is driven by the fact that it is free to use and is tightly integrated with the database, thereby allowing any DB-knowledgeable person to create powerful and elegant database applications. This is a sweet spot, as many database gurus are not too familiar with other development platforms.

APEX also represents a popular application architecture. It deploys to the Web, and the application developers and users can connect to the database using shared and proxy schemas. APEX also manages much of the application-level security itself. Both of these aspects make APEX a prime case study and valuable aid in understanding how to work with those types of architectures.

OBI is popular and represents a standard way that people interact with Oracle Database. Learning the integration and synchronization points between the BI server and the Oracle Database security technologies proves a valuable, and more importantly, a repeatable lesson in securely connecting applications to the database.

In Chapter 1 we explain our top motivation for writing this book—namely, technology changes have been made to match developer and user behaviors for building and using secure database applications. This is a major theme behind the new technologies discussed.

The discussion then moves to the primary drivers for security. These security motivators are important because they imply what needs to be done to “secure” your database applications. More importantly, they identify the design and business goals that need to be satisfied to ensure that the applications meet an acceptable level of security standards. Put another way, the motivators help you define your business and technical targets. If you cannot reach specific goals, you cannot determine whether you have achieved success. You need to be able to answer the question, Is it secure enough? To do this, you must understand what you need to accomplish and why. Your job is to ensure that the applications and security implementations are aimed at the proper targets and thereby satisfy your business and technical goals.

The security motivators give way to four principles for implementing security correctly. The products, technologies, and blueprints discussed throughout the book are aligned with these principles. The chapter concludes with concrete examples of the architectural evolution of database security. In analyzing applications over the years, we have noted three main blueprints or design patterns that people have employed to varying degrees of success. We examine those patterns and analyze the pros and cons. This serves as a basis for future chapters, where we will build and connect various database applications. You will see the common architectures and best practices for each.

## Database Security Today

Database security has changed radically over the years. In some ways, it has outpaced the growth of the general security market. The creation of record-level access via transparent query modifications—aka *virtual private database*—and the ability to perform conditional auditing—aka *fine-grained auditing*—are two examples of these changes. However, there is another side to this discussion, because we have to recognize that many of the design patterns and the Oracle products and technologies are focused on an era about 15 years ago—the mainframe and client-server days.

To achieve the security designs required for today’s environment, you must understand not only how things work but also why they work. The intent of a product or technology is the first clue in understanding its usefulness and applicability to your current projects. This book applies (security) technologies to the problems and architectures used today. Before we get into that, though, you need to understand how technology has evolved.

## Evolving Technologies

Simple technology models have always been used to explain complex systems, and a model can be used to explain security as well. Security can be described as an understanding of *who* gets access to *what*, from *where*, *when*, and *how*. Physical security and digital security are largely focused on the ability to control all aspects of people and things interacting with people and things.

With this in mind, consider how security has evolved as technology has evolved. Security implications center around two things: user identification for auditing and accountability purposes, and access controls to allow or prevent users from performing specific actions or accessing specific data. In sequence, we tend to think of the security process as identification and authentication—who (authorization and access controls) gets access to what, from where and when; and how (via auditing). Let’s translate this into how Oracle technology has evolved over time.

In the early years, much of Oracle’s security was based on the concept of a database user, in which the user logs in directly to the database and has a private, dedicated account. As you know, in an Oracle database, a user and a schema are considered one in the same. The reasons for this are numerous, but for the security architect, this can pose a few problems. Access controls and auditing in the Oracle database are optimized for the notion of users connecting directly to database schemas. The problem is, however, that building an application today is different from what it was when many of the baseline security models were designed.

While appropriate at the time, direct database logins have given way to connection pools and middle tier applications running on application servers that in many ways break the direct association of an end user and a database user. Furthermore, as you will see, a significant manageability challenge exists in creating and managing individual database accounts for individual application users who just happen to be accessing an application that uses the database.

### Proxy Authentication Addresses Secure, Fast DB Connections

Oracle has slowly acknowledged this pattern over the years, and it has changed its technology as a result. *Proxy authentication* allowed developers to use connection pools and start lightweight database sessions for end users. This solved the challenge of quickly connecting many users to a database.

A problem still persisted, however, because the proxy call required the user to start a session with an existing database account, meaning that while proxy authentication solved a performance challenge presented by the connection pool architecture (and it did so securely by not requiring the connection to maintain the end user's password), it did not address user administration and management issues. That was addressed by enterprise user security, and proxy authentication supported that architecture as well. Manageability is an important principle in achieving an effective security implementation.

