Oracle® AI Database

Oracle Deep Data Security Guide

26ai
G50191-01
April 2026

Oracle AI Database Oracle Deep Data Security Guide, 26ai

G50191-01

Primary Author: Sacheth

Contributors: Tanvir Ahmed, Roger Wigenstam, Ji-Won Byun, Anita Patel, Srividya Tata, Quan Yang, Ivan Wu, Frank Hong Xiong, Chao Liang, Tulika Das, Marudha Sudharshan, Rizan Farooqui, Richard Evans, Luna Tan, Jinglei Xie, Yuechen Chen

Contents

19 Oracle Deep Data Security Data Dictionary Views

A Sample Scripts for Configuring Oracle Deep Data Security

Preface

This guide provides comprehensive instructions for configuring the Oracle Deep Data Security (Deep Sec) feature in Oracle AI Database to enforce fine-grained data authorization. It details how to integrate the database with external identity and access management (IAM) systems and implement declarative access controls at the row, column, and cell levels using Deep Sec capabilities. By following these guidelines, organizations can securely protect sensitive data at its source across agentic AI systems, analytics platforms, and enterprise applications.

Audience

This guide is intended for security architects, security administrators, application developers, and others tasked with configuring the Oracle Deep Data Security feature in Oracle AI Database.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Part I

Understand Oracle Deep Data Security

Discover Oracle Deep Data Security (Deep Sec), the next-generation solution for data access control across agentic AI, analytics, and enterprise applications.

Topics:

1

Introduction to Oracle Deep Data Security

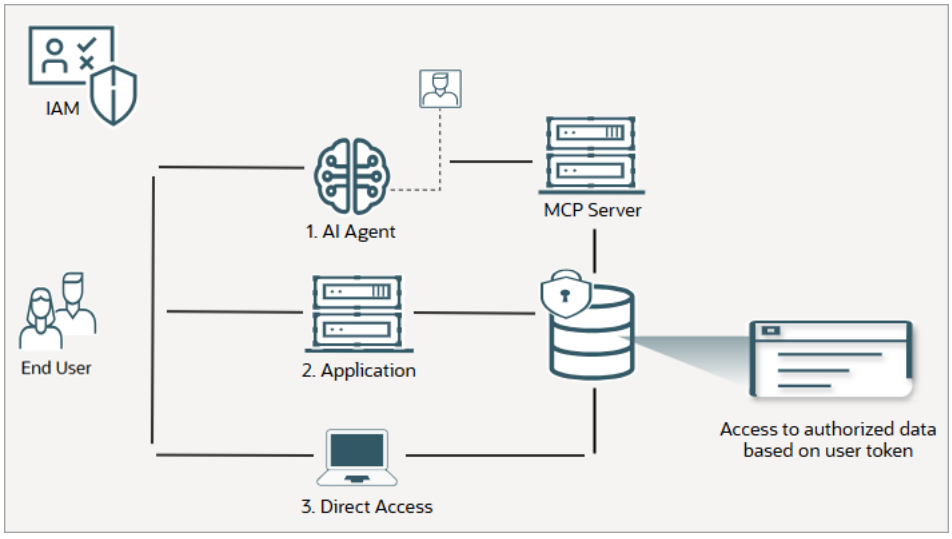
Learn about Oracle Deep Data Security (Deep Sec) and its authorization model built for the era of agentic AI. Explore the security challenges that agentic systems introduce, and discover how Deep Sec addresses the limitations of existing access control mechanisms to deliver fine-grained data security across agentic AI systems, analytics platforms, and enterprise applications.

Topics:

1.1 What Is Oracle Deep Data Security

Oracle Deep Data Security (Deep Sec) is a database-enforced data authorization framework. It enables application developers and security architects to define and enforce application-level security requirements directly at the database layer.

As a comprehensive, data-centric authorization platform, Deep Sec addresses the security needs of modern applications, including enterprise software, analytics, and agentic AI systems. It enforces fine-grained access control at the row, column, and cell levels, limiting users strictly to the data they are authorized to access. By securing data at its source, Deep Sec provides a unified security architecture across application and database tiers, protecting all access paths to sensitive data.



Topics:

1.1.1 The Deep Sec Authorization Model

The Deep Sec authorization model is a robust, centralized, and declarative framework designed to streamline fine-grained data access control in modern applications. It introduces a straightforward SQL syntax for specifying authorization at the row, column, and cell levels. You can define policies using clear, human-readable SQL statements, without the need for complex procedural code or opaque API-based approaches.

Deep Sec's authorization model is built on the following principles:

- **Centralized administration:** Authorization policies are defined during application development and enforced centrally at runtime. Access decisions are based on the end user's role or attributes through role-based access control (RBAC) or attribute-based access control (ABAC), rather than on database object ownership.
- **Declarative independence:** Authorization policies are expressed independently of application code. This separation allows developers and security administrators to update or refine policies without altering application logic, thereby simplifying version control, testing, and CI/CD pipelines.
- **Versatile policy definition:** Fine-grained authorization controls use a simplified SQL syntax that enables developers and administrators to apply precise restrictions within the database and manage policies efficiently at scale. Additionally, this syntax supports complex real-world requirements, such as cell-level authorization, column masking, authorization APIs, and policy lifecycle management through CI/CD.

For more information, see Fine-Grained Data Authorization.

1.1.2 Core Capabilities

Deep Sec provides the following capabilities for securing applications, analytics tools, and agentic AI systems.

- **Identity-aware and context-sensitive enforcement:** Authorization decisions are based on trusted identities, applications, and claims. An extensible end-user security context evaluates user identities, application settings, and environmental attributes (such as a user's geographic location) to determine data access.
- **Granular security:** Attribute-based access control (ABAC) is applied at the row, column, and cell levels, providing precise control over which data elements each user can access.
- **Dynamic masking:** Sensitive data is masked dynamically based on cell-level authorization decisions and runtime context, ensuring that unauthorized values are never exposed.
- **Authorization APIs:** Specialized APIs enable applications to pre-authorize access, supporting streamlined and secure end-user experiences.
- **Controlled privilege elevation:** Trusted application code can temporarily elevate user privileges to execute specific authorized operations, without granting those privileges to the user permanently.
- **Mandatory access control:** Non-discretionary access control is enforced by a central administrator, applying security rules uniformly across all subjects (end users, applications, agents) and database objects (tables and views).

1.2 Why Choose Oracle Deep Data Security

Oracle Deep Data Security (Deep Sec), integrated with Oracle AI Database, provides a database-native, declarative access control framework. By securing data at its source, it helps you adopt AI while prioritizing data security, privacy, and compliance.

Most high-value enterprise data resides within the database. As organizations integrate agentic AI (autonomous systems that observe environments, reason through complex problems, and execute actions) to interact directly with this private and regulated data, they face a new set of privacy risks. While these agents boost productivity and decision-making, their autonomy introduces vulnerabilities that traditional security models cannot address. Historically, organizations have enforced complex privacy rules within application code. However, this approach is no longer sufficient; AI-generated applications can inadvertently omit or misapply these rules, and autonomous agents can bypass them entirely through direct SQL access.

Deep Sec addresses these vulnerabilities by enforcing privacy rules and access controls directly at the database layer, rather than relying on application code, prompts, or other fragile guardrails. By applying fine-grained, least-privilege policies, Deep Sec restricts users and AI agents to only the data they are explicitly authorized to access. This helps you adopt agentic AI, analytics, and enterprise applications with confidence, making it easier to address regulatory compliance and data privacy without constraining innovation.

Topics:

1.2.1 The Security Challenge in Agentic AI

Agentic AI systems frequently use protocols such as the Model Context Protocol (MCP) to inspect database structures, identify relevant data, construct queries, and execute database operations autonomously.

This dynamic access to the database introduces a new class of threats:

- **Prompt injection:** Malicious inputs that manipulate an AI agent into performing unintended actions, such as bypassing system instructions or retrieving unauthorized data.
- **Excessive agency:** Shared, highly privileged database connections that grant AI agents broad access, enabling them to disclose sensitive information or execute unapproved transactions. In this scenario, an AI agent can generate and execute arbitrary SQL using the application's inherited privileges.
- **Unintended disclosure:** Accidental or malicious disclosure of confidential data, including business, health, or personal information.

Traditional security controls, designed for conventional applications with predictable patterns, often fail to handle the dynamic and autonomous nature of agentic AI. Deep Sec addresses this gap by enforcing access control directly at the data source, rather than relying on application code, prompts, or other potentially vulnerable AI guardrails—allowing you to harness agentic AI capabilities without compromising oversight.

1.2.2 Limitations of Existing Access Control Methods

Existing access control techniques work well for predefined reporting and transactional workflows, but they are often inadequate for the dynamic SQL patterns that agentic AI introduces. Common approaches include traditional database security, application-level controls, and external authorization systems. Each carries limitations that can weaken privacy enforcement, increase bypass risk, or impede innovation.

Traditional database security

Standard database controls, such as row-level security and column-level security, lack the necessary precision and efficiency required for dynamic AI workflows.

- **Performance overhead and obscurity:** Traditional row-level security often relies on policies implemented through conditional logic in stored procedures. This code-based approach can impose performance costs and bury security rules within scripts, which can hinder oversight and complicate governance.
- **Coarse granularity:** ANSI SQL column security operates on an all-or-nothing basis; users either have full access to a column's values or no access at all. This lacks the precision needed for granular, cell-level access control, such as allowing a manager to update a team member's job title (write access) while having read-only access to their own job title. Furthermore, queries that succeed for authorized users can trigger SQL errors for those lacking column-level access, causing application instability.
- **Masking limitations:** Data masking solutions are often limited to basic rules or suffer performance degradation with complex logic. They also remain susceptible to inference attacks, where users manipulate query filters to deduce masked values.
- **Administrative strain:** Mirroring external identities (such as those in Microsoft Entra ID) in the database creates significant provisioning burdens and synchronization challenges. Despite using external OAuth 2.0 authentication, databases typically require local user accounts and fail to propagate key IAM attributes, such as organization or location, required for real-time access decisions.

Application-level controls

Developers often embed hard-coded logic within applications to restrict data access and user actions. This approach creates liabilities that are amplified by modern AI workflows and by application security variability.

- **Static rules and maintenance burden:** Fixed rules are difficult to update, and adapting to new requirements demands code changes and deployment cycles that often delay critical security updates.
- **Role explosion:** Advanced applications implementing role-based access control (RBAC) often require thousands of roles. This complexity overwhelms administrators and increases the risk of policy misconfiguration.
- **Bypass risk:** In organizations with numerous applications and analytics tools, fragmented application policies are easily circumvented when data is accessed through alternative paths. Additionally, applications and analytics tools frequently use shared database user accounts with unrestricted database access, which can be exploited through SQL injection.
- **Agentic evasion:** Agents using the Model Context Protocol (MCP) can inspect schemas and execute SQL directly, bypassing application protections entirely. Workarounds to prevent this are impractical; embedding SQL parsers is prohibitively complex, while exposing REST APIs as MCP tools, with access rules enforced in the API logic, leads to a proliferation of interfaces and degraded performance compared to direct SQL execution.

- **RAG vulnerabilities:** Retrieval-Augmented Generation (RAG) application workflows chunk documents into vector embeddings for semantic search, creating unique security challenges. To authorize prompts for these searches, application code often relies on a secondary language model to validate and categorize prompts, then determines access based on the results. However, using another language model for authorization checks is prone to tampering and errors, while the alternative approach of post-processing search results to filter unauthorized content severely degrades performance.

External authorization systems

External authorization systems centralize policy management and often support attribute-based access control (ABAC). While they offer stronger governance than hard-coded rules by evaluating user, environment, and resource attributes, they face significant limitations in database environments.

- **Lack of database context:** As general-purpose systems, they are not optimized for database-specific constructs, such as schemas, table relationships, and SQL semantics. This can lead to policies with limited depth and precision.
- **Integration overhead:** Each database operation and privilege check requires custom integration, increasing development and maintenance costs. Authorization decisions must ultimately be enforced by application logic at runtime to adjust queries, perpetuating the reliance on custom code.
- **Persistent vulnerabilities:** Despite centralized management, many challenges associated with application-level controls remain. AI-generated SQL is difficult to secure, alternative access paths can still bypass controls, and continued reliance on privileged database connections elevates the risk of data breaches.
- **Latency and scaling challenges:** Runtime queries to external authorization services introduce latency, particularly when per-row or per-cell privilege checks are required to enforce access. SQL-intercepting proxies add comparable overhead and typically depend on nonstandard client drivers that either lag certification for new database releases or introduce risks not present in standard drivers.

1.2.3 The Deep Data Security Solution

Oracle Deep Data Security (Deep Sec) overcomes the limitations of traditional strategies by embedding a declarative access control framework directly into Oracle AI Database.

By enforcing policies at the database layer for every query, whether from an application, an analytics tool, or an AI agent, it reduces bypass risks. It helps align access with organizational security, privacy, and compliance requirements while improving operational efficiency.

Architected for AI with cell-level precision

Deep Sec enforces cell-level access control, enabling policies that target specific intersections of rows and columns. AI agents access only the exact data points they need, overcoming the limitations of broad row or column security. This minimizes the risks of data inference and excessive privilege, and enforces policy-compliant access to sensitive data.

Transparent governance through declarative SQL

Deep Sec replaces complex procedural logic (such as PL/SQL) and API-driven approaches with clear, human-readable declarative SQL for defining access rules.

This design provides the following benefits:

- **Clear and auditable controls:** Policies are decoupled from application logic, providing transparent, easy-to-audit access rules.

- **Rapid adaptation:** Centralized policy management integrates with the full continuous integration/continuous deployment (CI/CD) cycle, allowing you to adjust policies quickly in response to evolving security or regulatory requirements.
- **Low learning curve:** Standard SQL policy definitions make Deep Sec easy to adopt and administer.

Zero trust through native IAM integration

Deep Sec enforces *zero trust* security by seamlessly integrating with external identity and access management (IAM) systems, and applying identity-aware, context-driven controls at the row, column, and cell levels. Zero trust is a security approach that assumes no implicit trust for any user or application; instead, access is determined dynamically based on verified identity and policy at the time of each request.

This design provides the following key security capabilities:

- **Secure identity propagation:** Deep Sec supports native identity and context propagation, ensuring that end-user identities flow from the IAM system directly to the database without requiring a separate database user for every individual. The propagation mechanism uses standard OAuth 2.0 tokens to carry end-user and application identities from IAM to the database. Tokens travel with each SQL request through Oracle client drivers, and the database validates these tokens to establish the security context, differentiating between the end user's identity (for policy enforcement) and the application's authorization (for database access). See End-User Security Context.
- **Least-privilege enforcement:** Deep Sec eliminates the need for highly privileged, shared database connections. At runtime, only the specific privileges associated with the end user and application identity are applied, ensuring least-privilege access. For AI workloads, this mitigates excessive-agency risk by authorizing AI-generated SQL based on the end user who initiated the task, not a broadly privileged shared account.
- **Application isolation through protection domains:** Deep Sec uses the IAM-issued credentials to identify each application or AI agent and maps it to a database identity that acts as a protection domain—a boundary that defines which data roles the application can use. For each SQL execution, the database creates a lightweight sandbox scoped to those roles. This restricts the application to only the data its roles permit, even if the end user who initiated the task has broader access.

Unified, enterprise-ready security

Deep Sec provides a unified, standards-based security model that applies consistently across agentic AI, analytics, and traditional applications. It also offers the following distinct advantages for enterprises:

- **Optimized performance:** Unlike external authorization services that introduce latency, Deep Sec's database-native enforcement reduces external network round-trips to evaluate security policies. By filtering data at the source, Deep Sec can reduce database I/O operations and improve query performance.
- **Scalability:** Designed for deployments of any size or complexity, Deep Sec streamlines development and administration while ensuring consistent, auditable governance.
- **Support for modern data types:** Deep Sec handles diverse data formats within the Oracle AI Database, including relational data, JSON through Duality Views, and vector embeddings for RAG workflows that ground AI responses in enterprise knowledge.

2

Oracle Deep Data Security Fundamentals

Before you begin configuring Oracle Deep Data Security (Deep Sec), develop a clear understanding of its core concepts, components, and security model.

This chapter introduces key terminology and explains how IAM registrations and database entities (such as data roles and application identities) converge into a runtime end-user security context. It also details how the fine-grained authorization model enforces row-level and column-level security through data grants, access check functions, and mandatory access controls.

Topics:

2.1 Key Terminology

Familiarize yourself with the key terminology used within the Oracle Deep Data Security (Deep Sec) environment.

Identity and users

These terms describe the users and identities that Deep Sec recognizes for authentication and authorization.

- **IAM**

  An identity and access management (IAM) system, such as Microsoft Entra ID or Oracle Cloud Infrastructure Identity and Access Management (OCI IAM), that manages user identities and role assignments outside the database.

- **End user**

  A user of an application who does not own database schemas or database objects. An end user can be:

  – A user whose identity is managed in IAM. Typically, they connect to the database through the application. They can also log in directly using token-based authentication.

  – A user created and managed in the database using the `CREATE END USER` statement. End users managed in the database can log in directly using password authentication. Additionally, they can be mapped by user name to users managed in the application's own user store, so that those users can connect through the application's trust, without requiring direct password authentication on the database server.

- **Application identity**

  A database-resident identity that represents a specific application. During each session, the database enables data roles granted to this identity for all users connecting through that application.

Roles and authorizations

These terms distinguish between roles defined in IAM and their corresponding representations within the database.

- **Application role**

  A role your application defines to control access to protected resources. You usually create and assign these roles to users in IAM, and then create a corresponding data role in the database that maps to each application role.

- **Data role**

  A role in the database used for fine-grained access to data. You can grant data privileges (through data grants) and standard database roles to a data role. A data role can be:

  – Mapped to an application role in IAM using the `MAPPED TO` clause. The database automatically enables it when the end user's token includes the corresponding role claim.

  – Managed locally in the database. A data role managed locally in the database can be granted to end users, application identities, or other data roles (that are managed locally).

- **Database role**

  A standard role that exists within the Oracle AI Database environment, separate from the application-specific roles defined above.

Application and context

The following terms describe the component bridging the user interface and the database, and the security context for end-user operations.

- **Application**

  A client application in a two-tier architecture, or a mid-tier application in a three-tier architecture, that accesses the database using language-specific client drivers such as JDBC, Python, or ODP.NET.

- **End-user security context** *(runtime values within a database session)*

  The session-level object that holds the active end user's identity, their enabled data roles, and all live attribute values. The database creates an end-user security context automatically when an application sends an `EndUserSecurityContext` payload with the following components:

  – **End-user identity:** The end user's name as asserted in the IAM access token. For end users managed locally, this is the name of the end user created in the database.

  – **Data roles (optional):** Additional data roles that the application can enable for the end-user security context, beyond those mapped to application roles in IAM and those enabled by default for the application identity.

  – **End-user context attributes (optional):** A dictionary of application-defined key-value pairs to include in the security context. Used when data grants or application logic rely on custom end-user context attributes.

  – **Database-access token:** An on-behalf-of (OBO) token or an OAuth client-credential token that the application obtains from IAM to authorize its access to the database. This token ensures the database accepts requests only from trusted applications. Only an authorized application (with its application secret) can obtain this token from IAM.

- **End-user context** *(database-side definition)*

  A database schema object you create with the `CREATE END USER CONTEXT` command that defines a set of attributes, their data types, default values, and optional PL/SQL routines to populate them. At runtime, the database uses it as a template for JSON-style name-value pairs and instantiates it on first use in the current end-user security context — either when the application payload is attached or when an attribute is required for authorization checks.

Access control mechanisms

These terms define how Deep Sec protects specific data elements.

- **Data grant**

  A fine-grained access control policy that allows access to database records at row, column, and cell levels.

- **Predicate**

  A logical condition expressed as a SQL predicate that identifies a specific set of rows for access control purposes.

- **Data privilege**

  A specific right or permission that you grant through data grants to an end user or a data role.

2.2 Fine-Grained Data Authorization

Learn how Oracle Deep Data Security (Deep Sec) enforces row and column security, validates privileges through access check functions, and applies mandatory access control for comprehensive data protection.

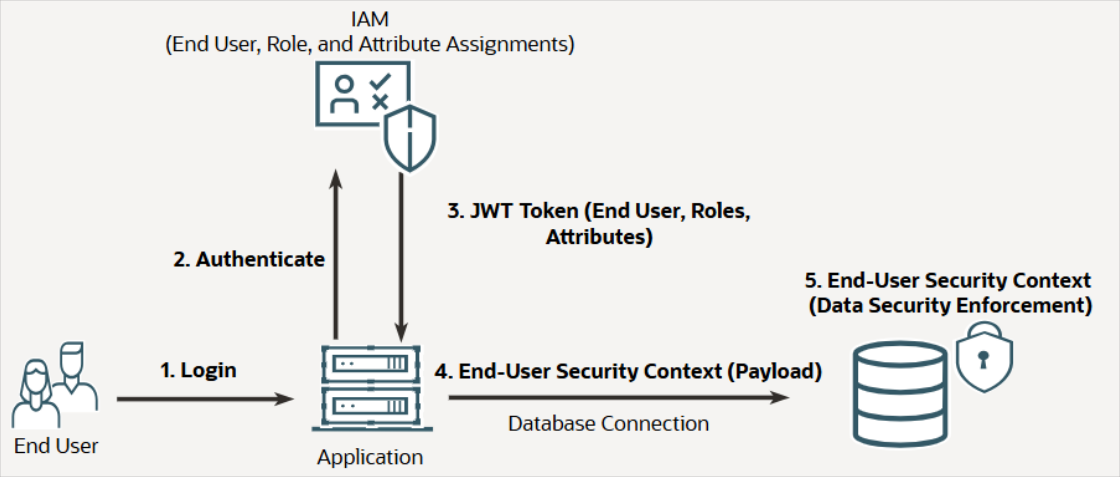
Topics:

2.2.1 Architectural Overview

To provide context for Deep Sec's fine-grained authorization model, let’s review the authentication and authorization flow in a typical three-tier architecture and examine the components involved.

Authorization flow

The following figure illustrates the authentication and authorization flow in a typical three-tier application architecture with Deep Sec enabled:



Key components

- **End-user management**

  End users of applications and analytics tools are typically managed in an external IAM system, and they're authenticated when they access the application. Unlike traditional database users, Deep Sec end users do not own database schemas or objects; however, they require access to specific rows and columns within tables and views.

  To grant this access, you assign roles to users in IAM, then create corresponding data roles in the database and map them to the IAM roles. Note that the fine-grained authorization model described in this guide applies *only* to application users. Database users (such as schema owners or administrators) continue to use database-native access control solutions, such as Oracle Virtual Private Database (VPD).

- **Application layer**

  The application represents a mid-tier processing unit, running either inside or outside the database server. It authenticates the user and acts on their behalf to perform tasks, which may include calling other servers or services. The application may also use automated agents, including agentic AI or generative AI, to process requests and generate SQL statements.

- **End-user security context**

  The database establishes the end-user security context (comprising the user's identity, roles, and attributes) based on claims from identity providers and application-specific logic (such as, business unit, job function, or profile options). In the typical flow depicted above, this security context payload is propagated through JSON Web Tokens (JWT) over the OAuth 2.0 protocol, though other authentication methods may be used. At runtime, the application attaches this security context payload to the database connection. The database then uses this payload to enforce fine-grained security, granting the end user access to specific rows and columns based on the established policy.

2.2.2 About Data Grants and Security Context

Deep Sec introduces *data grants* to precisely control access at the row, column, and cell levels. Learn about data grants and how to use the runtime security context to create dynamic, identity-based predicates within data grants.

Data grant

Central to the Deep Sec authorization model is the *data grant*, a policy statement that authorizes access to specific rows and columns within a table or view. Through multiple data grants, a user can hold different privileges on different row sets and column values (cells) within the same table or view; for example, SELECT privilege on some rows, UPDATE on certain columns for a subset of those rows, and full modification privileges on another set of rows. The grants are *additive*, meaning a user's data access is determined based on the union of all data grants assigned to them.

You define data grants using the `CREATE DATA GRANT` statement. Each grant specifies the privileges (`SELECT, UPDATE, INSERT, DELETE`), the target object, an optional `WHERE` clause (predicate) to identify rows, and the grantee.

Syntax:

```
CREATE [OR REPLACE] DATA GRANT [schema.]grant_name AS
    <SELECT | UPDATE | INSERT | DELETE> [(column [, column] ...)]
    ON <object>
    [WHERE <predicate>]
    TO {<end_user> | <data_role>};
```

Where:

- `object` is the target table or view for the data grant.
- `predicate` is a SQL expression that defines which rows the user can access. It supports dynamic logic, including subqueries and references to the end-user security context. The default is `TRUE`, representing all rows.
- `end_user or data_role` is the local end user or data role for the privilege grant. See User and Role Management in the Database.

See Configure Data Grants for complete details on the syntax and data privilege semantics for DML operations.

Security context reference

To define data grants, it is essential to understand how to reference the end-user security context in SQL. Data grants in Deep Sec typically rely on the end-user security context payload (the user's identity, roles, and attributes) that the application propagates to the database at runtime.

You can access this context in the database using the `ORA_END_USER_CONTEXT` SQL function. It returns the context parameters as JSON accessible through standard dot-notation. For example, `ORA_END_USER_CONTEXT.username` returns the current end user’s name.

You'll typically use this function in data grant predicates to identify rows based on the current user's identity or attributes. For more details, see Context Attributes.

2.2.3 Row and Column Controls

Understand how data grants enforce fine-grained security at the row, column, and cell levels.

You can create data grants for data roles or directly for local end users. In an IAM-managed environment, data grants are created for data roles that are mapped to external application roles. End users are provisioned in an IAM system and assigned application roles. In the database, data roles are mapped to the external roles, and these data roles hold the data grants. In a locally managed environment, you can create data grants for data roles that are managed within the database or directly for local end users. This flexibility supports both centralized identity governance and simpler standalone configurations. For details on data roles and role mapping, see User and Role Management in the Database.

When an end user queries a table protected by data grants, the database performs the following actions:

- **Policy evaluation:** Identifies the data grants that apply to the user or their active data roles, and verifies that the required data privilege (such as `SELECT`) is granted for the target table.
- **Row filtering:** Dynamically appends a `WHERE` clause to the query based on the grant's predicate, filtering rows transparently at the database level without requiring changes to the application's SQL.
- **Column restriction:** Masks or restricts columns according to column-level rules (for example, returning `NULL` for unauthorized columns or disallowing `UPDATE` on specific fields).

> **Note**
>
> Data grant examples throughout this section are based on the following `hr.employees` table.

```
EMPLOYEE_ID  FIRST_NAME  LAST_NAME  EMAIL       MANAGER
SSN          SALARY  PHONE
-----------  ----------  ---------  ----------  ----------
-----------  ------  --------
100          Victoria    Williams   vwilliams
219-09-9999  13000   555-0100
200          Marvin      Anderson   manderson   vwilliams
457-55-5462  12030   555-0200
300          Chris       Evans      cevans      vwilliams
321-12-4567   6900   555-0300
400          Emma        Baker      ebaker      manderson
733-02-9821   8200   555-0400
500          Taylor      Mills      tmills      manderson
558-76-1243   9000   555-0500
```

Row-level security

Row-level security is enforced through data grants that restrict operations (such as `SELECT`, `UPDATE`, `INSERT`, or `DELETE`) to a specific set of rows, defined by a SQL predicate (`WHERE` clause). The predicate can reference the end-user security context or include subqueries for

more complex logic. The database evaluates the predicate for each row: if it evaluates to `TRUE`, the row is accessible; if `FALSE`, the row is filtered out.

Example: Basic row security

The data grant in this example permits employees to view only their own record.

The following SQL statement creates a data grant named `employees_own_record` in the `hr` schema. This grant allows users with the `employee_role` data role to access their own record in the `hr.employees` table. In an IAM-managed environment, any end user whose application role maps to `employee_role` can view only the row that corresponds to their own identity.

```
-- Grant 1: Allow employees to see their own record
CREATE OR REPLACE DATA GRANT hr.employees_own_record
  AS SELECT
  ON hr.employees
  WHERE email = ORA_END_USER_CONTEXT.username
  TO employee_role;
```

When Emma, who is an application user and an employee, queries the table, she sees only her own record.

```
EMMA> SELECT * FROM hr.employees;

EMPLOYEE_ID FIRST_NAME LAST_NAME EMAIL   MANAGER    SSN         SALARY PHONE
----------- ---------- --------- ------- ---------- ----------- ------
--------
400         Emma       Baker     ebaker  manderson  733-02-9821  8200
555-0400

1 row selected.
```

Column and cell-level security

While row security determines which records are visible, column security determines which attributes within those records are exposed. By specifying row predicates with column restrictions, you achieve cell-level security—rules that govern which column values a user may `SELECT`, `INSERT`, or `UPDATE` for a specific set of rows.

