Database Security in Oracle8*i*™

An Oracle Technical White Paper

November 1999

Database Security in Oracle8*i*

THE NEED FOR SYSTEMS SECURITY

The opening of mission-critical systems to partners and customers over the Internet poses new challenges to traditional notions of enterprise security. Data access must now be controlled at a very fine level of granularity, often to the level of individual customers or users. Organizations providing "hosting" environments seek to deploy common applications which nonetheless can incorporate customer-specific preferences, and operate on customer-specific data. Oracle8*i* addresses these requirements by providing highly granular, server-enforced access control and flexible privilege models. Users can be strongly authenticated, even remotely, and data is protected in transit by network encryption. Oracle8*i* enforces the same strong security whether users access data directly, or through middle tiers, such as application servers or transaction processing (TP) monitors.

Another challenge of system security is ease of management. Organizations spend significant time and resources managing multiple user accounts and privileges. Additionally, they often must implement security in multiple applications accessing the same data — which is duplicative and often leads to security vulnerabilities — instead of building security once, in the data server. Security is often complex and expensive to implement. Oracle8*i* addresses these needs by offering integrated security and directory services, which enables Public Key Infrastructure (PKI)-based single sign-on. Single Station Administration allows organizations to manage users and their privileges centrally, with greater ease and lower cost. Flexible, granular security can be built once in the data server, instead of in multiple applications, and business logic may be divorced from actual privileges and data, which means that applications can be developed once, then reused and redeployed at significant cost savings.

Within the enterprise, information is stored on physically separate computers in different locations. Therefore, it is essential that users be able to access all information easily and consistently. Consequently, a database server must provide the technology to hide the complexity of data access from users, allowing them to access distributed information as if it were all stored on the same computer. Oracle8*i* addresses this requirement by providing a transparent interface to all data in the system, improving access to information and simplifying application development.

Oracle8*i* addresses all these security and functionality needs by providing complete and robust facilities for managing data and implementing a strong, yet flexible, security policy. This paper describes these security facilities, and how an organization can use them to enforce an overall security policy throughout Oracle8*i*.

SECURITY CONCEPTS

Basic computer security concepts require that an information system be able to identify and control critical aspects such as:

- Who the authorized users are (identification and authentication).
- What users should have access to (object access controls).
- What types of operations users can perform on those objects (also part of object access control).
- What types of activities have occurred (e.g., the ability to maintain accountability via auditing).
- Extended security concepts further address issues such as data and system integrity, reliability and availability, further conditional access controls (such as for special business rules), and assurance that all the above are operating properly and consistently. The following sections discuss these security concepts as supported by Oracle8*i*.

Strong User Authentication For Accountability

The basis for system security is strong user identification and authorization; if you cannot establish, with certainty, who a user is, then it is impossible to hold users accountable for their actions, and to ensure that users only have access to the data they need to do their jobs, but no more. Oracle8*i* supports a number of choices for user authentication: Oracle-based (by password, or by industry-standard X.509 certificates), host-based (by the underlying operating system), or third-party based (network authentication services, smart cards and biometric devices).

Oracle Password-Based Authentication

In Oracle password-based authentication, each Oracle8™ user must have a username and password. To connect to the database, a properly-authenticated operating system user must supply his database username and password. However, password-based schemes, to be secure, must ensure that passwords can be changed regularly, are of sufficient complexity, and are not easily guessed.

Oracle8 provides built-in, robust password management facilities to enable administrators to:

- Enforce minimal password length.
- Ensure password complexity (i.e., that passwords contain symbols or numbers as well as alphabetic characters).
- Disallow passwords that are easily guessed words, such as a user's last name or company name.

Administrators can prevent password-guessing attempts by locking accounts automatically after a number of incorrect password entries; an administrator can also lock an account "on the fly" if he detects a security breach. Passwords can be forced to expire over any period (every ninety days, for example) to ensure that users change their passwords regularly. Administrators can also prevent passwords from being reused, either permanently, or for a specified period of time. Password preferences may be assigned to an entire enterprise, groups of users, or individual users by means of user profiles, providing complete flexibility for an organization to implement desired security preferences.

In distributed systems, a password passing from a client to server may pose a security risk. If the password is passed in clear text (unencrypted), any eavesdropper snooping for data can also read the password. The Oracle password protocol provides security for client-server and server-server password communication by encrypting passwords passed over a network. The Oracle password protocol uses a session key valid for a single database connection attempt to encrypt the user's password. Each connection attempt uses a separate key for encryption, making the encryption more difficult to decipher. After the key-encrypted password is passed to the server, the server decrypts it, then re-encrypts it using a Data Encryption Standard (DES) based one-way encryption algorithm and compares it with the password stored in the database. If they match, the user successfully connects to the database. The Oracle password protocol is used to encrypt all passwords upon an attempted connection — whether local connection, client to server, or server to server. Oracle8*i* also supports secure remote administration protected by password, even when the database is not available. Users connecting as SYSDBA and SYSOPER connect using user-specific passwords, providing individual accountability for these privileged users.

Host-Based Authentication

Oracle8*i*'s identification and authentication facility also allows you to specify that users should be authenticated by operating system mechanisms, consolidating username and password information and allowing users to enter an application without having to specify a username and password.

Third Party-Based Authentication

Oracle Advanced Security, an option to Oracle8*i*, supports multiple third-party authentication technologies, such as Kerberos, DCE, smart cards and biometric authentication (Identix), as well as integration with Bull’s ISM and ICL’s Access Manager. These hardware and software technologies verify a user’s identity in a stronger way than passwords. For example, SecurID cards provide two-factor authentication — something you have (the card) and something you know (a personal identification number (PIN)). Many of these network authentication services also provide single sign-on for users. Users authenticate themselves once to a central service (e.g., Kerberos), and may then connect to multiple applications or databases without providing additional credentials. In addition, any device compliant with RADIUS (Remote Authentication Dial-In User Service) is capable of integrating with Oracle8*i* to provide strong user authentication. Oracle8*i*’s integration with third-party security providers offers customers a choice among a number of strong authentication and single sign-on services.

Public Key Infrastructure-Based Authentication

Oracle8 introduced single sign-on for Oracle users through X.509 (version 1) digital certificates and a proprietary authentication protocol. The advantage of X.509 certificates is that they may be used to uniquely identify an individual within an organization and thus enable strong authentication. Also, instead of remembering multiple passwords, a user need only remember the password that unlocks his Oracle wallet. The certificate and private key contained in the wallet are used to authenticate the user to multiple services, including application servers and data servers, which need no longer store and manage local passwords for users.

Oracle Advanced Security offers enhanced PKI-based single sign-on through use of interoperable X.509 (version 3) certificates for authentication over Secure Sockets Layer (SSL), the standard for Internet authentication. In addition to strong user authentication, SSL also provides network data confidentiality and data integrity for multiple types of connections: LDAP (Lightweight Directory Access Protocol), IIOP (Internet Intra-ORB Protocol), and Net8™.

Oracle Wallet Manager provides secure management of PKI (public key infrastructure)-based user credentials: a user's private key, his certificate, and a list of trustpoints, the list of root certificates that the user trusts. Wallets are protected using password-based, strong encryption.

In most cases, a user need never access a wallet once it has been configured, but can easily access his wallet using Oracle Enterprise Login Assistant, a very simple-to-use login tool that hides the complexity of a private key and certificate from users. Once users have securely opened their wallets, they can then connect to multiple databases over SSL, without providing additional passwords. This provides the benefit of strong authentication as well as single sign-on.

Remote Authentication

Oracle Advanced Security supports remote authentication of users through RADIUS, a standard lightweight protocol used for user authentication, authorization, and accounting. RADIUS, a proposed standard of the Internet Engineering Task Force (IETF), is a popular means of enabling remote authentication of users. For example, a user accessing his corporate network remotely first authenticates himself to RADIUS; then, after successful authentication, the user is able to access applications within his corporate network.

Oracle Advanced Security provides an interface which can be used with any third-party authentication service that supports the RADIUS protocol. The advantage to customers is that multiple authentication devices (for example, tokens or smart cards) may be used for authentication to the Oracle8*i* database, as long as the mechanism or device supports the RADIUS protocol.

Authentication Through a Middle Tier

In applications which use a heavy middle tier, such as a transaction processing monitor, it is important to be able to preserve the identity of the client connecting to the middle tier. Yet, one advantage of a middle tier is connection pooling, to allow multiple users to access a data server without each of them needing a separate connection. In such environments, you need to be able to set up (and break down) connections very quickly, without the overhead of establishing a separate, authenticated database session for each connection. For these environments, Oracle8*i* offers n-tier authentication, "lightweight session" creation via the Oracle® Call Interface, so that applications can have multiple user sessions within a single database session. These "lightweight sessions" allow each user to be authenticated by a database password, without the overhead of a separate database connection, as well as preserving the identity of the real user through the middle tier. See "Authentication through a Middle Tier," described later in this paper.