### Enterprise User Security Addresses Manageability

To adapt the technology to meet the challenges of managing large-scale user populations, Oracle created *Enterprise Users*, a major step forward. The end users (or application users) are managed in a central Lightweight Directory Access Protocol (LDAP) repository. The directory includes an entry for each user that maps the user to a shared database schema. Also included in the user's entry are *role mappings*. This effectively gives us centralized administration and, to some extent, centralized authorizations.

The ability to grant and manage authorizations was not totally complete, however, as database roles in Oracle Procedural Language/Structured Query Language (PL/SQL), or definer's rights, procedures were disabled. Definer's rights procedures still provide the default model and is the prominent mechanism, if not the most popular one, for granting user privileges. As the roles were disabled from within any executing (definer's rights) procedure, any privileges assigned to the roles were unavailable, thereby rendering the roles useless, a less-than-optimal solution for privilege management. While invoker's rights procedures remedied this problem, many applications did not employ them and many architectures today have failed to incorporate them. Nevertheless, centralized user management resolves many of the user manageability challenges and has made Enterprise User Security a very useful architecture.

With enterprise users, identity preservation occurred, but just barely. *Identity preservation* means that you are able to preserve the identity of the end user from source application all the way to the database. You can then implement security and auditing controls on a user level. There are two problems with this, however. First, the DB security and auditing work at a schema level—that is, the identity of the user—is presumed to be the schema. Security is all too often implemented with some reliance on a `SELECT USER FROM DUAL`. You could, however, write your own fine-grained security via virtual private database, oracle label security, encryption, views, triggers and so forth. (This sentence probably should not say *could write*; it should say *must write*. Otherwise, you have no way of applying different security enforcements for users sharing the same database schema.)

The second issue—and perhaps the more important issue—concerning identity preservation is that security architectures of tomorrow, which to some extent we are already seeing, do not use end user identity as the sole mechanism for access enforcement. These security and control mechanisms will be based on many factors, of which the user's identity may or may not be a relevant piece. End user identity by itself is useful mostly for auditing purposes and not so much for security and access controls.

Security and access controls today and tomorrow will largely be based on authorization models that use roles, group memberships, and data attributes. This is because the users in many situations are unknown not only to the database, but sometimes even to the application! Therefore, no user entry will exist in an application's USERS table (for example), nor will an entry exist in the Oracle Database USER\$ table and perhaps not even in the local LDAP directory. With no user entry, you have no access control list (ACL). Without a way to capture users ahead of time, identity is meaningless for security enforcement purposes.

To help you understand a bit better, consider an example that consists of a Web services architecture that federates or makes many calls as part of a single business transaction. Each of these services may be on separate servers, using separate databases, providing separate information for separate and potentially shared purposes. As such, the ability to execute a multi-call transaction requires some way of conveying to all the services, and subsequently the programs that access the databases, that the user is authorized or unauthorized to perform an action. Ideally, this model needs to support an infinite number of users and needs to be as adaptable as the standards on which it relies. The actual user identities will therefore not be stored locally. Authorization information and other aspects about the data and how the information is being accessed or used will be employed to ensure that proper access is controlled. User identities, if properly propagated, will be used only for auditing and accountability reasons.

Hopefully, you now are starting to see the new thinking of today. These highly distributed architectures supporting vast numbers of unknown users are forcing radical changes in architectures and implementations. In addition, another paradigm shift concerns how you address security concerns: What are these concerns? How do you know if your data is secure? What are you protecting and why? You can address all these questions by looking at what motivates the security end of businesses today.

## Security Motivators

It used to be that to sell security, you had to *sell* security. That is, other than a few exceptions for a few customers, security was considered a nice-to-have luxury that might be considered at the end of a development cycle when and if enough money and time remained. In fact, most data was not considered sensitive and therefore not worth the effort to secure.

Many people used fear, uncertainty, and doubt (FUD) as primary tools to motivate people to adopt a good security posture. Statistics of insider attacks, network packet sniffing, and computer security hacks could create a level of anxiety sufficient for people to take proactive actions to protect their data. Many times, however, the statistics were insufficient in motivating anyone to do anything. The threat did not seem viable or the examples were irrelevant to the business of the day for that specific organization.

Today's world has radically changed its temperament on security—what it means and how to do it. With service oriented architecture (SOA) representing today's major infrastructure thinking, and business intelligence, collaborative, and social Web 2.0 technologies representing higher order thinking, maintaining a security environment to protect people and assets is just not at the top of the list from an IT viewpoint. But security can be viewed another way, and this has everything to do with the sensitivity of data, the pervasiveness and exposure of data, and the consequences for not sufficiently protecting the data.

