
# The Virtual Private Database in Oracle9*i*R2

*Understanding Oracle9i Security for Service Providers*

*An Oracle Technical White Paper*

*January 2002*

# Virtual Private Database in Oracle9*i*R2

## INTRODUCTION

The ubiquity of the Internet has modernized business practices by reducing costs, improving communication, and enabling fast access to distributed data. The Internet makes for an efficient use of resources, now that customers, partners, and suppliers can easily access centralized data. It has changed the way that companies do business, it has enriched the way schools teach our children, and it has enhanced the approach to data sharing between businesses and their customers. Such information sharing, however, is not without its challenges—a principal concern being the *security* of the data that individuals and organizations dispense. An organization's data is among its most important assets, so business managers and technical leads must choose and implement appropriate products and devices to protect the very assets they wish to share.

### Product Security

Collectively, there countless terabytes of data stored electronically, and we store a measurable portion of that data in a core information storage unit—the relational database management system (RDBMS). With new, innovative technology products hitting the market virtually every day, it may be a surprise that a system invented over twenty years ago (several lifetimes in "Internet time") thrives as the place for information storage. The relational database serves as the underlying resource for information storage that powers information sharing and data availability today. Oracle9*i* builds upon over two decades of development and deployment to deliver the leading database server on the market.

It is well known that a principal reason Oracle leads the database market is its ability to scale to the onerous requirement placed on database servers today. Of commensurate importance is the security built into the database. Simply put, Oracle9i's secure infrastructure makes it an attractive foundation for building and deploying applications.

That said, any vendor can claim to build a secure product, but what assurance of a product's security does one have? There is no equivalent of a TPC benchmark for security, and with the database battles heating up, customers hear conflicting claims from competing companies. How can you be assured of the security built into a product? Independent security evaluations against internationally established security criteria provide assurance of vendors' security claims.

The Oracle9*i* database, introduced earlier this year, builds upon 14 independent security evaluations (undertaken at Oracle's own cost)of its server software. Nine of those evaluations examined, in exceptional detail, the security of the Oracle RDBMS. No other database vendor approaches this number, nor can they claim the years of experience from the efforts behind these evaluations. The evaluation process lasts up to a full year (and sometimes longer) for an independent, licensed and accredited organization to complete; each security evaluation is no small task.

Oracle's main database competitors have completed an insignificant number of evaluations of their database products. IBM has not completed any evaluations of DB2™, and Microsoft has just one for SQL Server™.

They are light-years behind Oracle in this area and can therefore claim little security worthiness of their products. Security evaluations are perhaps the most effective way to qualify a vendor's assertions about its security implementations because such evaluations provide independent evidence of properly implemented security against established criteria.

Other database vendors do not approach Oracle's lead in security awareness, features, or function. Unlike Oracle, IBM builds virtually no security into the database itself, deferring almost all security to an add-on line of security products requiring complex integration work. Building all security outside of the database would be akin to a bank locking the front door, but keeping the vault inside unlocked. You cannot retrofit security into a database—or any other software product for that matter. It is best built-in from the ground-up, and Oracle has been building security into its relational database management system for many years.

### The Need for Granular Access Control

Who would consider opening production systems, such as order entry, inventory and customer support, to customers and partners without the ability to strictly limit data access? Internet-based systems have a strong requirement for access control at a very fine level of granularity, often to the level of individual customers or users.

Another trend sweeping the corporate community is an increased focus on core competencies and the outsourcing of routine tasks. Many examples of this can be found, including human resources, customer support, online ticketing, and Web storefronts. Many companies are interested in providing "hosting" environments, with a well-designed and well-managed computing infrastructure, but face tremendous challenges in designing systems that keep the data of each "hosted" corporation separate and secure from each other, while allowing personalizations and data access methods which best meet their individual needs.

A recurring challenge organizations face is the "application security problem." When access control is embedded in an application (instead of being enforced directly on the data), users who have access to ad-hoc query or reporting tools bypass the security mechanisms of the application. Strong security policies, centrally managed and applied directly to data, enables security to be enforced no matter how a user gets to the data, whether through an application, by a query, or using a report-writing tool. A centrally-managed and applied security policy also offers a lower cost of ownership: organizations can build security once, in the data server, instead of building security into every application which accesses the data.

Many existing applications are also faced with problems of enforcing complex access control policies, required to safeguard sensitive information (the disclosure of which might have severe social and legal ramifications). Human resources and medical information systems, for example, have strict requirements for security and privacy. These applications typically enforce multiple security rules, depending upon who is accessing the data, and what his function is. Organizations within diverse industries endeavor to simultaneously manage data centrally (for administrative ease and cost reduction purposes), while limiting access to centralized data. Consider the following examples:

*A bank* wishes to allow its customers to do on-line banking. The bank needs to ensure that customers can only review transactions and account balances for their own accounts, and not anyone else's account.

*A large telecommunications company* provides long distance service for several local calling areas, and maintains call information for multiple local companies in a central database. The local calling companies must be able to review long distance calling information for their local customers, but they must not ever be able to see customer information for their local calling competitors. Violation of this rule is subject to a large fine for the long distance telecommunications company, per incident.

*A Department of Defense organization* wishes to limit data access based on a security label (Unclassified, Secret, or Top Secret). Users (with appropriate privileges) can only query records at their security classification and below. For example, a user cleared to Secret can retrieve records labeled Secret and Unclassified, while a user cleared to Unclassified can only review Unclassified records.