Mutual Authentication For Secure Distributed Computing

While user authentication is important, it is equally important in distributed systems to ensure that a number of network principals — including application servers, web servers, and database servers — are who they say they are. For example, database A, attempting to connect to database B, needs assurance that database B really is database B, just as database B needs to be sure of database A’s identity.

Oracle8*i* enables secure distributed transactions — without compromising user credentials — by means of mutual authentication of databases, and by strong user authentication without disclosure of credentials. Mutual database authentication and strong user authentication are accomplished by industry-standard X.509 (version 3) certificates, without using passwords or any other “hard-coded,” potentially vulnerable means of authentication. Furthermore, administrators can configure their systems so that databases are only trusted to connect as certain users. For example, an AP application might need to retrieve information about employees from the HR database in order to perform expense reporting processing. Not only could the AP and HR databases mutually authenticate, but the HR database could grant access to only those users in AP who need to query the employee information in order to process expense reports.

Privileges That Protect Data

To insure data security, Oracle8*i* implements "security by default." A user can only perform an operation on a database object (such as a table or view) if that user has been authorized to perform that operation. A privilege is an authorization to perform a particular operation; without privileges, a user cannot access any information in the database. To ensure data security, a user should only be granted those privileges that he needs to perform his job functions. This is known as the principle of “least privilege.”

To allow you to grant users only those specific privileges they need to perform their jobs, and not any more, Oracle8*i* provides a large number of very granular privileges. These privileges fall into two categories: system privileges and object privileges.

System Privileges

A system privilege authorizes a user to perform a specific operation. One example of a system privilege is the CREATE USER privilege, which allows a user to create a database username; another is SELECT ANY TABLE, which allows a user to query any table in the database. Oracle8*i* provides over 100 different system privileges, such as permission to connect to the database and permission to change a table's attributes. A privilege can be granted to a user “with ADMIN option.” This allows the grantee authority to further grant and revoke privileges from other users.

Object Privileges

An object privilege authorizes a user to perform a specific operation on a specific object. For example, you can grant a user the ability to select from the EMP table by granting him the SELECT privilege on that table. With this privilege, the user can query the EMP table but cannot query any other tables in the database nor update the EMP table. You can also grant object privileges "with GRANT option." This allows the grantee authority to further grant the object privilege to other users. Oracle8*i* provides a varying number of object privileges per object type, such as permission to insert into a table and permission to select from a sequence.

By providing these two types of very granular privileges, Oracle8*i* allows you to implement separation of function and to control access to information at a very fine level, ensuring that database users are only authorized to perform those specific operations required by their job functions. In addition, other Oracle8*i* features (like roles and stored procedures, described later in this paper) not only allow you to control which privileges a user has, but under what conditions he can use those privileges.

Secure Metadata

Oracle8*i* also provides protection for the data dictionary, ensuring that only those individuals making a database administrator-privileged connection can alter the data dictionary. In Oracle8*i*, users granted ANY privilege (such as ALTER ANY TABLE, DROP ANY VIEW) can exercise these privileges on any appropriate object in any schema, except the SYS schema, which includes the data dictionary. This allows developers and others who need privileges on objects in multiple schemas (e.g., ALTER ANY TABLE) to continue to have that access via ANY privileges, while ensuring that they do not inadvertently alter the data dictionary. Users making SYS-privileged connections (that is, connecting as SYSDBA or SYSOPER) are able to modify the data dictionary, as one would expect a DBA to be able to do.

Views To Customize Access To Information

While privileges allow you to control which operations a user can perform on database objects, views allow you to further limit the data that a user can access within these objects. A view is a content- or context-dependent subset of one or more tables (or views). For example, you can define a view that allows a manager to view only the information in the EMP table that is relevant to employees in his own department. The view may contain only certain columns from the base table (or tables), such as the example below, in which only the employee name and salary information are contained in a view. Content may also be limited to a subset of the rows in the base table, such as a view of the employee table which contains records for employees assigned to department 20.

Similarly, you can define a view that allows payroll clerks to update payroll information on certain days of the month only. This flexibility allows you to restrict the data that a user can see or modify to only that data that he truly needs to access, at only the times that access is appropriate. This allows you to enforce your unique business rules within the database. View can be created with additional business considerations in mind. For example, views may be created "with check option," which enforces that inserts and updates performed through the view must be accessible by the view query itself. This helps ensure data consistency from the user's viewpoint.

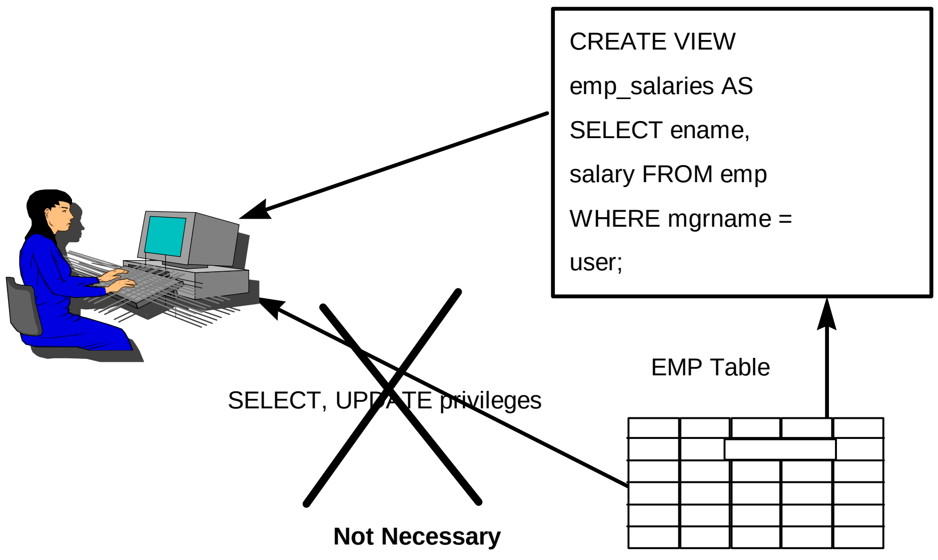


Figure 1: A View Controlling Salary Access By Manager

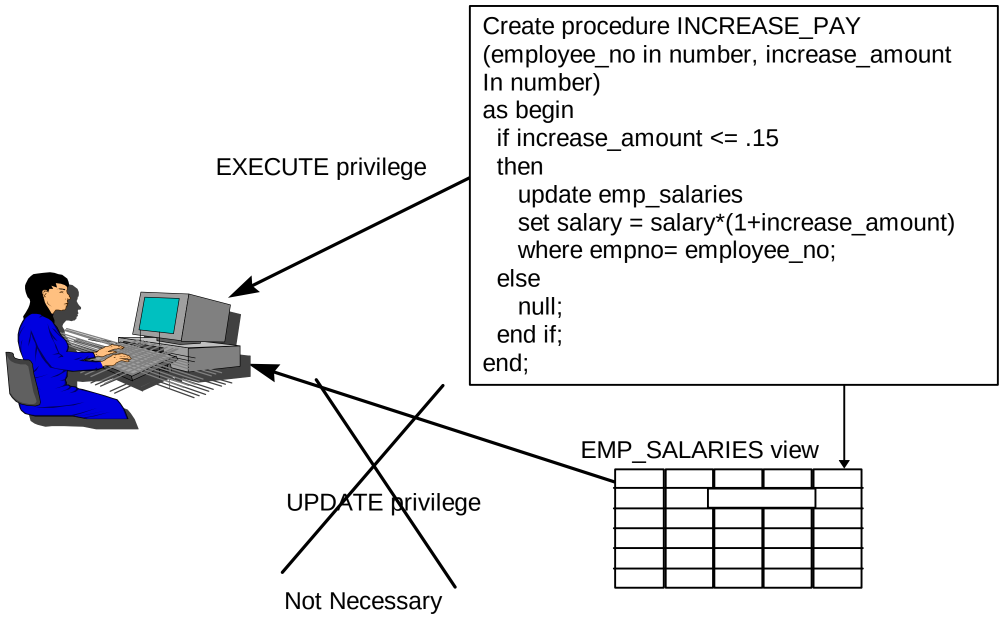
Stored Procedures To Customize Operations On Data

Oracle8*i* stored procedures offer another powerful and flexible way for you not only to limit those privileges a user has and the data that he can access, but to define a limited set of related operations that he can perform within the database. It is often desirable to encapsulate business rules into stored procedures for several reasons. One of them is that, if security is written in the front-end application, the user can bypass all the security of the application if the user has direct privileges in the database. Another reason is that stored procedures help enforce least privilege as well as business integrity, by ensuring that users have the minimum privileges they need to perform their job functions, and only access data according to well-formed business rules.