Many people now believe that security is more important than ever for two major reasons. First, the primary drivers have shifted, from acting out of fear of direct data theft of hard-to-find and hard-to-obtain information to complying with government and regulatory policies set up to

protect information that at one time was not considered sensitive but today is considered highly sensitive. Personally identifiable information is a prime example of such data. The explosion of information and information use has increased the need for security.

The second reason for an increase in the importance of security centers around the results and negative impacts that a compromise or data breach can have on an organization, its reputation, the future employability of those considered accountable, and the always motivating financial penalties and threat of incarceration. A lot of data needs to be protected. This data is shared, transmitted, transferred, analyzed, and integrated, and each action represents an increased risk of compromise. With compromise comes demise. With corporate brands and public perception influencing stock prices and future viability, security is indeed more important now than ever.

Let's explore a little more deeply to clarify this sensitive information and consider how we should protect it.

## **Sensitive Data Categorization**

To satisfy a security requirement properly, you must identify what you are protecting, from whom, and why. By understanding the characteristics and use of the data, you can then understand its vulnerabilities and subsequently derive a plan to protect it.

At a high level, many organizations today break up their data into four top-level categories:

- Personally identifiable information
- Protected health information
- Intellectual property
- Data within the realms of governance

### **Personally Identifiable Information**

The first category, personally identifiable information (PII), includes any information that can be used to obtain or create a false identity. It includes names, addresses, Social Security numbers, and other private information that can be used for nefarious purposes as a way to spoof or pretend to be someone else. The alarming thing about PII is that it is data about people—not just special people such as celebrities and politicians; it is data about essentially everyone. Furthermore, this data is used many times a day to perform the most mundane tasks, such as paying bills, registering for a license, applying for a loan, and applying for a job. These are just a few examples of where highly sensitive personal information can be found.

Identity theft is a growing concern, and organizations are struggling with ways to protect the identities of their customers, employees, and partners. Fortunately, some best practices for how to protect PII are developing, and these will be discussed in later chapters.

### **Protected Health Information**

Protected health information (PHI) is privacy information that deals with a person's health or medical conditions. It is more formerly described and governed in the US Health Insurance Portability and Accountability Act (HIPAA).

PHI pertains not just to healthcare providers (such as hospitals) and healthcare payers (such as health insurance companies). Many organizations collect some PHI, and this data is scattered throughout their IT systems in employee benefit submissions, paid time off, disability insurance, and so forth.

The challenge here is to employ the correct amounts of security to protect the individuals' privacy without hampering the general business flows necessary to operate an organization efficiently. Authorized persons must be able to perform authorized tasks easily, and unauthorized persons should not be able to perform unauthorized tasks easily. The issue regarding ease of use has to do once again with human behavior and incorporating security controls in a transparent or unobtrusive way. This is particularly important in day-to-day tasks and applications that support such things such as HR applications and customer relationship management (CRM) applications.

### **Intellectual Property**

Safe guarding proprietary information in a growing and global economy is more important today than it has ever been. Trade secrets, business transactions, and merger and acquisition strategies are among the top information categories that organizations are struggling to secure.

Often, organizations will (and should) create classifications around the data. Federal governments use classification schemes all the time to help control their information. Confidentiality markings are used to dictate the handling procedures of the data. It may be that some information is not permitted to leave the company, or information can be shared only with a partner who has signed a nondisclosure agreement, and so forth.

What makes this information difficult to secure is that large amounts of information in many forms is distributed to many people for many reasons. As it flows through different media to different people, it becomes increasingly more difficult to control and thus secure. In this book, we will keep our discussions simple while maintaining the point that this category of information is as important as it is large—it concerns lots of data and lots of people.

### **Governance and Regulations**

The biggest challenge that has everyone's attention centers on financial statements in regards to the Sarbanes Oxley (SOX) Act of 2002. As a result of several disastrous events, the legislation holds companies accountable for the precision and authenticity of their financial statements. This is by no means a legal definition, as one is not required here, but suffice it to say that legal penalties can be levied for noncompliance. However, the results of illegal disclosure are often more important outside the legal domain. A publicly traded company's reputation, branding, and public image are constantly at stake. A negative event can have disastrous effects on stock prices and the future viability of a company.

The challenge for this scenario arises from several factors, among them are ensuring the timeliness of reporting the financial information, controlling access to the information, and naturally trying to guarantee that the information is as accurate. Once again, this involves a must-have need to manage who gets access to the internal data and under what conditions.

### **Principles**

Regardless of the classification of data and the need to protect it, you can adhere to a few principles when considering a solution to a business problem and contemplating the correct level and appropriateness of security. These principles are adaptations and evolutions of those cited in *Effective Oracle Database 10g Security By Design*. These principles should serve as guidelines to help you drive decisions and effective postures and are echoed throughout the book. Understanding them and why they are important is essential to your understanding the complementary technologies, architectures, and best practices presented herein.