This enables scenarios where a user has access to a row but is restricted from viewing sensitive columns (such as social security number) or modifying critical columns (such as salary).

Example: Cell-level write restrictions

You can define data grants to limit `UPDATE` privileges to specific columns. If a user attempts to update an unauthorized column in an authorized row, the operation is silently ignored.

The data grant in this example allows employees to view their own record and update only the phone number.

```
-- Grant 2: Allow employees to see their own record and update phone number
CREATE OR REPLACE DATA GRANT hr.employees_own_record
  AS SELECT, UPDATE (phone)
  ON hr.employees
```

```
  WHERE email = ORA_END_USER_CONTEXT.username
  TO employee_role;
```

Example: Hierarchical access with column masking

You can grant access to rows while excluding sensitive columns (for example, social security number). The rows are returned, but the unauthorized columns appear as `NULL`.

The data grant (`manager_direct_reports`) in this example allows managers to view the records of their direct reports, with the social security number (SSN) column excluded. It also allows managers to update only the salary column for their direct reports. The grant is defined for the data role named `manager_role`.

```
-- Grant 3: Allow managers to see their direct reports (with SSN masked) and
update their salaries
CREATE OR REPLACE DATA GRANT hr.manager_direct_reports
  AS SELECT (ALL COLUMNS EXCEPT ssn), UPDATE (salary)
  ON hr.employees
  WHERE manager = ORA_END_USER_CONTEXT.username
  TO manager_role;
```

Marvin, who is both a manager and an employee, holds both the `employee_role` and `manager_role` data roles. As an employee, he sees his own record, including his SSN. As a manager, he also sees the records of his direct reports (Emma and Taylor) but with their SSN values masked as `NULL`. The manager role additionally grants him the ability to edit the salaries of his direct reports.

```
MARVIN> SELECT * FROM hr.employees;

EMPLOYEE_ID FIRST_NAME LAST_NAME EMAIL      MANAGER    SSN         SALARY
PHONE
----------- ---------- --------- ---------- ---------- ----------- ------
--------
200         Marvin     Anderson  manderson  vwilliams  457-55-5462 12030
555-0200
400         Emma       Baker     ebaker     manderson              8200
555-0400
500         Taylor     Mills     tmills     manderson              9000
555-0500

3 rows selected.
```

2.2.4 Access Check Functions

Deep Sec provides SQL functions to detect unauthorized values for conditional masking and to evaluate user privileges at the row and column level.

Verify column access (`ORA_IS_COLUMN_AUTHORIZED`)

This function differentiates between a genuine `NULL` stored in the database and a `NULL` returned due to authorization restrictions. This allows developers to handle restricted data gracefully in the presentation layer.

- **Behavior:** Returns `TRUE` if the current user is authorized to access the column value for the given row, or if the column is not protected by a data grant. Returns `FALSE` if the user lacks authorization to view the value.
- **Use case:** Displaying a mask instead of a blank field for unauthorized values.

Example:

Marvin can use this function in the `SELECT` list to display:

- The actual SSN value for rows where he is authorized to view it.
- A mask (`000-00-0000`) for rows where he is not authorized, instead of a blank field.

```
MARVIN> SELECT first_name, last_name,
               DECODE(ORA_IS_COLUMN_AUTHORIZED(ssn),
                       FALSE, '000-00-0000',
                       TRUE, ssn) AS ssn,
               email,
               manager
        FROM   hr.employees;

FIRST_NAME LAST_NAME  SSN          EMAIL       MANAGER
---------- ---------- ------------ ----------- ----------
Marvin     Anderson   457-55-5462  manderson   vwilliams
Emma       Baker      000-00-0000  ebaker      manderson
Taylor     Mills      000-00-0000  tmills      manderson

3 rows selected.
```

Validate operational privileges (`ORA_CHECK_DATA_PRIVILEGE`)

This function allows applications to check a user's privileges at the row or column level. This is essential for enforcing security in APIs and controlling dynamic UI elements (such as enabling or disabling an **Edit** button) based on the user's access rights.

- **Behavior:** Returns `TRUE` if the user holds the specified `<privilege>` for the given row or column. Returns `FALSE` otherwise.
- **Use case:** Pre-validating whether a user is authorized to perform an operation before executing it.

Example:

The following query verifies which operations Marvin (who is both an employee and a manager) can perform. It checks whether he has `SELECT` privileges on each row and `UPDATE` privileges specifically on the phone column.

```
MARVIN> SELECT first_name, last_name,
              manager,
              ORA_CHECK_DATA_PRIVILEGE(emp, 'SELECT') AS can_view,
              ORA_CHECK_DATA_PRIVILEGE(emp, 'UPDATE', phone) AS
can_update_phone
       FROM   hr.employees emp;

FIRST_NAME LAST_NAME  MANAGER    CAN_VIEW  CAN_UPDATE_PHONE
---------- ---------- ---------- --------- ----------------
Marvin     Anderson   vwilliams  TRUE      TRUE
Emma       Baker      manderson  TRUE      FALSE
```

```
Taylor     Mills      manderson  TRUE      FALSE

3 rows selected.
```

The output confirms that Marvin can view his own record and those of his direct reports, but can only update his own phone number.

2.2.5 Mandatory Access Control

Understand how adopting Mandatory Access Control (MAC) helps you enforce consistent security policies across all access paths to your data objects.

When your application handles sensitive data, like Personally Identifiable Information (PII) or Protected Health Information (PHI), authorization must remain consistent regardless of the access path. Whether a user queries a base table directly or accesses it indirectly through a view, the resulting data exposure must be identical.

The risk of inconsistent access

A security gap can arise when a user has access to both a base table and a view defined on that table. In standard configurations, views typically execute with the privileges of the view owner rather than the querying user. Consequently, a user restricted from certain rows in the base table may see all rows when querying the view.

The following example illustrates this scenario, using the same hr.employees table referenced earlier.

- An administrator may create a view on the `hr.employees` table that exposes all employee data.

  ```
  -- Create a view in hr schema
  CREATE VIEW hr.employees_view AS
    SELECT * FROM hr.employees;
  ```

- The administrator then grants the `employee_role` full access to the view, but *own-record-only* access to the base table.

  ```
  -- Grant broad access to the VIEW
  CREATE OR REPLACE DATA GRANT hr.employees_view_grant
    AS SELECT ON hr.employees_view
    TO employee_role;

  -- Grant restricted access to the BASE TABLE
  CREATE OR REPLACE DATA GRANT hr.employees_own_record
    AS SELECT ON hr.employees
    WHERE email = ORA_END_USER_CONTEXT.username
    TO employee_role;
  ```

- **Table access:** When Emma queries the `hr.employees` table directly, the database restricts her to her own record, in accordance with the `hr.employees_own_record` data grant.

  ```
  EMMA> SELECT first_name FROM hr.employees;

  FIRST_NAME
  ----------
  ```

```
Emma
1 row selected.
```

- **View access:** However, when Emma queries `hr.employees_view`, she can view all rows. This occurs because the view accesses the base table using the permissions of the view owner (`hr`), bypassing the user-specific restriction.

```
EMMA> SELECT first_name FROM hr.employees_view;

FIRST_NAME
----------
Victoria
Marvin
Chris
Emma
Taylor

5 rows selected.
```

MAC enforcement

To resolve the inconsistent access behavior and enforce a Mandatory Access Control (MAC) model, Deep Sec provides the `USE DATA GRANTS ONLY` configuration setting.

When `USE DATA GRANTS ONLY` is enabled on a table:

- End users cannot access the table unless they hold the required data grant. Access is denied even if the user possesses database object or system privileges (like `SELECT` or `SELECT ANY TABLE`).
- Access control policies are enforced uniformly, regardless of whether the user accesses the table directly or through a view.

Example: Enforce MAC on the `hr.employees` table

To ensure that access restrictions defined on the base table are not circumvented through views or alternative access paths, enable MAC on the table using the following command:

```
SET USE DATA GRANTS ONLY ON hr.employees ENABLED;
```

With MAC enabled, employees see only their row, whether they query the table or the view.

```
EMMA> SELECT first_name FROM hr.employees;

FIRST_NAME
----------
Emma

1 row selected.


EMMA> SELECT first_name FROM hr.employees_view;

FIRST_NAME
----------
Emma
```

```
1 row selected.
```

2.3 Application Registrations, Users, and Roles

Learn about the application registrations, user accounts, and role assignments required across both the IAM system and database to support Oracle Deep Data Security (Deep Sec). These configurations are fundamental to establishing identity, enforcing access policies, and securing the flow of data across your environment. This section begins with an overview of common data access patterns that guide these configuration choices.

- Data Access Patterns
- Application Registrations in IAM
- User and Role Management in IAM
- User and Role Management in the Database

2.3.1 Data Access Patterns

Explore the primary patterns in which IAM-managed end users and applications access the database using token-based authentication. Understanding these patterns helps you determine the application registrations, user accounts, and role mappings required to implement Deep Sec in your environment.

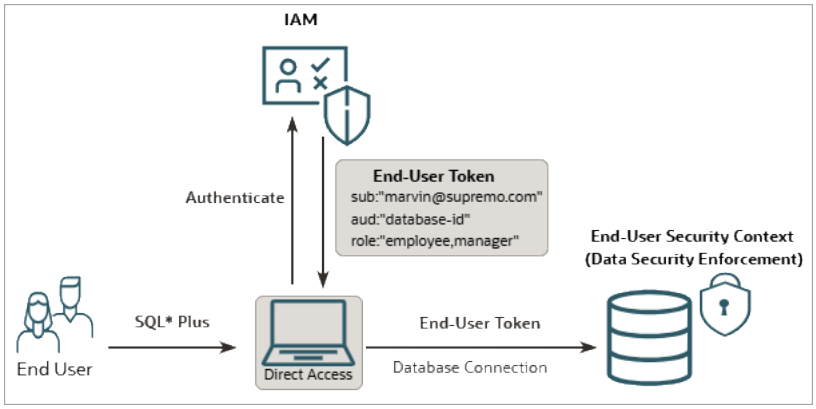
- **Connect Directly to the Database:** End users, applications, or AI agents access data by connecting directly to the database.
- **Connect Through an Application:** End users access data through an application, which propagates their identity to the database. This pattern has two variants:
  - Applications or AI Agents Access the Database on Behalf of End Users: Data access is based *entirely* on end-user authorization. An application, such as an analytics tool or MCP (Model Context Protocol) server, accesses the database using an end user's IAM role.
  - End Users Operate in the Context of an Application or AI Agent: Data access is based on *both* end-user and application authorization.

2.3.1.1 Connect Directly to the Database

In this pattern, end users log on directly to the database using a SQL client, such as SQL*Plus. No application is involved.

Use case

This pattern supports simple applications, administrative tasks, and development or testing environments where users query the database directly.



Database authentication flow

Prerequisite setup:

To enable Deep Sec for this pattern, you must complete the following configuration:

- **IAM setup:** Register the database as an OAuth resource with required roles (for example, `employee` and `manager`). Additionally, create an end user (Marvin) and assign them these roles.
- **Database setup:** Configure IAM as an external identity provider, create corresponding Deep Sec data roles, and map them to IAM roles (for example, create data roles named `employee_role` and `manager_role` that map to the IAM roles).

The runtime flow is as follows:

1. **User login:** The end user (Marvin) logs in through a SQL client configured for token-based authentication (for example, SQL*Plus).
2. **Token retrieval:** The client obtains an access token from IAM and passes it to the database. This token includes the following information:
   - End user (Marvin) as the subject
   - The database as the audience
   - Marvin's assigned IAM roles (`employee, manager`)
3. **Security context establishment:** The database validates the token, establishes an end-user security context, and activates the data roles corresponding to the end user's IAM roles (that is, `employee_role` and `manager_role` data roles are activated).
   Marvin receives the privileges associated with these data roles. Unlike traditional database users, Marvin does not have a dedicated database user or schema. Marvin's identity and privileges are provided dynamically at runtime in the end-user security context.

Configuration path

For the complete configuration path, see IAM-Managed Users Connecting Directly.

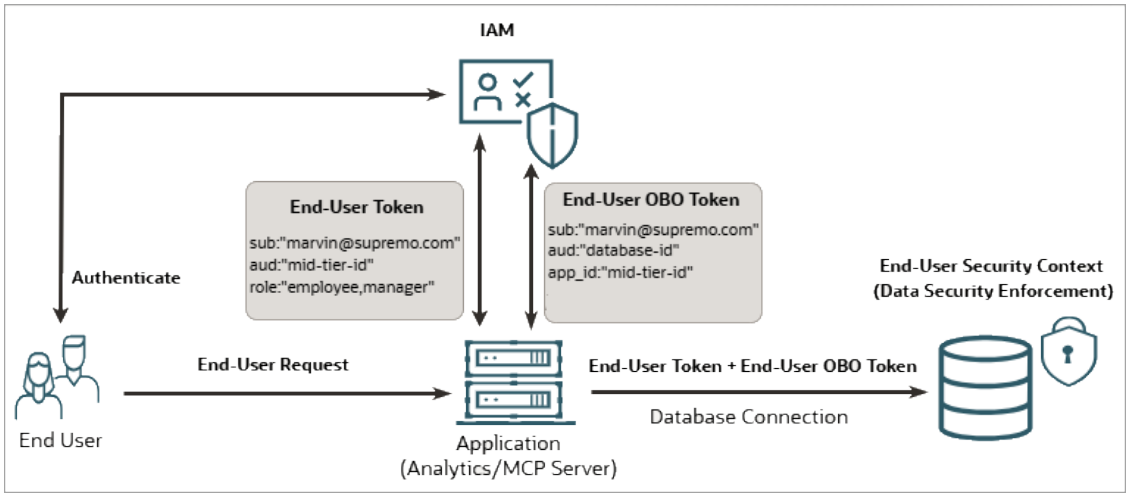
2.3.1.2 Applications or AI Agents Access the Database on Behalf of End Users

In this pattern, an application (such as an analytics tool or MCP server) executes SQL against the database using the end user's authorization. The application acts on behalf of the end user,

exchanging the end user's token for a database-scoped on-behalf-of (OBO) token from the IAM (for example, Microsoft Entra ID).

Use case

This pattern is ideal for reporting tools and MCP servers. In these scenarios, the application does not possess its own data privileges; instead, it acts on behalf of the user to retrieve only the data that specific user is authorized to access.



Database authentication flow

Prerequisite setup:

To enable Deep Sec for this pattern, you must complete the following configuration:

- **IAM setup:** Register the database as an OAuth resource with required roles (for example, `employee` and `manager`). Next, create an end user (Marvin) and assign them these roles. Additionally, configure the application to act on behalf of the end user.
- **Database setup:** Configure IAM as an external identity provider, create corresponding Deep Sec data roles, and map them to IAM roles (for example, create data roles named `employee_role` and `manager_role` that map to the IAM roles).

The runtime flow is as follows:

1. **User login:** The end user (Marvin) logs in to the application through IAM and receives a token scoped specifically to that application.
2. **Token exchange:** The application presents the end-user token as an assertion, along with its own credentials, and receives an on-behalf-of (OBO) token for Marvin. This OBO token is the database-access token, and it has Marvin as the subject (`sub` claim).
3. **Security context payload propagation:** The application uses this database-access token to connect to the database. Additionally, it passes the end-user token for role resolution.
4. **Security context establishment:** The database validates the tokens, establishes an end-user security context, and activates the data roles corresponding to the end user's IAM roles (that is, the `employee_role` and `manager_role` data roles).
   Because Marvin is the subject in the database-access token, the application is essentially accessing the database as Marvin. The database activates the data roles according to the claims in the end-user token. The application itself does not define or control the roles; the user's own IAM role assignments drive database access.

Configuration path

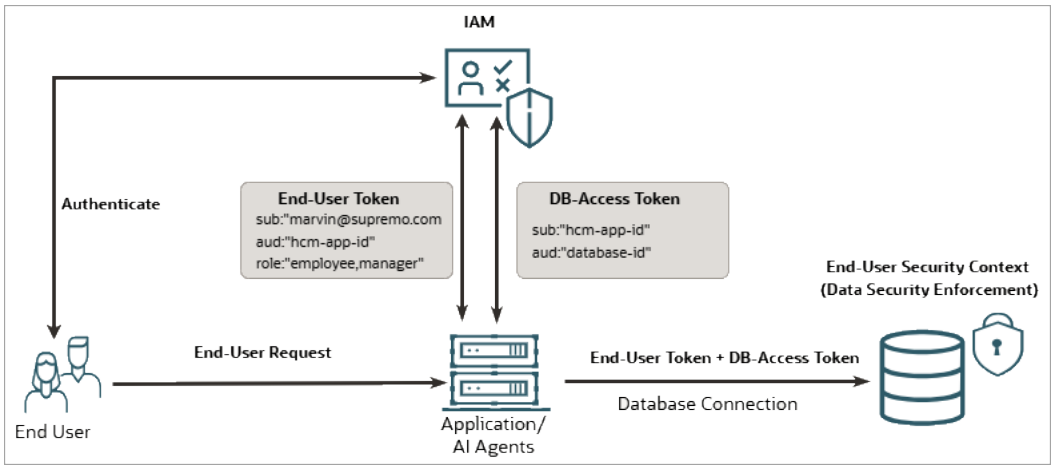
For the full configuration path for IAM-managed users with application connections, see IAM-Managed Users Connecting Through an Application.

2.3.1.3 End Users Operate in the Context of an Application or AI Agent

In this pattern, data access is controlled by both end-user and application authorization. The application is registered as an OAuth application in IAM with its own roles, and end users are assigned those roles. As a result, an end user acquires the combined privileges of their own account and the application (or AI agent).

Use case

This is the most common pattern for enterprise and agentic AI applications. This approach is ideal when an AI agent or enterprise application must connect to the database securely, but restrict the data it retrieves based on the end user interacting with the application.



Database authentication flow

Prerequisite setup:

To enable Deep Sec for this pattern, you must complete the following configuration:

- **IAM setup:** Register the database as an OAuth resource. Next, register the application (for example, HCM application) with required roles, such as `employee` and `manager`. Create an end user (Marvin) and assign them these roles. Additionally, grant the application access to the database resource.
- **Database setup:** Configure IAM as an external identity provider, create corresponding Deep Sec data roles, and map them to IAM roles (for example, create data roles named `employee_role` and `manager_role` that map to the IAM roles).

The runtime flow is as follows:

1. **User login:** The end user (Marvin) logs in to the application (HCM) through IAM and receives a token scoped specifically to that application. The token includes the roles assigned to the end user.
2. **Application authentication:** The HCM application obtains its own database-access token, with the application as the subject (`sub` claim) and the database as the audience.

3. **Security context payload propagation:** The application uses its own database-access token to authorize the connection. Additionally, the application forwards the end-user token as an attribute assertion for Marvin's identity and roles.
4. **Security context establishment:** The database validates the tokens, establishes an end-user security context, and activates the data roles corresponding to the end user's IAM roles (that is, the `employee_role` and `manager_role` data roles).
   In this case, the application accesses the database as itself, and the database activates the data roles corresponding to the application's roles, which are present as claims in the end-user token.

Configuration path

For the full configuration path for IAM-managed users with application connections, see IAM-Managed Users Connecting Through an Application.

2.3.2 Application Registrations in IAM

When an end user interacts with a modern application, their request traverses two distinct trust boundaries: the application server that orchestrates business logic and the database that enforces data-level security.

Each of these components has a unique security posture and must be represented as a separate application registration in your IAM.

The chain of trust

Because these components are registered independently, the authentication architecture of Oracle Deep Data Security (Deep Sec) relies on a transitive chain of trust, where each component delegates identity assertion to the next.

- **User to application (API call):** The user authenticates directly with IAM through a browser-based redirect. IAM issues an authorization code, which the application exchanges for an end-user token. This token asserts the end user's identity and is scoped to the application API. The application validates this token to confirm the user's identity and authorized scopes.
- **Application to database:** Subsequently, the application must obtain a separate database-scoped token (the database-access token) through one of the following flows:
  - On-behalf-of (OBO) flow: The application presents the end-user token along with its own credentials to the IAM token endpoint and requests a database-scoped token. The resulting database-access token carries the original user's identity (`sub` claim) but is addressed to the database audience.
  - Client credentials flow: The application obtains a database-access token using its own credentials, without exchanging the end-user token. The resulting database-access token carries the application's identity (`sub` claim), not the user's.

  In both flows, the application attaches two tokens to the database connection: the end-user token and the database-access token. The end-user token supplies the user's identity and role claims. The database-access token authorizes the application's connection to the database.
- **Database validation:** The Oracle AI Database validates both tokens against the IAM's public signing keys and confirms that the audience claim in the database-access token matches its own application ID, thereby authorizing the connection. It then uses the end-user token to extract the user's identity and role claims and establish the end-user security context. Once established, all subsequent SQL statements on that connection are subject to the data-security policies that apply to that specific end user.

About scopes and delegation

A scope is a named permission that a resource server (such as the application or the database) exposes. When a client requests a token, it specifies which scopes it needs, and IAM evaluates whether the client is authorized to request those scopes.

In Deep Sec's security architecture, scopes define what a component is authorized to do. For example, the database exposes a scope (such as `sessions:scope:connect`) that the application requests when acquiring a database-access token. In an OBO flow, the scope authorizes the application to connect to the database on behalf of the user. In a client credentials flow, the scope authorizes the application to connect to the database using its own identity. In both flows, the database uses the end-user token to establish the end-user security context, ensuring that data access is governed by the actual end user's identity and roles, regardless of how the application obtained its database-access token.

For steps to configure application registrations in your IAM, see IAM, Database, and Application Configuration.

2.3.3 User and Role Management in IAM

You create and manage application users and roles in your IAM. Subsequently, you create corresponding Deep Sec data roles in the database, and map the data roles to IAM roles for authorization.

This section explains how users and roles are structured in IAM and how they connect to database-level access control.

User registrations in IAM

Typically, you'll manage user identities for your application centrally in an IAM system. Each application user requires a user account in your IAM. IAM assigns every user a unique principal identifier, such as a user principal name (UPN) in Microsoft Entra ID or a user name in OCI IAM. This identifier is the canonical representation of the user's identity and is embedded in every token IAM issues for that user.

When the database receives a token, it extracts this principal identifier to establish the end-user security context for the session. The database does not store a separate local identity for externally managed users; instead, it relies entirely on the identity asserted in the token. This means that IAM remains the single source of truth for user lifecycle operations, such as provisioning, deprovisioning, and credential management.

Role definitions in IAM

Roles define what an authenticated user is authorized to do within the application and, by extension, within the database. The specific mechanism for defining roles differs between IAM providers, but the principle remains the same: roles appear as claims in the tokens issued to users, and the database maps those claims to its internal data roles.

Microsoft Entra ID: App roles

In Entra ID, you define app roles within each application registration. App roles are scoped to a specific application, meaning a role named `employee` on the HCM application registration is distinct from a role named `employee` on a different application registration. You assign app roles to users (or to other service principals) through the Entra ID portal or API. When Entra ID issues a token for that application, it includes the user's assigned app roles in the token's `roles` claim. This gives the database a reliable, application-specific set of role assertions for each user.

OCI IAM: Groups

In OCI IAM, groups serve as the role mechanism. You create groups that represent logical roles within your application, such as `employee` or `manager`, and assign users to them. When OCI IAM issues a token, it includes the user's group memberships in the token's claims (typically through a custom claim configuration or a default groups claim). The database reads these group claims and treats them as the equivalent of role assertions for authorization.

Despite the difference in terminology (app roles in Entra ID versus groups in OCI IAM), both serve the same function in this architecture; they provide the role claims that the database needs to determine which data roles to enable for an end-user security context.

Role mappings

The Deep Sec feature in Oracle AI Database supports *data roles* that can be mapped to external identity claims. When the database receives a token containing role or group claims, it matches those claims to pre-configured data roles. For each match, the database automatically enables the corresponding data role for that session, granting the user the associated fine-grained data authorizations.

Each data role has a one-to-one relationship with an external role claim. This means that any changes to a user's role assignments in IAM (such as adding the user to a new app role in Entra ID, or changing the user's group membership in OCI IAM) take effect automatically the next time that user's token is presented to the database, without requiring any changes to the database configuration itself. For details on data roles, see User and Role Management in the Database.

For steps to configure application users and roles in your IAM, see IAM, Database, and Application Configuration.

2.3.4 User and Role Management in the Database

Deep Sec provides a specialized framework for administering application users and roles within the database, designed to meet the security requirements of modern applications.

Traditional database user accounts pose two significant problems for application-level access control: they own database schemas and objects, and they typically carry excessive privileges, especially when multiple application users share a single account.

The Deep Sec framework addresses these problems by introducing *end users* and *data roles* as distinct entities from traditional database users and roles. This separation enables you to enforce fine-grained data authorization without granting application users direct ownership of database objects.

You can manage application users and roles in two ways:

- **Externally:** Through an IAM system, where data roles in the database map to application roles in IAM.
- **Locally:** Directly within the database, for simple applications, demonstrations, or development and test environments.

Topics:

2.3.4.1 Entity Types

Learn about the principal entities in Deep Sec and how they interact within the database environment.

End user

A user that an application defines. Unlike a traditional database user, an end user does not own database schemas or database objects. Instead, the Deep Sec framework grants end users access to specific rows and columns of tables (and views) through *data grants*, which are assigned to data roles that correspond to their application roles.

End users can create lightweight database sessions (end-user security contexts) through the application, rather than establishing full traditional database sessions.

Deep Sec recognizes the following two categories of end users:

- **End user managed in IAM**
  A user whose identity is created and managed in IAM. Typically, they connect to the database through the application. They can also log in directly using token-based authentication. These end users do not require local identity mappings in the database because the database derives their data authorization from the application roles that IAM assigns to them.
- **End user managed locally**
  A user created and managed in the database using the `CREATE END USER` statement. End users managed in the database can log in directly using password authentication. Additionally, they can be mapped by user name to users managed in the application's own user store, so that those users can connect through the application's trust, without requiring direct password authentication on the database server. Locally managed end users do not map to any entities in IAM, making them suitable for simple applications, or for demo, development, and test environments. You can grant fine-grained data grants to local end users directly or indirectly through data roles that are managed locally.

  The following SQL statement creates a local end user named `emma` who can log in directly with a password stored in the database:

  ```
  CREATE END USER emma IDENTIFIED BY <password>;
  ```

Data role

A data role is defined in the database specifically for fine-grained data grants. The keyword `DATA` in SQL statements distinguishes these roles from standard database roles.

Deep Sec supports the following two categories of data roles.

- **Data role that is externally mapped**
  A data role that is a database-side representation of an application role managed in IAM and assigned to application users. You create this data role with the `MAPPED TO` clause followed by the identifier string of the application role. This mapping establishes a one-to-one relationship between the data role in the database and the application role in IAM.

  When an end user in IAM attaches to a session, the database automatically finds and enables the data roles that map to the user’s application roles. This mechanism allows the end user to use the fine-grained data grants associated with the data roles.

The following example creates a data role that maps to a Microsoft Entra ID role named `employee`:

```
CREATE DATA ROLE employee_role MAPPED TO 'AZURE_ROLE=employee';
```

The following example creates a data role that maps to an OCI IAM group named `employee`:

```
CREATE DATA ROLE employee_role MAPPED TO 'IAM_OAUTH_GROUP=employee';
```

- **Data role that is locally managed**
  A data role that is created and managed entirely within the database, without any mapping to an external application role. You can grant this data role to local end users, application identities, or to other data roles that are managed locally. Also, you can grant database roles or data grants to this data role.

  The following SQL statements create a data role named `employee_role` and grant it to a local end user, `emma`:

  ```
  CREATE DATA ROLE employee_role;
  GRANT DATA ROLE employee_role TO emma;
  ```

Application identity

An identity that represents an application within the database. An application identity allows you to grant common privileges to all users connecting through a specific application. To do this, create data roles (that are managed locally in the database) with the required privileges and grant them to the application identity. When an end-user security context is established at runtime, the database enables the data roles only if the associated application identity has the grants on them. Because these roles are tied to the application, all users connecting through that application can inherit the associated privileges.