A package is a group of one or more stored procedures that are stored and managed together. Stored procedures and functions are sets of PL/SQL™ (Oracle's procedural language) or Java™ statements stored in compiled form within the database. You can define a procedure so that it performs a specific business function, then grant a user the ability to execute that procedure without granting him any access to the objects and operations that the stored procedure uses. This prevents users from exercising privileges to perform operations outside of the context of the pre-defined authorized procedure.

For example, the INCREASE_PAY stored procedure illustrated in Figure 2 allows managers to increase their employees' salaries. By executing this stored procedure, managers are allowed to increase employees' salaries by no more than 15 percent.

While you could have just granted these managers the ability to update the EMP_SALARIES view, the stored procedure allows you to enforce your business rules within the database by restricting managers from giving their employees increases that violate these rules. Note that the managers need not have access to the EMP_SALARIES view in order to execute this procedure; because a stored procedure performs an explicitly defined operation, users only need permission to execute this procedure, not permission to access the underlying objects. This prevents users from accessing the procedure's underlying objects outside the context of your business rules.



```
Create procedure INCREASE_PAY
(employee_no in number, increase_amount
In number)
as begin
  if increase_amount <= .15
  then
     update emp_salaries
     set salary = salary*(1+increase_amount)
     where empno= employee_no;
  else
     null;
  end if;
end;
```

Figure 2. Stored Procedure To Update Salary

Flexible Procedures To Lower Cost of Ownership

The type of procedure described above relies on a "definer's rights" privilege model; for example, users who have EXECUTE permission on Chuck's (the definer's) procedure access Chuck's data with Chuck's privilege set, for the duration of the transaction only. "Definer's rights" procedures are useful for encapsulation of privileges within a business context; that is, users need not have direct privilege on objects, merely the privilege to execute a procedure which accesses objects according to well-defined business rules.

However, object-oriented technology and the use of new programming languages such as Java require a more flexible privilege model, in which business logic is separate from data and the privileges required to access an object. For example, an Enterprise JavaBeans™ that updates a bank account balance should update Jane's account balance if Jane accesses the bean, but John's account balance if John accesses the bean. Furthermore, the Enterprise JavaBeans may be deployed in a bean store, and the beans may actually act upon different databases, or different schemas within the same database. Alternatively, developers of data cartridges wish to deploy application libraries, in which business logic must remain independent of specific users' privileges. To support these requirements, Oracle8*i* extends its privilege model by offering “invoker’s rights” procedures, available in both PL/SQL and Java, which execute with an invoker’s privilege set, on an invoker’s schema.

Invoker’s rights procedures enable organizations to lower their cost of deploying applications, since business logic — for example, a procedure which updates account balances — is not tied to a particular user’s privilege set or a particular schema, and thus can be used (and reused) by many applications and users. For example, an organization may have a common set of applications which multiple divisions use, but the data upon which the applications act are separated from one another. Division 1 employees never access Division 2’s data, and vice versa. One approach to this problem would be to physically separate data on different servers, which is expensive, and makes it difficult to do necessary summaries at a corporate level. Another approach is to maintain the data of each application in a separate schema, and have the application reside in an application-owned schema. Invoker’s rights procedures enable users from each division to access the same application, while acting upon their own data only. Invoker’s rights procedures thus enhance the ability of organizations to deploy common applications which nonetheless “act” differently for different sets of users. The result is stronger security at a lower cost of deployment.

The Virtual Private Database

Giving customers and partners direct access to mission-critical systems over the Internet may yield reduced cost, better service, and more timely information, but it also offers new challenges. Organizations must not only keep data safe from prying eyes, but they must segregate data appropriately, often to the level of individual customers or users. Also, many companies are interested in providing Internet “hosting” environments, with a well-designed and well-managed computing infrastructure, but must keep the data of each “hosted” corporation separate and secure from each other, while allowing customizations and data access methods which best meet their individual needs.

Within the Intranet, organizations continue to struggle with traditional access control problems, such as the classic “application security problem”: when access control is embedded in an application, users who have access to ad-hoc queries or reporting tools bypass the security mechanisms of the application.

Oracle8*i* addresses these diverse security needs by introducing the Virtual Private Database — server-enforced, flexible, fine-grained access control, together with a secure application context, enabling multiple customers and partners to have secure direct access to mission-critical data. The Virtual Private Database enables, within a single database, per-user or per-customer data access with the assurance of physical data separation. For Internet access, the Virtual Private Database can ensure that online banking customers see only their own accounts, and that web storefront customers see their own orders only. Web hosting companies can maintain multiple companies’ data in the same Oracle8*i* database, while allowing each company to see only their own data.

The Virtual Private Database enables fine-grained access control by associating one or more security policies with tables or views. Direct or indirect access to a table with an attached security policy causes the data server to consult the policy function. The policy function returns an access condition known as a predicate (a WHERE clause) which the data server appends to the SQL statements, dynamically modifying the user’s data access. For example, if an organization’s security policy is that customers can see their own orders, a user issuing the following query:

```
    SELECT * FROM orders;
```

could have her query transparently and dynamically rewritten by Oracle8*i* as follows:

```
SELECT * FROM orders WHERE cust_num = SYS_CONTEXT (‘userenv’,
‘session_user’);
```

This limits access to only those orders for which the customer matches the logged-in user.

The Virtual Private Database enables dynamically modified data access, transparently to both users and applications, based on any criteria; an organization can have different access conditions per user, per group of users, or per application.

Flexible Implementation

The Virtual Private Database offers flexible policy implementation, to allow customers to fine-tune their security policies based on their specific needs:

- *Attach security policies to tables or views.* Many applications already use views for security reasons, or to enforce business rules. Attaching security policies to either views or tables allows organizations to add fine-grained access to their existing applications without completely rewriting them.
- *Add security policies to only those tables or views where it is needed.* For example, to implement the policy ‘customers can see only their own orders,’ one need only add security policies to the ORDERS and ORDER_LINES table.
- *Enable different policies for different types of access, (e.g., select, insert, delete, and update).* For example, you could implement a policy on the EMP table that enables users to query name and address information for any employee, but allows them to update only their own records.
- *Add multiple policies per table.* For example, a hosting application may allow different companies’ HR systems to enable different access control conditions. Companies can add additional security policies on top of the base HR application security policy (e.g., that data access is limited by company), without affecting the base security enforcement and data separation policies.

Context-Based Security Enforcement

To make the Virtual Private Database easy to implement, Oracle8*i* offers application contexts: secure, application-specific attributes on which you can base your fine-grained access control policies. Application contexts are completely user-definable, as are their attributes. A human resources application may base its security policy on ‘organization,’ ‘employee number,’ and ‘position.’ For example, a user in the ‘manager’ position can see the employee records of all employees in his ‘organization,’ while a user in the ‘employee’ position can only see and update records matching his own ‘employee number.’ Alternatively, a general ledger application may base its security policy on ‘set of books,’ and ‘cost center.’ You can use application contexts within policy functions to determine the correct access condition (predicate) to return. You can also use application contexts within a predicate. Oracle8*i* ensures that application contexts are secure, by enforcing that only trusted packages implement them and can set context values.

Oracle8*i* also provides access to session primitives — information the database maintains about a user session — which can be used for access control. The USERENV application context provides access to these session primitives, including external name, proxy user and userid, protocol, port number, and full DN (distinguished name) from an X.509 certificate.

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 DN to perform access control; for example, you could use the OU (Organizational Unit) component of a DN to limit users to viewing their own organization's records only.

Scaleable Security

Fine-grained access control is highly scaleable; rewritten statements are fully parsed, optimized and available to be shared by other users. Use of application context with fine-grained access control offers even greater performance benefits, because an application context functions as a secure data cache. For example, if you were to rewrite a query to limit data access based on a user's position, organizational unit, and employee number, you could either use a subquery to retrieve all these values from metadata tables (which may involve several full table scans), or you could retrieve these values into an application context and reference the context within your security policy whenever you need to access attributes. As a result, you can have highly granular access control, with excellent performance.

Strong, Server-Enforced Security

The Virtual Private Database provides 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.
- *Eliminate the "Application Security Problem."* Users can no longer bypass security policies embedded in applications because security policies are associated directly with 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.
- *Enable Applications You Could Never Build Before*. In the past, organizations could not give customers and partners direct access to their production systems, because there was no way to secure the data. Internet hosting companies could not 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 the Virtual Private Database gives you server-enforced, fine-grained access control with the assurance of physical data separation.

Triggers To Customize Functionality

Like stored procedures, database triggers are user-defined sets of PL/SQL or Java statements, also stored in compiled form. While users explicitly execute stored procedures, database triggers are automatically executed (or "fired") within the data server based on pre-specified events. A trigger is defined to execute either before or after an insert, update, or delete, so that when that operation is performed on that table, the trigger automatically fires. Four types of triggers are available for definition on a table: BEFORE statement, BEFORE row, AFTER statement, and AFTER row. Statement triggers are executed once regardless of the number of rows affected by the triggering statement. Row triggers are fired once for each row affected by the triggering statement.