Principles can serve to prove due diligence or, in some cases, negligence. Incorporating security is a delicate balance of preserving ease of use, performance, and manageability. As these factors often compete with security, it is important that you are able to justify and implement a

balanced, prudent, and rational solution. Discounting security entirely is rarely, if ever, an option. The point is for you to be able to prove that the correct level of security is being implemented in the correct way. Doing so may assist you in preserving company brand, reputation, and viability and also in protecting your reputation and employability.

### **Layers of Security**

You probably know that dressing in layers of clothing is a prudent approach to staying warm in places where the weather can change quickly and dramatically. Security in layers has an analogous benefit. A removal of one layer—say, by compromise—does not expose the entire system. When you look at how to apply security, you want to look at incorporating multiple layers whenever it makes sense to do so. There is no such thing as being too secure (which should not be confused with too cumbersome a security implementation to be usable).

Of all the things that compete with a security implementation, other security implementations should not be one of them. In fact, quite the opposite is true. The more, the better is the suggestion as long as it does not present a significant cost in money, time, labor, effort, or performance. While that may seem an impossible task, it is not. New technologies such as Transparent Data Encryption (TDE) can add a layer of security, thereby increasing security and keeping coding to a minimum, along with minimal costs, effort, and performance degradation.

Another best practice is to apply a security layer as close to the data as possible. This is an optimization technique in that it allows you to get the biggest return for the effort. If the data can be secured in the database, then do it there. This will ensure that a data security layer is available to anyone accessing the data. Adding security at the middle tier (application server) and within the application itself are the complementary steps advocated in a best practices doctrine.

### **Manageable Security**

As alluded to already, being able to set up, develop, deploy, control, and adapt security is critical to any successful security strategy. You may initially look at these desirable qualities and decide that they are impossible to realize. But you will find ways to move toward effective security management, and you will ultimately achieve a better solution.

In fact, common ways for achieving manageable security have already been realized. Centralization of identities and authorizations—aka identity management—uses this very principle as its selling point. Centralization of security is in fact a major facilitator in managing security efficiently. The problem is that you cannot always consolidate everything into a single, centralized environment. You will see how technology has adapted to this reality and ways in which you can get the benefits of single control with the reality of distributed and autonomous applications and enforcements.

### **Business Congruency**

The next principle to achieving security success is in aligning it with existing business policies and technology architectures. If the business needs analytical tools to investigate the deep meaning and relationships of its data, then the security needs to align with how those tools function and how the users are accustomed to using the application. You will see this incorporated into BI applications.

Another example of this comes from the predominant architecture of the day—SOA. Although this book is not about SOA, we will simply say that the goodness that SOA represents lies in its inherent ability to allow anyone to link into any service, from anywhere, at any time. The security challenge lies in protecting anyone from linking to any service, from anywhere, at any time. The point here is that SOA exists and will continue to exist, and to work securely in it requires an implementation that is congruent with how this architecture is basically employed. You'll see that

you can leverage the identity management components and implement techniques that are aligned with many SOA designs.

### Transparency

You already know that changing behavior is more difficult than adapting technology. A simple way, then, to employ new technology, and specifically security technology, is to make it transparent. Barring any substantial issues in manageability and performance, transparency will ensure a successful implementation. It practically eliminates the usability issues often associated with an unsuccessful security implementation. Transparency may allow users to continue to behave as always, without your needing to give up the ability to insert enhanced security controls and auditing.

Throughout this book, you will read how many of the technologies have incorporated these principles in their design, creation, and implementation. You should take these concepts and apply them to your thinking, designs, and implementations.

## Modeling Secure Schemas

There are many aspects to creating a secure database application. Here we will cover a “jugular” topic that at first seems basic, but is in fact a bit thought-provoking. Our goal here is to answer what seems to be a simple question: When building and deploying an application, which schema do users connect to in the database? As you will see, the answer to this question has the greatest security implications on your overall design.

Two important points need to be made here. First, by understanding these models now, you will save development time in the future, as you will not be agonizing over what is the proper way to set up your application-database connections. If you don’t agonize over such decisions today, employing the logic presented here will allow you to create more secure applications—because we have already agonized over the matter.

The second objective is to address the case that you are not architecting a new application but rather installing, maintaining, or securing some other application—that is, a third-party or architected application developed by someone else. The objective here is to provide you with the information you need to ascertain the correct risks in such designs and provide initial guidance on what you might do better to fortify the design.

In later chapters, we will use the information presented here to create or modify our schema models. Defining these models now as a reference will provide consistency for the remaining parts of this book. It will also provide a meaningful baseline to your future design and development endeavors.