*A large manufacturing company* has a centralized human resource database which incorporates data from multiple subsidiaries, divisions and departments. There are multiple users of HR information, and different security policies apply to each type of access:

- Employees can view their own HR records, and modify information such as marital status, number of dependents, address, and phone number, but they cannot modify their own salary.
- Managers can view all information for employees who work for them, directly or indirectly.
- HR specialists can review and update employee records within their area only. A specialist might only be able to update records in the Engineering division with last names beginning with the letters A-F.

*A Web hosting company* wishes to run the HR and Payroll business of other companies. Different companies want different personalizations. Some want access to raw data to run business analysis reports that best suit their corporate standards. The hosting company wants to use an HR application, but creating a completely new system for the company is not economically-viable.

In every one of these scenarios, the organization faces the same fundamental challenge: the need to make data available while mediating data access at a very fine level of granularity.

## CURRENT APPROACHES

Oracle7 introduced many features which enabled application developers to limit data access based on well-known security concepts, such as "least privilege." The principle of least privilege states that users should have only the minimum privilege set required to perform their jobs, and no more. These features include:

- Granular privileges as a means of limiting access rights
- Roles to provide ease-of-administration (by encapsulating groups of privileges)
- Views to provide content- or context-based data access
- Stored procedures to enable well-formed business transactions, without direct privilege grants

While many of these features can be used to enforce least privilege, and provide access control at a level greater than table-level (e.g. the ability to access all data in a table), these features aren't always well-suited to access control at much finer levels of granularity. A discussion of one way in which Oracle users can obtain more granular access control, and the limitations of this method, follows.

### Views

Views are the foundation for many applications' security mechanisms. Because views can limit access to information contained in a base table by content or context, they are widely used in many applications. For example, suppose an HR clerk needs access to routine employee information contained in the EMP table (such as name, address and department), but the EMP table also includes salary information which the HR clerk is not allowed to see. A view, which includes only name, address, and department, allows HR clerks to see only the selected columns in EMP which they are authorized to see. A clerk is granted SELECT privilege on the view, not on the EMP table itself. Also, if you want to allow the manager of department 20 to see employee data for only her department, you could create a view which selects department 20 data from the EMP table, and then grant the manager SELECT privilege on the view. These are examples of view usage based on data content, but views are also useful for controlling access based on a different approach.

Views can be used to limit data access according to context. For example, suppose your company policy is that employees can only view salary information during normal business hours. You could create a view (which selects from the EMP table the information you want to see) with the additional restriction that the view can only be accessed from 9:00am to 5:00pm.

LIMITATIONS OF VIEWS

While views can provide fairly granular access control, they have limitations which make them less than optimal for very fine-grained access control:

*Views are not always practical when you need a lot of them to enforce your security policy*. For example, using views to restrict access to customer data by region is probably feasible if there are 10 customer regions (and hence 10 views). On the other hand, using views to limit customers' access to their own records if there are 100,000 customers (and hence 100,000 views) is not practical.

*Views are best suited to access control conditions the database can evaluate simply*. For example, you can create a view of the EMP table for employees who are in department 20 and whose salaries are less than $50,000 only if department and salary are columns in the table, and the database can evaluate the condition "less than 50,000." A more complex access control policy, or one in which the database cannot evaluate the access control condition, simply does not lend itself to views. For example, if your access control policy is "a user accessing the EMP table as a Payroll clerk through the Payroll application is allowed to see all EMP information, including SALARY, but only for employees in her division," this is probably not possible to express in a view, since you can't determine what application the user is accessing at the time you create the view.

*If users access base tables, they bypass view security.* While applications may incorporate and enforce security through views, users often need access to base tables to run reports or conduct ad-hoc queries. Users who have privileges on base tables are able to bypass the security enforcement provided by views. Note that this is a general problem of embedding security in applications instead of enforcing security through database mechanisms, but it is exacerbated when security is enforced on views and not on the data itself (that is, on the table containing the data).

*Views may complicate administration of security policy.* A security administrator cannot tell the difference between the parts of a view definition based on logical object definition, and those designed to enforce security. When a security policy is added, changed, or removed, it's difficult to determine what exactly to do with each view. An administrator cannot tell whether, by changing security policies through altering or dropping a view, he is breaking an application.

## INTRODUCING THE VIRTUAL PRIVATE DATABASE

*The Virtual Private Database (VPD) is the aggregation of server-enforced, fine-grained access control, together with a secure application context in the Oracle9i database server.* It provides a flexible mechanism for building applications that enforce the security policies customers want enforced, only where such control is necessary. By dynamically appending SQL statements with a predicate, VPD limits access to data at the row level and ties the security policy to the table (or view or synonym) itself.

The Virtual Private Database offers the following benefits:

- ***Lower cost of ownership.*** Organizations can reap huge cost savings by building security once, in the data server, instead of implementing the same security in each application that accesses data.
- ***Elimination of the "application security problem."*** Users cannot bypass security policies embedded in applications because the security policy is attached to the data. The same security policy

is automatically enforced by the data server, no matter how a user accesses data, whether through a report-writing tool, a query, or through an application.

- ***Application transparency.*** Virtual Private Database is enforced at the database layer and takes into account application-specific logic used to limit data access within the database. Both commercial off-the-shelf applications and custom-built applications can take advantage of its granular access control, without the need to change any lines of application code.
- ***New business opportunities.*** In the past, organizations couldn't give customers and partners direct access to their production systems because there was no way to secure the data. Hosting companies couldn't have data for multiple companies reside in the same data server, because they could not separate each company's data. Now, all these scenarios are possible, because fine-grained access control gives you server-enforced data security with the assurance of physical data separation.