You create an application identity with the `CREATE APPLICATION IDENTITY` statement. The following example creates an application identity that maps to a Microsoft Entra ID application:

```
CREATE APPLICATION IDENTITY hcm_app MAPPED TO
'AZURE_CLIENT_ID=f1fab37e-7aa2-4ff8-849c-7e731fea3b48';
```

The following example grants the data role `hcm_role` to the application identity `hcm_app`, allowing the database to enable that data role for all HCM application's sessions:

```
GRANT DATA ROLE hcm_role TO hcm_app;
```

Application identities also enable privilege elevation, where trusted application code temporarily activates additional data roles on behalf of a user. See Privilege Elevation.

> **Note**
>
> You can only grant data roles that are managed locally to an application identity. The database enables data roles that are externally mapped automatically based on the end-user token's role claims; you cannot grant these to an application identity.

Summary of entity types

The following table summarizes the principal entity types in Deep Sec's user and role administration framework.

Table 2-1 Entity types in Deep Sec

Entity	Description
End user	A user of an application who does not own database schemas or database objects. An end user can be: • A user whose identity is managed in IAM. Typically, connect to the database through the application. Can also log in directly using token-based authentication. • A user created and managed in the database using the `CREATE END USER` statement. Can log in directly using password authentication. Can be mapped by user name to users managed in the application's own user store, so that those users can connect through the application's trust, without requiring direct password authentication on the database server.
Data role	It can be of either of the following types: • A data role that maps one-to-one to an application role in IAM. Enables fine-grained data authorization for external users. • A data role created and managed entirely within the database. You can grant it to local end users, application identities, other data roles (that are managed locally), or assign data grants to it.
Application identity	A registered identity for an application. Enables data roles and common privileges for all users connecting through that application.

For steps to configure Deep Sec entity types, see User and Role Administration.

2.3.4.2 SQL Quick Reference

Use this quick reference to find SQL statements for managing users, roles, and application identities in Deep Sec.

Task	SQL Statement
Create an end user in the database	`CREATE END USER <name> [IDENTIFIED BY <password>];`
Create a data role that is externally mapped (Microsoft Entra ID)	`CREATE DATA ROLE <name> MAPPED TO 'AZURE_ROLE=<role>';`
Create a data role that is locally managed in the database	`CREATE DATA ROLE <name>;`
Grant a data role (that is locally managed) to a local end user	`GRANT DATA ROLE <role> TO <user>;`
Create an application identity	`CREATE APPLICATION IDENTITY <name> MAPPED TO '<mapping>';`
Grant a data role (that is locally managed) to an application identity	`GRANT DATA ROLE <role> TO <app_identity>;`

2.4 End-User Security Context

The end-user security context is the security boundary through which the database identifies an end user and determines what they can see and do. When an application executes a database operation, it passes a set of identity and authorization details (such as tokens and data roles) as the payload. The database uses this payload to establish the end-user security context, which governs the end user's access for the duration of the Oracle Deep Data Security (Deep Sec) session.

Traditionally, applications connect to databases using a highly privileged database user account. In contrast, the end-user security context constrains the privileges assigned to the database connection to match strictly those of the end user. Deep Sec enforces this by replacing the underlying database session's security domain (the database user and their roles) with the end-user security context.

Once this replacement occurs, the database authorizes all SQL queries from the application based solely on the end user, their roles, and their attributes, rather than the broad privileges of the application's database user account. Consequently, the database user account used by the application requires only basic privileges, such as `CREATE SESSION` and `CREATE END USER SECURITY CONTEXT`, effectively mitigating the risk of attacks like SQL injection.

Explore the following topics to learn more about the end-user security context.

- Components of the Security Context
- The Security Context Life Cycle
- Context Attributes
- Privilege Elevation

> **See Also**
>
> About the End-User Security Context.

2.4.1 Components of the Security Context

The end-user security context is an extensible, session-level object that functions as a security context scoped to an individual user for access control enforcement. It holds the active end user's identity, their enabled data roles, and all live attribute values.

The database creates an end-user security context automatically when an application sends an `EndUserSecurityContext` payload with the following components:

- **End-user identity:** The end user's name as asserted in the IAM access token. For end users managed locally, this is the name of the end user created in the database.
- **Data roles (optional):** Additional data roles that the application can enable for the end-user security context, beyond those mapped to application roles in IAM and those enabled by default for the application identity.
- **End-user context attributes (optional):** A dictionary of application-defined key-value pairs to include in the security context. Used when data grants or application logic rely on custom end-user context attributes.
- **Database-access token:** An on-behalf-of (OBO) token or an OAuth client-credential token that the application obtains from IAM to authorize its access to the database. This token

ensures the database accepts requests only from trusted applications. Only an authorized application (with its application secret) can obtain this token from IAM.

As discussed previously, for the application to obtain the end-user and database-access tokens, you must perform the required configurations in your IAM. During token issuance, IAM authenticates and authorizes both the end-user identity and the application's identity.

2.4.2 The Security Context Life Cycle

Learn how the end-user security context flows through a Deep Sec environment. Understand how your application builds the security context payload, propagates it to the database, and how the database establishes and manages that security context across SQL operations and connection reuse.

Security context population and transmission

The security context life cycle begins when the application initiates a database operation. During API calls, applications use an Oracle client driver to access data in the database. The Oracle client driver (such as JDBC, Python, or ODP.NET) allows application code to populate the end-user security context payload, which includes the end-user token (containing identity and roles) and application-specific attributes. Additionally, the application provides a database-access token. This token represents the application's authorization to access the database. For a complete list of end-user security context components, see Components of the Security Context.

Once populated, the client driver attaches this end-user security context payload, including the database-access token, to the database connection as metadata. This occurs before the request is sent to the database. The driver forwards this payload to the database with every subsequent SQL request.

Database validation and session attachment

When the database receives the security context payload, it validates the OAuth 2.0 tokens against three criteria:

- **Trust:** The token must be signed by a trusted IAM provider.
- **Audience:** The token's intended recipient must match the database's configured identity (set through the `init.ora` parameter).
- **Validity:** Neither token can be expired. Both the end-user and database-access tokens have finite lifetimes, typically around one hour.

If validation succeeds and the tokens contain the required scopes, the database grants the application access, establishes the end-user security context on the session, and enables the required data roles in the security context. If the database detects an existing, identical end-user security context, it reuses it. Otherwise, it creates a new security context. All SQL operations in the session execute within the boundaries of this end-user security context (end user’s identity, roles, and attributes), ensuring strict policy enforcement throughout.

The Deep Sec security mechanism can't be bypassed, as establishing a valid session requires the application to present three elements simultaneously: a valid end-user token, a database-access token, and the database credentials for a connection that supports the end-user security context payload attachment.

Security context detachment and cleanup

The database automatically manages the detachment and reattachment of an end-user security context to a database session through the following mechanisms:

- *Connection release:* When your application code releases the connection, the security context is detached from the database session.
- *Security context replacement:* When your application reuses a connection, the database seamlessly replaces the previous security context with the new one. If the incoming request does not use an end-user security context, the server simply detaches the old one.
- *Garbage collection:* The database automatically garbage-collects any end-user security contexts that remain inactive (not attached to any database session) after a timeout duration of one hour.

Driver support and configuration

Oracle client drivers, namely JDBC, Python, and ODP.NET drivers, support propagating the end-user security context payload to the database. You can implement this support in your Java, Python, or .NET applications using either of the two methods:

- API extension methods
- A configuration-driven Service Provider Interface (SPI)

For details, see Update Application Configuration with IAM Details.

2.4.3 Context Attributes

In Deep Sec, you use end-user context attributes in data grant predicates to make authorization decisions at runtime. Additionally, you can use these attributes to drive application logic, simplify SQL queries, and avoid complex joins.

Authorization policies in Deep Sec often reference multiple contexts, such as user, application, and system contexts. These contexts are composed of attribute-value pairs sourced from IAM systems, application logic, or the database itself.

Examples of context attributes:

- **User attributes:** `username, employee_id, organization_id, customer_id`
- **Environment attributes:** `NLS settings, device, location (IP address)`
- **Application profile settings:** `default business unit, org hierarchy type, territory assignment type`

To support Deep Sec policies, Oracle AI Database provides a flexible *end-user context* feature that lets you define, pass, and reference context attributes in a developer-friendly JSON format.

Context definition

Before the application can pass attributes in the *end-user security context* payload at runtime, you must define the attributes and their corresponding *end-user contexts* in the database. You do this by creating an `END USER CONTEXT` object, which acts as a template that specifies allowed attributes, data types, and associated logic.

Oracle AI Database supports two types of contexts: custom (user-defined) and default (system-defined).

- **Custom context**

  You can create your own end-user contexts using JSON schemas to store application-specific attributes. For example, a Human Capital Management (HCM) application might own an end-user context `hr.hcm_context` containing `emp_id` and `org_id`, while a Customer Relationship Management (CRM) application can own an end-user context

`ct.crm_context` containing `territory_id`. This isolation prevents conflicts between different applications running on the same database.

At runtime, the application instantiates and initializes the required end-user context by passing the context and attributes through client drivers. Attribute values can be set directly in code or through event-handler functions that run on first read (lazy loading). Only authorized code can set or modify application-specific attributes.

- **Default context**

  The database provides two predefined, system-managed end-user contexts that are always available and do not require explicit instantiation. You cannot modify attributes in these contexts directly; they are initialized by the database server based on the user identity propagated from IAM.

  - **`USER.DEFAULT`**: Contains standard end-user identifiers and database session information (such as `USERNAME, DB_NAME`).
  - **`USER.TOKEN`**: Contains claims extracted directly from the end-user security context token. Only the following claims are available: `iss [issuer]`, `sub [subject]`, and `aud [audience]`.

> **See Also**
>
> Configure End-User Contexts and Attributes.

The `ORA_END_USER_CONTEXT` function

You can reference end-user context attributes using JSON dot-path notation with the new SQL function, `ORA_END_USER_CONTEXT`. For example, you can reference a system attribute from the `USER.DEFAULT` context in a data grant as follows:

```
ORA_END_USER_CONTEXT.username
```

Similarly, if an HCM application instantiates an end-user context `hr.hcm_context` with an attribute `org_id`, the application code can refer to the `org_id` value as follows:

```
ORA_END_USER_CONTEXT.hr.hcm_context.org_id
```

For the complete syntax details, see Read End-User Context Attributes.

2.4.4 Privilege Elevation

An application can dynamically enable additional data roles within an end-user security context to temporarily elevate privileges.

This allows the application to perform restricted database operations on behalf of users who are not directly authorized to perform them, without permanently granting those privileges to the users.

For example, a product marketing manager may need to view total sales for a product but must not have access to individual orders or customer data. The application temporarily elevates privileges in the end-user security context to run a sales-summary query, then reverts to the user's original access level, ensuring that the underlying granular data remains protected.

The following sections illustrate how you can implement privilege elevation for this example scenario.

Configure data roles and grants

To implement privilege elevation, you must first create a data role managed locally that contains the necessary privileges. You can configure these roles to be disabled by default, ensuring they are only active when explicitly required by the application.

Use the following command to create a data role as disabled specifically for viewing the sales summary:

```
CREATE OR REPLACE DATA ROLE summarize_sales_role DISABLED;
```

Next, define a data grant to assign specific permissions to the data role.

In this example, the `oe.order_items` table contains individual order line items, including product, quantity, and unit price. The sales-summary query aggregates this data without exposing individual order details.

The following command grants the data role permission to select from the `oe.order_items` table:

```
CREATE OR REPLACE DATA GRANT SummarizeSalesOrderItems AS
SELECT
ON oe.order_items
TO summarize_sales_role;
```

Establish and authorize the application identity

To authorize the application, first create an application identity in the database that maps to the application's external client identifier (such as its client ID in Microsoft Entra ID). Then, grant the required data roles to that identity. The application can only enable data roles that have been explicitly granted to its identity.

Use the following command to create an application identity `hcm_app`, and map it to a specific client ID in Microsoft Entra ID:

```
CREATE OR REPLACE APPLICATION IDENTITY hcm_app
MAPPED TO 'AZURE_CLIENT_ID=f1fab37e-7aa2-4ff8-849c-7e731fea3b48';
```

Grant the previously created data role to the application identity, so the application can elevate privileges at runtime.

```
GRANT DATA ROLE summarize_sales_role TO hcm_app;
```

Runtime execution

At runtime, the application enables its assigned data roles through client driver APIs such as JDBC or Python drivers. The application elevates privileges only for the duration of a specific query, establishing a confined security context for that execution.

This approach provides two key security benefits:

- **Confined context:** Elevated privileges are scoped to a specific operation, limiting access even for SQL generated by AI agents.

- **Risk mitigation:** Because elevated privileges are bound to the application's logic and are active only during execution, the risk of SQL injection attacks gaining unintended access is significantly reduced.

For an example of controlled privilege elevation, see Configure Oracle Deep Data Security for a Sample Application.

Part II

Quick Starts: Configure Oracle Deep Data Security

Explore quick-start tutorials that demonstrate various Oracle Deep Data Security (Deep Sec) configuration scenarios, ranging from basic setups to more advanced environments. Use them as a starting point for your own implementation.

Topics:

3

Configure Oracle Deep Data Security for Direct Logon with Local End Users

In this quick-start chapter, you learn how to configure Oracle Deep Data Security (Deep Sec) in an Oracle AI Database environment to enable direct logon for locally managed end users.

This is the simplest scenario to configure and is ideal for development and testing environments, or product demonstrations.

As part of this chapter, you will:

- Create a sample HR schema with employee records.
- Create end users (Marvin and Emma) with password authentication.
- Create data roles and grant them to the end users.
- Define data grants to enforce role-based access control, so Marvin and Emma see only the data they are authorized to access.
- Validate the configuration by logging in as each user and querying the protected table.

No external identity and access management (IAM) system, TLS configuration, or application is required. End users authenticate directly to the database using their user name and password.

> **Note**
>
> For a sample script that runs this entire scenario, see Scripts for Direct Logon with Local End Users.

Overview of tasks

Task	Topic
Review prerequisites	Before You Begin
Create a sample HR schema and populate it with employee records	Create Sample Data
Create local end users with password authentication	Create Local End Users
Create data roles and grant database privileges	Configure Data Roles
Define data grants to enforce row-level and column-level access control	Configure Data Access Control
Log in as each user and verify role-based data access	Validate Data Access Control

3.1 Before You Begin

Ensure that you meet the following prerequisites before you begin.

- **Software:** Oracle AI Database (23.26.2 or later) installed on a Linux host.
- **Database access:** A named database user with the DBA role.
- **SQL client:** SQL*Plus or another SQL client to connect to the database.

3.2 Create Sample Data

Create a sample HR schema with an `employees` table to test data access control.

1. Connect to the database as a named user with the DBA role and switch to the target pluggable database.

   ```
   ALTER SESSION SET CONTAINER = <your-target-PDB>;
   ```

2. Create a sample HR schema and populate it with an employees table. If you already have an `hr.employees` table, drop it before proceeding.

   > **Note**
   >
   > The `CREATE USER` statement below uses the `USERS` tablespace. If this tablespace does not exist in your database, create it.
   >
   > ```
   > CREATE TABLESPACE users
   >   DATAFILE '<data_file_path>' SIZE 100M
   >   AUTOEXTEND ON;
   > ```
   >
   > Replace `<data_file_path>` with the full path for the data file.

```
-- Create the HR user
CREATE USER hr NO AUTHENTICATION
  DEFAULT TABLESPACE users
  QUOTA UNLIMITED ON users;

-- Create the employees table
CREATE TABLE hr.employees (
    employee_id   NUMBER PRIMARY KEY,
    first_name    VARCHAR2(50),
    last_name     VARCHAR2(50),
    email         VARCHAR2(128),
    manager       VARCHAR2(128),
    ssn           VARCHAR2(20),
    salary        NUMBER(10,2),
    phone         VARCHAR2(20)
);
```

3. Populate the table with sample employee records.

```
INSERT INTO hr.employees VALUES
  (100, 'Victoria', 'Williams', 'vwilliams',
   NULL, '219-09-9999', 13000, '555-0100');

INSERT INTO hr.employees VALUES
  (200, 'Marvin', 'Anderson', 'manderson',
   'vwilliams', '457-55-5462', 12030, '555-0200');

INSERT INTO hr.employees VALUES
  (300, 'Chris', 'Evans', 'cevans',
   'vwilliams', '321-12-4567', 6900, '555-0300');

INSERT INTO hr.employees VALUES
  (400, 'Emma', 'Baker', 'ebaker',
   'manderson', '733-02-9821', 8200, '555-0400');

INSERT INTO hr.employees VALUES
  (500, 'Taylor', 'Mills', 'tmills',
   'manderson', '558-76-1243', 9000, '555-0500');

COMMIT;
```

After loading the data, the `hr.employees` table contains the following records:

```
EMPLOYEE_ID  FIRST_NAME  LAST_NAME  EMAIL       MANAGER     SSN
SALARY  PHONE
-----------  ----------  ---------  ----------  ----------  -----------
------  --------
100          Victoria    Williams   vwilliams               219-09-9999
13000   555-0100
200          Marvin      Anderson   manderson   vwilliams   457-55-5462
12030   555-0200
300          Chris       Evans      cevans      vwilliams   321-12-4567
6900   555-0300
400          Emma        Baker      ebaker      manderson   733-02-9821
8200   555-0400
500          Taylor      Mills      tmills      manderson   558-76-1243
9000   555-0500
```

3.3 Create Local End Users

Create two local end users with password authentication. Unlike traditional database users, local end users do not own schemas or database objects. They receive fine-grained data access through data grants assigned to their data roles.

When an end user logs in, `ORA_END_USER_CONTEXT.username` returns their user name. The data grants in this chapter use predicates that compare this value against the `email` and `manager` columns of the `hr.employees` table. For the predicates to match, the local end-user names you create must match the values stored in those columns.

1. For Marvin, create an end user named `manderson`.

   ```
   CREATE END USER "manderson" IDENTIFIED BY <password>;
   ```

2. For Emma, create an end user named `ebaker`.

   ```
   CREATE END USER "ebaker" IDENTIFIED BY <password>;
   ```

3.4 Configure Data Roles

Create data roles for the employee and manager job functions.

1. First, create the data roles, then create a standard database role with the `CREATE SESSION` privilege and grant it to each data role that must support direct logon. In this scenario, grant it to the `employee_role` data role, which you'll assign to the end users in this example.

   ```
   -- Create data roles that are managed locally in the database
   CREATE DATA ROLE employee_role;
   CREATE DATA ROLE manager_role;

   -- Create a standard database role for connection privileges
   CREATE ROLE db_role;
   GRANT CREATE SESSION TO db_role;

   -- Grant the connection privileges to the data role used for direct logon
   GRANT db_role TO employee_role;
   ```

2. Grant the data roles to the local end users. Marvin receives both the manager and employee roles. Emma receives only the employee role.

   ```
   -- Grant data roles to Marvin (manager and employee)
   GRANT DATA ROLE manager_role TO "manderson";
   GRANT DATA ROLE employee_role TO "manderson";

   -- Grant the employee data role to Emma
   GRANT DATA ROLE employee_role TO "ebaker";
   ```

3.5 Configure Data Access Control

Define data grants to control which rows and columns each end user can access. In data grant predicates, `ORA_END_USER_CONTEXT.username` returns the end user's name from the current end-user security context.

> **Note**
>
> To query `ORA_END_USER_CONTEXT` directly or to use it in data grant predicates, you must set the database instance's `COMPATIBLE` initialization parameter to `20.0` or greater.

1. Create a data grant to allow employees to view their own record.

This data grant allows users with the `employee_role` data role to view only the row in `hr.employees` where the `email` column matches their user name.

```
CREATE DATA GRANT hr.employees_own_record
  AS SELECT
  ON hr.employees
  WHERE email = ORA_END_USER_CONTEXT.username
  TO employee_role;
```

2. Create a data grant to allow managers to view their direct reports (with SSN excluded).

   This data grant allows users with the `manager_role` data role to view the rows in `hr.employees` where the `manager` column matches their user name. The `ALL COLUMNS EXCEPT ssn` clause excludes the social security number column, so it returns `NULL` for these rows.

```
CREATE DATA GRANT hr.manager_direct_reports
  AS SELECT (ALL COLUMNS EXCEPT ssn)
  ON hr.employees
  WHERE manager = ORA_END_USER_CONTEXT.username
  TO manager_role;
```

3.6 Validate Data Access Control

Log in as each end user and query the `hr.employees` table to verify that the data grants enforce the expected access control.

1. Verify manager access (Marvin).

   a. Log in as Marvin.

```
sqlplus '"manderson"'/<password>@//<host>:<port>/<PDB-service-name>
```

   b. Confirm that the session is using Marvin's identity.

```
SELECT ORA_END_USER_CONTEXT.username FROM dual;
```

   You see the following output.

```
USERNAME
----------
"manderson"
```

   c. Query the employees table.

```
SELECT * FROM hr.employees;
```

   Marvin holds both the `employee_role` and `manager_role` data roles. As an employee, he can view his own record (including his SSN). As a manager, he can also view his direct reports (Emma and Taylor), but their SSN values are returned as `NULL`.

```
EMPLOYEE_ID  FIRST_NAME  LAST_NAME  EMAIL       MANAGER
SSN          SALARY  PHONE
-----------  ----------  ---------  ----------  ----------
```

```
-----------  ------  --------
200          Marvin      Anderson   manderson   vwilliams
457-55-5462  12030   555-0200
400          Emma        Baker      ebaker
manderson                 8200   555-0400
500          Taylor      Mills      tmills
manderson                 9000   555-0500
```

2. Verify employee access (Emma).
   a. Log in as Emma.

      ```
      sqlplus '"ebaker"'/<password>@//<host>:<port>/<PDB-service-name>
      ```

   b. Confirm that the session is using Emma's identity.

      ```
      SELECT ORA_END_USER_CONTEXT.username FROM dual;
      ```

      You see the following output.

      ```
      USERNAME
      ----------
      "ebaker"
      ```

   c. Query the employees table.

      ```
      SELECT * FROM hr.employees;
      ```

      Emma holds only the `employee_role` data role. She can view only her own record.

      ```
      EMPLOYEE_ID  FIRST_NAME  LAST_NAME  EMAIL   MANAGER    SSN
      SALARY  PHONE
      -----------  ----------  ---------  ------  ---------  -----------
      ------  --------
      400          Emma        Baker      ebaker  manderson  733-02-9821
      8200   555-0400
      ```

4

Configure Oracle Deep Data Security for Direct Logon with End Users in IAM

In this quick-start chapter, you learn how to configure Oracle Deep Data Security (Deep Sec) in an Oracle AI Database environment to enable direct logon for users provisioned in Microsoft Entra ID.

As part of this chapter, you will:

- Set up a scenario in which users connect directly to an application database, establish a Deep Sec session, and perform basic operations.
- Implement role-based access control (RBAC) for two users, Marvin and Emma, by provisioning them in Microsoft Entra ID and assigning different roles. Their roles determine their level of access to data in the application database.
- Use a single-host setup in which the database client and server run on the same machine. You'll use SQL*Plus, included with Oracle AI Database, as the client.
- Use the OAuth 2.0 authorization flow natively supported by Oracle AI Database to authenticate users through browser-based sign-in.

> **Note**
>
> For a sample script that performs the database-side configuration for this scenario, see Scripts for Direct Logon with End Users in IAM. You must complete the Microsoft Entra ID, TLS, and client configuration steps manually.

Overview of tasks

Task	Topic
Review prerequisites for the tutorial	Before You Begin
Register applications and define application-specific roles in Microsoft Entra ID	Create Application Registrations in Microsoft Entra ID
Create users in Microsoft Entra ID and assign roles	Manage Users and Role Assignments in Microsoft Entra ID
Generate client and server credentials (Oracle wallets and certificates) for encryption	Generate Wallets and Certificates
Configure the database listener for secure TCPS connections	Configure the Database Listener
Configure the SQL*Plus client	Configure the Client for Interactive Sign-In
Configure data access control using Oracle Deep Data Security capabilities	Configure Data Access Control
Verify the signed-in identity and role-based data access	Validate User Connections and Data Access Control

4.1 Before You Begin

Ensure that you meet these prerequisites before starting the tasks in this chapter.

- **Software:** Oracle AI Database (23.26.2 or later) installed on a Linux host.
- **Microsoft Entra ID access:** A Microsoft Entra account with permissions to create application registrations, define roles, and manage users in Microsoft Entra ID.
- **Web browser:** A modern browser on the host (used for OAuth sign-in).

4.2 Create Application Registrations in Microsoft Entra ID

In your Entra ID Default Directory, create two application registrations: one for your database (resource) and one for SQL*Plus (client). This step establishes trust between your client application, database resource, and Microsoft Entra ID.

> **Note**
>
> The Microsoft Entra portal interface may be updated over time. If a specific label or navigation path differs from the instructions provided here, look for the closest matching option.

4.2.1 Register the Database Resource

Register your database by creating an application in Microsoft Entra ID so that access tokens can be issued specifically for the database. Subsequently, expose the database application as a web API to manage client access, and define your application-specific roles that govern user permissions.

1. Create an application registration in Microsoft Entra ID to represent the database.
   a. Log in to the Microsoft Entra Portal.
   b. In the left navigation pane, expand **Entra ID**, click **App registrations**, and then click **New registration**.
   c. Perform the following tasks on the Register an application page:
      i. Enter `OracleDB_Resource` in the **Name** field.
      ii. For **Supported account types**, select **Single tenant only - Default Directory**.
      iii. Click **Register**.

      The application is successfully created for your database.
   d. From the application's Overview page, copy and save the following values for later use:
      - **Application (client) ID** (referenced later as `[DB_APP_ID]`).
      - **Directory (tenant) ID** (referenced later as `[TENANT_ID]`).

2. Expose the database application as a web API, and define a scope to control access for client applications.
   a. On the application's Overview page, click **Add an Application ID URI**.

      The Expose an API page opens.
   b. Add an application ID URI.
      i. Click **Add** next to Application ID URI.
      ii. In the panel that appears, update the default URI by replacing `api://` with `https://<your-entraID-domain>/`, then click **Save**.
      The resulting application ID URI should resemble: `https://supremo.onmicrosoft.com/fe58fefb-0925-4c8f-9b14-598a0d2f4552`.
      iii. Copy this URI for later use (referenced later as `[DB_APP_ID_URI]`).
   c. Add a scope.
      i. Under **Scopes defined by this API**, click **Add a scope**.
      ii. In the panel that appears, enter the following information:
         - **Scope name:** `sessions:scope:connect`.
         - **Who can consent:** Select **Admins and users**.
         - Enter `Access Oracle Database` as the value in all remaining fields.
         - Click **Add scope**.
3. Define application-specific roles in Entra ID to centrally manage job functions (for example, manager or employee).

   Later, you can create data roles in the database and map them to these roles.
   a. On the `OracleDB_Resource` application page, under **Manage**, click **App roles**.

      The App roles page opens.
   b. Click **Create app role** to create the **Manager** role:
      - **Display name:** Enter `MANAGER`.
      - **Allowed member types:** Select **Users/Groups**.
      - **Value:** Enter `MANAGER`.
      - **Description:** Enter `Full access to all records`.
      - Click **Apply**.
   c. Click **Create app role** to create the **Employee** role:
      - **Display name:** Enter `EMPLOYEE`.
      - **Allowed member types:** Select **Users/Groups**.
      - **Value:** Enter `EMPLOYEE`.
      - **Description:** Enter `Access to own records only`.
      - Click **Apply**.

4.2.2 Register the Client Application

Register SQL*Plus by creating an application in Microsoft Entra ID, and authorize it to request access tokens for your database on behalf of signed-in users.

1. Create an application registration in Microsoft Entra ID to represent SQL*Plus.
   a. On the Microsoft Entra portal's Home page, click **App registrations** in the left navigation pane under **Entra ID**, and then click **New registration**.
   b. Perform the following tasks on the Register an application page:
      i. Enter `SQLPlus_Client` in the **Name** field.
      ii. For **Supported account types**, select **Single tenant only - Default Directory**.
      iii. Under **Redirect URI**, select **Public client/native (mobile & desktop)** from the drop-down field, and enter `http://localhost`.