## Schema Profiles

As mentioned earlier, the Oracle Database does not differentiate between a user and a schema. That is somewhat disappointing, because those two things are remarkably different, especially when security is concerned. In this section, you’ll see that this is actually somewhat trivial to overcome once you develop a reference methodology for thinking about your application and database design.

Within the database, you can think of the database users or schemas as serving some distinct purpose. For starters, you can generally separate users from database objects. User accounts are the schemas in which end users connect to the database. Schemas then can be loosely defined as a collection of data, objects, procedures, and so on.

Note that we are not suggesting that only two schemas are involved, only that it is logical to break them into these two broad categories. This argument is based on the design practice that says it's best to separate accounts by purpose and function. This not only simplifies the design logically and intuitively, but it also improves security. It is a basic database design best practice that rationalizes the overall database architecture and simplifies many of the database operations such as backups, version control, and patching.

The security angle to separate database accounts is based on the well-known and well-practiced security tenet that says you should always maintain *least privileges*. This tenet was well described in *Effective Oracle Database 10g Security By Design*, which summarized least privileges as the following: “Least privileges means you give people only the permissions they need to do their job and nothing more: you give them the least amount of privileges.” One of the easiest ways to compromise a system is to exploit accounts that have been granted too many privileges.

Using this as your driving factor, you can start to categorize your database accounts into one of the following two:

- **Object owner accounts** The schemas that hold application code, logic, and data structures
- **User access accounts** The schemas to which users connect when they execute application code and query and update application data

Note that for user access accounts, a direct correlation exists between user, function, and security privileges. Users and thus schemas perform some specific function, and each function requires privileges. For simplicity, you can group users/schemas by functional role, which will naturally require a set of security privileges.

## Object Owner Accounts

Object owner accounts are the accounts that you typically think of when you think of database *schemas*. The accounts serve as a container for execution code and other database objects such as tables, views, indexes, triggers, and so forth, that are part of an application or database option. The default database accounts that end in “SYS”, which includes the schema SYS, are examples of object owner accounts. These accounts are often synonymous with a database option. For example, CTXSYS is the schema for the (Con)Text database option, MDSYS is the schema for the Spatial data option, and SYS is the schema for the database engine itself.

Putting the objects together makes certain tasks such as patching and backup and recovery activities easier to do. It also simplifies some code writing, as object resolution, object creation, and object access can be accomplished without requiring any further privileges—meaning that if you can create a table in your account, then, barring any fine-grained access controls, you can also read from it, write to it, and drop it.

The first practice, then, is to identify and separate the object owner accounts into meaningful yet distinct schemas. Depending on the size and complexity of the application, it may have from one schema to many, many schemas.

The difficult part is deciding at what level to break apart the code and objects into other (related) schemas. At first, keeping everything in a single schema might appear to be the simplest thing to do. However, you might later find this suboptimal for supporting many tasks such as patching, backups, administration, and, for our purposes, providing access and security.

For a single application, for example, it's not uncommon to see base tables in one schema, code in another schema, and metadata or summary data in a third schema. Isolating these might allow procedural code updates to the code schema to be done without significant (if any) impact on the other schemas. Likewise, building new summary data structures would have little impact on the procedural and data schemas. And, finally, backups may be more frequently done on the data schema as the data probably changes more often than do the procedural and code structures.

It's not our intent here to illustrate all the possible ways or possible reasons (barring the security reason) to separate objects into different schemas. Architecture and application design and implementation is as much an art as it is a science. The important point conveyed here is not so much stating the laws on how to organize schemas, but that you should apply a functional and logical organization to your schemas.

### Security Concerns

You should understand security implications associated with schemas, as this is critical to ensuring a secure application design and implementation. It's important that we point out that no object-level security within a database schema guards *itself* from *itself*. The schema owner has full access to all the objects within the schema. It may sound a bit ridiculous to say this, because, naturally, that is why you may decide to co-locate the objects, code, and so forth. The obvious interrelationships and interactions among objects that are necessary for the application to work need to exist without security barriers.

It is a fairly common worst practice to allow users—general application users in this reference—to log into the object owner accounts (that is, the schemas where the objects reside). When this is done, the thinking is usually that the application code will protect the objects from any malicious or malevolent user intent. It is most often done to expedite development and minimize execution errors that may arise from an incorrectly or improperly configured database. Unfortunately, there is no way to distinguish between an improper configuration and a genuine security exception until it happens.