These benefits contribute to Oracle9i's industry-leading security solutions. No other RDBMS vendor offers a competitive feature set that can limit access to data at a comparatively granular level, uniquely placing Oracle9*i* as the database of choice for any security-conscious or cost-sensitive application developers and customers.

The following sections describe the functionality of the Virtual Private Database—fine-grained access control and a related feature, secure application context—provided in Oracle9*i*.

### Dynamically Modified Queries

Fine-grained access control relies upon "dynamic query modification" to enforce security policies on the objects with which the policies are associated. Here, "query" refers to any selection from a table or view, including data access through a query-for-update, insert or delete statements, or a subquery, not just statements which begin with SELECT.

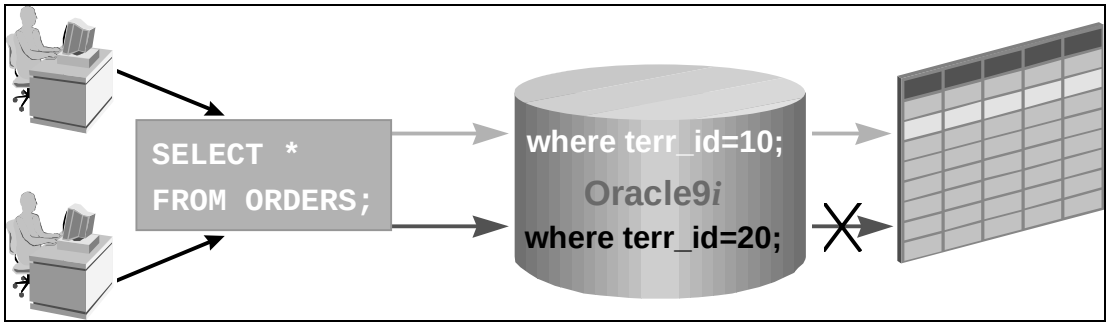


*Figure 1: Oracle9i with Virtual Private Database dynamically modifies SQL queries. Both users execute identical SQL statements, and with VPD, the database returns customized results to each user.*

As shown in figure 1, a user directly or indirectly accessing a table, view or synonym with an associated security policy causes the server to dynamically modify the statement based on a "WHERE" condition (known as a *predicate*) returned by a function which implements the security policy. The user's SQL statement is modified dynamically, transparently to the user, using any condition which can be expressed in, or returned by a function. Functions which return predicates can also include callouts to other functions; you could embed a C or Java callout within your PL/SQL package that could either access operating system information or return WHERE clauses from an operating system file or central policy store. You have great flexibility within a policy function, which can return different predicates for each user, each group of users, or each application.

At its purest, Virtual Private Database can limit access to data in certain tables for all users, as required by a corporate policy. For example, a company involved in trading might need to limit access to certain tables so they are only accessible during trading hours. This organization can implement a simple VPD policy that acquires the time of day and the day of week from the system's SYSDATE. The policy allows users to query the table during normal trading hours, but returns no rows when any user attempts to access the table outside of trading hours.

Other unrelated tables, views or synonyms within the same database need not have the policy applied to them so that users can access them after hours and on days in which the market is closed. This simple example illustrates a very straightforward implementation of Virtual Private Database, but it does not take advantage of more powerful ways to use the feature.

Consider an HR clerk who is allowed only to see employee records in the Engineering Division. When the user initiates the query "SELECT * FROM emp," the function implementing the security policy returns the predicate "division = 'ENGINEERING'", and the database transparently rewrites the query, so that the query actually executed becomes "SELECT * FROM emp WHERE division = 'ENGINEERING'". The database could obtain the name of the division from a subquery on another table, or the dynamically modified query implementation could be enhanced to use application context, which is explained in the next section.

### Secure Application Contexts

Many organizations want to make access control decisions based on something about the user, such as the user's position within the organization, his organizational unit, whether he is a customer or partner. Application contexts give application developers an easy mechanism to define, set, and validate the security attributes on which to base fine-grained access control and thus enhance the ability of developers to implement the Virtual Private Database within Oracle9*i*.

***Application contexts act as secure caches of data that may be applied to a fine-grained access control policy on a particular table, view or synonym.*** Upon logging into the database, Oracle9*i* sets up an application context in the user's session. Information in the application context is defined by a developer based on information relevant to the particular application. For example, an Order Entry application that will query data from an Orders table can base its access control on the user's position and geographical location. The application, in this case, could initially set up an application context for each user as he/she logs in and populate it with data queried from the Employees and Sales tables for the user's position and area (region), respectively. The package implementing the VPD policy on the Orders table references this application context to populate the user's position and area for each query. As such, Application Context obviates the need to execute sub-queries, which otherwise hinder performance.

Following are the salient points on Application Context that an application developer should understand:

***Application contexts are completely definable by an application developer***, as are their attributes. As a result, each application can have its own application-specific context, with different attributes. For example, your order entry application might base access control on customer number, position (whether you are an order entry clerk, a customer, or a sales rep), and sales region; "customer number," "position," "sales region," etc. are all *attributes* of an application context, which you could call ORDER_ENTRY_CONTEXT. A human resource application might base access control on position, organizational unit, and management hierarchies, for which you could define an application context called HR_CTX, having attributes "position," "org_unit," and "hierarchy." Thus, application contexts are extensible and useful for a variety of applications.

***Application contexts are easy to use*** — Applications can set, verify and retrieve a particular context attribute conveniently and unambiguously. For example, if a GL user changes the set of books she is referencing within an application, the application is able to reset a "set_of_books" attribute without resetting all other attributes, or without parsing a long string of attributes to change the attribute of interest. Oracle9*i* offers a system function (SYS_CONTEXT) which allows you to specify the exact context and attribute you want to set.