      > **Note**
      >
      > This URI allows the browser to return the authentication token to SQL*Plus running on your local machine.

      iv. Click **Register**.

      The application is successfully created for your client.
   c. From the application's Overview page, copy and save the **Application (client) ID** (referenced later as `[CLIENT_APP_ID]`).
2. Grant permissions to the client application.

   You must explicitly authorize the client application (`SQLPlus_Client`) to access the database application (`OracleDB_Resource`). This permission allows SQL*Plus to request valid authentication tokens on behalf of the signed-in user.
   a. On the `SQLPlus_Client` application page, under **Manage**, click **API permissions**, and then click **Add a permission**.
   b. Perform the following tasks on the Request API permissions panel:
      i. Click **APIs my organization uses**, and then click **OracleDB_Resource**.
      ii. Select **Delegated permissions** and check the box for `sessions:scope:connect`.
      iii. Click **Add permissions**.

      The permission is successfully added and appears on the API permissions page.
   c. Click **Grant admin consent for Default Directory** (or your specific directory name) to authorize the permissions, and select **Yes** in the confirmation dialog box.

4.3 Manage Users and Role Assignments in Microsoft Entra ID

Create two test users and assign them distinct application roles to simulate an organizational hierarchy in a sample HR application schema.

1. Create new test users in your Microsoft Entra ID domain.
   a. On the Microsoft Entra portal's Home page, click **Users** in the left navigation pane under **Entra ID**.

      The Users page opens.
   b. Click **New user**, and then **Create new user**.
   c. Create the first user (Marvin) with the following details:
      - **User principal name:** `marvin@<your-directory>.onmicrosoft.com`

- **Display name:** `marvin`
- **Password:** Set a password for the user
- **Account enabled:** Leave this field checked

Click **Review + create**, and then click **Create**. From the Users page, copy Marvin's full user principal name (UPN).

d. Create the second user (Emma) with the following details:

- **User principal name:** `emma@<your-directory>.onmicrosoft.com`
- **Display name:** `emma`
- **Password:** Set a password for the user
- **Account enabled:** Leave this field checked

Click **Review + create**, and then click **Create**. From the Users page, copy Emma's full user principal name (UPN).

2. Assign users to the database application in Entra ID to enable sign-in.

Select specific application roles to define their data access privileges (manager or employee).

a. In the left navigation pane under **Entra ID**, click **Enterprise apps**, and then select **OracleDB_Resource**.

b. Click **Assign users and groups** in the Getting Started section of the application's Overview page.

c. On the Users and groups page, click **Add user/group** to assign a new user to the application.

d. To assign Marvin, perform the following actions:

i. Select **Marvin** under **Users**.

ii. Select **Manager** under **Select a role**.

iii. Click **Assign**.

iv. Repeat the steps for Marvin, but this time, add the **Employee** role.

e. To assign Emma, perform the following actions:

i. Select **Emma** under **Users**.

ii. Select **Employee** under **Select a role**.

iii. Click **Assign**.

3. Initialize Microsoft Entra ID users.

New users you created in Microsoft Entra ID have temporary passwords. Update these to permanent passwords in a browser before attempting to sign in with SQL*Plus, as SQL*Plus cannot handle the *Force Change Password* prompt.

a. Open a private or incognito browser window.

b. Go to `https://myapps.microsoft.com`.

c. Sign in as Marvin using the user principal name (`marvin@<your-directory>.onmicrosoft.com`) and the temporary password, then change the password.

d. If the sign-in flow requires additional security, follow the prompts to configure the Microsoft Authenticator app as your second verification method.

When the My Apps dashboard appears, close the browser. The user is now active.

e. Repeat the previous steps to set a permanent password to Emma.

4.4 Generate Wallets and Certificates

Secure the client-to-database connection with Transport Layer Security (TLS). Without TLS, the database rejects token-based authentication. Use the orapki (Oracle Public Key Infrastructure) utility to create a server wallet and a self-signed certificate. In this single-host setup, SQL*Plus also uses the same server wallet as its trust store.

> **Note**
>
> - Run all commands as the operating system user that owns the Oracle software (typically `oracle`).
> - Set Common Name (CN) to the database server's fully qualified domain name.
> - For more details on TLS configuration, see Configuring Transport Layer Security Encryption in *Oracle AI Database Security Guide*.

1. Open a terminal on the Linux host.
2. Create a directory for the server wallet.

```
mkdir -p /u01/app/oracle/wallets/server
```

3. Create the server wallet (identity) and a self-signed certificate.

```
orapki wallet create -wallet /u01/app/oracle/wallets/server -pwd
<wallet_password> -auto_login

orapki wallet add -wallet /u01/app/oracle/wallets/server -dn "CN=<your-vm-
fqdn>,O=Oracle,C=US" -keysize 2048 -self_signed -validity 3650 -pwd
<wallet_password>
```

> **Note**
>
> This guide uses a self-signed certificate and a single wallet directory for simplicity. For production deployments, use CA-signed certificates, separate client and server trust stores, and a secret manager for wallet passwords. Oracle recommends configuring the `WALLET_ROOT` initialization parameter and storing the TLS wallet under `WALLET_ROOT/<PDB GUID>/tls`. See Configuring Transport Layer Security Encryption in *Oracle AI Database Security Guide*.

4.5 Configure the Database Listener

Configure the database listener to accept TCPS connections on port 2484 using the server wallet you created.

1. Go to the database network configuration directory.

   ```
   cd $ORACLE_HOME/network/admin
   ```

2. Open `listener.ora` in a text editor (for example, vi or nano).

   ```
   vi listener.ora
   ```

3. Add or update the wallet location and the TCPS address as shown below and save the file.

   Use the database server's fully qualified domain name for `HOST`.

   ```
   WALLET_LOCATION = (SOURCE = (METHOD = FILE) (METHOD_DATA = (DIRECTORY
   = /u01/app/oracle/wallets/server)))

   TLS_CLIENT_AUTHENTICATION = FALSE

   LISTENER =
    (DESCRIPTION_LIST =
     (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = <your-vm-fqdn>)(PORT = 1521))
      (ADDRESS = (PROTOCOL = TCPS)(HOST = <your-vm-fqdn>)(PORT = 2484))
     )
    )
   ```

4. Restart the listener.

   ```
   lsnrctl stop
   lsnrctl start
   ```

4.6 Configure the Client for Interactive Sign-In

Configure SQL*Plus to trigger the browser-based OAuth sign-in.

1. Configure network and token authentication (`sqlnet.ora`).

   This step enables token-based authentication and directs the client to the security wallet.

   a. In `$ORACLE_HOME/network/admin`, open `sqlnet.ora`.

      ```
      vi sqlnet.ora
      ```

   b. Add the following configuration.

      > **Note**
      >
      > As this is a single-host setup, you'll configure both the client and the server to use the server wallet to avoid key-certificate mismatch issues during TLS authentication.

      ```
      # --- Authentication ---
      SQLNET.AUTHENTICATION_SERVICES = (BEQ, TOKENAUTH)

      # --- TLS Configuration ---
      ```

```
TLS_CLIENT_AUTHENTICATION = FALSE
TLS_SERVER_DN_MATCH = TRUE

# Prevents conflicts between Native Encryption and TLS
SQLNET.IGNORE_ANO_ENCRYPTION_FOR_TCPS = TRUE

# Pointing both Client and Server to the Server wallet avoids
# the "missing private key" issue in single-box setups.
WALLET_LOCATION = (SOURCE = (METHOD = FILE) (METHOD_DATA = (DIRECTORY
= /u01/app/oracle/wallets/server)))
```

2. Create the database connection alias (`tnsnames.ora`).

   This file defines the connection string for your database. It includes the Microsoft Entra ID application details required to trigger the browser sign-in.

   a. In `$ORACLE_HOME/network/admin`, open `tnsnames.ora`.

   ```
   vi tnsnames.ora
   ```

   b. Add the following entry. Ensure you replace the placeholder values with the IDs you saved earlier. See Register the Database Resource and Register the Client Application.

   Additionally, use the database server's fully qualified domain name (FQDN) for both `HOST` and `CN` fields, and specify the service name of your target PDB.

   ```
   db_entraid =
     (DESCRIPTION =
       (ADDRESS = (PROTOCOL = TCPS)(HOST = <your-vm-fqdn>)(PORT = 2484))
       (CONNECT_DATA =
         (SERVICE_NAME = <your-pdb-service-name>)
       )
       (SECURITY =
         (TLS_SERVER_CERT_DN = "CN=<your-vm-fqdn>,O=Oracle,C=US")

         # Use OAuth 2.0 (Authorization Code Flow)
         (TOKEN_AUTH = AZURE_INTERACTIVE)

         # App Registration Details
         (CLIENT_ID = <[CLIENT_APP_ID]>)
         (TENANT_ID = <[TENANT_ID]>)
         (AZURE_DB_APP_ID_URI = <[DB_APP_ID_URI]>)
       )
     )
   ```

4.7 Configure Data Access Control

In this section, you'll configure data access control using Oracle Deep Data Security (Deep Sec) capabilities.

You'll perform the following tasks:

- Configure the database to accept external identities.
- Create data roles, which map to Entra ID roles.
- Create sample HR application data.

- Define data grants to restrict access to Marvin and Emma based on their roles.

1. Connect to the database as a named user with the DBA role.
2. Switch to the target pluggable database (for example, `ORCLPDB`). Run the following command.

   ```
   ALTER SESSION SET CONTAINER = <your-target-PDB>;
   ```

3. Enable direct logon for Microsoft Entra ID users. Use the values from Register the Database Resource to replace the placeholders.

   ```
   ALTER SYSTEM SET IDENTITY_PROVIDER_TYPE=AZURE_AD SCOPE=BOTH;
   ALTER SYSTEM SET IDENTITY_PROVIDER_CONFIG = '{
      "application_id_uri": "[DB_APP_ID_URI]",
      "tenant_id": "[TENANT_ID]",
      "app_id": "[DB_APP_ID]"
   }' SCOPE=BOTH;
   ```

4. Configure data roles and permissions.

   Instead of creating a database user for each employee (Marvin or Emma), define data roles that map to the application roles defined in Microsoft Entra ID. You can then assign access privileges to these data roles as required.

   Additionally, create a standard database role with privileges to establish a database session and grant this role to each data role.

   ```
   -- Create data roles mapped to Microsoft Entra ID roles
   CREATE DATA ROLE employee_role MAPPED TO 'AZURE_ROLE=EMPLOYEE';
   CREATE DATA ROLE manager_role MAPPED TO 'AZURE_ROLE=MANAGER';

   -- Create a standard database role for connection privileges
   CREATE ROLE db_role;
   GRANT CREATE SESSION TO db_role;

   -- Grant the connection privileges to the data roles
   GRANT db_role TO employee_role;
   GRANT db_role TO manager_role;
   ```

5. Create sample data.

   To test the data access control capabilities of Deep Sec, create a sample HR application schema and populate it with an employees table.

   a. To create a sample HR schema, begin by creating the `hr` user. Next, create the `employees` table. If you already have an `hr.employees` table, drop it before proceeding.

> **Note**
>
> The `CREATE USER` statement below uses the `USERS` tablespace. If this tablespace does not exist in your database, create it.
>
> ```
> CREATE TABLESPACE users
>   DATAFILE '<data_file_path>' SIZE 100M
>   AUTOEXTEND ON;
> ```
>
> Replace `<data_file_path>` with the full path for the data file.

```
-- Create the HR user
CREATE USER hr NO AUTHENTICATION
  DEFAULT TABLESPACE users
  QUOTA UNLIMITED ON users;

CREATE TABLE hr.employees (
    employee_id   NUMBER PRIMARY KEY,
    first_name    VARCHAR2(50),
    last_name     VARCHAR2(50),
    email         VARCHAR2(128),
    manager       VARCHAR2(128),
    ssn           VARCHAR2(20),
    salary        NUMBER(10,2),
    phone         VARCHAR2(20)
);
```

b. Populate the `employees` table with the following sample data.

   Replace `<your-entraID-domain>` in the SQL statements with your actual Microsoft Entra ID domain (for example, `supremo.onmicrosoft.com`).

```
INSERT INTO hr.employees VALUES (100, 'Victoria', 'Williams',
  'victoria@<your-entraID-domain>', NULL,
  '219-09-9999', 13000, '555-0100');

INSERT INTO hr.employees VALUES (200, 'Marvin', 'Anderson',
  'marvin@<your-entraID-domain>', 'victoria@<your-entraID-domain>',
  '457-55-5462', 12030, '555-0200');

INSERT INTO hr.employees VALUES (300, 'Chris', 'Evans',
  'chris@<your-entraID-domain>', 'victoria@<your-entraID-domain>',
  '321-12-4567', 6900, '555-0300');

INSERT INTO hr.employees VALUES (400, 'Emma', 'Baker',
  'emma@<your-entraID-domain>', 'marvin@<your-entraID-domain>',
  '733-02-9821', 8200, '555-0400');

INSERT INTO hr.employees VALUES (500, 'Taylor', 'Mills',
  'taylor@<your-entraID-domain>', 'marvin@<your-entraID-domain>',
  '558-76-1243', 9000, '555-0500');

COMMIT;
```

After you load the sample data, the `hr.employees` table contains the following records:

```
EMPLOYEE_ID  FIRST_NAME  LAST_NAME  EMAIL
MANAGER                          SSN          SALARY  PHONE
-----------  ----------  ---------  ----------------------------
-----------------------------   -----------  ------  --------
100          Victoria    Williams   victoria@<your-entraID-
domain>                                  219-09-9999  13000   555-0100
200          Marvin      Anderson   marvin@<your-entraID-domain>
victoria@<your-entraID-domain>   457-55-5462  12030   555-0200
300          Chris       Evans      chris@<your-entraID-domain>
victoria@<your-entraID-domain>   321-12-4567  6900    555-0300
400          Emma        Baker      emma@<your-entraID-domain>
marvin@<your-entraID-domain>     733-02-9821  8200    555-0400
500          Taylor      Mills      taylor@<your-entraID-domain>
marvin@<your-entraID-domain>     558-76-1243  9000    555-0500
```

6. Implement data access control.

   You'll use data grants to define table-level access rules and column masks. In data grant statements, you'll use a SQL function, `ORA_END_USER_CONTEXT`, to retrieve session-context parameters for the signed-in user as a JSON data type. For example, `ORA_END_USER_CONTEXT.username` returns the user's name (UPN) from the current session. Create the following two data grants in the `hr` schema.

   a. Create the `employees_own_record` data grant for the employee data role (`employee_role`).

      This data grant ensures that users assigned the `employee_role` can access only those rows where the value in the `email` column matches their login user name.

      ```
      -- Grant 1: Allow employees to see only their own record
      CREATE DATA GRANT hr.employees_own_record
        AS SELECT
        ON hr.employees
        WHERE email = ORA_END_USER_CONTEXT.username
        TO employee_role;
      ```

   b. Next, create the `manager_direct_reports` data grant for the manager data role (`manager_role`).

      This data grant ensures that users assigned the `manager_role` (for example, Marvin) can access all those rows where the value in the `manager` column matches their login user name.

      Additionally, in the data grant, you'll use the clause `ALL COLUMNS EXCEPT ssn`. This ensures that, although Marvin can view the rows for his direct reports, the social security number (SSN) column returns `NULL` for those rows, preventing access to this sensitive information.

      ```
      -- Grant 2: Allow managers to see their direct reports (with SSN
      excluded)
      CREATE DATA GRANT hr.manager_direct_reports
        AS SELECT (ALL COLUMNS EXCEPT ssn)
        ON hr.employees
        WHERE manager = ORA_END_USER_CONTEXT.username
        TO manager_role;
      ```

4.8 Validate User Connections and Data Access Control

In this final section, you'll validate the security configuration. You'll log in as different users (Marvin and Emma), confirm the signed-in identity, and then query the `employees` table to verify that role-based access is applied as intended.

> **Note**
>
> - If your database server requires egress through a network proxy to validate Microsoft Entra ID tokens, set the HTTP proxy and restart the database listener.
>
>   ```
>   export http_proxy=http://<your-proxy-host>:<port>/
>   export https_proxy=http://<your-proxy-host>:<port>/
>   lsnrctl stop
>   lsnrctl start
>   ```
>
> - To make the query output easier to read in SQL*Plus, set page and column widths before you run the queries. For example, you can use the following settings:
>
>   ```
>   SET LINES 200 PAGES 100
>   COL employee_id  FOR 9999
>   COL first_name   FOR a10
>   COL last_name    FOR a10
>   COL email        FOR a10
>   COL manager      FOR a10
>   COL ssn          FOR a15
>   COL salary       FOR 99999
>   COL phone        FOR a10
>   ```

1. Verify manager access (Marvin).
   a. Open a new terminal window.
   b. Initiate a connection using the slash / syntax and your TNS alias.

      ```
      sqlplus /@db_entraid
      ```

   c. When your browser opens the Microsoft sign-in page, sign in as Marvin.
   d. After the browser displays *Authentication Complete*, close the window and return to your terminal.

      You're now logged in to the database.

   e. Run the following query to confirm the database session is using Marvin's identity.

      ```
      SELECT ORA_END_USER_CONTEXT.username FROM dual;
      ```

      You see the following output.

      ```
      USERNAME
      --------------------------------------------------------------------------------
      ```

```
--------
"marvin@<your-entraID-domain>"
```

f. Check Marvin's access to the employees table.

```
SELECT * FROM hr.employees;
```

You see Marvin's record and all his direct reports (Emma and Taylor), but the SSN column for others is `NULL`.

```
EMPLOYEE_ID  FIRST_NAME  LAST_NAME  EMAIL
MANAGER                       SSN          SALARY  PHONE
-----------  ----------  ---------  ----------------------------
----------------------------  -----------  ------  --------
200          Marvin      Anderson   marvin@<your-entraID-domain>
victoria@<your-entraID-domain>  457-55-5462  12030   555-0200
400          Emma        Baker      emma@<your-entraID-domain>
marvin@<your-entraID-domain>                 8200   555-0400
500          Taylor      Mills      taylor@<your-entraID-domain>
marvin@<your-entraID-domain>                 9000   555-0500
```

2. Verify employee access (Emma).
   a. Exit the current session and initiate a new connection.

```
exit
sqlplus /@db_entraid
```

   b. When your browser opens the Microsoft sign-in page, sign in as Emma.
   c. After the browser displays *Authentication Complete*, close the window and return to your terminal.

      You're now logged in to the database.

   d. Run the following query to confirm the database session is using Emma's identity.

```
SELECT ORA_END_USER_CONTEXT.username FROM dual;
```

      You see the following output.

```
USERNAME
--------------------------------------------------------------------------------
--------
"emma@<your-entraID-domain>"
```

   e. Check Emma's access to the employees table.

```
SELECT * FROM hr.employees;
```

      You only see Emma's record.

```
EMPLOYEE_ID  FIRST_NAME  LAST_NAME  EMAIL
MANAGER                       SSN          SALARY  PHONE
-----------  ----------  ---------  --------------------------
----------------------------  -----------  ------  --------
```

```
400          Emma        Baker      emma@<your-entraID-domain>
marvin@<your-entraID-domain>  733-02-9821   8200   555-0400
```

5

Configure Oracle Deep Data Security for a Sample Application

In this quick-start chapter, you'll learn how to configure Oracle Deep Data Security (Deep Sec) in an Oracle AI Database environment for a sample Spring Boot application.

This chapter demonstrates how to propagate end-user identities through a Spring Boot application to the database, enabling Deep Sec to enforce row-level and column-level access controls. It also demonstrates how to implement dynamic privilege elevation to securely perform restricted operations without permanently granting excessive permissions.

> **Note**
>
> - For a sample script that performs the database-side configuration for this scenario, see Scripts for the Employee Records Application. You must complete the Microsoft Entra ID, TLS, and Spring Boot application setup manually.
> - This quick start uses Microsoft Entra ID as the identity provider. You can also use Oracle Cloud Infrastructure Identity and Access Management (OCI IAM). For the corresponding configuration steps, see Configure OCI IAM for Application-Mediated Access.

Overview of tasks

The table below outlines the tasks you'll complete in this chapter.

Task	Topic
Learn about the sample application, the end-to-end security flow, and the Deep Sec capabilities you'll implement.	Concepts and Architecture
Review host, database, network, and software prerequisites	Before You Begin
Register the database and Spring Boot application in Microsoft Entra ID to establish trust and define application roles	Create Application Registrations in Microsoft Entra ID
Create a test user in Entra ID and assign required roles	Manage Users and Role Assignments
Create a server wallet and a self-signed certificate for TLS encryption	Generate Wallets and Certificates
Configure the database listener and network layer for TCPS connections and token-based authentication	Configure Database Network Settings
Create the sample `hr` schema, set up identity provider details, and define the Deep Sec data roles and data grants	Configure Data Access Control

Task	Topic
Clone the sample Spring Boot application, configure the JDBC provider, and set up environment variables	Set Up the Spring Boot Application
Build the application, obtain an Entra ID access token, and verify data access control and privilege elevation	Build, Run, and Verify

5.1 Concepts and Architecture

Explore the sample application, end-to-end security flow, and the Oracle Deep Data Security (Deep Sec) capabilities you'll implement.

About the sample application

In this chapter, you'll deploy the Employee Records API, which is a sample Spring Boot REST API serving employee data from a Human Resources (`hr`) schema. The application does not contain any custom access-control code. Instead, it relies entirely on Deep Sec data grants to determine which rows and columns a user can see, with the Oracle JDBC Spring Boot provider handling all security context propagation automatically.

The API exposes two endpoints:

- **GET /api/employees:** Returns the signed-in employee's own record. Access is controlled by the data role, `EMPLOYEE_ROLE`, which filters rows based on the user's email.
- **GET /api/employees/salary-summary:** Returns team-wide aggregate salary statistics (min, max, average, count). The application temporarily activates the `COMPENSATION_ANALYST` data role for the duration of the API call, securely calculating aggregates without exposing the individual salaries of other employees.

The end-to-end flow

In this application scenario, a client (`curl`) sends requests to the Spring Boot REST API, which then connects to Oracle AI Database.

The end-to-end security flow works as follows:

1. **End-user authentication:** The end user authenticates to Microsoft Entra ID using the OAuth 2.0 authorization-code flow with Proof Key for Code Exchange (PKCE) and obtains an access token.
2. **API request:** The user calls the Spring Boot REST API, passing the access token in the authorization header.
3. **End-user token validation:** Spring Security validates the incoming end-user token against Entra ID.
4. **Database-access token acquisition:** The Oracle JDBC Spring Boot provider uses the application's client credentials (client ID and client secret) to obtain a database-scoped access token from Entra ID. This occurs at pool initialization and on token refresh.
5. **Connection pool authentication:** The provider authenticates the application's connection pool user account to the database using the database-access token.
6. **Security context payload propagation:** On each request, the Spring Boot provider reads the end-user's JWT from the Spring Security context and attaches it to the database

connection, along with the application's database-access token. Together, these form the end-user security context payload.

7. **Database enforcement:** The database validates both tokens. It validates the database-access token and authorizes the application connection. It validates the end-user token and establishes the end-user security context, enabling the user's data roles (that are mapped to external application roles) and corresponding data grants. The database returns only the rows and columns the user is authorized to see.
8. **Privilege elevation:** For the salary-summary endpoint, the application temporarily activates an additional data role (`COMPENSATION_ANALYST`) through the `@RunWithDataRoles` annotation present in the application's code. This elevation is scoped to a single method call and deactivated automatically afterward.

What you'll implement

You'll implement two key security capabilities of Deep Sec in this quick start:

- **Security context propagation using only configuration changes** *(GET /api/employees)*: You'll configure the Oracle JDBC Spring Boot provider to automatically propagate the end-user security context payload to the database without making any application code changes.
- **Dynamic privilege elevation** *(GET /api/employees/salary-summary)*: You'll implement temporary privilege elevation so the application can perform restricted operations on behalf of end users without permanently granting those privileges. The sample application already includes the required code for this; however, if you were building your own application, adding the `@RunWithDataRoles` annotation to the target method would be your only application code change.
  - *How it works:* In the database, you'll create a data role as disabled and a corresponding data grant. You then grant this data role exclusively to the application identity so that only the application can enable it. At runtime, the application activates the data role using the `@RunWithDataRoles` annotation in its code. For the `/api/employees/salary-summary` endpoint, the service method in `EmployeeService.java` is decorated with the `@RunWithDataRoles` annotation. This method runs with the `COMPENSATION_ANALYST` data role whenever it is called:

    ```
    @RunWithDataRoles(
            dataRoles = {"COMPENSATION_ANALYST"}
    )
    @Transactional(readOnly = true)
    public SalarySummaryDto getSalarySummary() {
        ...
    }
    ```

    The framework keeps the data role active for the duration of this single method call and automatically deactivates it when the method returns.
  - *Why it matters:* By scoping elevated privileges to specific operations rather than granting them permanently to end users, you tightly confine access, limit exposure to AI-generated SQL, and significantly reduce the risk of SQL injection attacks.

  See Privilege Elevation.

5.2 Before You Begin

Ensure that you meet these prerequisites before starting the tasks in this chapter.

- **Oracle AI Database installation**: Version 23.26.2 or later, installed on a Linux host.
- **Command-line utilities:** `curl`, `openssl`, and Python 3 installed on the build or test host. Required to generate PKCE challenges, execute API requests, and format JSON responses.
- **Java installation:** Version 17 or later. Required for the Spring Boot application. If multiple Java versions are installed, set `JAVA_HOME` explicitly:

  ```
  export JAVA_HOME=/usr/lib/jvm/java-17-openjdk
  export PATH=$JAVA_HOME/bin:$PATH
  java -version # verify: openjdk version "17.x.x" or later
  ```

- **Maven installation:** Version 3.9 or later. Required to build the application. If installed manually or not on `PATH`, set `MAVEN_HOME` explicitly:

  ```
  export MAVEN_HOME=/opt/maven/apache-maven-3.9.x
  export PATH=$MAVEN_HOME/bin:$PATH
  mvn -version # verify: Apache Maven 3.9.x or later
  ```

- **Microsoft Entra ID access:** A Microsoft Entra account with domain or application administrator privileges. Required to create application registrations, define roles, and manage users in Microsoft Entra ID.
- **Network access:** Host connectivity to external endpoints. If your host is behind a firewall, configure an HTTP proxy for the following services:
  - **Microsoft Entra ID** (`login.microsoftonline.com`, `sts.windows.net`): Required for the database-access token validation. Set the `http_proxy` and `https_proxy` environment variables and restart the listener:

    ```
    export http_proxy=http://<your-proxy-host>:<port>/
    export https_proxy=http://<your-proxy-host>:<port>/
    lsnrctl stop
    lsnrctl start
    ```

  - **Maven Central** (`repo.maven.apache.org`): Required for downloading build dependencies. Create or edit `~/.m2/settings.xml` as follows:

    ```
    <settings>
          <proxies>
              <proxy>
                  <id>http-proxy</id>
                  <active>true</active>
                  <protocol>http</protocol>
                  <host><your-proxy-host></host>
                  <port><your-proxy-port></port>
              </proxy>
              <proxy>
                  <id>https-proxy</id>
                  <active>true</active>
                  <protocol>https</protocol>
    ```

```
                <host><your-proxy-host></host>
                <port><your-proxy-port></port>
            </proxy>
        </proxies>
    </settings>
```

5.3 Create Application Registrations in Microsoft Entra ID

In your Microsoft Entra ID directory, create two application registrations: one for the database and one for the Spring Boot application. This step establishes trust between the application, the database resource, and Microsoft Entra ID.

> **Note**
>
> The Microsoft Entra portal interface may be updated over time. If a specific label or navigation path differs from the instructions provided here, look for the closest matching option.

- Register the Database Resource
- Register the Spring Boot Application

5.3.1 Register the Database Resource

Register your database by creating an application in Microsoft Entra ID so that access tokens can be issued specifically for the database. Subsequently, expose this application registration as a web API to manage client access, and define a scope to control which applications can request database-access tokens.

1. Create an application registration in Microsoft Entra ID to represent the database.
   a. Log in to the Microsoft Entra Portal.
   b. In the left navigation pane, expand **Entra ID**, click **App registrations**, and then click **New registration**.
   c. Perform the following tasks on the Register an application page:
      i. Enter `EmployeeRecordsDB` in the **Name** field.
      ii. For **Supported account types**, select **Single tenant only - Default Directory**.
      iii. Click **Register**.