The security benefits of database triggers are similar to those of stored procedures: more granular access control and consistent rule enforcement. In addition to those benefits, database triggers allow you to perform behind-the-scenes operations based on user activity. For example, you could define a BEFORE UPDATE trigger on the EMP table that automatically records the existing values in the table before a user updates them. That way, you have a record of both the old and new values in any updated rows. You can also define multiple triggers of each type (statement or row) on a single table, to audit several different types of operations. Triggers can be used to apply security rules to the database. For example, if employee salary information should only be updated on a weekday between 8 a.m. and 6 p.m., a trigger can be defined to implement this business rule:

```
CREATE TRIGGER check_salary_access
before delete or insert or update
ON scott.emp
BEGIN
/* If today is a Saturday or Sunday, then return an error.*/
IF(TO_CHAR(SYSDATE, 'DY') = 'SAT' OR
      TO_CHAR(SYSDATE, 'DY') = 'SUN')
      THEN raise_application_error( -20501,
      'May not change employee table during the weekend');
ENDIF;
/*If the current time is before 8:00AM or after 6:00PM, then
return an error.  */
IF (TO_CHAR(SYSDATE, 'HH24') < 8 OR
      TO_CHAR(SYSDATE, 'HH24') >=18)
      THEN raise_application-error(-20502,'May only change
employee
      table during working hours');
ENDIF;
END;
```

All actions and checks done as a result of the SQL statement in a trigger must succeed. If any step is not successful, then all transactions are rolled back, ensuring data integrity.

While database triggers allow you to extend security based on actions involving specific database tables, event triggers allow you to extend security (e.g., access control and auditing) on larger events occurring within the database. Oracle8*i* provides event triggers on multiple database events, including login, logoff, startup, shutdown, as well as create, alter, and drop. Event triggers may be defined at the database level (e.g., startup), or for individual schemas (e.g., CREATE statements in the Order Entry schema).

For example, you could enable security policies immediately on login, based on how a user logged in or where he connected from. Or, you could use a login trigger to immediately set an application context for a user, to limit his access to data. You could use a login trigger to automatically enable more stringent auditing if a user connects to the database outside normal working hours, and disable these auditing options with a logoff trigger. Event triggers can be used to extend the innate security mechanisms of the Oracle8*i* data server, giving organizations more control over how and when users access data.

Roles To Manage Privileges

While Oracle8*i*'s granular privileges let you closely restrict the types of operations a user can perform in the database, managing these privileges may be complex. For example, if ten payroll clerks are responsible for maintaining payroll information, you would be required to grant each of these ten users the privileges necessary for him to perform his job. If their managers also must have access to information, you must grant these privileges to the managers, as well. If you decide to reduce the number of privileges these users require, you must revoke privileges from each of these users individually, making privilege management time-consuming and complex.

To address the complexity of privilege management, Oracle7™ introduced roles. Roles are user-defined collections of privileges that can be granted to and revoked from users, and even from other roles. For example, you can create the PAYROLL_CLERK role, grant it all privileges necessary for payroll clerks to perform their jobs, then simply grant this single role to all payroll clerks. In addition, you can create the PAYROLL_MANAGER role, grant it the PAYROLL_CLERK role and any other necessary privileges, then grant it to all payroll managers. To later grant an additional privilege to all payroll clerks and their managers, you need only grant an additional privilege to the PAYROLL_CLERK role; similarly, to revoke a privilege from all payroll clerks and managers, you need only revoke the privilege from the role. A role can also be defined to prompt the user for a password when that role is invoked, thus providing another layer of security for the system.

In addition to using roles to simplify privilege management, you can use roles to restrict the set of privileges accessible to a user at any time. For example, you can specify "default" roles that are enabled automatically for a user whenever he connects to the database, and specify additional roles that can only be enabled explicitly (by the user or within an application). You can also explicitly disable a role for a user to prevent him from using a certain collection of privileges when it is no longer appropriate (such as when he changes jobs). In addition, a role can be dropped completely from the database, making it no longer available to any user.

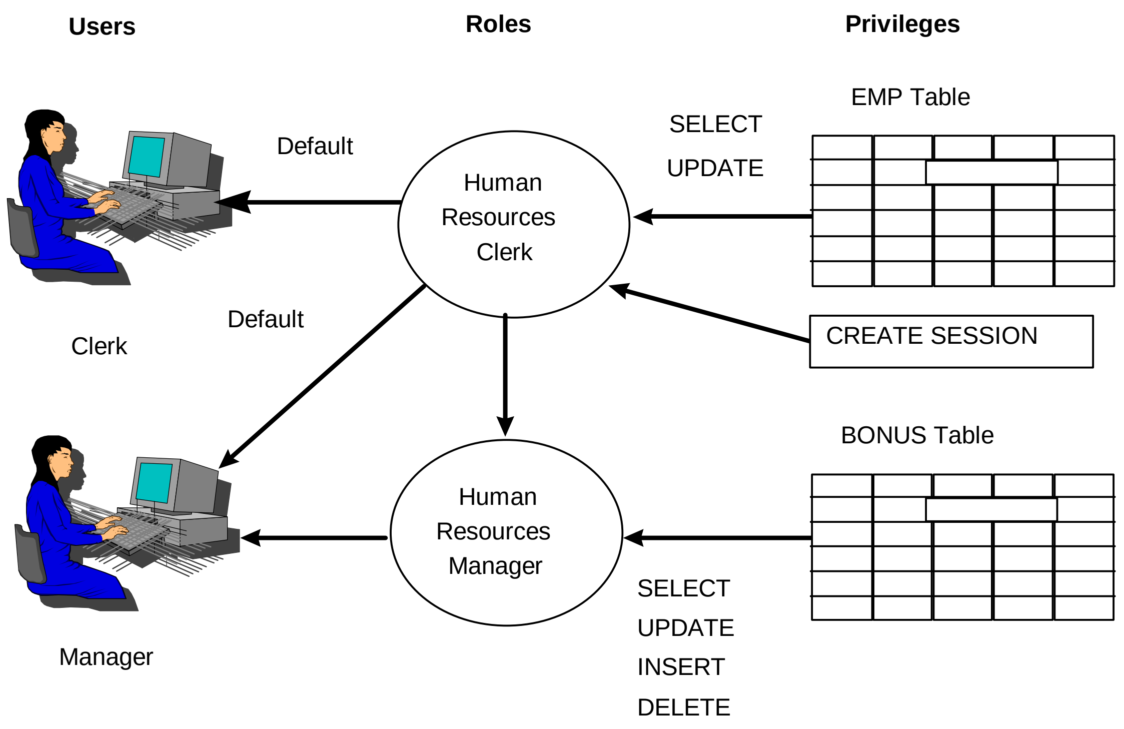


Figure 3: Use Roles To Assign Users Groups of Privileges

Roles To Manage Application Security

One beneficial use of the ability to dynamically enable and disable roles is to associate a role with an application. You can specify that a certain role be enabled for authorized users at the beginning of the application, then disabled at the end of the application. This restricts users from exercising the privileges within the role outside of the application. For example, you can enable the PAYROLL_CLERK role at the beginning of the payroll application, then automatically disable it when a payroll clerk exits from the application. This ensures that payroll clerks do not use the privileges of the role in any way other than that allowed by business rules defined within the payroll application.

Roles can also be useful in managing privileges in an application development environment. For example, certain privileges can be granted to developers to enable them to create their own objects. These privileges are not required by users of an application but are needed by developers. You can also associate a role with a database tool, allowing you to control which operations a user can perform using that tool. For example, you can associate the database roles PAYROLL_CLERK and PAYROLL_MANAGER with the menu roles in Oracle® Developer so that the tool only displays those menu entries that are accessible to clerks or managers, based on the enabled roles.

Enterprise Roles For Centralized Privilege Management

The challenge of managing user accounts and privileges is magnified in large enterprises, which often have a number of employees dedicated to creating user accounts, assigning privileges to them, and reassigning privileges as necessary. To address this need, Oracle8 introduced enterprise roles: a container of one or more global roles (encompassing one or more data servers), centrally administered, maintained in a proprietary data schema. Through Oracle Advanced Security, Oracle8*i* extends the benefits of enterprise roles by storing and retrieving them from Oracle Internet Directory (or selected other LDAP v3-compliant directory servers) via LDAP, an Internet standard for directory access.

Enterprise roles enable centralized authorization of users; for example, a user may be granted the enterprise role "HR Clerk," which contains the global role "HR User" on the Human Resources database, and the "Employee" global role on the Corporate Information database. If a user changes jobs, an administrator can simply change his enterprise role assignment, which alters his privileges in multiple databases throughout the enterprise. Also, you can add capabilities to enterprise roles (granted to multiple users) without having to update the authorizations of each user independently.