#### NOTE

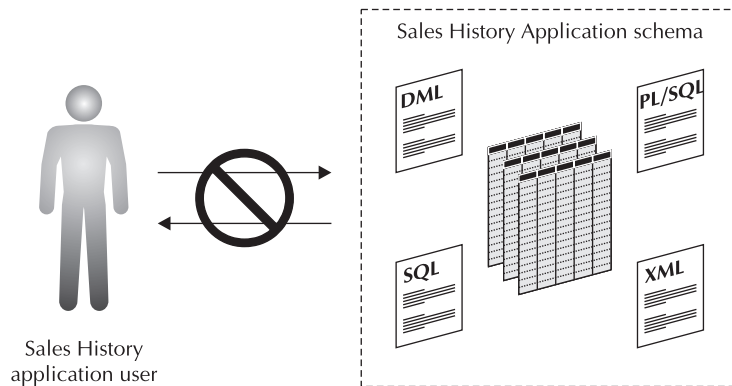
*It is a bad practice to allow general application users to connect directly to the schema that owns the objects.*

This is a bad practice for several reasons. First and foremost, when users are executing within the object owner's schema, no database security will prevent users from having their way with the data or the data structures (recognizing that fine-grained access controls do provide a safety net here, but the point is still valid). Second, as stated earlier, you want layers of security. Turning off the database access security by way of improper design creates a huge risk.

As you can see in Figure 1-1 for a Sales History (SH) schema, you should not allow application users to connect to the database by way of the SH schema. If you do, nothing in the database will prevent users from mutilating the data structures and basically having their way with everything contained within the SH. Obviously, your application would need to provide some sort of security, but even with some security present, allowing user to access the database via the schema is a bad idea.

## User Access Accounts

User access accounts are the schemas in which end users connect to the database. These user connections generally occur in one of two ways: The users may be directly connected to the database via a program such as SQL\*Plus, or the users may be connected via an application,



**FIGURE 1-1** A security worst-practice is to allow application users to connect directly to the schema that contains the data structures, objects, and code of the application.

typically running through an application server that is utilizing some form of connection pool, connection sharing, and/or connection cache. For simplicity, consider only that from the database's perspective, the end user is connected to a schema.

### Dedicated Accounts and Shared Accounts

Irrespective of how the users connect, they will be connected either to a dedicated account or a shared account. At the risk of stating the obvious, a dedicated account associates one distinct user with one distinct account. A shared account is used when users share the same functional roles and thus the same sets of privileges.

Security implications center around two things: user identification for auditing and accountability purposes, and access controls to allow or prevent users from performing specific actions or accessing specific data. From a security perspective, the simplest situation occurs when users have dedicated accounts, because the database applies object-level security and auditing controls at the schema level. If the user has logged in directly or used proxy authentication to a dedicated account, maintaining user identity for accountability is accomplished by default. Additionally, you can leverage all the native object-level security controls provided by the database to enforce access controls on a user-by-user basis.

Given all these benefits, you may wonder why everyone doesn't just go with dedicated accounts as the model. The answer to this was presented earlier in our guiding principles. The three reasons for not architecting dedicated user accounts are as follows:

1. User identity may not be known; the database invocation may be done via nested Web services or a federated call, which is unable to convey actual end user identity.
2. User identity alone does not provide enough of a security basis for many security decisions.
3. A one-to-one user-to-account ratio is a management nightmare.

It has long been argued that you should not allow users to share accounts. Whether you agree with this or not, shared accounts happen quite frequently, especially when the users share

functions and thus privileges. The scary aspect of shared accounts is that they are often applied to administrator accounts such as root, SYS, or SYSTEM for the database. Shared accounts allow multiple users to access the same account but in doing so, it obfuscates their unique identities. This would therefore limit, if not altogether defeat, any auditing. If someone does something bad, no one knows for sure who did it.

Nevertheless, shared accounts still make sense for manageability reasons, and you can employ such techniques as setting the CLIENT\_IDENTIFIER to propagate the real end user's identity. Another prime example on UNIX systems is the use of SUDO, which allows users to execute privileged commands (run as root) while preserving their identity. Notionally, and assuming that the accountability and security privileges are maintained, shared accounts are an ideal way to proceed, which is why it is a popular model used today.

Recall the notion of creating accounts by function. One familiar example of a default account is SYSTEM, the default DBA account for Oracle Database. Using SYSTEM, the DBA can log into this account to conduct most administrative activities. Note that the DBA is not logging into the SYS account unless it is absolutely necessary, as would be the case to grant privileges on a SYS-owned object. Ideally, individual accounts are created for the DBAs, and the DBA role is granted to each of the individual accounts. The authorizations and access controls are then based on the privileges assigned to the DBA role, and each DBA's identity is captured by his or her individual logins.