      The application is successfully created for your database.
   d. From the application's Overview page, copy and save the following values for later use:
      - **Application (client) ID** (referenced later as `[DB_APP_ID]`).
      - **Directory (tenant) ID** (referenced later as `[TENANT_ID]`).
2. Expose the database's application registration as a web API, and define a scope to control access for client applications.
   a. On the application's Overview page, click **Add an Application ID URI**.

      The Expose an API page opens.

b. Add an application ID URI.

   i. Click **Add** next to **Application ID URI**.

   ii. In the panel that appears, update the default URI by replacing `api://` with `https://<your-entraID-domain>/`, then click **Save**.
   The resulting application ID URI should resemble: `https://supremo.onmicrosoft.com/fe58fefb-0925-4c8f-9b14-598a0d2f4552`.

   iii. Copy this URI for later use (referenced later as `[DB_APP_ID_URI]`).

c. Add a scope.

   i. Under **Scopes defined by this API**, click **Add a scope**.

   ii. In the panel that appears, enter the following information:

   - **Scope name:** `sessions:scope:connect`.
   - **Who can consent:** Select **Admins and users**.
   - Enter `Access Oracle Database` as the value in all remaining fields.
   - Click **Add scope**.

5.3.2 Register the Spring Boot Application

Register the Spring Boot application by creating an application in Microsoft Entra ID and authorize it to request access tokens for your database.

This application serves two purposes: it is a confidential client that authenticates to Entra ID using a client secret, and it is also the resource server that end users authenticate against when calling the API. The application role defined on this registration flows into the user's JWT `roles` claim, which the database reads to activate the matching data role.

1. Create an application registration in Microsoft Entra ID to represent the Spring Boot application.

   a. On the Microsoft Entra portal's Home page, click **App registrations** in the left navigation pane under **Entra ID**, and then click **New registration**.

   b. Perform the following tasks on the Register an application page:

      i. Enter `EmployeeRecordsAPI` in the **Name** field.

      ii. For **Supported account types**, select **Single tenant only - Default Directory**.

      iii. Under **Redirect URI**, select **Web** from the drop-down field, and enter `http://localhost:3000`.

      iv. Click **Register**.

      The application is successfully created for your Spring Boot application.

   c. From the application's Overview page, copy and save the **Application (client) ID** (referenced later as `[EMP_RECORDS_APP_ID]`).

2. Set the application ID URI.

   This makes `EmployeeRecordsAPI` a resource, allowing users and other applications to request tokens scoped to it.

   a. On the application's Overview page, click **Add an Application ID URI**.

   b. Perform the following tasks on the Expose an API page:

      i. Click **Add** next to **Application ID URI**.

ii. In the panel that appears, accept the default value `api://<your-app-id>`, then click **Save**.

iii. Copy this URI for later use (referenced later as `[EMP_RECORDS_APP_ID_URI]`).

3. Expose a delegated scope so that end users can request access tokens for the Spring Boot API.

   This scope ensures that the access token issued to the user includes their identity and application role assignments. Perform the following tasks on the Expose an API page:

   a. Under **Scopes defined by this API**, click **Add a scope**.

   b. In the panel that appears, enter the following information:

      i. **Scope name:** Enter `access_as_user`.

      ii. **Who can consent:** Select **Admins and users**.

      iii. Enter `Access EmployeeRecordsAPI` as the value in all remaining fields.

      Click **Add scope**.

4. Create a client secret.

   A client secret is the application's password. The Spring Boot application uses it to authenticate to Entra ID when requesting database-scoped access tokens.

   a. On the `EmployeeRecordsAPI` application page, under **Manage**, click **Certificates & secrets**.

      The Certificates & secrets page opens.

   b. Click **New client secret** and enter the following details:

      - Enter a description; for example, `employee-records-api-secret`.
      - Set an expiry period according to your organization's policy.

      Click **Add**.

   c. Copy the **Value** column immediately. It is only shown once. Save it as `[CLIENT_SECRET]`.

5. Grant permissions to the Spring Boot application.

   You must explicitly authorize the application (`EmployeeRecordsAPI`) to request tokens scoped to the database resource (`EmployeeRecordsDB`).

   a. On the `EmployeeRecordsAPI` application page, under **Manage**, click **API permissions**, and then click **Add a permission**.

   b. Perform the following tasks on the Request API permissions panel:

      i. Click **APIs my organization uses**, and then click **EmployeeRecordsDB**.

      ii. Select **Delegated permissions** and check the box for `sessions:scope:connect`.

      iii. Click **Add permissions**.

      The permission is successfully added and appears on the API permissions page.

   c. Click **Grant admin consent for Default Directory** (or your specific directory name) to authorize the permissions, and select **Yes** in the confirmation dialog box.

6. Define application roles in Entra ID to manage job functions.

   The application role is how user permissions flow into the JWT token. When a user is assigned the `EMPLOYEE` role, that string appears in their token's `roles` claim. The database reads it and activates the matching data role.

a. On the `EmployeeRecordsAPI` application page, under **Manage**, click **App roles**.

   The App roles page opens.

b. Click **Create app role** and enter the following details:

   - **Display name:** Enter `EMPLOYEE`.
   - **Allowed member types:** Select **Users/Groups**.
   - **Value:** Enter `EMPLOYEE`.
   - **Description:** Enter `Employee role with own-record access`.

   Click **Apply**.

7. Authorize the application as a client of the database application.

   a. At the top of the page, locate the breadcrumb trail, click **App registrations**.

   b. On the App registrations page, click the database registration, **EmployeeRecordsDB**.

   c. Under **Manage**, click **Expose an API**.

   d. In the Authorized client applications section, click **Add a client application**.

   e. In the panel that appears, enter the following details:

      i. Paste the application (client) ID of `EmployeeRecordsAPI` (`[EMP_RECORDS_APP_ID]`).

      ii. Select the database scope.

   Click **Add application**.

5.4 Manage Users and Role Assignments

Create an end user called Emma in Microsoft Entra ID and assign her the `EMPLOYEE` application role.

1. Create the user.

   a. On the Microsoft Entra portal's Home page, click **Users** in the left navigation pane under **Entra ID**.

      The Users page opens.

   b. Click **New user**, and then **Create new user**.

   c. On the Create new user page, enter the following details:

      - **User principal name:** `emma@<your-directory>.onmicrosoft.com`
      - **Display name:** `emma`
      - **Password:** Set a password for the user
      - **Account enabled:** Leave this field checked

      Click **Review + create**, and then click **Create**. From the Users page, copy Emma's full user principal name (UPN).

2. Assign the user to the Spring Boot application in Entra ID to enable sign-in.

   Select the specific application role to define their data access privileges (for example, `EMPLOYEE`).

   a. In the left navigation pane under **Entra ID**, click **Enterprise apps**, and then select **EmployeeRecordsAPI**.

b. Click **Assign users and groups** in the Getting Started section of the application's Overview page.

c. On the Users and groups page, click **Add user/group** to assign a user to the application.

d. On the Add Assignment page, perform the following actions:

i. Select **Emma** under **Users**.

ii. Select **EMPLOYEE** under **Select a role**.

iii. Click **Assign**.

3. Initialize Microsoft Entra ID users.

New users you create in Microsoft Entra ID have temporary passwords. Update these to permanent passwords in a browser before attempting to test this example scenario.

a. Open a private or incognito browser window.

b. Go to `https://myapps.microsoft.com`.

c. Sign in as Emma using the user principal name (`emma@<your-directory>.onmicrosoft.com`) and the temporary password, then change the password.

d. If the sign-in flow requires additional security, follow the prompts to configure the Microsoft Authenticator application as your second verification method.

When the My Apps dashboard appears, close the browser. The user is now active.

5.5 Generate Wallets and Certificates

Secure the application-to-database connection with Transport Layer Security (TLS). Without TLS, the database rejects token-based authentication. Use the orapki (Oracle Public Key Infrastructure) utility to create the server wallet and a self-signed certificate. In this single-host setup, the Spring Boot application also uses the server wallet as its trust store.

> **Note**
>
> - Run all commands as the Oracle software owner user (typically `oracle`).
> - Set Common Name (CN) to the database server's fully qualified domain name.
> - For more details on TLS configuration, see Configuring Transport Layer Security Encryption in *Oracle AI Database Security Guide*.

1. Open a terminal on the Linux host.
2. Create a directory for the server wallet.

```
mkdir -p /u01/app/oracle/wallets/server
```

3. Create the server wallet (identity) and a self-signed certificate.

```
orapki wallet create -wallet /u01/app/oracle/wallets/server -pwd
<wallet_password> -auto_login

orapki wallet add -wallet /u01/app/oracle/wallets/server -dn "CN=<your-vm-
```

```
fqdn>,O=Oracle,C=US" -keysize 2048 -self_signed -validity 3650 -pwd
<wallet_password>
```

> **Note**
>
> This guide uses a self-signed certificate and a single wallet directory for simplicity. For production deployments, use CA-signed certificates, separate client and server trust stores, and a secret manager for wallet passwords. Oracle recommends configuring the `WALLET_ROOT` initialization parameter and storing the TLS wallet under `WALLET_ROOT/<PDB GUID>/tls`. See Configuring Transport Layer Security Encryption in *Oracle AI Database Security Guide*.

5.6 Configure Database Network Settings

Configure the database listener and network layer to accept secure TCPS connections on port 2484 and enable token-based authentication. Both configurations use the server wallet you created in the previous section.

1. Configure the database listener.
   a. Go to the database network configuration directory.

      ```
      cd $ORACLE_HOME/network/admin
      ```

   b. Open `listener.ora` in a text editor (for example, vi or nano).

      ```
      vi listener.ora
      ```

   c. Add or update the wallet location and the TCPS address as shown below and save the file.

      Use the database server's fully qualified domain name for `HOST`.

      ```
      WALLET_LOCATION = (SOURCE = (METHOD = FILE) (METHOD_DATA = (DIRECTORY
      = /u01/app/oracle/wallets/server)))

      TLS_CLIENT_AUTHENTICATION = FALSE

      LISTENER =
       (DESCRIPTION_LIST =
        (DESCRIPTION =
         (ADDRESS = (PROTOCOL = TCP)(HOST = <your-vm-fqdn>)(PORT = 1521))
         (ADDRESS = (PROTOCOL = TCPS)(HOST = <your-vm-fqdn>)(PORT = 2484))
        )
       )
      ```

2. Configure network and token authentication (`sqlnet.ora`).

   This step enables token-based authentication and directs the database network layer to the server wallet for TLS.

   a. In `$ORACLE_HOME/network/admin`, open `sqlnet.ora`.

      ```
      vi sqlnet.ora
      ```

b. Add the following configuration.

```
# --- Authentication ---
SQLNET.AUTHENTICATION_SERVICES = (BEQ, TOKENAUTH)

# --- TLS Configuration ---
TLS_CLIENT_AUTHENTICATION = FALSE
TLS_SERVER_DN_MATCH = TRUE

# Prevents conflicts between Native Encryption and TLS
SQLNET.IGNORE_ANO_ENCRYPTION_FOR_TCPS = TRUE

WALLET_LOCATION = (SOURCE = (METHOD = FILE) (METHOD_DATA = (DIRECTORY
= /u01/app/oracle/wallets/server)))
```

c. Restart the listener.

Configure any required proxy environment variables in the same terminal session used to restart the listener. The listener inherits its environment from that session.

```
export TNS_ADMIN=$ORACLE_HOME/network/admin
lsnrctl stop
lsnrctl start
```

> **Note**
>
> In Oracle AI Database 26ai, the `WALLET_LOCATION` parameter in `sqlnet.ora` is deprecated for the database server. Use the `WALLET_ROOT` initialization parameter instead, as described in Generate Wallets and Certificates. However, `WALLET_LOCATION` remains valid for the listener and for client-side configurations.

5.7 Configure Data Access Control

In this section, you configure data access control using Oracle Deep Data Security (Deep Sec) capabilities.

You'll perform the following tasks:

- Configure the database to accept external identities from Microsoft Entra ID.
- Create sample employee records data.
- Set up the connection pool user account and application identity for the Spring Boot application.
- Create data roles, which map to Entra ID application roles.
- Define data grants that limit Emma to her own employee record, and enable privilege elevation for the salary-summary endpoint.

1. Connect to the database as a named user with the DBA role.

2. Switch to the target pluggable database (for example, `ORCLPDB`). Run the following command.

```
ALTER SESSION SET CONTAINER = <your-target-PDB>;
```

3. Enable Microsoft Entra ID as the identity provider.

   Configure the database to validate tokens from your Entra ID tenant. Use the values from Create Application Registrations in Microsoft Entra ID to replace the placeholders.

```
ALTER SYSTEM SET IDENTITY_PROVIDER_TYPE = AZURE_AD SCOPE = BOTH;

ALTER SYSTEM SET IDENTITY_PROVIDER_CONFIG = '{
  "application_id_uri": "[DB_APP_ID_URI]",
  "tenant_id": "[TENANT_ID]",
  "app_id": "[DB_APP_ID]"
}' SCOPE = BOTH;
```

4. Create sample data.

   Create a sample HR schema with an employees table. Replace `<your-entraID-domain>` in the SQL statements with your actual Microsoft Entra ID domain (for example, `supremo.onmicrosoft.com`).

> **Note**
>
> If you completed the previous quick start (Configure Oracle Deep Data Security for Direct Logon with End Users in IAM) and the `hr.employees` table already exists, you can reuse it.

   a. Create the `hr` user and the employees table.

> **Note**
>
> The `CREATE USER` statement below uses the `USERS` tablespace. If this tablespace does not exist in your database, create it.
>
> ```
> CREATE TABLESPACE users
>   DATAFILE '<data_file_path>' SIZE 100M
>   AUTOEXTEND ON;
> ```
>
> Replace `<data_file_path>` with the full path for the data file.

```
-- Create the hr user
CREATE USER hr NO AUTHENTICATION
  DEFAULT TABLESPACE users
  QUOTA UNLIMITED ON users;

-- Create the employees table
CREATE TABLE hr.employees (
    employee_id   NUMBER PRIMARY KEY,
    first_name    VARCHAR2(50),
```

```
    last_name     VARCHAR2(50),
    email         VARCHAR2(128),
    manager       VARCHAR2(128),
    ssn           VARCHAR2(20),
    salary        NUMBER(10,2),
    phone         VARCHAR2(20)
);
```

b. Populate the `employees` table with the following sample data.

```
INSERT INTO hr.employees VALUES (100, 'Victoria', 'Williams',
  'victoria@<your-entraID-domain>', NULL,
  '219-09-9999', 13000, '555-0100');

INSERT INTO hr.employees VALUES (200, 'Marvin', 'Anderson',
  'marvin@<your-entraID-domain>', 'victoria@<your-entraID-domain>',
  '457-55-5462', 12030, '555-0200');

INSERT INTO hr.employees VALUES (300, 'Chris', 'Evans',
  'chris@<your-entraID-domain>', 'victoria@<your-entraID-domain>',
  '321-12-4567', 6900, '555-0300');

INSERT INTO hr.employees VALUES (400, 'Emma', 'Baker',
  'emma@<your-entraID-domain>', 'marvin@<your-entraID-domain>',
  '733-02-9821', 8200, '555-0400');

INSERT INTO hr.employees VALUES (500, 'Taylor', 'Mills',
  'taylor@<your-entraID-domain>', 'marvin@<your-entraID-domain>',
  '558-76-1243', 9000, '555-0500');

COMMIT;
```

5. Create the connection pool user account and assign privileges.

   Create a database user account for the Spring Boot application's connection pool. The application uses this account to maintain its connection pool. You must grant this user account the privileges to establish a database session and transmit the end-user security context payload. This user authenticates to the database using the database-access token.

```
CREATE USER hr_app_user IDENTIFIED GLOBALLY AS
    'AZURE_CLIENT_ID=[EMP_RECORDS_APP_ID]';

GRANT CREATE SESSION TO hr_app_user;
GRANT CREATE END USER SECURITY CONTEXT TO hr_app_user;
```

6. Create an application identity.

   Create an identity for the Spring Boot application in the database that maps to its client ID in Entra ID. This identity is used to authorize the application for privilege elevation.

```
CREATE OR REPLACE APPLICATION IDENTITY hr_app
  MAPPED TO 'AZURE_CLIENT_ID=[EMP_RECORDS_APP_ID]';
```

7. Configure data roles.

a. Define a data role that maps to Emma's application role defined in Microsoft Entra ID.

```
CREATE OR REPLACE DATA ROLE employee_role MAPPED TO
'AZURE_ROLE=EMPLOYEE';
```

b. Next, create a data role as disabled for privilege elevation. This data role is never active by default; it is only enabled temporarily by the application at runtime. Grant it to the application identity so that only the Spring Boot application can activate it.

```
CREATE OR REPLACE DATA ROLE compensation_analyst DISABLED;
GRANT DATA ROLE compensation_analyst TO hr_app;
```

8. Create data grants.

   Define two data grants: one that limits employees to their own record, and one that enables the salary-summary endpoint through privilege elevation.

   a. Create the `EMPLOYEES_OWN_RECORD` data grant.

      This data grant ensures that users assigned the `EMPLOYEE_ROLE` data role can access only those rows where the value in the `email` column matches their login user name (UPN).

```
CREATE OR REPLACE DATA GRANT hr.employees_own_record AS
  SELECT
  ON hr.employees
  WHERE email = ORA_END_USER_CONTEXT.username
  TO employee_role;
```

   b. Create the `EMPLOYEES_SALARY_SUMMARY` data grant.

      This data grant gives the `COMPENSATION_ANALYST` data role read access to the `salary` column across all employees. Because the data role is disabled by default and granted exclusively to the application identity, it can only be activated when the application explicitly elevates privileges at runtime.

```
CREATE OR REPLACE DATA GRANT hr.employees_salary_summary AS
  SELECT (salary)
  ON hr.employees
  WHERE 1 = 1
  TO compensation_analyst;
```

5.8 Set Up the Spring Boot Application

In this section, you clone the sample application, review its dependencies, and configure the application properties and environment variables.

Topics:

5.8.1 Get the Application Source Code

The sample application is part of the Oracle JDBC Extensions project. Clone the repository and navigate to the sample application directory.

```
cd $HOME
git clone https://github.com/oracle/ojdbc-extensions.git
cd ojdbc-extensions/ojdbc-provider-spring/samples/employee-records-api
```

This directory is referred to as `APP_DIR` throughout the rest of this chapter. Here are the key files within it:

Path	Description
`APP_DIR/pom.xml`	Maven build file
`APP_DIR/.env`	Environment variables (copy from `.env.example`)
`APP_DIR/src/main/resources/application.properties`	Spring Boot configuration

5.8.2 Install the JDBC Driver

The core Oracle JDBC driver is required for database connectivity. For this sample application, the driver dependency is already included in `pom.xml`.

If you are using a different application, add the following dependency to your `pom.xml` file:

```
<!-- Oracle JDBC driver -->
<dependency>
    <groupId>com.oracle.database.jdbc</groupId>
    <artifactId>ojdbc11</artifactId>
    <version>23.26.2.0.0</version>
</dependency>
```

5.8.3 Install the Provider

The Oracle JDBC Spring provider automates end-user security context payload propagation through the JDBC Service Provider Interface (SPI). For this sample application, the provider dependencies are already included in `pom.xml`.

If you are using a different application, add the following dependencies. These artifacts are part of the Oracle JDBC Extensions project:

```
<!-- Propagate end-user security context to Oracle Database -->
<dependency>
    <groupId>com.oracle.database.jdbc</groupId>
    <artifactId>ojdbc-provider-spring</artifactId>
    <version>${ojdbc.provider.version}</version>
</dependency>
```

The `ojdbc-provider-spring` library handles the end-user security context payload propagation automatically. Oracle JDBC discovers the library on the class path and invokes it on every connection; no additional code is required in your application.

5.8.4 Review Additional Dependencies

The following additional dependencies are also required and are already included in the sample application's `pom.xml`.

If you are using a different application, add each of these manually:

```
<!-- Azure token provider for JDBC (authenticates the connection pool to
Oracle via Entra ID) -->
<dependency>
    <groupId>com.oracle.database.jdbc</groupId>
    <artifactId>ojdbc-provider-azure</artifactId>
    <version>1.0.6</version>
</dependency>

<!-- Oracle Wallet/SSL support (reads cwallet.sso for TCPS certificate
validation) -->
<dependency>
    <groupId>com.oracle.database.security</groupId>
    <artifactId>oraclepki</artifactId>
    <version>${oracle.security.version}</version>
</dependency>

<!-- OAuth 2.0 Resource Server (validates incoming user JWTs in the
Authorization header) -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
</dependency>

<!-- JPA (Hibernate) with HikariCP -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<!-- AOP (for @RunWithDataRoles privilege elevation) -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
</dependency>

<!-- Actuator for health/info endpoints -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

5.8.5 Configure the Application

Now that the dependencies are in place, configure the application to connect to your Oracle AI Database and Entra ID tenant.

The application uses two configuration files that work together:

- **application.properties:** The primary Spring Boot configuration file. It defines application-level settings (such as connection pools, security providers, and OAuth 2.0 settings) using `${VARIABLE}` placeholders to keep sensitive and environment-specific data out of the source code.
- **.env:** A key-value file that stores the actual environment variables (such as client credentials and database URLs). When sourced at runtime, Spring Boot uses these values to dynamically resolve the placeholders in `application.properties`.

> **Note**
>
> Spring Boot does not read `.env` files automatically. Use the `set -a && source .env && set +a` command to export the variables into the current shell session.

Review `application.properties`

The sample application includes a pre-configured `application.properties` file with all required settings. If you are configuring a different application or if the file is missing after cloning, create it using the following commands:

```
mkdir -p APP_DIR/src/main/resources
vi APP_DIR/src/main/resources/application.properties
```

Verify that the file contains the following configuration sections, and add any that are missing.

JWT validation

Configures the application to validate incoming end-user tokens.

```
spring.security.oauth2.resourceserver.jwt.issuer-uri=${JWT_ISSUER_URI}
```

Database connection

Configures the connection pool for TCPS with token authentication using your environment variables.

```
spring.datasource.url=${ORACLE_JDBC_URL}
spring.datasource.driver-class-name=oracle.jdbc.OracleDriver
spring.datasource.hikari.data-source-properties.oracle.jdbc.clientId=$
{HRAPP_CLIENT_ID}
spring.datasource.hikari.data-source-properties.oracle.jdbc.clientSecret=$
{HRAPP_CLIENT_SECRET}
spring.datasource.hikari.data-source-properties.javax.net.ssl.trustStore=$
{ORACLE_WALLET_TRUSTSTORE}
spring.datasource.hikari.data-source-
properties.javax.net.ssl.trustStoreType=SSO
```

OAuth 2.0 client registration

Defines the OAuth 2.0 client used to acquire tokens for the database connection.

```
spring.security.oauth2.client.registration.hrapp.client-name=hrapp
spring.security.oauth2.client.registration.hrapp.client-id=${HRAPP_CLIENT_ID}
spring.security.oauth2.client.registration.hrapp.client-secret=$
{HRAPP_CLIENT_SECRET}
spring.security.oauth2.client.registration.hrapp.scope=${HRAPP_SCOPE}
spring.security.oauth2.client.registration.hrapp.authorization-grant-
type=client_credentials
spring.security.oauth2.client.provider.hrapp.token-uri=${HRAPP_TOKEN_URI}
spring.security.oauth2.client.provider.hrapp.authorization-uri=$
{HRAPP_AUTH_URI}
```

End-user security context propagation

Enables the Oracle JDBC Spring Boot Provider to propagate the user's JWT to the database. These properties are mandatory for Oracle Deep Data Security (Deep Sec) to function.

```
# End-user security context provider — propagates the user's JWT to Oracle
spring.datasource.hikari.data-source-
properties.oracle.jdbc.provider.endUserSecurityContext=ojdbc-provider-spring-
end-user-security-context
spring.datasource.hikari.data-source-
properties.oracle.jdbc.provider.endUserSecurityContext.registrationId=hrapp
```

> **Note**
>
> The `registrationId` value (`hrapp`) must exactly match the client name used in the OAuth 2.0 client registration section above.

Configure environment variables

You manage the application's environment variables using a `.env` file located in the application directory.

Navigate to the directory, copy the example template provided with the sample application, and open the new `.env` file for editing.

```
cd APP_DIR
cp .env.example .env
vi .env
```

Update the variables in the `.env` file using the values from your Entra ID application registrations. See Create Application Registrations in Microsoft Entra ID. Use the following table as a reference.

Variable	Description	Example / Format
JWT_ISSUER_URI	Entra ID token issuer URI for JWT validation. The value must be exactly the same as the iss claim from the Microsoft Entra ID access token.	https://sts.windows.net/[TENANT_ID]/
ORACLE_JDBC_URL	Oracle TCPS connection string.	See format after the table.
ORACLE_WALLET_TRUSTSTORE	Path to the server wallet for TLS.	/u01/app/oracle/wallets/server/cwallet.sso
HRAPP_CLIENT_ID	Application (Client) ID of EmployeeRecordsAPI. [EMP_RECORDS_APP_ID]	b821ac28-11aa-2222-c3c3-5b3e31c8cb8e
HRAPP_CLIENT_SECRET	Client secret generated for EmployeeRecordsAPI. [CLIENT_SECRET]	Jkk8Q~F4t0Hof6~2SecRETe14SHRYtQ.i7wwwcC0
HRAPP_SCOPE	Scope that the application requests through the client credentials flow.	[DB_APP_ID_URI]/.default https://supremo.onmicrosoft.com/fe58febf-0724-4d8f-9b14-598a0d2f4662/.default
HRAPP_TOKEN_URI	Entra ID token endpoint.	https://login.microsoftonline.com/[TENANT_ID]/oauth2/v2.0/token
HRAPP_AUTH_URI	Entra ID authorization endpoint.	https://login.microsoftonline.com/[TENANT_ID]/oauth2/v2.0/authorize
HRAPP_DELEGATED_SCOPE	Delegated scope for Auth Code + PKCE (user login).	api://[EMP_RECORDS_APP_ID]/access_as_user

ORACLE_JDBC_URL Format

Construct your JDBC URL using the following format. Ensure that you replace the placeholder values ([your-vm-fqdn], [DB_APP_ID_URI], [TENANT_ID], and [your-pdb-service]) with your specific environment details.