User/Schema Separation

A benefit of managing users in a directory is that it may reduce the number of user accounts needed. In most cases, users do not need their own accounts – or their own schemas – in a database. Typically, users merely need to access an application schema. For example, suppose users John, Firuzeh and Jane are all users of the Payroll application, and they need access to the Payroll schema on the Finance database. None of them needs to create their own objects in the database; in fact, they need only access Payroll objects. For most applications, users should be able to share a schema since they don't need to create their own objects in the database.

Oracle Advanced Security supports mapping many enterprise users to the same *shared schema* on an individual database. This separation of users from schemas reduces the cost of user administration. Instead of creating a user account (that is, a user schema) in each database a user needs to access, as well as creating the user in the directory, you can create a user once, in the directory, and "point" the user at a shared schema that many other enterprise users can also access. In the previous example, if John, Firuzeh and Jane all access the Sales database, you need only create a single schema, e.g. 'SALES_APPLICATION' which all three users can access, instead of creating an account for each user on the Sales database.

User/schema separation reaps the benefit of deploying a directory within the enterprise. Centralized user administration and far fewer user accounts provides thus provides greater security as well as lower cost of ownership.

Single Station Administration

Managing thousands of user accounts is one of the largest administration challenges facing large organizations. Creating user accounts and assigning privileges is often a multi-step process, requiring multiple tools. Significant new functionality has been added in Oracle8*i* to address this need. Oracle Enterprise Security Manager (an extension to Oracle's traditional database security manager) provides "single station administration." From a single console, an administrator can perform the following:

- Create a enterprise users in Oracle Internet Directory
- Create a user in multiple Oracle8*i* databases
- Create a shared schema
- Map enterprise users to shared schemas
- Create enterprise roles that span multiple databases

- Assign one or more enterprise roles to a user

Oracle Enterprise Security Manager provides one tool to centrally manage enterprise users and their roles, as well as administer enterprise domains — groups of databases and enterprise roles — resulting in a lower cost of user administration throughout the enterprise. Another benefit of single station administration is that if security is easy to administer, organizations are more likely to implement security well throughout the enterprise.

Auditing To Monitor Database Activity

A critical aspect of any security policy is maintaining a record of system activity to ensure that users are held accountable for their actions. To address this requirement, Oracle8*i* provides an extensive audit facility.

Granular Auditing

The Oracle8*i* audit facility allows you to audit database activity by statement, by use of system privilege, by object, or by user. For example, you can audit activity as general as all user connections to the database, and as specific as a particular user creating a table. You can also audit only successful operations, or unsuccessful operations. For example, auditing unsuccessful SELECT statements may catch users on ‘fishing expeditions’ for data they are not privileged to see. You can also set default object auditing options so that new objects automatically have auditing enabled from object creation (for example, any new tables are audited automatically for unsuccessful selects). Audit trail records are stored in an Oracle8*i* table, making the information available for viewing through ad hoc queries or any appropriate application or tool.

Efficient Auditing

Oracle8*i* implements auditing efficiently: statements are parsed once for both execution and auditing, not separately. Also, auditing is implemented within the server itself, not in a separate, add-on server which may be remotely situated from the statements which are being executed (thereby incurring network overhead). The granularity and scope of these audit options allow you to record and monitor specific database activity without incurring the performance overhead that more general auditing entails. And, by setting just the options of interest to you, you avoid the “catch-all, and throw away” audit methods which intercept and log all statements, and then filter them to retrieve the ones of interest.

Extensible Auditing

To record customized information that is not automatically included in audit records, you can use triggers (described in "Triggers to Customize Functionality") to further design your own audit auditing conditions and audit record contents. For example, you could define a trigger on the EMP table to generate an audit record whenever an employee's salary is increased by more than 10 percent and include selected information, such as before and after values of SALARY:

```
CREATE TRIGGER audit_emp_salaries
AFTER INSERT OR DELETE OR UPDATE ON employee_salaries
for each row
begin
if (:new.salary> :old.salary * 1.10)
      then
      insert into emp_salary_audit values (
      :employee_no,
      :old.salary,
      :new.salary,
      user,
      sysdate);
      endif;
end;
```

Furthermore, you can use event triggers to enable auditing options for specific users on login, and disable them on logoff.

Oracle8*i* also gives you the option of sending audit records to the database audit trail or your operating system's audit trail, when the operating system is capable of receiving them. This option, coupled with the broad selection of audit options and the ability to customize auditing with triggers or stored procedures, gives you the flexibility of implementing an auditing scheme that suits your specific business needs.

Auditing For Three-Tier Applications

Many three-tier applications authenticate users to the middle tier, then the TP monitor or application server connects as super-privileged user, and does all activity on behalf of all users. With Oracle8*i*, you are not only able to preserve the identity of the real client over the middle tier and enforce “least privilege” through a middle tier, but you can audit actions taken on behalf of the user by the middle tier. Oracle8*i*’s audit records capture both the logged-in user (e.g., the TP monitor) who initiated the connection, and the user on whose behalf an action is taken. For example, to capture all SELECTs on the BONUS table done by a middle tier called ‘appsrv,’ you would enabled the following audit option:

```
AUDIT SELECT ON bonus BY appsrv ON BEHALF OF ANY;
```

Audit records capture both the user taking the action and the user on whose behalf the action was taken. Auditing user activity, whether users are connected through a middle tier or directly to the data server, enhances user accountability, and thus the overall security of multi-tier systems.

Active Auditing

While an auditing facility can record attempts to breach database security or actual breaches, they do not alert administrators or security officers that the breach is happening. In fact, it can be hours, days, or months before analysts detect a security breach by examining the audit trail. Consequently, if one of the purposes of your auditing policy is to detect potential breaches of security, you need an alarm facility to alert the appropriate administrator when the database or operating system detects suspicious behavior.

When you use database triggers to perform customized auditing, Oracle8*i* can send an alarm to a waiting process that a potential breach of security is occurring.

AUTHENTICATION THROUGH A MIDDLE TIER

The growth of three-tier systems (for example, browser to application server to database) has increased dramatically with the growth of the Internet. Indeed, three-tier applications are often referred to as the "Internet computing model." Oracle8*i* n-tier authentication addresses many security difficulties that arise in three-tier applications. It enables organizations to reap the benefits of Internet computing, while minimizing the security risks of three-tier systems. This section includes:

- Advantages of n-Tier Authentication
- Security Challenges of Three-tier Computing
- Oracle8*i* n-Tier Authentication Solutions

Advantages of n-Tier Authentication

Three-tier systems provide many benefits to organizations. Application servers and web servers enable users to access data stored in legacy applications. Users like using a familiar, easy-to-use browser interface. Organizations can separate application logic from data storage, partitioning the former in application servers and the latter in databases. Organizations can also lower their cost of computing by replacing many "fat clients" with a number of "thin clients" and an application server.

In addition, Oracle n-tier authentication delivers the following security benefits:

- A limited trust model, by controlling the users on whose behalf middle tiers can connect, and the roles the middle tiers can assume for the user
- Scalability, by supporting lightweight user sessions through OCI, and eliminating the overhead of reauthenticating clients
- Accountability, by preserving the identity of the real user through to the database, and enabling auditing of actions taken on behalf of the real user

Security Challenges of Three-tier Computing

While three-tier computing provides many benefits, it raises a number of new security issues:

- Who Is the Real User?
- Does the Middle Tier Have Too Much Privilege?
- How to Audit? Whom to Audit?
- Can the User Be Reauthenticated to the Database?

Who Is the Real User?

Most organizations want to know the identity of the actual user who is accessing the database, for reasons of access control or auditing. User accountability is diminished if the identity of the users cannot be traced through all tiers of the application.

Furthermore, if only the application server knows who the user is, then all per-user security enforcement must be done by the application itself. Application-based security is very expensive. If each application that accesses the data must enforce security, then security must be reimplemented in each and every application. It is often preferable to build security on the data itself, with per-user accountability enforced within the database.

Does the Middle Tier Have Too Much Privilege?

Some organizations are willing to accept three-tier systems within the enterprise, in which "all-privileged" middle tiers, such as transaction processing (TP) monitors, can perform all actions for all users. In this architecture, the middle tier connects to the database as the same user for all application users. It therefore needs to have all privileges that application users need to do their jobs.

This computing model may be undesirable in the Internet, where the middle tier resides outside, on, or just inside a firewall. More desirable, in this context, is a limited trust model, in which the identity of the real client is known to the data server, and the application server (or other middle tier) has a restricted privilege set. Also useful is the ability to limit the users on whose behalf a middle tier can connect, and the roles the middle tier can assume for the user. For example, many organizations would prefer that users have different privileges depending on where they are connecting from. A user connecting to a web server or application server on the firewall might only be able to use very minimal privileges to access data, whereas a user connecting to a web server or application server within the enterprise might be able to exercise all privileges she is otherwise entitled to have.

How to Audit? Whom to Audit?

Accountability through auditing is a basic principle of information security. Most organizations want to know on whose behalf a transaction was accomplished, not just that a particular application server performed a transaction. A system must therefore be able to differentiate between a user performing a transaction, and an application server performing a transaction on behalf of a user.