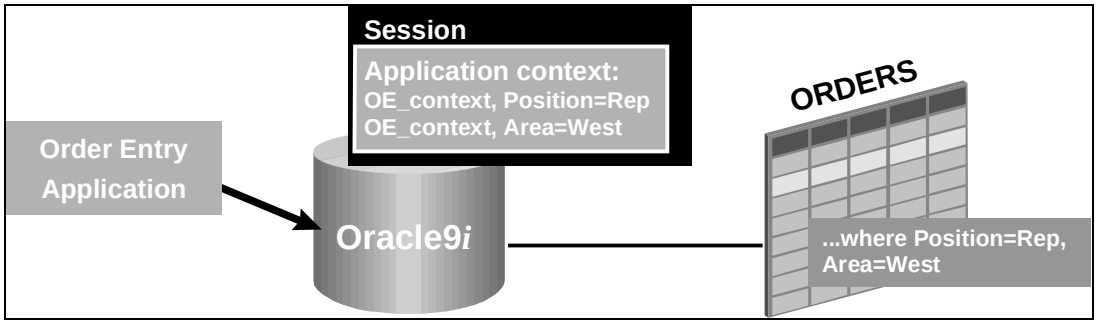


*Figure 2: Application Context stores a cache of information useful for access control decisions.*

***Application contexts simplify the implementation of fine-grained access control*** in two significant ways:

> *Predicate selection*. You can access an application context within the function implementing a security policy to determine the correct predicate to return. For example, if an attribute of your Order Entry context is "position," you can return different predicates depending on position; e.g. if the user has the "clerk" position, then the predicate returned results in a query that retrieves all orders, but if the user has the "customer" position, then the predicate returned results in a query that returns records for only that customer.
>
> *Providing a bind variable within a predicate*. A context attribute can be used within the predicate itself, to provide a bind variable. For example, to limit customers to seeing their own records, you could return a predicate which limits records returned based on a "cust_num" attribute of your order entry context. The "cust_num" attribute will be different for every user. Note that you've now created one SQL statement which is shareable by all users, which nonetheless executes differently for each user.

***Application contexts are secure*** so that application contexts may be safely used to enforce fine-grained access control. Application contexts provide security in the following ways:

> *Context uniqueness*. Oracle9*i* enforces that context names are unique across an entire database, to ensure that contexts can't be duplicated or spoofed by individual users, either inadvertently or maliciously. For example, if your human resources application uses the context HR_CONTEXT, you do not want any user to be able to create her own HR_CONTEXT (which might potentially allow her to access more information in HR than she is otherwise privileged to see). Also, the ability to create a security context is a separate system privilege; only suitably-privileged users are able to create a context.
>
> *Attribute validation*. For example, suppose a user accessing GL changes his set of books from 01 to 02. The application context can both ensure that 02 is a valid set of books, and that the user has the privilege to access set of books 02 (for example, by querying application metadata tables).
>
> *Secure attribute setting*. The database ensures that whenever a context attribute is set, it is the trusted package (implementing the context) and only the trusted package that sets the context attribute. Oracle9*i* does not allow users to make changes to the package; only the package itself can write to the user session. The database accomplishes this by checking the call stack, thereby ensuring that the trusted package is issuing a call to set an attribute. As a result, system security officers can comfortably allow applications to base security decisions on application contexts, because they can be assured that the context is set correctly, by a trusted and known package (and not a malicious user or process).

***Built-In Session Primitives enable application context to use information the database already has available*** regarding a user session, in order to perform access control. Oracle9*i* provides a built-in application context namespace, USERENV, which provides access to predefined attributes called *session primitives*—information which the database captures regarding a user's session. For example, the IP address from which a

user connected, the distinguished name (DN) from a user's public-key (X.509) certificate, the username, and a proxy username (in cases where a user connection is proxied through a middle tier), are all available as predefined attributes through the USERENV application context.

Predefined attributes can be very useful for access control. For example, if you are using a three-tier application which creates lightweight user sessions through OCI, you can access the PROXY_USER attribute in the USERENV application context to determine whether the user's session was created by a middle tier application. Your policy function could allow a user to access data only for connections where the user is proxied. If not (that is, in cases where the user is connecting directly to the database), the user would not be able to access any data. You could use the information in the user's DN to perform access control; for example, if the Organizational Unit (OU) is "Acme Corporation," you could limit the data accessed to "Acme Corporation" data. Predefined attributes can be accessed through the USERENV application context, but cannot be changed.

### VPD FEATURES INTRODUCED IN ORACLE9I

The Oracle9*i* Database release marks the introduction of three new Virtual Private Database features that add to the already-powerful Oracle8*i* Virtual Private Database:

- ***Partitioned Fine-grained Access Control*** – which provides the ability to create unique application contexts per-application.
- ***Global Application Context*** – which supports connection pooling common to multi-tier deployments while preserving Oracle9*i*'s ability to make fine-grained access control decisions based on user information.
- ***Oracle Policy Manager*** – the graphical user interface (GUI) tool used for managing VPD policies.
- ***Support for Synonyms*** – which enables the application of VPD policy functions on synonyms (in addition to tables and views) so that applications that rely on synonyms can take advantage of fine-grained access control. Introduced in Oracle9*i*R2.

Following is a discussion defining the new features and their value in enhancing VPD. It is followed by an introduction to an Oracle9*i* database add-on option, ***Oracle9i Label Security***, which is built on top of VPD.