```
jdbc:oracle:thin:@(DESCRIPTION=
    (ADDRESS=(PROTOCOL=tcps)(HOST=[your-vm-fqdn])(PORT=2484))
    (SECURITY=
        (TLS_SERVER_CERT_DN="CN=[your-vm-fqdn],O=Oracle,C=US")
        (TOKEN_AUTH=AZURE_SERVICE_PRINCIPAL)
        (AZURE_DB_APP_ID_URI=[DB_APP_ID_URI])
        (TENANT_ID=[TENANT_ID]))
    (CONNECT_DATA=(SERVICE_NAME=[your-pdb-service])))
```

5.9 Build, Run, and Verify

Compile the application, start the Spring Boot server, authenticate an end user through Microsoft Entra ID, and verify that the data grants correctly enforce fine-grained data access control and privilege elevation.

For this section, you'll need two separate terminal sessions:

- **Terminal 1 (build and run):** Use to compile the project, load environment variables, and start the Spring Boot application. This terminal remains occupied while the application runs.
- **Terminal 2 (test):** Use to generate Proof Key for Code Exchange (PKCE) values, obtain Entra ID access tokens, and execute `curl` requests against the running API.

Topics:

5.9.1 Build the Application

Configure your Java and Maven environments, then compile the parent project, the Spring provider module, and the sample API into an executable JAR. Perform these tasks in Terminal 1.

1. Set the active JDK.

   Ensure Java 17 is your active JDK (if it isn't already your system default).

   ```
   export JAVA_HOME=/usr/lib/jvm/java-17-openjdk
   export PATH=$JAVA_HOME/bin:$PATH
   ```

2. Set up Maven.

   If Maven is not on your system path, point `MAVEN_HOME` to your installation directory. (If `mvn -version` already works, you can skip this step.)

   ```
   export MAVEN_HOME=/opt/maven/apache-maven-3.9.x
   export PATH=$MAVEN_HOME/bin:$PATH

   # Verify installation (requires Apache Maven 3.9.x or later)
   mvn -version
   ```

3. Build the project.

   Build the project components in the following order. Starting from the repository root, compile the parent project, the Spring provider module, and finally the sample application.

   ```
   # a. Build the parent project
   cd $HOME/ojdbc-extensions
   mvn -DskipTests clean install
   ```

```
# b. Build the Spring provider module
cd $HOME/ojdbc-extensions/ojdbc-provider-spring
mvn -DskipTests clean install

# c. Build the sample application
cd APP_DIR
mvn -DskipTests clean package
```

> **Note**
>
> A successful build ends with `BUILD SUCCESS`. The executable JAR is located at `target/employee-records-api-0.0.1-SNAPSHOT.jar`.

5.9.2 Run the Application

Continuing in Terminal 1, source your environment variables and launch the Spring Boot application.

1. Load environment variables.

   Ensure that you are in the application directory, then source your `.env` file.

   ```
   cd APP_DIR
   set -a && source .env && set +a
   ```

2. Start the Spring Boot application.

   Run the application using the Maven Spring Boot plug-in.

   ```
   mvn -DskipTests spring-boot:run
   ```

   Leave this terminal running. Wait for the console to display `Started EmployeeRecordsApiApplication` before continuing with the next steps.

> **Note**
>
> If your host requires an HTTP proxy to reach Entra ID, append the following Java Virtual Machine (JVM) arguments to the `run` command:
>
> ```
> mvn -DskipTests spring-boot:run \
>     -Dspring-boot.run.jvmArguments="\
>     -Dhttps.proxyHost=<proxy-host> \
>     -Dhttps.proxyPort=<port> \
>     -Dhttp.proxyHost=<proxy-host> \
>     -Dhttp.proxyPort=<port> \
>     -Dhttp.nonProxyHosts=localhost"
> ```

5.9.3 Get an Access Token

Switching to Terminal 2, execute the Authorization Code with PKCE flow to simulate a user login, ultimately exchanging an Entra ID authorization code for a JWT access token.

1. Create the PKCE helper script.

   Navigate to the application directory and create a utility script. You can use this script to easily generate fresh PKCE verifier and challenge codes each time you need to fetch an access token.

   ```
   cd APP_DIR

   cat > generate_pkce.sh << 'EOF'
   #!/bin/bash
   CODE_VERIFIER=$(openssl rand -base64 64 | tr -d '=+/' | tr -d '\n' | cut -
   c1-64)
   CODE_CHALLENGE=$(echo -n "$CODE_VERIFIER" | openssl dgst -sha256 -binary |
   base64 | tr '+/' '-_' | tr -d '=')

   # Update .env file in place
   sed -i "s|CODE_VERIFIER=.*|CODE_VERIFIER='$CODE_VERIFIER'|" .env
   sed -i "s|CODE_CHALLENGE=.*|CODE_CHALLENGE='$CODE_CHALLENGE'|" .env

   echo "Generated and updated .env:"
   echo "CODE_VERIFIER=$CODE_VERIFIER"
   echo "CODE_CHALLENGE=$CODE_CHALLENGE"
   EOF

   chmod +x generate_pkce.sh
   ```

2. Run the script and load variables.

   Execute the script to update your `.env` file with verifier and challenge codes, then source the file to load the new variables into your terminal session.

   ```
   ./generate_pkce.sh
   set -a && source .env && set +a
   ```

3. Generate the authorization URL.

   Construct the Entra ID login URL and print it to the console.

   ```
   AUTH_URL="${HRAPP_AUTH_URI}?client_id=${HRAPP_CLIENT_ID}"
   AUTH_URL+="&response_type=code&redirect_uri=${REDIRECT_URI}"
   AUTH_URL+="&scope=${HRAPP_DELEGATED_SCOPE}"
   AUTH_URL+="&response_mode=query&prompt=consent"
   AUTH_URL+="&code_challenge=${CODE_CHALLENGE}&code_challenge_method=S256"

   echo -e "\nOpen this URL in your browser:\n$AUTH_URL\n"
   ```

4. Authenticate in the browser.

   a. Copy the generated URL and open it in a web browser.

   b. Sign in as your test user (Emma).

c. After authenticating, your browser redirects to `http://localhost:3000?code=<AUTH_CODE>`. It is normal for this page to fail to load.

d. From the URL bar, copy the authorization code (everything after `code=` and before the next `&`).

5. Exchange the code for an access token.

   Use `curl` to swap the authorization code for a JWT access token. In the command below, replace `<AUTH_CODE_FROM_REDIRECT>` with the code you copied.

```
curl -s -X POST "$HRAPP_TOKEN_URI" \
    -H "Content-Type: application/x-www-form-urlencoded" \
    --data-urlencode "grant_type=authorization_code" \
    --data-urlencode "client_id=$HRAPP_CLIENT_ID" \
    --data-urlencode "client_secret=$HRAPP_CLIENT_SECRET" \
    --data-urlencode "code=<AUTH_CODE_FROM_REDIRECT>" \
    --data-urlencode "redirect_uri=$REDIRECT_URI" \
    --data-urlencode "code_verifier=$CODE_VERIFIER" \
    --data-urlencode "scope=$HRAPP_DELEGATED_SCOPE"
```

   In the JSON response, find the `access_token` value and save it.

5.9.4 Verify Employee Access

In Terminal 2, call the `/api/employees` endpoint using Emma's access token you obtained previously.

```
curl -s --http1.1 -X GET "http://localhost:8080/api/employees" \
    -H "Authorization: Bearer $ACCESS_TOKEN" \
    -H "Accept: application/json" | python3 -m json.tool
```

Emma's access token carries `EMPLOYEE` in its `roles` claim, which activates the `EMPLOYEE_ROLE` data role in the database. The `EMPLOYEES_OWN_RECORD` data grant on this data role ensures only Emma's record is returned.

```
[
    {
        "id": 400,
        "name": "Emma Baker",
        "salary": 8200,
        "phone": "555-0400"
    }
]
```

5.9.5 Verify Privilege Elevation

Now, use the same access token and call the `/api/employees/salary-summary` endpoint. This demonstrates how the application temporarily elevates Emma's database privileges, allowing her to view aggregate salary statistics without exposing individual employee salaries.

```
curl -s --http1.1 -X GET "http://localhost:8080/api/employees/salary-summary"
\
```

```
    -H "Authorization: Bearer $ACCESS_TOKEN" \
    -H "Accept: application/json" | python3 -m json.tool
```

Emma's token still carries only the `EMPLOYEE` role claim, but for this endpoint the application temporarily activates the `COMPENSATION_ANALYST` data role. The corresponding data grant (`EMPLOYEES_SALARY_SUMMARY`) allows access to aggregate salary data across all employees, without exposing individual records:

```
{
    "minSalary": 6900,
    "maxSalary": 13000,
    "averageSalary": 9826.00,
    "employeeCount": 5
}
```

What happens behind the scenes

1. The application invokes the `getSalarySummary()` method, which is tagged with the `@RunWithDataRoles(dataRoles = {"COMPENSATION_ANALYST"})` annotation.
2. A Spring AOP interceptor (`RunWithDataRolesAopConfig.java`) catches this request and temporarily adds the `compensation_analyst` data role to Emma’s end-user security context.
3. The database activates the `hr.employees_salary_summary` data grant, permitting the aggregate salary query.
4. Emma receives the minimum salary, maximum salary, average salary, and employee count of her team; however, individual salaries remain hidden.
5. After the method returns, the privilege elevation is immediately deactivated. Emma's security context reverts to her standard `EMPLOYEE_ROLE` data role.

This temporary privilege elevation is scoped strictly to the execution of that specific method, controlled by the application, and securely enforced at the database level.

Part III

IAM, Database, and Application Configuration

Learn how to configure your external identity and access management (IAM) system, the Oracle AI Database, and your application layer to support authentication and authorization with Oracle Deep Data Security (Deep Sec).

Begin by identifying the deployment scenario that matches your environment. Then complete the applicable configuration steps.

- **IAM system setup:** Register applications, users, and roles in your IAM system.
- **Database configuration:** Configure the database to establish end-user security contexts for both application-mediated connections and direct connections, using either IAM OAuth 2.0 tokens or local end-user authentication.
- **Application setup:** Configure the application to build and propagate the end-user security context payload.
- **SQL client configuration:** For direct logon scenarios with token-based authentication, set up a SQL client.

To understand the trust chain (end user, application, and database) and how scopes, roles, and user identities flow through the architecture, see Application Registrations, Users, and Roles.

Topics:

6

Identify Your Deployment Scenario

Before configuring Oracle Deep Data Security (Deep Sec), identify your deployment scenario based on how your end users are managed (locally or in IAM) and how they connect to the database (directly or through an application).

See the following topics for the configuration paths for different scenarios.

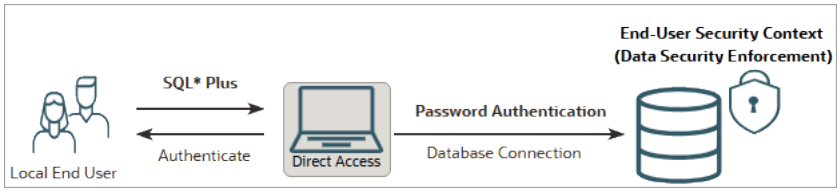
Topics:

6.1 Local End Users Connecting Directly

In this scenario, end users are created and managed in the database, and connect using a SQL client (for example, SQL*Plus) and password-based authentication. No application or IAM system is involved.

Use case

Because it is the simplest scenario to configure, this setup is ideal for development and testing environments, product demonstrations, or applications with a small, manageable number of users who connect directly to the database.



Database authentication flow

Prerequisite setup: To enable Deep Sec for this scenario, you must create an end-user account identified by password in the database and assign it data roles that are managed locally. In addition, you must grant the end user the system privilege to create a database session.

The runtime flow is as follows:

1. **User login:** The end user connects directly to the database using their user name and password.
2. **Database authentication:** The database authenticates the user against their local end-user account.
3. **Security context establishment:** Upon successful connection, the database establishes an end-user security context. The end user's granted data roles are activated, and the corresponding data grants are enforced for all subsequent SQL operations of the end user.

Configuration path

Configure the database for local end-user authentication. See Set Up Local Authentication for Direct Logon.

Example configuration

For a step-by-step example of this configuration, see Configure Oracle Deep Data Security for Direct Logon with Local End Users.

6.2 IAM-Managed Users Connecting Directly

In this scenario, end users are managed in an IAM system (Microsoft Entra ID or OCI IAM), and connect to the database directly using a SQL client (for example, SQL*Plus) and their own IAM credentials. No application is involved. This scenario is typical for data analysts, developers, and DBAs.

For an overview of the use case and authentication flow, see Connect Directly to the Database.

Configuration path

1. In your IAM system, register applications to represent the database and the SQL client. Subsequently, create end users and assign roles. Depending on your IAM provider, see one of the following topics:
   - For Microsoft Entra ID, see Create Application Registrations in Microsoft Entra ID.
   - For OCI IAM, see Configure OCI IAM for Application-Mediated Access. Substitute the SQL client for the application.
2. Configure the database. See Set Up IAM Integration for Direct Logon.
3. Set up your SQL client for direct logon. See Configure a SQL Client for Interactive Logon (Direct Logon).

Example configuration

For a step-by-step example of this configuration using Microsoft Entra ID, see Configure Oracle Deep Data Security for Direct Logon with End Users in IAM.

6.3 IAM-Managed Users Connecting Through an Application

In this scenario, end users are managed in an IAM system (Microsoft Entra ID or OCI IAM) and connect to the database through an application, such as a REST API, web application, MCP server, or analytics tool.

There are two variations of this scenario:

- Applications or AI agents accessing the database on behalf of end users.
- End users operating in the context of applications or AI agents.

For a complete overview of the use cases and authentication flows, see *Connect Through an Application* in Data Access Patterns.

Configuration path

1. Register applications, create end users, and assign roles in your IAM system. Depending on your IAM provider, see one of the following topics:

- Configure Microsoft Entra ID for Application-Mediated Access.
- Configure OCI IAM for Application-Mediated Access.

2. Configure the database. See Set Up IAM Integration for Application-Mediated Connections.
3. Configure the application. See Update Application Configuration with IAM Details.

Example configuration

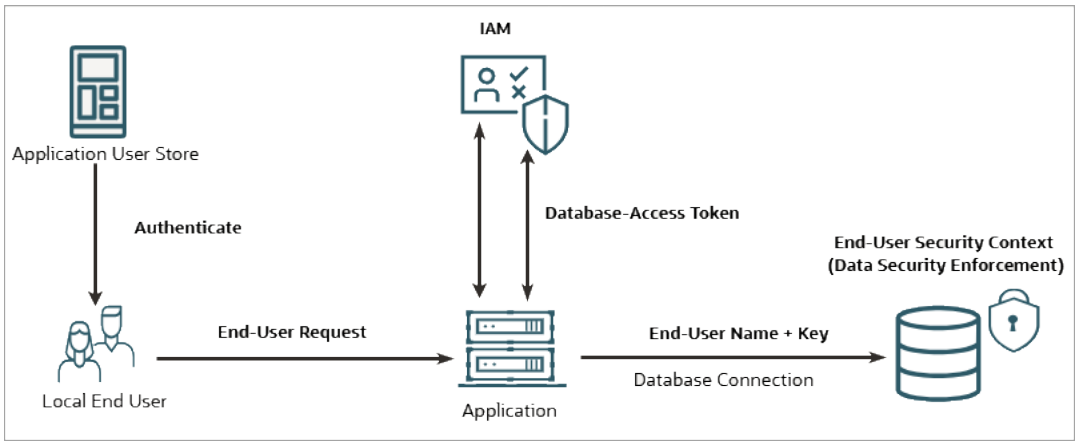
For a step-by-step example of this configuration using Microsoft Entra ID and a Spring Boot application, see Configure Oracle Deep Data Security for a Sample Application.

6.4 Local Application Users Connecting Through the Application

In this scenario, end users are managed in an application's own user store, such as a separate database, a Lightweight Directory Access Protocol (LDAP) directory, or an application-managed identity system. These users then connect to the database through the application.

Use case

This scenario is ideal when your application maintains its own user directory, and you want to enforce fine-grained data authorization at the database layer without requiring your end users to have external IAM accounts.



Database authentication flow

Prerequisite setup:

To enable Deep Sec for this pattern, you must complete the following configuration:

- **IAM setup:** Register the database as an OAuth resource. Next, register the application with a client secret, and grant the application access to the database resource.
- **Database setup:** Configure IAM as an external identity provider, and create local end-user accounts with names that match the users in the application's user store. Create these accounts without passwords, because they rely on the application's trusted connection instead of direct database authentication. Finally, grant data roles that are managed locally to these accounts.

The runtime flow is as follows:

1. **End-user authentication:** The application authenticates the end user against its own local user store.

2. **Application authentication:** The application obtains its own database-access token, with the application as the subject and the database as the audience.
3. **Security context payload propagation:** The application uses its database-access token to authorize the connection. As the end-user security context payload, the application forwards the end user's name and a security context lookup key.
4. **Security context establishment:** The database validates the token, matches the supplied name to the local end-user account, establishes the end-user security context, and activates the data roles granted to the user.

Configuration path

1. Register the database and the application in your IAM system. Depending on your IAM provider, see one of the following topics:
   - Configure Microsoft Entra ID for Application-Mediated Access.
   - Configure OCI IAM for Application-Mediated Access.
2. Configure the database. See Set Up Local Authentication for Application-Mediated Connections.
3. Configure the application. See Update Application Configuration with IAM Details.

7

Configure Microsoft Entra ID for Application-Mediated Access

Register applications, create users, and assign roles in Microsoft Entra ID to support authentication and authorization for application-mediated access to the database.

The registration steps in this chapter apply regardless of which OAuth 2.0 flow your application uses (on-behalf-of or client credentials). You select the flow when you configure the application. See Update Application Configuration with IAM Details.

> **Note**
>
> - For all available Oracle AI Database integration options with Entra ID, see Authenticating and Authorizing Microsoft Azure Users for Oracle AI Databases in *Oracle AI Database Security Guide*.
> - For the full range of Entra ID capabilities, see the Microsoft identity platform documentation.
> - If your application users connect directly to Oracle AI Database using Entra ID tokens with tools such as SQL*Plus (without an application), see Configure Oracle Deep Data Security for Direct Logon with End Users in IAM.
> - The Microsoft Entra portal interface may be updated over time. If a specific label or navigation path differs from the instructions provided here, look for the closest matching option.

Topics:

7.1 Register the Database in Microsoft Entra ID

Register your Oracle AI Database as an application in Microsoft Entra ID so that your application can request access tokens scoped to the database.

This registration represents the database as a resource server. You will also expose the database as a web API and define a scope that controls which client applications can request database-access tokens.

1. Create an application registration for the database.
   a. Log in to the Microsoft Entra Portal.
   b. In the left navigation pane, expand **Entra ID**, click **App registrations**, and then click **New registration**.

c. Perform the following tasks on the Register an application page:

   i. Enter `OracleDB_Resource` in the **Name** field.

   ii. For **Supported account types**, select **Single tenant only - Default Directory**.

   iii. Click **Register**.

   The application is successfully created for your database.

d. From the application's Overview page, copy and save the following values for later use:

   - Application (client) ID.
   - Directory (tenant) ID.

   You'll use these values to configure the identity provider configuration in the database.

2. Expose the database's application registration as a web API, and define a scope to control access for client applications.

   a. On the application's Overview page, click **Add an Application ID URI**.

   The Expose an API page opens.

   b. Click **Add** next to **Application ID URI**.

   c. Update the default URI by replacing `api://` with `https://<your-entraID-domain>/`, and click **Save**.

   The resulting URI should resemble: `https://supremo.onmicrosoft.com/fe58fefb-0925-4c8f-9b14-598a0d2f4552`. Copy this URI for later use.

   > **Note**
   >
   > These instructions are for v1 access tokens, which use a `https://`-prefixed application ID URI. Oracle AI Database also supports v2 access tokens. To use v2 tokens, keep the default `api://` URI, add `upn` as an optional claim under **Token configuration**, and set `accessTokenAcceptedVersion` to `2` in the application manifest. For the detailed procedure, see Enabling Microsoft Entra ID v2 Access Tokens in *Oracle AI Database Security Guide*.

   d. Add a scope.

   i. Under **Scopes defined by this API**, click **Add a scope**.

   ii. In the panel that appears, enter the following information:

   - **Scope name:** `sessions:scope:connect`.
   - **Who can consent:** Select **Admins and users**.
   - Enter `Access Oracle Database` as the value in all remaining fields.
   - Click **Add scope**.

You have now created the database application registration in Entra ID with an application ID URI and a delegated scope. Client applications can reference this scope when requesting database-access tokens.

> **See also**
>
> Registering the Oracle AI Database Instance with a Microsoft Entra ID Tenancy in *Oracle AI Database Security Guide*.

7.2 Register the Application in Microsoft Entra ID

The application serves as both a confidential client (authenticating to Entra ID with a client secret) and a resource server that end users authenticate against.

Application roles defined on this registration flow into the user's JWT `roles` claim, which the Oracle AI Database reads to activate the corresponding data roles.

1. Create a registration for your application.
   a. On the Microsoft Entra portal's Home page, click **App registrations** in the left navigation pane under **Entra ID**, and then click **New registration**.
   b. Perform the following tasks on the Register an application page:
      i. Enter a name for the registration in the **Name** field; for example, `HCM APP`.
      ii. For **Supported account types**, select **Single tenant only - Default Directory**.
      iii. Under **Redirect URI**, select **Web** from the drop-down field, and enter `http://localhost:3000`.
      iv. Click **Register**.

      The application is successfully created.
   c. From the application's Overview page, copy and save the application (client) ID.

      You'll use this value to configure your application properties.
2. Set the application ID URI.

   This step makes the application a resource, allowing users and other applications to request tokens scoped to it.
   a. On the application's Overview page, click **Add an Application ID URI**.
   b. Perform the following tasks on the Expose an API page:
      i. Click **Add** next to **Application ID URI**.
      ii. In the panel that appears, accept the default value `api://<your-app-id>`, then click **Save**.
      iii. Copy this URI for later use.
3. Expose a delegated scope so that end users can request access tokens for the application.

   This scope ensures that the access token issued to the user includes their identity and application role assignments. Perform the following tasks on the Expose an API page:
   a. Under **Scopes defined by this API**, click **Add a scope**.
   b. In the panel that appears, enter the following information:
      - **Scope name**: Enter `user_access`.
      - **Who can consent**: Select **Admins and users**.
      - Enter `Access HCM APP` as the value in all remaining fields.

- **State**: Leave **Enabled** selected.

Click **Add scope**.

4. Create a client secret.

   The application uses a client secret to authenticate to Entra ID when requesting database-scoped access tokens.

   a. On the `HCM APP` application page, under **Manage**, click **Certificates & secrets**.

      The Certificates & secrets page opens.

   b. Click **New client secret** and enter the following details:

      - Enter a description; for example, `hcm-app-secret`.
      - Set an expiry period according to your organization's policy.

      Click **Add**.

   c. Copy and save the **Value** column immediately; it is displayed only once.

5. Grant permissions to the application.

   The application needs permission to request tokens scoped to the database resource (`OracleDB_Resource`).

   a. On the `HCM APP` application page, under **Manage**, click **API permissions**, and then click **Add a permission**.

   b. Perform the following tasks on the Request API permissions panel:

      i. Click **APIs my organization uses**, and then click **OracleDB_Resource**.

      ii. Select **Delegated permissions** and check the box for `sessions:scope:connect`.

      iii. Click **Add permissions**.

      The permission is successfully added and appears on the API permissions page.

   c. Click **Grant admin consent for Default Directory** (or your specific directory name) to authorize the permission, and select **Yes** in the confirmation dialog box.

6. Define application roles.

   Application roles control how user permissions flow into the JWT token. When a user is assigned an application role (for example, `MANAGER`), that value appears in the token's `roles` claim. Oracle AI Database reads this claim and activates the matching data role.

   a. On the `HCM APP` application page, click **App roles** under **Manage**.

   b. On the App roles page, click **Create app role**, and enter the following details:

      - **Display name:** Enter a name for the role; for example, `MANAGER`.
      - **Allowed member types:** Select **Users/Groups**.
      - **Value:** Enter a value; for example, `MANAGER`.
      - **Description:** Enter a description for the role.

      Click **Apply**.
      Repeat this step to add as many roles as needed.

7. Authorize the application as a client of the database.

   a. At the top of the page, locate the breadcrumb trail, click **App registrations**.

   b. On the App registrations page, click the database registration, **OracleDB_Resource**.

   c. Under **Manage**, click **Expose an API**.

d. Under the Authorized client applications section, click **Add a client application**.

e. In the panel that appears, enter the following details:

   i. Paste the application (client) ID of `HCM APP`.

   ii. Select the database scope.

   Click **Add application**.

You have now configured the application as both a resource server and a client. It is equipped with a client secret, application roles, delegated permissions, and database pre-authorization.

> **See also**
>
> Configure an application to expose a web API in the Microsoft identity platform documentation.

7.3 Create Users and Assign Roles in Microsoft Entra ID

Create users in Microsoft Entra ID and assign them application roles. You can create as many users as your organization requires. The role assignments flow into each user's JWT `roles` claim and determine which data roles the Oracle AI Database activates in an end-user security context.

1. Create new users in your Microsoft Entra ID domain.

   a. On the Microsoft Entra portal's Home page, click **Users** in the left navigation pane under **Entra ID**.

      The Users page opens.

   b. Click **New user**, then click **Create new user**, and enter the details as shown in the example below:

      - **User principal name:** `marvin@<your-directory>.onmicrosoft.com`
      - **Display name:** `marvin`
      - **Password:** Set a password for the user
      - **Account enabled:** Leave this field checked

      Click **Review + create**, and then click **Create**. From the Users page, copy the user's full user principal name (UPN).

   Repeat this step for each additional user your organization requires.

   > **Note**
   >
   > New users are assigned a temporary password at creation. Each user is prompted to change their password on first sign-in.

2. Assign users to your application (`HCM APP`) in Entra ID to enable sign-in. Select specific application roles to define their data access privileges.

   a. In the left navigation pane under **Entra ID**, click **Enterprise apps**, and then select **HCM APP**.

   b. Click **Assign users and groups** in the Getting Started section of the application's Overview page.

c. On the Users and groups page, click **Add user/group** to assign a new user to the application.

d. On the Add Assignment page, perform the following actions:

   i. Select **Marvin** under **Users**.

   ii. Select **MANAGER** under **Select a role**.

   iii. Click **Assign**.

Repeat this step to assign additional roles to the same user or to assign roles to other users.

You have now provisioned users in Entra ID and assigned them application roles on the `HCM APP` application.

> **See also**
>
> Managing App Roles in Microsoft Entra ID in *Oracle AI Database Security Guide*.

7.4 Validate the Microsoft Entra ID Configuration

Before you configure the database, confirm that the application registrations, scopes, client secret, and user and role assignments configured in Microsoft Entra ID are working as expected.

Validation consists of obtaining two tokens from Microsoft Entra ID and inspecting their claims:

- **Database-access token**: Issued to the application through the client credentials grant.
- **End-user token**: Issued to the test user through the authorization code grant with Proof Key for Code Exchange (PKCE).

To complete this task, you need the following:

- Command-line environment with `curl` and `openssl` installed.
- Application (client) ID, application ID URI, redirect URI, and the client secret value of `HCM APP`. See Register the Application in Microsoft Entra ID.
- Application ID URI of `OracleDB_Resource`, and directory (tenant) ID. See Register the Database in Microsoft Entra ID.
- User principal name (UPN) and password of a test user assigned at least one application role. See Create Users and Assign Roles in Microsoft Entra ID.
- JWT debugger, such as a command-line JWT decoder or a trusted web-based debugger, for inspecting token claims.

1. Set up environment variables.

   Create a `.env` file in your working directory with the following contents. Subsequent steps source variables from this file:

   ```
   # Application client ID and secret
   APPLICATION_CLIENT_ID='<application_client_id>'
   APPLICATION_CLIENT_SECRET='<application_client_secret>'

   # OAuth 2.0 endpoints
   TOKEN_URI='<token_uri>'
   ```

```
AUTH_URI='<auth_uri>'

# Scope for database-access token (client credentials flow)
DB_SCOPE='<db_scope>'

# Scope for end-user token (authorization code flow with PKCE)
APPLICATION_SCOPE='<application_scope>'

# PKCE values — populated by generate_pkce.sh in step 3.a
CODE_VERIFIER=''
CODE_CHALLENGE=''

# Redirect URI registered for HCM APP
REDIRECT_URI='<redirect_uri>'
```

The following list describes each placeholder:

- `<application_client_id>`: Application (client) ID of `HCM APP`.
- `<application_client_secret>`: Client secret value of `HCM APP`.
- `<token_uri>`: `https://login.microsoftonline.com/<tenant_id>/oauth2/v2.0/token`, where `<tenant_id>` is the directory (tenant) ID.
- `<auth_uri>`: `https://login.microsoftonline.com/<tenant_id>/oauth2/v2.0/authorize`, with the same `<tenant_id>` as above.
- `<db_scope>`: Application ID URI of `OracleDB_Resource`, followed by `/.default`. For example, `https://supremo.onmicrosoft.com/fe58fefb-0925-4c8f-9b14-598a0d2f4552/.default`.
- `<application_scope>`: Application ID URI of `HCM APP`, followed by `/user_access`. For example, `api://<hcm_app_client_id>/user_access`.
- `<redirect_uri>`: Redirect URI registered for `HCM APP`. For example, `http://localhost:3000`.
- `CODE_VERIFIER` and `CODE_CHALLENGE`: Leave empty. The script in *step 3.a* populates these.

Source the `.env` file into your shell to make the variables available:

```
set -a && source .env && set +a
```

2. Request a database-access token through the client credentials grant.

   Run the following command:

```
curl -s -X POST "$TOKEN_URI" \
  -H "Content-Type: application/x-www-form-urlencoded" \
  --data-urlencode "grant_type=client_credentials" \
  --data-urlencode "client_id=$APPLICATION_CLIENT_ID" \
  --data-urlencode "client_secret=$APPLICATION_CLIENT_SECRET" \
  --data-urlencode "scope=$DB_SCOPE"
```

   A successful response returns HTTP 200 with a JSON body containing an `access_token` field. Decode the token and verify the following claims in its payload (this section assumes v1 tokens, according to the configuration in Register the Database in Microsoft Entra ID):

   - `aud`: Matches the application ID URI of `OracleDB_Resource`.

- `appid`: Matches the application (client) ID of `HCM APP`.
- `iss`: The Entra ID issuer URL for your tenant (for example, for v1 tokens: `https://sts.windows.net/<tenant_id>/`).

If the request returns HTTP 401 with `AADSTS7000215` or a similar invalid client secret error, verify the `HCM APP` client secret value and its expiration date.

3. Request an end-user token through the authorization code grant with PKCE.

   a. Generate a PKCE verifier and challenge.

   Create the helper script `generate_pkce.sh`:

```
cat > generate_pkce.sh << 'EOF'
#!/bin/bash
CODE_VERIFIER=$(openssl rand -base64 64 | tr -d '=+/' | tr -d '\n' |
cut -c1-64)
CODE_CHALLENGE=$(echo -n "$CODE_VERIFIER" | openssl dgst -sha256 -
binary | base64 | tr '+/' '-_' | tr -d '=')

# Update .env file in place
sed -i "s|CODE_VERIFIER=.*|CODE_VERIFIER='$CODE_VERIFIER'|" .env
sed -i "s|CODE_CHALLENGE=.*|CODE_CHALLENGE='$CODE_CHALLENGE'|" .env

echo "Generated and updated .env:"
echo "CODE_VERIFIER=$CODE_VERIFIER"
echo "CODE_CHALLENGE=$CODE_CHALLENGE"
EOF
chmod +x generate_pkce.sh
```

   Run the script and reload the `.env` file to pick up the generated values:

```
./generate_pkce.sh
set -a && source .env && set +a
```

   b. Construct the authorization URL and sign in.

   Use the following command to build the authorization URL and print it to the console:

```
AUTH_URL="${AUTH_URI}?client_id=${APPLICATION_CLIENT_ID}"
AUTH_URL+="&response_type=code&redirect_uri=${REDIRECT_URI}"
AUTH_URL+="&scope=${APPLICATION_SCOPE}"
AUTH_URL+="&response_mode=query&prompt=consent"
AUTH_URL+="&code_challenge=${CODE_CHALLENGE}&code_challenge_method=S256"

echo -e "\nOpen this URL in your browser:\n$AUTH_URL\n"
```

   Copy the printed URL into a web browser and sign in as the test user. After successful authentication, the browser redirects to the redirect URI with an authorization code, appended as a query parameter. The redirect page typically fails to load; this is expected.

   From the browser's address bar, copy the authorization code value (everything after `code=` and before the next `&`, if any).

   c. Exchange the authorization code for the end-user token.

Run the following command, replacing <AUTH_CODE_FROM_REDIRECT> with the authorization code copied in the previous step:

```
curl -s -X POST "$TOKEN_URI" \
  -H "Content-Type: application/x-www-form-urlencoded" \
  --data-urlencode "grant_type=authorization_code" \
  --data-urlencode "client_id=$APPLICATION_CLIENT_ID" \
  --data-urlencode "client_secret=$APPLICATION_CLIENT_SECRET" \
  --data-urlencode "code=<AUTH_CODE_FROM_REDIRECT>" \
  --data-urlencode "redirect_uri=$REDIRECT_URI" \
  --data-urlencode "code_verifier=$CODE_VERIFIER" \
  --data-urlencode "scope=$APPLICATION_SCOPE"
```

After you obtain the end-user token, decode it and verify the following claims in its payload:

- `aud`: Matches the application ID URI of `HCM APP` (for example, `api://<hcm_app_client_id>`).
- `upn`: Matches the user principal name of the test user (for example, `marvin@<your-directory>.onmicrosoft.com`).
- `roles`: An array containing the application role values assigned to the test user (for example, `["MANAGER"]`).

If the `roles` claim is missing from the token, verify that the test user is assigned to `HCM APP` with at least one application role. See Create Users and Assign Roles in Microsoft Entra ID. If the `roles` claim is present but does not include the expected role values, verify the role definitions and confirm that the `Value` field of each role contains the required value. See Register the Application in Microsoft Entra ID.

> **Note**
>
> If your configuration uses v2 tokens, see Enabling Microsoft Entra ID v2 Access Tokens in *Oracle AI Database Security Guide*.

Successful completion of both token requests confirms that Entra ID is configured correctly for Oracle Deep Data Security. The `roles` claim verified in the end-user token is what Oracle AI Database reads at runtime to activate the corresponding data roles.

You can now proceed to configure Oracle AI Database to accept and validate these tokens.

8

Configure OCI IAM for Application-Mediated Access

Register applications, create users, and configure groups in Oracle Cloud Infrastructure Identity and Access Management (OCI IAM) to support authentication and authorization for application-mediated access to the database.

In OCI IAM, groups serve the same authorization function that application roles serve in Microsoft Entra ID. When OCI IAM issues a token, the user's group memberships appear as claims in the token. Oracle AI Database reads these group claims and activates the corresponding data roles.

> **Note**
>
> - This chapter covers only the OCI IAM configuration required for IAM-managed users connecting through an application. For the full range of OCI IAM capabilities, see Managing Identity Domains in Oracle Cloud Infrastructure documentation. For OAuth flow details, see Using OAuth 2 to Access the REST API.
> - The OCI console interface may be updated over time. If a specific label or navigation path differs from the instructions provided here, look for the closest matching option.

Topics:

8.1 Register the Database in OCI IAM

Register Oracle AI Database as a confidential application in your OCI IAM identity domain.

This application serves two roles:

- **Resource server:** Allows other applications to request tokens scoped to the database.
- **Client:** Allows the database to obtain an access token to fetch the IAM public signing keys for token validation.

1. Navigate to your identity domain.
   a. Log in to the OCI Console.
   b. In the search bar, type `domain` and select **Domains** under **Services**.

c. Select your identity domain (for example, **Default**), or click **Create domain** to create a new one.

d. On the domain's Details page, copy and save the **Domain URL** for later use (for example, `https://idcs-<unique_id>.identity.oraclecloud.com:443`).

2. Create a confidential application for the database.

   a. In the domain, click the **Integrated applications** tab.

   b. Click **Add application**, select **Confidential Application**, and click **Launch workflow**.

   c. Enter a name for the application (for example, `OracleDB`) and an application URL.

   d. Click **Submit**.

3. Configure the application as a resource server.

   a. On the application page, click the **OAuth configuration** tab.

   b. In the Edit OAuth configuration section, under **Resource server configuration**, select **Configure this application as a resource server now**.

   c. Under **Primary audience**, enter `OracleDB`.

   d. Toggle **Add scopes** on. Click **Add** and create a scope:

      - **Scope:** Enter `DB_ACCESS_SCOPE`.
      - **Display name**: Enter `DB Access`.
      - **Description**: Enter `Access the database`.

4. Configure the application as a client.

   The database must also be configured as a client, so it can obtain an access token to retrieve the IAM public signing keys.

   a. Under **Client configuration**, select **Configure this application as a client now**.

   b. Under **Authorization**, select **Client credentials**.

5. Activate the application.

   Click **Actions** in the top-right corner, and select **Activate**.

6. Record the following values for database configuration.

   a. On the application page, click the **Details** tab, and copy the **Application ID**.

   b. Click the **OAuth configuration** tab, scroll to **General Information**, and copy the **Client ID** and **Client secret**.

You have now registered the database in OCI IAM as both a resource server (with an audience and scope) and a client. Other applications can now request tokens scoped to this database resource.

> ⓘ **See also**
>
> Adding a Confidential Application in the OCI IAM documentation.

8.2 Register the Application in OCI IAM

Register your application as a separate confidential application in OCI IAM.

This application registration serves two roles:

- **Resource server:** Exposes a scope that end users authenticate against. When a user signs in, OCI IAM issues an end-user token scoped to this application.
- **Client:** Uses its client credentials to obtain a database-access token and allows users to obtain access tokens scoped to a resource it exposes. The application authenticates to Oracle AI Database with this database-access token and attaches the end-user token to the connection as part of the end-user security context payload.

1. Create a confidential application.
   a. In the same identity domain, click the **Integrated applications** tab.
   b. Click **Add application**, select **Confidential Application**, and click **Launch workflow**.
   c. Enter a name for the application (for example, `HCM APP`).
   d. Click **Submit**.
2. Configure the application as a resource server.

   This step exposes a scope so that end users can obtain access tokens scoped to the application.
   a. On the application's page, click the **OAuth configuration** tab.
   b. In the Edit OAuth configuration section, under **Resource server configuration**, select **Configure this application as a resource server now**.
   c. Under **Configure application APIs that need to be OAuth protected**, set the **Access token expiration (seconds)** as appropriate (for example, 3600).
   d. Under **Primary audience**, enter a value that identifies the application (for example, `OracleConfidentialClient`).
   e. Toggle **Add scopes** on. Click **Add** and create a scope:
      - **Scope:** Enter a scope name (for example, `APP_ACCESS_SCOPE`).
      - **Display name**: Enter a display name (for example, `APP_ACCESS_SCOPE`).
3. Configure the application as a client.

   This step allows the application to obtain database-access tokens and to receive end-user sign-ins through the authorization code flow.
   a. Under **Client configuration**, select **Configure this application as a client now**.
   b. Under **Authorization**, in **Allowed grant types**, select the grant types your application requires. For most deployments, select the following:
      - **Authorization code:** Allows end users to sign in through a browser and receive an end-user token scoped to this application.
      - **Client credentials:** Allows the application to obtain a database-access token, which it uses to authenticate its connection to the database.
   c. If you selected **Authorization code**, enter the URL under **Redirect URL** (for example, `https://hcm.example.com/oauth2/callback`). Click **Add redirect URL**.
4. Grant the application access to the database resource and its own scope.

   The application needs two scopes: the database scope (so it can obtain database-access tokens) and its own scope (so end-user tokens include the claims required for database authentication).
   a. Under **Client configuration**, toggle **Add resources** on.
   b. Click **Add scope**.

c. In the **Add scope** dialog, expand the database application (for example, **OracleDB**) and select the scope (for example, `OracleDBDB_ACCESS_SCOPE`).

d. Expand this application (for example, **HCM APP**) and select its own scope (for example, `OracleConfidentialClientAPP_ACCESS_SCOPE`).

e. Click **Submit** to save.

5. Activate the application.

   Click **Actions** in the top-right corner, and select **Activate**.

6. Record the following values for application configuration.

   a. On the application page, click the **Details** tab, and copy the **Application ID**.

   b. Click the **OAuth configuration** tab, scroll to **General Information**, and copy the **Client ID** and **Client secret**.

You have now registered the application in OCI IAM as both a resource server (so end users can obtain end-user tokens scoped to this application) and a client (so the application can obtain a database-access token through client credentials).

> **See also**
>
> Configuring OAuth in the OCI IAM documentation.

8.3 Configure Custom Claims for Group Information in OCI IAM

In OCI IAM, you must create a custom claim rule to populate a user's group memberships into a `group` claim in an access token. The Oracle AI Database reads this claim to activate the corresponding data roles.

The OCI Console does not provide a user interface for managing custom claims on tokens. You can create and manage custom claims only through the REST API. See Managing Custom Claims in the OCI IAM documentation.

To complete this task, you need the following:

- A user account with the Identity Domain Administrator role in the target identity domain.
- A command-line environment with `curl` installed, such as OCI Cloud Shell, a local terminal, or Windows Subsystem for Linux (WSL).

1. Obtain a personal access token with identity domain administrator privileges.

   You need an access token to authenticate the REST API call. OCI IAM allows identity domain administrators to generate a short-lived personal access token through the OCI Console.

   a. In the OCI Console, click your **Profile** icon (in the top-right corner of the Console header), then click your user name.

   b. Click the **Tokens and keys** tab.

   c. Under **My access tokens**, locate the row labeled **Invokes identity domain APIs** and click **Download token**.

   d. In the **Generate personal access token** dialog box:

      - From the **Select app role** drop-down menu, select **Identity Domain Administrator**.

- Set the **Token expires in mins** field to a value appropriate for completing this task (for example, 5).
- Click **Download token**.

e. Open the downloaded token file and copy the token value. This is a time-limited bearer token that you use in the next step.

2. Run the custom claim REST API command.

   Run the following curl command, replacing `<domain_url>` with your identity domain URL (recorded in Register the Database in OCI IAM) and `<access_token>` with the personal access token you downloaded in *step 1* above.

```
curl -X POST https://<domain_url>/admin/v1/CustomClaims \
  -H "Authorization: Bearer <access_token>" \
  -H "Content-Type: application/scim+json" \
  -d '{
    "schemas": [
      "urn:ietf:params:scim:schemas:oracle:idcs:CustomClaim"
    ],
    "name": "group",
    "value": "$user.groups.*.display",
    "expression": true,
    "mode": "always",
    "tokenType": "AT",
    "allScopes": true
  }'
```

   The following list describes each parameter in the request body:

   - `"name"`: The claim name to add to the access token. The database reads this specific claim.
   - `"value"`: The dynamic expression that resolves to the display names of the user's groups.
   - `"expression"`: The flag indicating if the value is a dynamic expression or a static string. Set to `true` to indicate a dynamic expression.
   - `"mode"`: The inclusion mode. Set to `always` to include the claim in every access token.
   - `"tokenType"`: The token type to which the claim is added. Set to `AT` to add the claim to access tokens only. Use `BOTH` to include identity tokens as well.
   - `"allScopes"`: The scope filter. Set to `true` to include the claim regardless of the requested scope.

3. Verify the response.

   A successful request returns an HTTP `201 Created` status code and a JSON response body. Verify that the response includes the following key fields:

   - `"name": "group"`
   - `"value": "$user.groups.*.display"`
   - An `"id"` field containing a unique identifier for the custom claim (for example, `"60b9904b5895431e90f2b60a7edeb28e"`). Record this identifier in case you need to modify or delete the claim later.

If you receive an HTTP `401 Unauthorized` error, verify that the personal access token has not expired and that you copied the full token value. If you receive an HTTP `403 Forbidden` error, confirm that your user account has the Identity Domain Administrator role.

A custom claim rule is now active on the identity domain. From this point forward, every access token issued by this domain includes a group claim containing the display names of the authenticated user's group memberships.

8.4 Create Users and Assign Groups in OCI IAM

Create users and groups in OCI IAM and assign users to the appropriate groups.

Group memberships appear in the token's `group` claim (configured in the previous section), which the Oracle AI Database reads to activate data roles. You can create as many users and groups as your organization requires.

1. Create groups representing application roles.
   a. In your identity domain, click the **Groups** tab.
   b. Click **Create group** and enter the following details:
      - **Name**: A name for the group; for example, `MANAGER`.
      - **Description**: Enter `Managers group with full access`.

      Click **Create**. Repeat this step for each additional group your organization requires (for example, `EMPLOYEE`).
2. Create users.
   a. In the identity domain, click the **User management** tab, and go to the Users section.
   b. Click **Create** and enter the following details:
      - In the **First name** and **Last name** fields, enter the user's name (for example, `Marvin Anderson`).
      - In the **Username / Email** field, enter the user's email address (for example, `marvin@<yourdomain>.com`).

      Click **Create**. Repeat this step for each additional user your organization requires.
3. Assign users to groups.
   a. In the Users section, click a user's name, then click the **Groups** tab.
   b. Click **Assign user to groups**, select the groups to assign, and click **Assign user**.

      Repeat this step for each user.
4. Assign users to the application (`HCM APP`).
   a. In your identity domain, click the **Integrated applications** tab.
   b. Click the application (for example, **HCM APP**).
   c. Click the **Users** tab, and then click **Assign users**.
   d. Select the users who need access. Alternatively, click the **Groups** tab, assign a group to the application, and then add users to that group.

You have now provisioned users and groups in OCI IAM and assigned them to the application. When these users authenticate and obtain end-user tokens, their group memberships appear in the `group` claim.

> **See also**
>
> Creating a User and Managing Groups in the OCI IAM documentation.

8.5 Validate the OCI IAM Configuration

Before you configure the database, confirm that the application registrations, scopes, custom claim rule, and user assignments you configured in OCI IAM are working as expected.

Validation consists of obtaining two tokens from OCI IAM and inspecting their claims:

- **Database-access token**: Issued to the application through the client credentials grant.
- **End-user token**: Issued to the test user through the authorization code grant.

To complete this task, you need the following:

- Command-line environment with `curl` installed.
- Client ID and client secret for the database registration, identity domain URL, and database-access scope. See Register the Database in OCI IAM.
- Client ID and client secret for the application, application scope, and redirect URL. See Register the Application in OCI IAM.
- User name and password of a test user assigned to at least one group. See Create Users and Assign Groups in OCI IAM.
- JWT debugger, such as a command-line JWT decoder or a trusted web-based debugger, for inspecting token claims.

1. Request a database-access token through the client credentials grant.

   Run the following command, substituting values as described below:

   ```
   curl -i \
     -H "Authorization: Basic <encoded_app_credentials>" \
     -H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8" \
     --request POST https://<domain_url>/oauth2/v1/token \
     -d "grant_type=client_credentials&scope=<database_scope>"
   ```

   The following list describes each placeholder:

   - `<encoded_app_credentials>`: Base64 encoding of the application's client ID and client secret joined by a colon (`CLIENT_ID:CLIENT_SECRET`). On Linux or macOS, generate the value as follows:

     ```
     echo -n "<CLIENT_ID>:<CLIENT_SECRET>" | base64
     ```

   - `<domain_url>`: Identity domain URL. For example, `idcs-<unique_id>.identity.oraclecloud.com:443`.
   - `<database_scope>`: Database application registration's primary audience concatenated with its scope name. For example, if the primary audience is `OracleDB` and the scope is `DB_ACCESS_SCOPE`, use `OracleDBDB_ACCESS_SCOPE`.

   A successful response returns HTTP 200 with a JSON body containing an `access_token` field. Decode the token and verify the following claims in its payload:

   - `aud`: Matches the database application's primary audience (for example, `OracleDB`).

- `scope`: Contains the database access scope (for example, `OracleDBDB_ACCESS_SCOPE`).
- `sub` or `client_id`: Matches the client ID of the application.

If the request returns HTTP 401, verify the application's client ID and secret. If the request returns HTTP 400 with `invalid_scope`, verify that you added the database scope to the application's Resources section. See Register the Application in OCI IAM.

2. Request an end-user token through the authorization code grant.

   a. Obtain an authorization code.

   Paste the following URL into a browser and sign in as the test user:

   ```
   https://<domain_url>/oauth2/v1/authorize?
   client_id=<app_client_id>&response_type=code&redirect_uri=<url_encoded_r
   edirect_url>&scope=<application_scope>
   ```

   The following list describes each placeholder:

   - `<domain_url>`: Same identity domain URL used in the database-access token request.
   - `<app_client_id>`: Client ID of the application.
   - `<url_encoded_redirect_url>`: URL-encoded form of the registered redirect URL. For example, `https://hcm.example.com/oauth2/callback` becomes `https%3A%2F%2Fhcm.example.com%2Foauth2%2Fcallback`.
   - `<application_scope>`: Application's scope (primary audience concatenated with scope name). For example, `OracleConfidentialClientAPP_ACCESS_SCOPE`.

   After sign-in, the browser redirects to the registered redirect URL, with an authorization code appended as a query parameter (for example, `?code=<authorization_code>`). Copy the authorization code value.

   b. Exchange the authorization code for an end-user token.

   Run the following command:

   ```
   curl -i \
     -H "Authorization: Basic <encoded_app_credentials>" \
     -H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8" \
     --request POST https://<domain_url>/oauth2/v1/token \
     -d "grant_type=authorization_code&code=<authorization_code>"
   ```

   The following list describes each placeholder:

   - `<encoded_app_credentials>`: Same Base64-encoded credentials used in the database-access token request.
   - `<domain_url>`: Same identity domain URL used in the database-access token request.
   - `<authorization_code>`: Authorization code from the redirect URL in the previous task.

   A successful response returns HTTP 200 with a JSON body containing an `access_token`. Decode the token and verify the following claims in its payload:

   - `aud`: Matches the application's primary audience (for example, `OracleConfidentialClient`).

- `scope`: Contains the application scope (for example, `OracleConfidentialClientAPP_ACCESS_SCOPE`).
- `group`: Contains the display names of the groups the test user is assigned.

The token also includes subject and user identity claims (for example, `sub` and `user.name`) that identify the test user.

If the `group` claim is missing from the token, revisit Configure Custom Claims for Group Information in OCI IAM and confirm that the REST API call returned HTTP 201 and that the custom claim's `name` field is `group`. If the `group` claim is present but does not list the expected groups, verify the test user's group memberships and whether the test user is assigned to the application. See Create Users and Assign Groups in OCI IAM.

Alternative: Obtain an end-user token using the Resource Owner Password Credentials grant

For non-production validation, you can skip the browser sign-in by requesting an end-user token through the Resource Owner Password Credentials (ROPC) grant. This approach is suitable only for automated testing in non-production environments.

To use the ROPC grant, first select the **Resource owner** grant type on the application registration in Register the Application in OCI IAM, then run:

```
curl -i \
  -H "Authorization: Basic <encoded_app_credentials>" \
  -H "Content-Type: application/x-www-form-urlencoded;charset=UTF-8" \
  --request POST https://<domain_url>/oauth2/v1/token \
  -d
"grant_type=password&username=<username>&password=<password>&scope=<applicatio
n_scope>"
```

The following list describes each placeholder:

- `<encoded_app_credentials>`: Same Base64-encoded credentials used in the database-access token request.
- `<domain_url>`: Same identity domain URL used in the database-access token request.
- `<username>` and `<password>`: Credentials of the test user created in Create Users and Assign Groups in OCI IAM.
- `<application_scope>`: Same application scope used in the authorization code request.

Decode the returned token and verify the same claims as in the authorization code grant.

Successful completion of both token requests confirms that OCI IAM is configured correctly for Oracle Deep Data Security. The `group` claim verified in the end-user token is what Oracle AI Database reads at runtime to activate the corresponding data roles.
You can now proceed to configure Oracle AI Database to accept and validate these tokens.

9

Configure the Database and Application

Learn how to configure your Oracle AI Database and application layer for Oracle Deep Data Security (Deep Sec) data authorization.

This chapter guides you through the database setup (identity provider integration, connection pool user accounts, and data roles), followed by application configuration for propagating the end-user security context payload, and SQL client configuration for direct logon scenarios.

Topics:

9.1 Configure Oracle AI Database

Configure Oracle AI Database to accept end-user security context payloads by setting up identity provider configuration, connection pool user accounts, data roles, and local end-user accounts.

The procedures differ depending on whether your end users are managed in an identity and access management (IAM) system or managed locally — either in the application's own user store or in the database.

> **Note**
>
> Connect to the database as a named user with the DBA role to perform the configuration tasks in this section.

Topics:

9.1.1 Configure the Database for IAM Integration

Complete the required configuration in your database to integrate with an IAM system and implement Oracle Deep Data Security (Deep Sec). Your specific configuration steps vary depending on whether your IAM-managed users connect through an application or directly through a SQL client.

9.1.1.1 Set Up IAM Integration for Application-Mediated Connections

If your IAM-managed users connect to the database through an application, complete the following configuration steps in your database to enable end-user security context establishment.

> **Note**
>
> - On Oracle AI Database, run the SQL and PL/SQL statements in this section as a named user with the DBA role. Run the statements that create the OCI IAM credential object as `SYS`.
> - On Oracle Autonomous AI Database, run the statements as the `ADMIN` user.

1. Enable transport layer security (TLS).

   Transmission of the end-user security context payload requires a TLS-secured connection between the Oracle client driver and the database server. If TLS is not already configured, see Configuring Transport Layer Security Encryption in *Oracle AI Database Security Guide*.

2. Set up the connection pool user account.

   Create a connection pool user account and grant it both the `CREATE SESSION` and `CREATE END USER SECURITY CONTEXT` privileges. Without the latter privilege, the database server rejects any attempt to attach an end-user security context payload to the session.

   You can provision the connection pool user account in either of the following two ways.

   a. IAM-authenticated connection pool user account

      Create the connection pool user account by identifying it with the application's IAM client ID. This user authenticates to the database using the database-access token. This authentication method for the connection pool user account is supported only for Microsoft Entra ID.

      ```
      CREATE USER hr_app_user IDENTIFIED GLOBALLY
        AS 'AZURE_CLIENT_ID=${HRAPP_CLIENT_ID}';

      GRANT CREATE SESSION TO hr_app_user;
      GRANT CREATE END USER SECURITY CONTEXT TO hr_app_user;
      ```

   b. Password-authenticated connection pool user account

      Alternatively, create a standard database user account with a password. This user authenticates to the database using the password.

      ```
      CREATE USER hr_app_user IDENTIFIED BY <password>;

      GRANT CREATE SESSION TO hr_app_user;
      GRANT CREATE END USER SECURITY CONTEXT TO hr_app_user;
      ```

3. Configure the identity provider.

   Set the database identity provider parameters to match your database's IAM application registration. This allows the database server to validate the audience (`aud`) claim in the application's database-access token.

On Oracle AI Database, connect to your target pluggable database (PDB) before applying configuration changes.

> **Note**
>
> On Oracle Autonomous AI Database, this step is not required; you connect directly to the PDB as the `ADMIN` user.