Auditing in three-tier systems should be tied to the issue of knowing the real user: if you cannot preserve the user's identity through the middle tier of a three-tier application, you cannot audit actions on behalf of the user.

Can the User Be Reauthenticated to the Database?

In client/server systems, authentication tends to be straightforward: the client authenticates to the server. In three-tier systems authentication is more difficult, because there are several potential authentications.

- Client to Middle Tier Authentication
- Middle Tier to Database Authentication
- Client Reauthentication Through Middle Tier to Database

Client to Middle Tier Authentication Client authentication to the middle tier is clearly required if a system is to conform with basic security principles. The middle tier is typically the first gateway to useful information that the user can access. Users must, therefore, authenticate to the middle tier. Note that such authentication may be mutual; that is, the middle tier authenticates to the client just as the client authenticates to the middle tier.

Middle Tier to Database Authentication Since the middle tier must typically initiate a connection to a database to retrieve data (whether on its own behalf or on behalf of the user), this connection clearly must be authenticated. In fact, Oracle8*i* does not allow unauthenticated connections. Again, middle tier to database authentication may also be mutual.

Client Reauthentication Through Middle Tier to Database Client reauthentication from the middle tier to the database is problematic in three-tier systems. The username may not be the same on the middle tier and the database. In this case, users may need to reenter a username and password, which the middle tier uses to connect on their behalf. Or, more commonly, the middle tier may need to map the username provided, to a database username. This mapping is often done in an LDAP-compliant directory service, such as Oracle Internet Directory.

For the client to reauthenticate himself to the database, the middle tier either needs to ask the user for a password (which it then must be trusted to pass to the database), or the middle tier must retrieve a password for the user and use that to authenticate the user. Both approaches involve security risks, because the middle tier is trusted to handle the user's password properly, and not use it maliciously.

One of the only cases for which reauthentication does not involve trusting the middle tier occurs when a middle tier downloads an applet to a client, and the client connects directly to the database via the applet. In this case, the application server is literally just that: it serves the application (applet) to the user, and has no part in further authentication of the user.

Reauthenticating the client to the back-end database is not always beneficial. First, two sets of authentication handshakes per user involves considerable network overhead. Second, you must trust the middle tier to have authenticated the user. (You clearly must trust the middle tier if it retrieves or otherwise is privy to the user's password.) It is therefore not unreasonable for the database to simply accept that the middle tier has performed proper authentication. In other words, the database accepts the identity of the real client without requiring the real client to authenticate herself.

For some authentication protocols, client reauthentication is just not possible. For example, many browsers and application servers support the Secure Sockets Layer (SSL) protocol. Both the Oracle8*i* database (through Oracle Advanced Security) and Oracle Application Server support the use of SSL for client authentication. However, SSL is a point-to-point protocol, not an end-to-end protocol. It cannot be used to reauthenticate a browser client (through the middle tier) to the database.

The reason for this is that a user cannot securely give up his private key to the middle tier in order for the reauthentication of the client to occur. Once the user's private key is compromised, the user's very identity is compromised. In addition, there is no way to "tunnel" through a middle tier so that the authentication of the browser client to the database can occur directly.

In short, organizations deploying three-tier systems require flexibility as regards reauthentication of the client. In some cases, they cannot reauthenticate the client; in other cases, they may choose whether or not to reauthenticate the client.

Oracle8*i* n-Tier Authentication Solutions

The following sections explain how Oracle8*i* addresses each of the challenges listed above.

- Passing Through the Identity of the Real User
- Limiting the Privilege of the Middle Tier
- Reauthenticating the Real User
- Auditing Actions Taken on Behalf of the Real User

Passing Through the Identity of the Real User

Many organizations want to know who the real user is through all tiers of an application, without sacrificing the benefits of a middle tier. Oracle8*i* provides the ability to preserve client identity through the Oracle Call Interface (OCI). OCI enables a middle tier to set up, within a single database connection, a number of "lightweight" user sessions, each of which uniquely identifies a connected user. These lightweight sessions reduce the network overhead of creating separate network connections from the middle tier to the database. The application can switch between these sessions as required to process transactions on behalf of users.

The full authentication sequence from the client to the middle tier to the database occurs as follows:

1. The client authenticates to the middle tier, using whatever form of authentication the middle tier will accept. For example, the client could authenticate to the middle tier using a username/password, or an X.509 certificate by means of SSL.
2. The middle tier authenticates itself to Oracle8i, using whatever form of authentication Oracle8i will accept. This could be a password, or an authentication mechanism supported by Oracle Advanced Security, such as a Kerberos ticket or an X.509 certificate (SSL).
3. The middle tier then creates one or more sessions for users using the Oracle Call Interface. The lightweight session information must include username as a minimum. The middle tier may optionally provide a password for the client, and the roles for the client.
4. Since the database cannot require the middle tier to provide a password for the client, authentication is performed by OCI. To create the session for the client, the middle-tier server calls the `OCISessionBegin` function. Prior to calling `OCISessionBegin`, the `OCIAttrSet` function is called to provide the needed information about the client to the middle tier server. It is called in turn with the following attributes:
   - OCI_ATTR_USERNAME — Sets the database user name of the client. This attribute is mandatory.
   - OCI_ATTR_PASSWORD —If the client has provided a database password to be validated by the database, then the middle tier server passes it along with the username. If this attribute is not provided, then it is assumed that the middle tier server has authenticated the client.
   - OCI_ATTR_PROXY_CREDENTIALS — This attribute tells the server that the client is connecting through a middle tier server.
   - OCI_ATTR_INITIAL_CLIENT_ROLES — If the middle tier server wants to activate a set of roles upon connecting as the client, then the list is passed along with this attribute.
5. The database verifies that the middle tier is privileged to create sessions on behalf of the user, using the roles provided. (See "Limiting the Privilege of the Middle Tier," below).

The `OCISessionBegin` call will fail if the application server is not allowed to proxy on behalf of the client by the administrator, or if the application server is not allowed to activate the specified roles.

Limiting the Privilege of the Middle Tier

"Least privilege" is the principle that users should have the fewest privileges necessary to perform their duties, and no more. As applied to middle tier applications, this means that the middle tier should not have more privileges than it needs. Oracle8*i* enables you to limit the middle tier such that it can connect only on behalf of certain users, using only specific roles.

For example, suppose that user Sarah wants to connect to the database through a middle tier, appsrv (which is also a database user). Sarah has multiple roles, but it is desirable to restrict the middle tier to exercise only the clerk role on her behalf. A DBA could effectively grant permission for appsrv to initiate connections on behalf of Sarah using her clerk role only, using the following syntax:

```
ALTER USER Sarah GRANT CONNECT THROUGH appsrv WITH ROLE clerk;
```

By default, the middle tier cannot create connections for any client. The permission must be granted on a per-user basis.

To allow `appsrv` to use all of the roles granted to the client Sarah, the following statement would be used:

```
ALTER USER sarah GRANT CONNECT THROUGH appsrv WITH ROLE ALL;
```

Each time a middle tier initiates a lightweight (OCI) session for another database user, the database verifies that the middle tier is privileged to connect for that user, using the role specified.

Reauthenticating the Real User

As described above, it is not always beneficial to reauthenticate users to the database after they have been authenticated by the middle tier. However, if you wish to do this for an added measure of security, you can pass the database the user's password using the `OCI_ATTR_PASSWORD` attribute of the `OCIAttrSet` call.

Auditing Actions Taken on Behalf of the Real User

The n-tier authentication features of Oracle8*i* enables you to audit actions a middle tier performs on behalf of a user. For example, suppose an application server `hrappserver` creates multiple lightweight sessions for users Ajit and Jane. A DBA could enable auditing for `SELECT`s on the `bonus` table that `hrappserver` initiates for Jane as follows:

```
AUDIT SELECT ON bonuses BY hrappserver ON BEHALF OF Jane;
```

Alternatively, the DBA could enable auditing on behalf of multiple users (in this case, both Jane and Ajit) connecting through a middle tier as follows:

```
AUDIT SELECT ON bonuses BY hrappserver ON BEHALF OF ANY;
```

This auditing option only audits `SELECT` statements being initiated by `hrappserver` on behalf of other users. A DBA can enable separate auditing options to capture `SELECT`s against the `bonus` table from clients connecting directly to the database:

```
AUDIT SELECT ON bonuses;
```

Objects For Rapid, Flexible Application Development

With each wave of technology, businesses find that their product cycles are shortened, competitive pressures have increased, and they must continually reinvent themselves to match the pace of change. Many design and development teams have concluded that developing applications in an object-oriented manner can meet the challenges of this rapid technological change. Oracle8 marked the metamorphosis of the industry-leading relational server into an object-relational server.