#### Partitioned Fine-grained Access Control

*A database serving up data to multiple applications can run a different application context for each of the applications.* When deploying a Virtual Private Database on a server used by multiple applications, it is useful to maintain separate contexts for each application so that developers do not have to agree on a shared policy. This enhancement enables customers using the feature to deploy VPD on more systems because they can now centralize application data for a number of different applications that share some of the same tables, yet define completely separate VPD policies on them, returning appropriate results per application.

Consider two applications, an Order Entry (OE) and a Sales Analyzer (SA), that both rely on an Oracle9*i* database named Products. The Order Entry application is built in-house, and the Sales Analyzer is an off-the-shelf product from an enterprise software firm. Though both query many of the same tables and views in the Products database, they each have distinct access control conditions they must individually enforce. OE must prevent users from entering orders in someone else's name and limits users from entering orders outside of their own region, while Sales Analyzer needs users to examine sales figures based on their region and perform analysis on only the product line they're responsible for. A "driving" application context securely determines which application is accessing data, and policy groups facilitate managing the policies which apply by application. The value of the driving application context indicates which policy group shall be enabled. In this example, the policy group for OE (users can update only in their own orders and those within their region) is enforced when the OE driving context is active, and when the query is executed through SA, its policy group

(limit access by region and by product line) is enforced when SA is the driving application context. The driving application context *drives* which application context enforces the policy at a given time. By applying partitioned fine-grained access control, the SQL predicate appended to statements differs by application.

There might also be a default policy group which acts as the policy that is always enforced, onto which the application-specific contexts are added. For example, you might want the database to enforce a policy in which "users only see products manufactured by their own subsidiary" in addition to any application-enforced confines. This policy is ANDed together with the Application A-specific application context when a user accesses the table through that application. It is ANDed together with the Application B-specific context when that application accesses the table. Additionally, in the case that the database has no information on which application context should be in use (that is, the value of add_context is null or has not been set), the database enforces all policies for all applications. This is done to ensure that a user cannot get more data if she connects directly, bypassing the application (for example, with a query tool such as SQL*Plus) than if she executed the query through an application.

The partitioned fine-grained access control model uniquely allows application-driven security enforcement without relying on a less secure application security mode, in which all access control would otherwise be enforced by the application, leaving the data in the database exposed to direct queries. It allows you to mix and match custom-built applications with off-the-shelf products with the ability to set differing security policies appropriate for each application. You can still enjoy the financial and technical benefits of application transparency, as well as the security benefits derived from this powerful aggregation of application-based security logic with database-enforced access control.

#### Global Application Context

*Applications utilize global application context to supply user identity to the database, which in turn utilizes the identity for access control decisions. Secure application context can be shared across sessions.* The three-tier architecture is the cost common model for delivering highly effective, scalable, performant information systems, particularly for web-based applications. The middle tier application or application server establishes necessary application logic and performs application-specific operations, while the database provides the scalability, security and availability required for web-enabled applications. Oracle9*i*'s global application context increases performance for systems running in a three-tier environment. Because middle tier server does not create a new user sessions for each connection to the database, global application context enables applications to scale in a security-conscious manner. Good performance is the primary reason application developers use the feature, but consider the following additional reasons.

First, using global application context balances the benefits of utilizing database security functionality with applicability to oft-used architectures. That is, using this feature within three-tier environments makes it possible to use of fine-grained access control as well as audit the end user who need not be a database user.

Second, many application servers use connection pooling to enhance performance and reduce the number of physical network connections between the middle tier server and the database server. Connection pooling limits the use of network resources used for each process, supports large user populations, maximizes the number of client-server sessions over a limited number of process connections, and optimizes resource utilization. Global application context allows you to take advantage of connection pooling.

Most application servers do not start individual session for each user, as it weighs down the network with too much overhead. Instead, applications often connect to the database server simply as application_user. While this model scales well, it is not an extremely secure model because (a) almost all security must be built into the application, and (b) any user who subverts the application can potentially gain full access to the data with no access control protections. Global application context adds much-needed security to applications functioning in

this type of environment. Application developers do not have to re-architect the entire application; they need only build in support for global application context and pass to the database the relevant information in the client_identifier. The client identifier can refer to an attribute such as a user's name or virtually any type of group, enabling flexible options for using global application context in a variety of application environments. Following are examples of each of these two approaches.

Consider an online banking application running in a three-tier architecture that consists of users on web browser clients, the middle-tier banking application, and an Oracle9*i* database. First, the application authenticates users on the browsers over http/s (HTTP with SSL), and, for best performance, the application pools connections to the database. It connects to the database as BankApp and pools users' connections, yet endeavors to make access control decisions based on individual users. Because global application context is employed, the application connects as BankApp and sets a different client identifier (which can be defined and set by the application) for each of its users. The Oracle9*i* database uses the identifiers for fine-grained access control, successfully restricting the user's access at the row level as if she/he were a database user.