Note the immediate separation between SYS and SYSTEM. SYS owns the objects; SYSTEM has a functional role as DBA. This separation, which has been established as a precedent by Oracle for many years, serves as the basis on which you should consider your schema designs.

## Getting Started

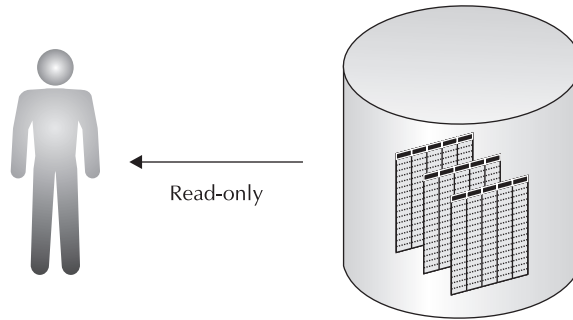
Now you know that you should segment schemas and user accounts. Within user accounts, you need to further divide by function and profile. One effective way to begin these tasks is to divide the user community into four coarse-level groups. These groups are coarse-level and are intended to be such so as to simplify the discussions. This provides the smallest, yet most prevalent, use cases as we discuss security patterns, so the focus can be on the categorizations as opposed to anything else.

We'll use the term *user profiles* to refer to the categories in which user populations are divided. These user profiles allow you to think about the basic requirements and then attach them to an appropriate design pattern. Quite simply, it's a needed first step to designing security correctly. You should be able to use this methodology when you build and design your applications. The intent here is not to solve or complete the design, but to provide a foundation and foundational understanding of what needs to be done and why from a security perspective.

## User Profiles

In its simplest form, four categories of user profiles exist:

- Read-only users
- Read-write users
- Application administrators/developers
- Database administrators



---

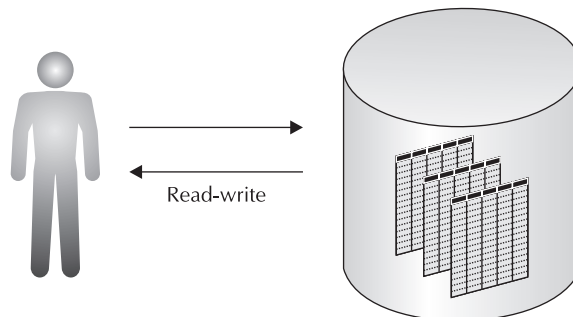
**FIGURE 1-2** *Read-only users only consume information.*

The first class of users will be the read-only users (Figure 1-2) who are generally running reports, such as those typically found in BI applications or data viewed via a portal. Since these users do not modify data, their access should be controlled to ensure that they are connected to a read-only account. You don't need to further define the type of data they will be accessing, except to specify that they should not be able to modify any of it. Typically, you would use other access control mechanisms to ensure that these users read only the data that they are supposed to.

You should remember several important security points when building applications for read-only users:

- The security should enforce the fact that the users cannot change the data.
- Users should be confined to the application's data or only the data they need to access to do their job.
- Users should not be able to change the data objects or perform DDL (such as create or drop tables).

The second class of users will be the read-write users (Figure 1-3) who not only read the information but also update, enter, delete, and perform other tasks that can change the data. Typical examples are transaction-based applications and applications that are self-service in nature.



---

**FIGURE 1-3** *Read-write users can read and update information.*

As with read-only users, read-write users should be prevented from accessing information outside of their application. They should also be prevented from manipulating the data objects themselves.

The third category of users is the application administrator or application developer, who are included together because the access they require is generally the same. The administrator/developer (Figure 1-4) will need to create, update, and delete objects as well as data. Configurations and some levels of fine-grained access controls for the application may also be required. This person's job is focused around an application, and they should not necessarily have free access to all information in the database.

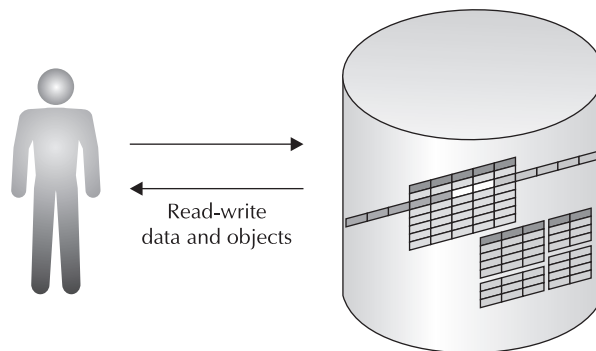
Lastly, we want to consider the DBA. As you know, the DBAs are generally concerned about the health and welfare of the database. Due to the enormous amounts of jobs and activities DBAs need to perform, they generally have superuser access that allows them to access any part of the database and do anything necessary.