Oracle8*i* includes a set of features that enables the creation and manipulation of user-defined Object Types in the database, allowing designers to model the structures of complex real-world entities, as well as operations that might be performed on these entities. Oracle8*i*'s object constructs have a close correspondence with the relational constructs that Oracle's customers are familiar with. For example, a REF is very similar to a foreign key, methods are really PL/SQL stored procedures and Oracle8*i* provides the same transactional semantics and behavior on objects as it does on relational data. Perhaps most importantly, the security model that operates on object types is exactly the same that Oracle defined for relational tables.

Object Types

Oracle has extended SQL to allow users to define their own types (that represent their business objects), store them as base or native types within the database, and query, insert, and update them. Instances of object types may be stored as rows in tables (row objects), or specified as the data types of columns (column objects). They can contain one business object inside of another, point from one business object to another (using a pointer called a REF) and access and manipulate collections or sets of these objects.

As with relational data, there are both system privileges and object privileges which apply to object types. System privileges include the ability to create, alter, and drop types as well as the ability to execute a type, for instance, to use and reference named types in a schema. The only object privilege that applies to types is the execute privilege, which allows the grantee to use the type to define a table, define a column in a relational table, or declare a variable of the named type.

For example, many applications use address information: an order entry application (customers), an accounts payable application (vendors), and a human resources application (employees). Sam could create the address type address_type (which contains street, city, state, and zip information), and grant execute privilege on address_type to Jane. Jane can now use address_type as an embedded type in the order entry application she is developing. Other developers could also use address_type in their human resources and accounts payable applications.

Oracle8*i* prevents dropping or revoking a type if there are dependencies on it. For example, Sam is prevented from dropping type address_type if Jane has used address_type in her order entry application. (Sam can use the FORCE option to revoke execute privilege on a type even if there are dependencies, but this is generally not recommended.)

Object Views

Object views are an extension to the basic relational view mechanism, to provide an object abstraction over existing relational or object data. By using object views, relational data can be retrieved, updated, inserted and deleted as if such data were stored as objects. Object views can be constructed on both relational and object data, as well as other object views. Object views also enable the coexistence of object applications with relational data, which facilitates a smooth migration from relational applications to object-oriented applications. Using object views to group logically-related data can also lead to better database performance; relational data that make up a row of an object traverse the network as a unit, potentially saving many round trips.

A current limitation of views is that a view is not updateable if the view query contains joins, set operators, group functions, GROUP BY or DISTINCT. Since object views often include joins, Oracle8*i* provides a mechanism to overcome these obstacles: INSTEAD OF triggers. INSTEAD OF triggers provide a transparent way to update object or relational views. You write the same SQL data manipulation language (DML) (INSERT, DELETE and UPDATE) statement as for an object table. Oracle8*i* invokes the appropriate trigger instead of the SQL statement, and the actions specified in the trigger body take place. INSTEAD OF triggers can also be used to customize regular data manipulation operations, such as insert, delete, and update.

Functionality To Integrate Your Security Policies

Most installations have an overall system security policy, of which database security is only a part. It is critical that all components of your system, including operating systems, networks, and databases, work together to enforce a cohesive and consistent policy. For example, Oracle8*i* relies upon operating system security mechanisms to protect against unauthorized access to the operating system files which contain database objects.

Oracle8*i* allows you to enforce your own unique security policy and business rules within the database, and to integrate your database security policy with that of the rest of your system. Oracle8*i*'s granular privileges and flexible roles allow you to customize privilege sets within the database. In addition, you can tie database roles to operating system roles (such as UNIX groups and VMS process rights identifiers) so that you can consolidate privilege and role management outside of the database.

Oracle8*i* also offers integrated security and directory services, which enables you to centrally manage users and privileges, among other benefits. Integrated security and directory services are built using many industry standards, such as LDAP for directory access, SSL for authentication and encryption, and X.509 (version 3) certificates for user authentication and single sign-on. Consequently, organizations using non-Oracle directory services and certificate authorities are able to incorporate them into Oracle8*i*'s security and directory framework.

Oracle Advanced Security supports multiple third party authentication mechanisms, thereby providing the benefits of customer choice and security integration. Many of these services also provide single sign-on, which means that Oracle8*i* customers can participate in the resultant greater ease of administration and centralization of user management. Users, of course, are happy to have fewer passwords to remember.

Oracle8*i* provides the option of sending all database audit records to the operating system audit trail, allowing you to consolidate audit information from all applications in one location. For additional security, Oracle8*i* guarantees that data can not be accessed once it is deleted, also known as object reuse. Oracle8*i* guarantees object reuse by allocating space for use by an object only after all traces of remnant data are removed.

HIGH AVAILABILITY FOR MISSION-CRITICAL APPLICATIONS

Availability is often thought of as continuity of service, ensuring that a database is available 24 hours a day, seven days per week. However, there are security aspects to availability. For example, if a user is able to manipulate system resources in order to deny their availability to other users, he is breaching security. This is referred to as "denial of service." Multiple Oracle8*i* mechanisms — including resource limits and user profiles, online backup and recovery, and advanced replication — help provide uninterrupted database processing and minimize denial of service in order to support today's on-line transaction processing and decision support environments.

User Profiles

Resource limitation and user profile mechanisms prevent "run-away" queries, or more deliberate and malicious manipulation of system resources by a particular user. A user profile is a set of administrator-defined resource limits assigned to a username; through the use of user profiles, Oracle8*i* allows the database administrator to define and limit the amount of certain system resource available to a user. System resources that can be limited include:

- Total Connect and Idle Time

- Total Amount Of Logical Input Or Output
- Number of Concurrent, Multiple Sessions Per Username
- Amount of Memory Used
- Composite System Usage, Based On a Site-Defined Weighting of the Above.

Through user profiles, Oracle8*i* prevents resource hogs from denying service to other users, either inadvertently or maliciously.

Online Backup and Recovery

Oracle8*i* also ensures high availability by providing robust online backup and recovery, so that mission-critical applications are not inhibited by these necessary activities. Oracle8*i* provides an integrated method for creating, managing, and restoring backups of a database, providing greater ease of management and administration of the backup and recovery operations, while maintaining superior performance and increased availability of the database. Oracle8*i* databases can be backed up on-line, even during periods of peak transaction processing activity. Server-managed backup and recovery improves database administrator productivity as well as simplifying the backup and recovery process. Oracle8*i* backup and recovery allows backing up of the entire database, or a subset of the database, in one operation, and minimizes time needed for backup and restore operations by performing automatic parallelization of backups and restores. Oracle8*i* backup and recovery also supports sequential input/output (I/O) devices for output during backup and for input during restore operations. Tape backups are supported in conjunction with vendor-provided tape management systems.

Advanced Replication

Oracle8*i*'s advanced replication facilities can be used to increase the availability of systems by off-loading large scale queries from transaction processing databases. For example, large tables of customer purchasing data may be replicated to customer service databases, so that data-intensive queries do not contend with transactions against the same tables. Advanced replication facilities can also be useful in protecting the availability of a mission-critical database. For example, symmetric replication can replicate an entire database to a failover site should the primary site be unavailable do to a system or network outage. Advanced replication for both read and write access ensures data consistency; refresh groups preserve referential integrity and transaction consistency and the table snapshots of related master tables. For example, customers, orders, order lines are all related, so could be refreshed as a group.

Data Partitioning

Data partitioning in Oracle8*i* is a powerful tool for dramatic improvements in the manageability, performance, and scale of applications deployed using the Oracle8*i* data server. Oracle8*i* allows range partitioning of tables and multiple partitioning strategies for indexes, providing very large database support, and improves administrative operations. In the real world, media failure, access balancing for performance, and table de-fragmentation are just a few of the areas where partitioning can reduce the impact of a outage or increase availability under high loads.

Oracle8*i* with the Partitioning option supports all DML operations in parallel today. In addition, scans of indexes, export and import of table data, and estimating and calculating statistics can also be performed in parallel on individual partitions. Partitions can be loaded individually and in parallel, with or without index pre-creation. Loading, backup, recovery, computing statistics, and import and export are all supported on a per partition basis. These can be performed individually without interfering with operations underway on other partitions. With every operation available on a per-partition basis, it is possible to have truly dramatic performance improvements.

Advanced Technology That Meets Standards

To give you confidence that your database provides all of the functionality and security you need, it should meet all relevant standards. Oracle Corporation has designed Oracle8*i* to meet all relevant standards; additionally, Oracle Corporation is at the forefront of creating new technology and working with standards groups to extend current standards.

Oracle8*i* meets functionality and open systems standards as well as being designed to meet security standards. Oracle8*i* is fully compliant with ANSI/ISO SQL standards. In addition, Oracle Corporation proposed its roles facility to the ANSI/ISO X3H2 SQL standards committee, and it was accepted into the SQL3 specifications.

Oracle8*i* and Oracle Advanced Security support a number of relevant standards, including SSL for authentication, X.509v3, as well as Public Key Certificate Standards, such as PKCS#10, for certificate signing requests. Oracle Internet Directory provides an LDAP v3 implementation built on the Oracle8*i* database.