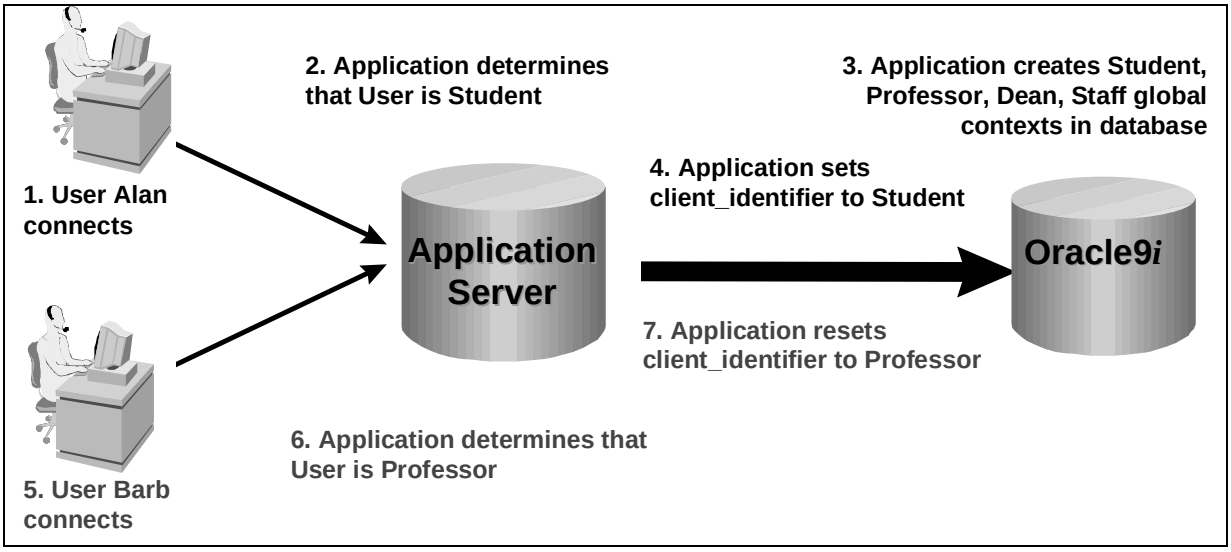


*Figure 3: Global Application Context combines application- and database-level controls in a secure, scalable environment.*

Another example (shown in Figure 3) that is well-suited for global application context is an application deployed at a university which shares the security decisions between the middle tier application and the database. The application server first authenticates users, then determines their category: Student, Professor, Dean, or Staff, then connects to the VPD-enabled database. When the first user connects, the application establishes that user Alan is a Student, then connects (as itself, the Application Server) to Oracle9*i* and creates four global application contexts of Student, Professor, Dean, Staff in the SGA. For the first user, Alan, it sets the client_identifier to Student. At this point, the database can use the application context information in access control. When user Barb connects to the application, it authenticates her and establishes that she is a Professor. Within the same database session, the application resets the client_identifier to Professor, at which point the database limits access to tables, views, and rows therein to those appropriate for Professors.

Oracle9*i* introduced global application context as a part of VPD, but, in fact, it has applicability beyond Virtual Private Database for secure three-tier systems employing connection pooling. The application server can provide such a client identifier to the database—*even if it's not employing VPD*—in order to share sessions among multiple end users who are not database users. It acts as a way to manage access control on a group- or user-specific level and maintain user identity throughout the tiers of a multi-tier application. Global application context thus exemplifies an exceptional model for deploying scalable and secure three-tier systems.

### Oracle Policy Manager

Oracle Policy Manager is the new Java-based GUI administration tool for managing Oracle9*i* Virtual Private Database fine-grained access control policies and application contexts. The same tool also manages policies for the database option, Oracle9*i* Label Security. Oracle Policy Manager provides a standard Oracle interface easing the administration of VPD policies and application contexts and is especially useful for large implementations employing multiple policies on various tables, views or synonyms. The tool does not replace the coding involved in deploying a Virtual Private Database. However, it greatly simplifies the administration involved in managing VPD policies, application contexts, and global application contexts. Oracle Policy Manager makes VPD so easy to use, in fact, that some administrators gain interest in VPD because of the tool itself.

### Oracle9*i* Label Security

*Built on top of VPD, Oracle9i Label Security enforces label-based access control.* Oracle Label Security is a security option for the Oracle9*i* database that mediates access to data by comparing a sensitivity label assigned to a piece of data with label authorizations assigned to an application user. Such access mediation allows data to be separated into different sensitivities within a single database. Application hosting, healthcare, national security, and privacy enforcement are just a few of the areas which can benefit from Oracle9*i* Label Security.

Labels are used extensively in commercial organizations. Examples of labels include [internal], [confidential], [sensitive:human resources], and [internal:ACME California]. Oracle Label Security policies are applied to individual tables or entire schemes. Oracle9*i* Label Security uses an Oracle-supplied security package to mediate access to data rows, and no coding or PL/SQL software development is required.

Oracle9*i* Label Security policies are comprised of a policy name, enforcement options, label definitions, user label authorizations and a list of protected objects. Using Oracle Policy Manager—the same tool used to manage VPD—you can create policies, define label components, create labels, establish user label authorizations, customize enforcement options, apply policies to schemes and tables, drop policies from schemes and tables, disable policies, and configure Oracle Label Security specific auditing options. In addition, SQL*Predicates, more commonly known as *where* clauses, can be added to Oracle9*i* Label Security policies using Oracle Policy Manager.

### Virtual Private Database Support for Synonyms

E-business applications built on top of databases often utilize synonyms as aliases for tables, views, or other objects. Synonyms are useful for masking the name and owner of a schema object, providing public access to a schema object, providing location transparency for tables, views, or program units of a remote database, and for simplifying SQL statements for database users.

In Release 2 of Oracle9*i* (Oracle9*i*R2), VPD has been enhanced to support the application of policy functions on synonyms (in addition to tables and views). Thus, applications that rely on synonyms can take advantage of fine-grained access control. Both public and private synonyms are supported for full flexibility in application development.