These four user profiles are meant to serve as a top-level grouping of user accounts. By thinking about the security implications to each of the user profiles, you can immediately decide what risks you will undertake by using dedicated or shared accounts as well as find easy ways to simplify designs and design discussions. Note that if you decide to share accounts, you will want to seriously consider the suggestions in the database vault chapters (Chapters 4 to 7), which will allow you to share accounts securely and maintain accountability.

## Schema Naming

A large part of this book is dedicated to conveying best practices and methodologies to assist you in designing and developing secure database applications. To that end, coming up with a predictable and consistent naming convention for your database schemas will prove invaluable. A sound naming convention will allow you to collaborate more easily and mitigate confusion for yourself and others.

A good way to start with naming is to give the database account/schema the same name or abbreviation for the application module in which it serves. For example, the sales history objects could be contained in the SH schema, and the order entry objects could be contained in the OE schema.



---

**FIGURE 1-4** *Application administrators/developers manipulate objects.*

Another common practice is to create a new schema for each new version of the application. Of course, this makes sense only when the database objects and relationships have been altered as part of the new version. You will often find the new schema with a version number appended to its name. For example, later in this book you will see the Application Express (APEX) examples. We started with Application Express version 3.1.0. This installed itself into a database schema named FLOWS\_030100. If you dissect the name, you will see the name of the application is Flows, not APEX. Flows is the Oracle internal project name that later evolved into HTML\_DB and now is Application Express. The version naming used groups of three sets of two digits. Therefore, 03 for the major version number of three, 01 for the minor version number of one, and 00 for the initial release. A patched and updated version was released called APEX 3.1.1. As this version contained code modifications only, it stayed in the FLOWS\_030100 schema.

The point to naming is not so you can decipher which version of APEX you are using so much as it is to have a consistent and deterministic method for naming your schemas. When you use a naming scheme, you gain the benefits of knowing which application modules (and often which version) are contained within each schema.

## Security Architecture Checklist

This chapter has focused on security architecture. With everything stated so far, you might find it useful to create a summarized checklist of questions to think about and answer with respect to your security architecture. Consider the following as a start for a potential checklist that you might need to ensure architectural security. A simple way to start is to consider what happens when the end user clicks something in an application that fetches data from the database.

Consider the following:

- At a high level, how does that user action translate to data access? Can you identify all the modules and connections?
- To which account(s) are users connected?
- What are the privileges on the account(s)? This will begin to tell you what possible things the user could do if application security breaks.
- If the user is supposed to have only a subset of the privileges for the account to which he or she is connected, what mechanisms are in place to ensure least privilege? It's better if this security is declarative and enforced inside the database or application server and not programmed within the application code.
- Are you connected directly to the DB through a private or shared connection? If shared, how can you ensure that no information is leaked between connection threads on the application side?
- Are DB sessions shared? If so, what clears the session cache between users?
- In the DB, can a privilege escalation path be achieved by exploiting procedures?
- Are connections occurring to the schema that owns the objects? That's great for development but not so great for runtime, especially if the desired functionality is read-only reporting.
- How is auditing done? How can it be done in a secure way—that is, not manipulated by the people creating the audit?

You should now understand the security relevance to the interrelationship among schemas, users, and the data and objects with which they interact. Who gets access to what? From where? When? And how? This chapter begins to address these questions. The rest are addressed with technology, tips, and tricks discussed in the upcoming chapters.

## Summary

Let's review some key points in this chapter. Computer security continues to change, and technology is moving quickly, so that increased computing capacity has allowed new capabilities as well as exploits. In conjunction with the changes in technology, new thinking about designs and architectures, risks, and requirements have radically changed the security landscape in a short period of time.

Understanding what you are trying to accomplish with an effective security posture is essential to creating a good plan and determining its success. The common security motivators serve as good reference markers for what people are trying to protect and why. Personally identifiable information, protected health information, intellectual property, and an abundance of government regulations are forcing people to think about the pervasiveness of sensitive data and the things they can do to protect this valuable information.

A few guiding principles serve as a practical way to deal with this challenge. We looked at layers of security, manageability, business congruence, and transparency as vital areas that make or break an effective security stance. With all technology—especially security—you need to take a practical approach to implementation. Likewise, manageability is a usability issue for administrators and therefore security must abide by this tenet as well.

You also learned about effective modeling of schemas, which involves comprehension and segmentation according to function. Likewise, connecting users to database accounts must be done with thought and clarity. A lackadaisical approach here can truncate any future successful security designs and worse: it can lead to a disastrous compromise.

In this chapter, we established a baseline from which the rest of the book can refer and which you can use to simplify the real issues around building and deploying a secure database application.