Oracle has participated in multiple security evaluations since the advent of Oracle7. Oracle7 has completed security evaluations at class C2 against the U.S. National Computer Security Center (NCSC) Trusted Computer System Evaluation Criteria (TCSEC or "Orange Book"), and Oracle7 has also been evaluated against the European Information Technology Security Evaluation Criteria (ITSEC) at assurance E3 (with functionality F-C2 in conjunction with an F-C2 operating system). Oracle7 and Oracle8 have also been evaluated against the Russian Criteria, at level III and level IV, respectively. Oracle7 has been evaluated and Oracle8 is in a Common Criteria evaluation against the Commercial Database Protection Profile. Oracle has also been a leader in adoption of the Common Criteria (CC), which has recently been adopted as ISO standard 15408, as well as developing multiple protection profiles. A benefit of a Common Criteria certificate is that it is recognized and accepted by multiple governments (US, UK, Germany, France, Netherlands, and Australia, to name a few).

Oracle Advanced Security is undergoing a Federal Information Processing (FIPS)-140 certification at level 2, with completion expected in Q4 1999.

A table of Oracle security evaluations' status follows:

Type of Evaluation	Server Release	Level	Status
TCSEC	Oracle7 release 7.0.13.1	C2	Completed
ITSEC	Oracle7 release 7.0.13.6	E3/F-C2	Completed
ITSEC	Oracle7 release 7.2.2.4.13	E3/F-C2	Completed
ITSEC	Oracle7 release 7.3.4	E3/F-C2	Completed
Common Criteria	Oracle7 release 7.2.2.4.13	EAL-4	Completed
Common Criteria	Oracle8 release 8.0.5	EAL-4	In evaluation
Common Criteria	Oracle8 release 8.1.6	EAL-4	In evaluation
Russian Criteria	Oracle7 release 7.3.4	III	Completed
Russian Criteria	Oracle8 release 8.0.3	IV	Completed

Database Encryption

Given the limitations of discretionary access control and the 'superuser' privileges typically enjoyed by Database Administrators (DBAs) and System Administrators, many organizations feel a need to safeguard data in the database via encryption. Database encryption can address threats to both the confidentiality and integrity of online data and data stored off-line, although it may not be the optimal solution to these threats. There are three broad categories of database encryption which this paper addresses: encryption of all data in on-line, operational environments; encryption of data stored off-line; and partial encryption of data in operational environments.

Full Database Encryption

Why might an organization consider encrypting an entire database? One reason is to limit the readability of the database files in the operating system. Clearly, access to database files in the operating system should be limited through groups or rights identifiers; however, an organization may also wish to make these files unreadable to a person or persons who otherwise has legitimate access to the database files, but has no database privileges, such as a System Administrator.

Encryption of an entire database is problematic. In an operational environment, encryption must not interfere with other access controls; meaning, it must not prevent users from accessing an object they are otherwise privileged to access. Otherwise, their ability to perform their jobs is impaired. For example, a user who has SELECT privilege on EMP should not be limited by the encryption mechanism from seeing all the data he is otherwise cleared to see. Consequently, there is little benefit to encrypting, (for example) part of a table with one key and part of a table with another key if users need to see all encrypted data in the table; it merely adds to the overhead of decrypting data before users can read it. Therefore, provided that access controls are implemented well, there is little additional security provided within the database itself from full database encryption; any user who has privilege to access data within the database has no more nor less privilege as a result of encryption.

If data within database tables is encrypted to provide additional security for database files, then the indexes which access those tables must also be encrypted, since they may also contain sensitive data.

The result of encrypting all data and all associated indexes is a drain on performance. For example, a user querying the EMP table must wait while the indexes are decrypted, the data in table EMP is decrypted, and the query is satisfied. The results of other operations which affect data — such as UPDATEs or INSERTs — must also be encrypted. Consequently, most organizations would find any additional security provided by full database encryption in an operational environment to be a poor tradeoff for the performance degradation that they would experience. Consequently, Oracle8*i* does not provide full database encryption.

Another drawback to full database encryption is the requirement to periodically change the encryption key to mitigate the threat of a compromised key. Changing the key requires that the entire database be decrypted and reencrypted using the new key or keys. This process is time-consuming and would likely have to be done when the data is not being accessed.

Off-Line Database Encryption

Some organizations who feel confident in the security of on-line data may wish to encrypt data stored off-line. For example, an organization may store backups for a period of six months to a year off-line in a remote location. Of course, the first line of protection is to secure the data in a facility to which access is controlled, a physical measure. In addition, there may be a benefit to encrypting this data before it is stored, and since it is not being accessed on-line, performance need not be a consideration. While Oracle8*i* does not provide this facility, there are vendors who can provide such encryption services.

> *A note of caution: Organizations considering this should thoroughly test that data which is encrypted before storage off-line can be decrypted and re-imported successfully before embarking on large-scale encryption of backup data.*

Partial Database Encryption

The growth of electronic commerce has resulted in an increase in the storage of highly sensitive information, such as credit card numbers, in the database. Countries with strict national privacy laws are often required to prevent national identity numbers from being viewed, even by DBAs or other "trusted" users. Companies with trade secrets, such as industrial formulas, may wish to zealously guard these valuable assets. Applications for which users are not database users may wish to store "application user" passwords, or session cookies, in encrypted form in the database.

Most issues of data security can be handled by Oracle8*i*'s authentication and access control mechanisms, ensuring that only properly identified and authorized users can access data. Data in the database, however, cannot normally be secured against the database administrator's access, since a DBA has all privileges.

For applications with special requirements to secure sensitive data from view, even from DBAs, Oracle8*i* provides a PL/SQL package (DBMS_OBFUSCATION_TOOLKIT) to encrypt (and decrypt) data, including string inputs and raw inputs, using the industry-standard Data Encryption Standard (DES), in exportable keylengths.

The ability to natively encrypt data in the server enables applications to guard their especially sensitive data. Furthermore, developers need no long "roll their own" encryption using algorithms they craft themselves, or download from the Internet.

In using the DBMS_OBFUSCATION_TOOLKIT package, key management must be handled programmatically; in other words, an application is responsible for generating an encryption key, passing it to the encryption package, and storing the encryption key securely. For example, the encryption key could be stored in an operating system file, or in a database table. Because of US government restrictions on the export of cryptographic products, developers cannot add their own encryption algorithms, alter the key length, or make multiple calls to the same function (thereby "double-encrypting" values) using this interface.

Network Encryption

Organizations operating in a distributed environment may have particular concerns about security which may necessitate encryption of data passing over a network. For these organizations, Oracle8*i* offers Oracle Advanced Security, which provides high speed data encryption over a network using such services as Secure Sockets Layer, the Internet standard for confidentiality of transmitted data.

To prevent modification or replay of data during transmission the Oracle Advanced Security can generate a cryptographically-secure message digest, which is included in each network packet. An immediate integrity check on each packet performed at the destination by the recipient makes it virtually impossible for an intruder to alter data without detection. To protect data from unauthorized viewing, Oracle Advanced Security includes an encryption module which uses the RSA Data Security RC4™ encryption algorithm as well as the Data Encryption Standard (DES). Using a secret, randomly-generated key, Oracle Advanced Security encrypts all data in a Net8 session — including all data values, SQL statements, and stored procedure calls and results — to fully safeguard data. The Oracle Advanced Security offers RC4 in 128-bit, 56-bit, and 40-bit key lengths, and DES in 56-bit and 40-bit key lengths. Oracle Advanced Security also provides high-strength encryption with Triple DES (3DES) supported with Secure Sockets Layer. The strength of encryption available to customers is limited by US export regulations.

Oracle Advanced Security also offers SSL encryption for Internet Intra-ORB Protocol (IIOP) communications, enabling secure Enterprise Java Beans (EJBs). Also, Oracle Advanced Security provides a Java version of its encryption libraries to secure thin JDBC connections. The Java implementation of Oracle Advanced Security provides DES encryption, with anonymous Diffie-Hellman key exchange, in 100% Java.

Oracle Advanced Security thus secures *all* protocols into the Oracle8*i* database, whether IIOP, thick or thin JDBC, or Net8.

Oracle Advanced Security has also completed the operational testing phase of FIPS-140 level 2 (Federal Information Processing Standard) certification, a United States government standard that relates to the security of cryptographic products. Completion of the FIPS-140 certification, which is expected in Q4 1999, is required by many organizations, among them the United States government and many financial institutions.

SUMMARY

Whether enterprises employ client-server or multi-tier architectures, data will continue to be stored and managed in database servers. With the expansion of the enterprise to customers and partners via the Internet or extranets, database security becomes an increasingly critical component of information systems. Oracle8*i* builds upon over 20 years of Oracle client-server expertise, providing robust, industry-standard security mechanisms to meet the challenges of securing the expanded enterprise.