The benefits of VPD support for synonyms are many. First, it means that applications can replace the use of views with the use of synonyms and reap the benefits of lower overhead due to fewer database objects. With fewer objects, users realize increases in performance. Also, there is no change to the existing PL/SQL API used in Virtual Private Database in order to support synonyms, so there is no new interface to master. This Oracle9*i*R2 feature enables wider deployment of Virtual Private Database for applications that must scale well and enforce security at the database layer.

### VPD Assists Application Development

With the powerful security infrastructure provided by Virtual Private Database, many application vendors and in-house application developers find VPD an effective platform for securely scaling their applications. Without it, developers might have to rely solely on views, build all security logic into the application, or simply not be able to deploy practical, secure and scalable applications. In this light, one of the most challenging applications to build is a hosting environment, with its stringent requirements for separation of data. Application Service Providers (ASPs) that host their customers' data face a very strict requirement to separate data of their customers—they would certainly lose business if they were to accidentally share proprietary information among them.

## ADDITIONAL VPD CAPABILITIES

The following sections discuss more advanced Virtual Private Database concepts.

### Applying VPD Policies to Existing Views

*Virtual Private Database enables customers to extend the discretionary access control mechanisms already provided by Oracle9i to a finer level of granularity than was previously possible.* VPD can further enhance the security already provided by existing views present in a database, not compete with them.

In some cases, users may need to have base table access to run reports, for example, or to do ad-hoc queries using SQL*Plus. Users who can access base tables underlying views thus bypass security policies attached to those views. In these cases, fine-grained access control can be applied to base tables, which ensures that, no matter how their users get to data — via an application, a report writer, or SQL*Plus — the same security policy is enforced. This ensures both data consistency (users can access the same set of data no matter how they access the data) and that there are no loopholes or "backdoors" by which users can violate the security policy.

On the other hand, many applications today already use views to limit data access. Security policies may be attached to views, for additional application development flexibility. Application developers (and customers of these existing applications) would like to extend the current view-based functionality they have, rather than completely rewrite their applications to use table-based security. For example, an HR application may use a view of the EMP table (EMP_VIEW) which includes all information from the EMP table except salary. The company would like to allow employees to view and update their base employee information online. Adding a security policy to the view EMP_VIEW, rather than the base EMP table, preserves the current application while supporting the desired functionality: that employees can only view or update their own employee records, and nobody else's.

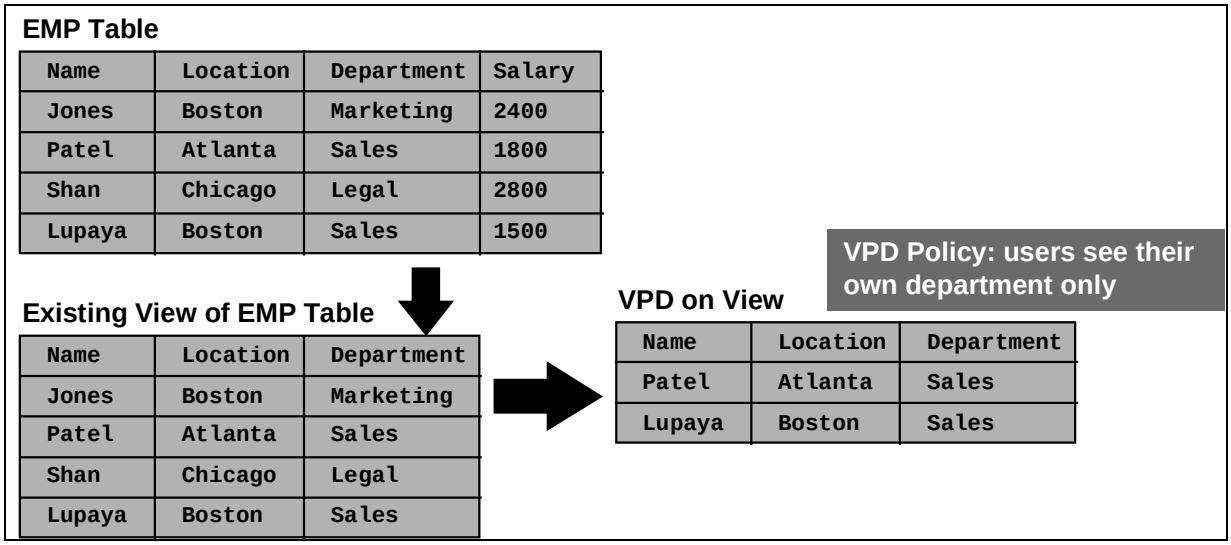


*Figure 4: VPD and views in collaboration save re-development.*

In considering whether to apply a policy to a table or view, note that it is possible to create a view reflecting a complex security policy, particularly when combined with the application context feature; however, you may also end up with an unwieldy predicate which results in a poor query plan. For example, if you create a view which may be accessed by multiple users, each with different access conditions, the view itself needs to contain a lot of data as well as potentially having a security policy with many OR conditions. While only a few conditions may be relevant to any particular user, if all of the (potential) access control conditions are associated with a view, the optimizer will have to incorporate them into everyone's query plan. However, if you apply fine-grained access control to the base object rather than the view, everyone can reference the base object, with only those predicates relevant to each user appended and optimized. That is, instead of having one large view with many access conditions to be evaluated, you have multiple, dynamically-created views, each of which only incorporates a small set of access conditions. Enabling fine-grained access control on the base object (rather than on the view) allows the view to be dynamic before execution, rather than during execution, so performance is much faster.

There are benefits to both view-based and table-based fine-grained access control. Allowing security policies to be attached to either tables or views provides customers with the flexibility they need to extend the security of existing applications based on views, or to associate their security policy directly with base tables, as they choose.

#### Granular Security Policies

*Fine-grained access control need only be implemented on those tables, views or synonyms where you want it.* For example, an Order Entry application, in order to enforce the security policy "customers can see their own orders, but nobody else's orders" might only need fine-grained access control on the ORDER and ORDER_LINES tables, not all the tables used by the application. In many cases, if users have the ability to SELECT from a table, they are allowed to select anything in the table, and thus no additional access control needs to be implemented. Attaching security policies to selected tables, views or synonyms (instead of making a policy apply system-wide) allows you to use fine-grained security only where you need to. Additionally, you can add, drop, or disable a policy on a table at any time, if you have appropriate privilege.

Another advantage of applying fine-grained access control to tables, views or synonyms is that you can continue to use existing applications, while enjoying the benefits of better security. You don't need to rewrite your entire application to use fine-grained accessed control, you need only add it to base tables or views.

### Multiple Policies per Table

*Oracle9i's Virtual Private Database capability provides maximum flexibility to support both built-in application security and site-specific customization.* For example, an off-the-shelf Order Entry application might provide fine-grained access control on the ORDERS table based on sales organization (sales representatives can see any customer orders from their sales organization, but not orders from any other sales organizations). A site which sells sensitive military equipment might want to customize the Order Entry application to limit access to customer orders based on the security clearance of the Order Entry clerk. The addition of an additional, custom security policy on the ORDERS table enables the desired customization without tampering with the base security policy of the packaged Order Entry application. If there are multiple predicates returned for a user, Oracle9*i* automatically ANDs them together to create the rewritten SQL statement. Also, if your security policy changes, you can drop, alter or disable it, without tampering with (and possibly altering) the security enforcement mechanisms of the base application.

### Statement-based Access Control

*Fine-grained access control allows you to implement your access control policies based upon statement type* (e.g. SELECT, INSERT, UPDATE, or DELETE). This allows application developers (and security specialists) maximum flexibility to implement desired security policies. For example, a divisional HR representative might be able to view (SELECT) all employee records in her division, but only create or change (INSERT, UPDATE, or DELETE) records for employees whose last names begin with A through F. Having different policies for different statement types (on the same object) provides customers with the flexibility to fine-tune their access control policies based on their needs and preferences.

Oracle9*i* also supports a "check option" on a security policy to automatically ensure that users inserting or updating a record can "see" the resulting record. This ensures that users don't become frustrated by altering or inserting a record they cannot later SELECT.

### Scalable Security

*The Virtual Private Database's fine-grained access control has been designed to be highly scalable, and to use the underlying optimization features of Oracle9i.* Under most circumstances, the addition of a security policy to a table should not adversely impact performance. The addition of a WHERE clause, appended dynamically to a statement, occurs before a statement is optimized. This means that the full statement (including the appended WHERE condition) participates in optimization, so that it is parsed and executed efficiently. And of course, the full statement can participate in shared memory, so that any user executing the same statement (including the WHERE condition) can reexecute the statement without reparsing it.

The use of application context with fine-grained access control can deliver even greater performance benefits. because application context can function as a secure data cache. For example, to implement the policy "customers can see their own orders," one could have the actual policy function determine the customer number for the logged-in user, by querying the CUSTOMERS table. Or, a developer can create an application context having a "cust_num" attribute; the policy function (or functions) can then access the "cust_num" attribute when needed instead of querying the CUSTOMERS table repeatedly. It's the difference between writing an often-used phone number on a Post-It and sticking it on your telephone (where you can access it readily), and looking the phone number up each time you need to use it.

While the value of using an application context may not seem evident in such a simple example, consider that many applications have a variety of access control attributes; your policy might be "customers can see their own orders, order entry clerks can update all orders for customers in their region only, sales reps can query orders for only their customers." In this case, your context attributes could include "customer_number," "position" (clerk, customer, rep, manager of rep), and "sales_region." Now you can clearly see the benefit of caching the attribute

values for the logged-in user (once), instead of doing multiple queries to retrieve multiple attribute values within a policy function.

## SUMMARY

The Virtual Private Database is key enabling technology for opening mission-critical systems to partners and customers over the Internet. Fine-grained access control, with secure application contexts, enable organizations to secure data in the Oracle9*i* server, and ensure that, no matter how a user gets to the data (through an application, a report writing tool, or SQL*Plus) the same access control policy will be enforced. Because it can be transparent to applications, it can help a commercial application vendor or an in-house application designer decide to run Oracle9*i* as the database of choice because no other vendor supplies a comparable means of implementing granular access control nor nearly as mature an implementation that combines security, scalability and performance.

The Virtual Private Database can help ASPs ensure that customers see their own data and nobody else's, that telecommunications firms can keep customer records safely segregated, and that human resources applications can support their complex rules of data access to employee records. The Virtual Private Database also helps lower your cost of development, by building security once, in the data server, instead of in every application that accesses the data. It thus curbs the "application security problem." Finally, it complements the most common application models to achieve secure, scalable three-tier deployments that combine the use of connection pooling with the ability to control access at the row level within Oracle9*i*.

**The Virtual Private Database in Oracle9*i*R2**
**January 2002**
**Authors: Kristy Browder and Mary Ann Davidson**
**Contributing Authors: John Heimann and Paul Needham**

**Oracle Corporation**
**World Headquarters**
**500 Oracle Parkway**
**Redwood Shores, CA 94065**
**U.S.A.**

**Worldwide Inquiries:**
**Phone: +1.650.506.7000**
**Fax: +1.650.506.7200**
**www.oracle.com**

**Oracle Corporation provides the software**
**that powers the internet.**