```
ALTER SESSION SET CONTAINER = <your-target-PDB>;
```

Configuration details vary by IAM system and token version in use. Complete the steps applicable to your environment.

a. **Microsoft Entra ID v1 Tokens**

For Entra ID v1 tokens, the database verifies that the database-access token's `aud` claim matches the `application_id_uri` field in the database's `identity_provider_config`.

- On Oracle AI Database, run the following statements:

```
ALTER SYSTEM SET IDENTITY_PROVIDER_TYPE = AZURE_AD SCOPE=BOTH;

ALTER SYSTEM SET IDENTITY_PROVIDER_CONFIG = '{
  "application_id_uri": "<DB_APP_ID_URI>",
  "tenant_id": "<TENANT_ID>",
  "app_id": "<DB_APP_ID>"
}' SCOPE=BOTH;
```

- On Oracle Autonomous AI Database, run the following statement:

```
BEGIN
  DBMS_CLOUD_ADMIN.ENABLE_EXTERNAL_AUTHENTICATION(
    type   => 'AZURE_AD',
    params => JSON_OBJECT(
      'tenant_id'          VALUE '<TENANT_ID>',
      'application_id'     VALUE '<DB_APP_ID>',
      'application_id_uri' VALUE '<DB_APP_ID_URI>'
    ),
    force  => TRUE
  );
END;
/
```

Replace the placeholders with values from your Entra ID instance. See Configure Microsoft Entra ID for Application-Mediated Access.

Parameter	Description
`application_id_uri`	The application ID URI of the database resource registered in Entra ID.
`tenant_id`	The directory (tenant) ID of your Entra ID environment.
`app_id`	The application (client) ID of the database resource.

b. Microsoft Entra ID v2 Tokens

For Entra ID v2 tokens, the database verifies that the database-access token's aud claim matches the app_id field in the database's identity_provider_config. The identity provider configuration follows the same format as v1 tokens.

c. Oracle Cloud Infrastructure Identity and Access Management (OCI IAM)

For OCI IAM environments, use the IDENTITY_PROVIDER_OAUTH_CONFIG parameter to set up the identity provider. Additionally, you must create a credential object.

- On Oracle AI Database, run the following statements:

```
ALTER SYSTEM SET IDENTITY_PROVIDER_TYPE = OCI_IAM SCOPE=BOTH;

ALTER SYSTEM SET IDENTITY_PROVIDER_OAUTH_CONFIG = '{
  "app_id": "<application_id>",
  "domain_url": "<domain_url>"
}' SCOPE=BOTH;
```

- On Oracle Autonomous AI Database, run the following statement:

```
BEGIN
  DBMS_CLOUD_ADMIN.ENABLE_EXTERNAL_AUTHENTICATION(
    type   => 'OCI_IAM',
    params => JSON_OBJECT(
      'app_id'     VALUE '<application_id>',
      'domain_url' VALUE '<domain_url>'
    )
  );
END;
/
```

Replace the placeholders with values from your OCI IAM instance. See Configure OCI IAM for Application-Mediated Access.

Parameter	Description
app_id	The application ID of the database application registered in OCI IAM.
domain_url	The OCI IAM domain URL. Used to retrieve the public signing key for OAuth 2.0 token validation.

- Create the credential object.
To obtain an access token for the public signing-key endpoint, the database requires the client ID and client secret of the database's application registration in OCI IAM.

  Create a credential object to store these values securely in the database. Obtain the client ID and client secret from the OAuth Configuration section of your database application in the OCI IAM console, and execute the following block.

  On Oracle AI Database, run the following block as SYS:

```
BEGIN
  DBMS_CREDENTIAL.CREATE_CREDENTIAL(
    credential_name => 'OCI_IAM_DOMAIN_DB_CRED$',
    username        => '<CLIENT_ID>',
```

```
    password        => '<CLIENT_SECRET>'
  );
END;
/
```

On Oracle Autonomous AI Database, run the following block as `ADMIN`:

```
BEGIN
  DBMS_CLOUD.CREATE_CREDENTIAL(
    credential_name => 'OCI_IAM_DOMAIN_DB_CRED$',
    username        => '<CLIENT_ID>',
    password        => '<CLIENT_SECRET>'
  );
END;
/
```

To verify the identity provider settings, run the following command from your SQL*Plus prompt:

```
SHOW PARAMETER identity;
```

4. Create data roles.

   Define the data roles that you want to enable in your end-user security contexts. Because you are using IAM-managed users, you must create data roles mapped to the application roles in IAM. For the complete syntax and options, see Configure Data Roles.

5. Perform optional additional configuration.

   The following tasks are not mandatory to establish a basic end-user security context, but may be required depending on your application's authorization model.

   a. Create end-user context definitions.

      If your application logic or data grants rely on custom end-user context attributes, create the corresponding `END USER CONTEXT` schema objects before deployment. See Configure End-User Contexts and Attributes.

   b. Create the application identity.

      Set up a database identity for your application that matches its IAM client ID. This identity authorizes the application to activate specific data roles within an end-user security context. See Configure Application Identities.

   c. Grant additional data roles to the application identity.

      Explicitly grant any additional or common data roles that your application is allowed to activate. If you skip this step, the database silently ignores the application's requests for ungranted roles. See Grant and Revoke Data Roles.

9.1.1.2 Set Up IAM Integration for Direct Logon

If your IAM-managed users (such as data analysts or developers) connect directly to the database using a SQL client and their own IAM access tokens, the database-side setup is more concise. Complete the following tasks.

1. Enable transport layer security (TLS).

Transmission of the end-user security context payload requires a TLS-secured connection between the client and the database server. If TLS is not already configured, see Configuring Transport Layer Security Encryption in *Oracle AI Database Security Guide*.

2. Configure the identity provider.

   Set the database identity provider parameters to match your database's IAM application registration. This enables the database server to validate the audience (`aud`) claim in the end-user token.

   Follow the procedure described in Set Up IAM Integration for Application-Mediated Connections.

3. Create data roles.

   Define the data roles that you want to enable in your end-user security contexts. Because you are using IAM-managed users, you must create data roles mapped to the application roles in IAM.

   ```
   CREATE DATA ROLE employee_role MAPPED TO 'AZURE_ROLE=EMPLOYEE';
   ```

   For the complete syntax and options, see Configure Data Roles.

4. Create and grant a generic database role.

   Create a generic database role and grant it the `CREATE SESSION` privilege. Then grant this generic role to your data role used for direct logon. See Grant Database Role to Data Role.

   ```
   -- Create a standard database role for connection privileges
   CREATE ROLE db_role;
   GRANT CREATE SESSION TO db_role;

   -- Grant the connection privileges to the data roles
   GRANT db_role TO employee_role;
   ```

9.1.2 Configure the Database for Local End-User Authentication

Complete the required configuration in your database to implement Oracle Deep Data Security (Deep Sec) when your end users are managed locally rather than through an IAM system. Your specific configuration steps vary depending on whether your local end users connect through an application or establish direct database sessions with password authentication.

- Set Up Local Authentication for Application-Mediated Connections
- Set Up Local Authentication for Direct Logon

9.1.2.1 Set Up Local Authentication for Application-Mediated Connections

If your application maintains its own user store (for example, a user registry in a separate database, LDAP directory, or an application-managed identity system) and those users connect to the database through the application, use the database configuration detailed here to implement Deep Sec, without requiring the users to have IAM accounts.

In this scenario, the local end user is identified by a user name and a security context lookup key supplied by the application.

Even though the end users are managed locally, the application must still be registered in your IAM system and must obtain a database-access token to authorize its own connection to the database. A TLS-secured connection, a connection pool user account, and the identity

provider configuration in the database are therefore required. For application registration in IAM, see Register the Application in Microsoft Entra ID and Register the Application in OCI IAM.

1. Configure the database for application sessions.

   Complete the TLS, connection pool user account, and identity provider setup described in Set Up IAM Integration for Application-Mediated Connections. The configuration for this scenario is identical.

2. Create end users in the database.

   Provision end users with user names that match the users in the application's user store. Because these users authenticate to the database through the application's trust rather than directly, create the end-user accounts *without* a password.

   ```
   CREATE END USER emma;
   ```

   For the complete syntax and options, see Configure Local End Users.

3. Create data roles that are managed locally in the database.

   Create the required data roles to enable within your end-user security contexts. See Configure Data Roles.

   ```
   CREATE DATA ROLE employee_role;
   ```

4. Grant the data roles to end users.

   Explicitly grant appropriate data roles to your end-user accounts. See Grant and Revoke Data Roles.

   ```
   GRANT DATA ROLE employee_role TO emma;
   ```

5. Perform optional additional configuration.

   The following tasks are not mandatory to establish a basic end-user security context, but may be required depending on your application's authorization model.

   a. Create end-user context definitions.

      If your application logic or data grants rely on custom end-user context attributes, create the corresponding `END USER CONTEXT` schema objects before deployment. See Configure End-User Contexts and Attributes.

   b. Create the application identity.

      Set up a database identity for your application that matches its IAM client ID. See Configure Application Identities.

9.1.2.2 Set Up Local Authentication for Direct Logon

For environments where local end users establish direct database sessions through a SQL client using credential-based authentication, perform the following configuration tasks to implement Deep Sec.

1. Create end users in the database.

   For this scenario, create end-user accounts with password authentication enabled.

   ```
   CREATE END USER emma IDENTIFIED BY <password>;
   ```

For the complete syntax and options, see Configure Local End Users.

2. Create data roles that are managed locally in the database.

   Create the required data roles to enable within your end-user security contexts. See Configure Data Roles.

   ```
   CREATE DATA ROLE employee_role;
   ```

3. Create and grant a generic database role.

   Create a generic database role and grant it the `CREATE SESSION` privilege. Then grant this generic role to your data role used for direct logon. See Grant Database Role to Data Role.

   ```
   CREATE ROLE db_role;
   GRANT CREATE SESSION TO db_role;

   GRANT db_role TO employee_role;
   ```

4. Grant the data roles to end users.

   Explicitly grant appropriate data roles to your end-user accounts. See Grant and Revoke Data Roles.

   ```
   GRANT DATA ROLE employee_role TO emma;
   ```

9.2 Update Application Configuration with IAM Details

To enforce data access control using Oracle Deep Data Security (Deep Sec), the application must pass an end-user security context payload to the database with every SQL operation through an Oracle client driver.

This section describes how to configure an application to build and propagate this payload for all deployment scenarios where end users connect to the database *through the application*, regardless of whether those users are IAM-managed or maintained in the application's own user store.

> **Note**
>
> You must configure the database server for TLS before configuring the application. The client-side trust store requires the server certificate to establish trust. For TLS configuration, see Configuring Transport Layer Security Encryption in *Oracle AI Database Security Guide*.

Topics:

- Understand Authentication Flow and Prerequisites
- Configure Java Applications
- Configure Python Applications
- Configure .NET Applications
- Advanced: Implement a Security Context Provider

9.2.1 Understand Authentication Flow and Prerequisites

Review the authentication concepts and gather the necessary credentials before modifying your application configuration.

Authentication flow

For the application to successfully construct and propagate the `EndUserSecurityContext` payload to the database, you must register your application as a confidential client in an IAM system. During runtime, the application must typically acquire and manage two distinct tokens to build the security context payload:

- **End-user token:** An OAuth 2.0 access token generated when the end user logs in through the browser. This token establishes the user's identity and role claims (app roles in Microsoft Entra ID or group memberships in OCI IAM).
- **Database-access token:** A token the application obtains to authorize itself to access the database resource. This is acquired through the on-behalf-of (OBO) flow or the client credentials flow.

The application passes these tokens as part of the `EndUserSecurityContext` payload on the database connection; the database-access token authorizes the connection, and the end-user token supplies the user identity and role claims used to enforce data security.

Required credentials

Before proceeding, ensure you have gathered the following values from your application's IAM registration:

- **Client ID:** The unique identifier assigned to your application.
- **Client secret:** The secret key generated for the application.
- **Tenant ID** (Entra ID) / **Domain URL** (OCI IAM): The identifier for your IAM instance.
- **Database Scope:** The scope required to access the database resource (for example, `[DB_APP_ID_URI]/sessions:scope:connect` in Entra ID or `[primary_audience]DB_ACCESS_SCOPE` in OCI IAM).

9.2.2 Configure Java Applications

The Oracle JDBC driver supports two approaches for propagating the end-user security context payload from your Java application to the database: API extension methods and a configuration-driven Service Provider Interface (SPI). API calls take precedence over provider-based configurations.

Topics:

9.2.2.1 Use the API Extension Methods

If you want to supply the end-user security context payload directly in your application code, use the API extension methods.

This approach involves interacting directly with the `oracle.jdbc.OracleConnection` interface within your Java code to manually set and clear the security context payload.

1. Build the security context payload.

   Use the `createWithToken` or `createWithUsername` method of `oracle.jdbc.EndUserSecurityContext` to create the security context object, with the end-user identity and optional data roles. For example, you can create a security context payload named `myEndUserSecurityContext`.

2. Set the security context payload.

   Before executing SQL statements, attach the security context payload to the database connection. If your connection object is a `java.sql.Connection` (as is typical when obtained from a `DataSource` or connection pool), you must first cast or unwrap it to `OracleConnection` and then attach the payload.

   ```
   OracleConnection oracleConnection =
       connection instanceof OracleConnection
           ? (OracleConnection) connection
           : connection.unwrap(OracleConnection.class);

   oracleConnection.setEndUserSecurityContext(myEndUserSecurityContext);
   ```

3. Perform database operations.

4. Clear the security context payload.

   Always clear the security context payload before returning the connection to a pool to prevent data leakage. Use a `finally` block to ensure the security context payload is cleared even if an exception occurs during database operations.

   ```
   OracleConnection oracleConnection =
       connection instanceof OracleConnection
           ? (OracleConnection) connection
           : connection.unwrap(OracleConnection.class);

   oracleConnection.setEndUserSecurityContext(myEndUserSecurityContext);
   try {
       executeSqlAsEndUser(connection);
   }
   finally {
       oracleConnection.clearEndUserSecurityContext();
   }
   ```

9.2.2.2 Use the Service Provider Interface

The Service Provider Interface (SPI) approach enables your Java application to supply the end-user security context payload without requiring modifications to your core application code.

Oracle JDBC defines a Java interface named `oracle.jdbc.spi.EndUserSecurityContextProvider`. This interface is designed to work as a

standard service provider interface that is dynamically loaded by Java's `java.util.ServiceLoader` class. This design allows you to implement the interface by integrating with different Java security technologies to provide a security context payload to Oracle JDBC.

By simply installing and configuring a provider implementation, you can enable your application to automatically propagate the security context payload to the database before each SQL operation. The SPI approach also allows you to integrate with different IAM-specific providers without changing how your application interacts with the JDBC driver.

To supply the security context payload using the SPI approach, perform the following tasks.

1. Install the driver.

   Ensure the core Oracle JDBC driver is included in your project dependencies by adding the following block to your `pom.xml` file.

   ```
   <!-- Oracle JDBC driver -->
           <dependency>
               <groupId>com.oracle.database.jdbc</groupId>
               <artifactId>ojdbc11</artifactId>
               <version>23.26.2.0.0</version>
           </dependency>
   ```

2. Install a provider implementation.

   Add an implementation of `oracle.jdbc.spi.EndUserSecurityContextProvider` to your application's class path. You can use a pre-built provider published to a repository such as Maven Central, or implement the interface yourself to integrate with your security framework. For a pre-built provider that integrates with Spring Security, see Example: Use the JDBC Spring Boot Provider (The SPI Approach).

3. Configure the data source to use the provider.

   Configure the application's data-source connection properties to use the installed provider. The specific properties depend on the provider implementation. See your provider's documentation for the required configuration.

9.2.2.3 Example: Use the JDBC Spring Boot Provider (The SPI Approach)

The JDBC Spring Boot provider is an out-of-the-box (OOTB) security context payload provider for the Oracle JDBC driver. It automates the propagation of the end-user security context payload from Spring Security (OAuth 2.0) to the database through the JDBC connection. In most cases, no application code changes are necessary.

For Spring Boot applications, both the IAM registration and the data source driver configuration are handled in the `application.properties` file. You must link the data source to the OAuth registration using the `registrationId`.

1. Install the provider.

   Add the Oracle JDBC Spring provider dependency to your `pom.xml`. This artifact is part of the Oracle JDBC Extensions project hosted on GitHub: `https://github.com/oracle/ojdbc-extensions`.

   ```
   <dependency>
       <groupId>com.oracle.database.jdbc</groupId>
       <artifactId>ojdbc-provider-spring</artifactId>
   ```

```
    <version>${ojdbc.provider.version}</version>
</dependency>
```

2. Add IAM credentials and endpoints.

   In your `application.properties` file, add the following properties to configure the application with your application's IAM registration details.

   > **Note**
   >
   > The registration name used below (for example, `hrapp`) acts as the identifier for the next step.

```
# Spring OAuth2 Configuration
spring.security.oauth2.client.registration.hrapp.client-name=hrapp
spring.security.oauth2.client.registration.hrapp.client-
id=<HRAPP_CLIENT_ID>
spring.security.oauth2.client.registration.hrapp.client-
secret=<HRAPP_CLIENT_SECRET>

# Endpoint Configuration
spring.security.oauth2.client.provider.hrapp.token-uri=<HRAPP_TOKEN_URI>
spring.security.oauth2.client.provider.hrapp.authorization-
uri=<HRAPP_AUTH_URI>
spring.security.oauth2.client.registration.hrapp.redirect-
uri=<REDIRECT_URI>

# Scope Configuration
spring.security.oauth2.client.registration.hrapp.scope=<DB_APP_ID_URI>/.def
ault

# --- SELECT ONE GRANT TYPE BELOW ---

# OPTION A: OBO Flow (User assertion)
spring.security.oauth2.client.registration.hrapp.authorization-grant-
type=urn:ietf:params:oauth:grant-type:jwt-bearer

# OPTION B: Client Credentials Flow (Service assertion)
# spring.security.oauth2.client.registration.hrapp.authorization-grant-
type=client_credentials
```

3. Enable the provider.

   Next, enable the provider in your connection pool's data-source configuration. The connection pool setup (data source URL, SSL wallet, pool sizing, and so on) is standard and is not covered here. Ensure the `registrationId` matches the name used in *step 2* (`hrapp`). For the complete connection pool configuration details, see Configure Oracle Deep Data Security for a Sample Application.

```
# Enable the Spring Security Context Provider
spring.datasource.hikari.data-source-
properties.oracle.jdbc.provider.endUserSecurityContext=ojdbc-provider-
spring-end-user-security-context
```

```
# Link to the OAuth2 Registration ID configured in Step 2
spring.datasource.hikari.data-source-
properties.oracle.jdbc.provider.endUserSecurityContext.registrationId=hrapp
```

4. Set additional data roles and context attributes.

   Optionally, you can set additional data roles and end-user context attributes in the Spring security context. The provider attaches these to the end-user security context payload and propagates them with every database operation.

   ```
   # (Optional) Configure default Data Roles or Attributes to pass with every
   connection
   spring.datasource.hikari.data-source-
   properties.oracle.jdbc.provider.endUserSecurityContext.dataRoles=COMPENSATI
   ON_ANALYST
   spring.datasource.hikari.data-source-
   properties.oracle.jdbc.provider.endUserSecurityContext.endUserContextAttrib
   ute={"HR_APP_NAMESPACE":{"APP":"HR"}}
   ```

5. Handle privilege elevation (Java code).

   For scenarios requiring temporary elevated permissions (for example, generating a compensation summary), use the `@RunWithDataRoles` annotation in your Java code. This adds specific data roles to the end-user security context payload for the duration of a method's execution. After the method finishes, the user's original security context is restored.

   a. Define a unique prefix for data roles through your application code (default is `ORACLE_DATA_ROLE_`).

   b. Annotate your service method.

      ```
      @RunWithDataRoles(
                  dataRoles = {"COMPENSATION_ANALYST"}
          )
          @Transactional(readOnly = true)
          public SalarySummaryDto getSalarySummary()
      ```

> **Note**
>
> The OOTB JDBC Spring Boot provider may not support custom or non-standard authentication and authorization workflows. If the provider doesn't meet your requirements, you can build a custom provider. See Advanced: Implement a Security Context Provider.

9.2.3 Configure Python Applications

The python-oracledb driver supports two approaches for propagating the end-user security context payload from your Python application to the database: API extension methods and a

configuration-driven Service Provider Interface (SPI). API calls take precedence over provider-based configurations.

Topics:

9.2.3.1 Use the API Extension Methods

If you want to supply the end-user security context payload directly in your application code, use the API extension methods.

The python-oracledb driver extends the standard Python Database API with the `Connection` methods to set and clear the end-user security context payload. Currently, you can use this feature only in python-oracledb's Thin mode.

To supply the security context payload using the python-oracledb driver, perform the following tasks.

1. Build the security context payload.

   In your application logic, use the `create_end_user_security_context` method to construct the security context object. You must provide both the end-user identity and the database-access token using the following supported values. Other parameters are optional, such as additional data roles to activate and end-user context attributes.

   - **End-user identity (`end_user_identity`):** Provide *one* of the following:
     - **`end_user_token`:** For users managed in IAM.
     - **`end_user_name` and `key`:** For users managed in the application's own user store. You can create the user name using the `create_end_user_pk` command, and key is the server-side string (also referred to as the *security context lookup key*).
   - **Database-access token (`database_access_token`):** Provide *one* of the following:
     - **On-behalf-of (OBO) token:** Obtained using the end-user token as an assertion.
     - **Client credentials token:** Obtained using the application's credentials.
   - **Data roles (`data_roles`):** Optional. Additional data roles that the application can enable for the end-user security context, beyond those mapped to application roles in IAM and those enabled by default for the application identity.
   - **End-user context attributes (`attributes`):** Optional. A dictionary of application-defined key-value pairs to include in the security context.

```
import oracledb
# ... authentication logic to retrieve tokens ...
data_roles = ["hr_role"]
attrs = {
    "emp_id": 3,
    "org_id": 5
}
ctx_attrs = {
        "hr.hcm_context": attrs
}
# Build the end-user security context
user_context = oracledb.create_end_user_security_context(
    end_user_identity = <user_token_or_local_end_user_name_and_key>,
```

```
    database_access_token = <db_token_issued_by_an_IAM>,
    data_roles = data_roles,   #optional
    attributes = ctx_attrs     #optional
)
```

2. Obtain a connection.

   You can obtain a connection using `oracledb.connect()` (standalone connection) or `pool.acquire()` (connection pool).

   ```
   connection = oracledb.connect(user="db_user", password=userpwd,
       dsn="orclpdb", config_dir="/opt/oracle/config",
       wallet_location="location_of_pem_file", wallet_password=walletpw)
   ```

3. Set and clear the security context payload on the connection.

   Add the security context payload to the connection before executing queries, and clear it afterward to return the connection to its original state. The security context payload is piggybacked on the next database operation.

   a. Set the security context payload.

      ```
      connection.set_end_user_security_context(user_context)
      ```

   b. Execute your SQL queries.

      ```
      with connection.cursor() as cursor:
          cursor.execute("select 1 from dual")
          row = cursor.fetchone()
          print(row)
      ```

   c. Clear the security context payload.

      As a best practice, clear the end-user security context payload explicitly before closing or releasing the connection. This ensures the next user does not inherit the previous user's privileges. If you do not call this method, the driver clears the security context payload only when the connection is closed with `connection.close()` or released back to the pool with `pool.release()`.

      ```
      connection.clear_end_user_security_context()
      connection.close()
      ```

> **Note**
>
> If no end-user identity is present, the plug-in does not attach the end-user security context payload, and the application receives a least-privileged connection or session (standard behavior) and no error is thrown.

For more information about the python-oracledb driver's support for Oracle Deep Data Security, see the python-oracledb documentation.

9.2.3.2 Use the Service Provider Interface

The Service Provider Interface (SPI) approach enables your Python application to supply the end-user security context payload with no code changes to your SQL or Object Relational Mapping (ORM) layer.

The python-oracledb driver's `end_user_sec_provider` plug-in acts as the SPI provider. By importing the plug-in and adding a configuration block to your application's database settings, you can enable your application to automatically propagate the security context payload on every database operation. The plug-in supports both the `client_credentials` and `on_behalf_of` (OBO) authentication flows for Microsoft Entra ID, and the `client_credentials` flow for OCI IAM.

To supply the security context payload using the SPI approach, perform the following tasks.

1. Install the driver.

   Ensure the python-oracledb driver is installed. The plug-in ships with the driver and uses the driver's Thin mode for security context payload propagation.

   ```
   python -m pip install oracledb
   ```

2. Import the Oracle Deep Data Security plug-in.

   Import the plug-in to register its parameter hook with the driver.

   ```
   import oracledb.plugins.end_user_sec_provider
   ```

3. Configure the Oracle Deep Data Security plug-in.

   Provide the identity provider settings in the `end_user_sec_params` configuration block. This block specifies the token provider type, authentication flow, IAM credentials, and optional data roles or context attributes. The location of this configuration block depends on your integration mode, which is detailed in the subsequent step.

   ```
   Example configuration:

   "end_user_sec_params": {
       "spi_type": "azure_tokens",
       "auth_flow": "client_credentials",
       "authority": "https://login.microsoftonline.com/<tenant_id>",
       "client_id": <client_id>,
       "client_credential": <secret>,
       "scopes": <scopes>,
       "data_roles": ["hr_role"],  #optional
       "attributes": {
           "hr.hcm_context": {"emp_id": 3, "org_id": 5}
       }
   }  #optional
   ```

Parameter	Description
`spi_type`	The name of the pre-supplied python-oracledb plug-in to use for database-access token acquisition. Valid values: `azure_tokens` (Microsoft Entra ID) or `oci_tokens` (OCI IAM).

Parameter	Description
`auth_flow`	The OAuth 2.0 grant type for acquiring the database-access token. Valid values: `client_credentials` or `on_behalf_of`. OCI IAM supports `client_credentials` only.
`authority`	The token endpoint authority URL. For example, for Entra ID, this is: `https://login.microsoftonline.com/<tenant_id>`.
`client_id`	The Application (Client) ID of the application registered in your IAM.
`client_credential`	The client secret for the application. Retrieve this value from an approved secret storage mechanism.
`scopes`	The scope required to access the database resource.
`data_roles`	(Optional) A list of additional data roles that the application can enable for the end-user security context.
`attributes`	(Optional) A dictionary of application-defined key-value pairs to include in the security context.

4. Configure the integration mode.

   For every request or execution context that interacts with the database, your application must supply an end-user identity to the plug-in. The identity payload is either an end-user token (for IAM-managed users) or a user name and key (for users managed in the application's own user store). How you supply this identity and where you place the `end_user_sec_params` configuration block depends on your application framework.

   If both the `end-user identity` and `end_user_sec_params` are present, database calls execute under a delegated end-user security context. If either is missing, database calls proceed without a delegated end-user security context, and the application receives a standard connection with default privileges.

   The following subsections describe three common integration patterns: Django, Flask, and other Python web frameworks. Pick the one that matches your application and adapt the example accordingly.

   a. **Django applications**

      Register the plug-in middleware in your `MIDDLEWARE` setting, and place the `end_user_sec_params` block inside your database configuration under `DATABASES['default']['OPTIONS']['extra_auth_params']` in the `settings.py` file. Your application is responsible for authenticating the end user and setting the end-user identity as a cookie named `identity`. The identity should be available on each request or execution. Its value is either the end-user token issued by your IAM system (for IAM-managed users) or a user name and key (for users managed in the application's own user store).

      The middleware reads this cookie from the request context (for example, `request.COOKIES["identity"]`) and makes it available to the plug-in for each database operation.

      Alternatively, after the end user authenticates with your application, your application can include the end user's bearer token in the `Authorization` header of the REST API call. The middleware then retrieves this token from the header (for example, `request.headers.get("Authorization")`).

      ```
      # settings.py
      import oracledb.plugins.end_user_sec_provider

      MIDDLEWARE = [
          'django.middleware.security.SecurityMiddleware',
          ...
      ```

```
    'oracledb.plugins.end_user_sec_provider.EndUserSecMiddleware',
]
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.oracle',
        'NAME': 'db_name',
        'USER': 'hr_db',
        'PASSWORD': '<hr_db connection pool user password>',
        'OPTIONS': {
            "extra_auth_params": {
                "end_user_sec_params": {
                    "spi_type": "azure_tokens",
                    "auth_flow": "client_credentials",
                    "authority": "https://login.microsoftonline.com/
<tenant_id>",
                    "client_id": <client_id>,
                    "client_credential": <secret>,
                    "scopes": <scopes>,
                    "data_roles": ["hr_role"],             # optional
                    "attributes": { "hr.hcm_context": {"emp_id": 3,
"org_id": 5}} # optional
                }
            }
        }
    }
}
```

b. Flask applications

Unlike Django, Flask does not use a middleware chain. So, python-oracledb's Oracle Deep Data Security plug-in is wired up inside the view function for each request. In each handler that talks to the database, extract the end-user identity from the incoming request, pass it to `set_end_user_identity()`, and then call `oracledb.connect()` with the `end_user_sec_params` block supplied as `extra_auth_params`.

The end-user identity is typically an OAuth 2.0 access token carried in the `Authorization` request header when the application receives the token from an upstream caller, or a token the Flask application acquires directly using a library such as MSAL when the application authenticates the end user itself. Either way, the token becomes the value passed to `set_end_user_identity()`.

```
from flask import Flask, request, jsonify
import oracledb
import oracledb.plugins.end_user_sec_provider as deep_data_sec_provider

app = Flask(__name__)

@app.route("/myapi", methods=["GET"])
def myfun():
    token = #get token from request headers if frontend passes token in
authorization headers or
            #get token from msal if frontend is not there

    deep_data_sec_provider.set_end_user_identity(token)

    conn = oracledb.connect(
                                user="hr_db",
```

```
                                    password="<database password>",
                                    dsn="<your connect string>",
                                    wallet_location="path to wallet file",
                                    wallet_password="<wallet password>",
                                    extra_auth_params={
                                        "end_user_sec_params": {
                                            "spi_type": 'azure_tokens',
                                            "auth_flow": 'client_credentials',
                                            "client_id":"<client_id>",
                                            "client_credential":"<secret>",
                                            "authority":"https://
login.microsoftonline.com/<tenant_id>",
                                            "scopes":"<scopes>",
                                            "data_roles": ["hr_role"],
# optional
                                            "attributes": { "hr.hcm_context":
{"emp_id": 3, "org_id": 5}}   # optional
                                        }
                                    }
                                )

    data = mygetdata(conn)

    return jsonify(data), 200
```

The plug-in reads the identity set by `set_end_user_identity()`, acquires the database-access token using the configured `end_user_sec_params`, and attaches the end-user security context payload to every database call made on that connection.

c. Other web frameworks

For other Python web frameworks, such as FastAPI, apply the same pattern as the Flask example above: set the end-user identity for the current execution context with `set_end_user_identity()` before acquiring a connection, and pass the `end_user_sec_params` block in the `extra_auth_params` parameter of your connection or pool-creation call. The hook point (middleware, dependency, before-request handler) depends on your framework; the identity-and-configuration handoff to the plug-in is the same.

The following example shows this pattern applied to a FastAPI application.

```
import oracledb
import oracledb.plugins.end_user_sec_provider as deepsec_provider
from fastapi import FastAPI, Header

app = FastAPI()

@app.get("/myapi")
def myfun(authorization: Optional[str] = Header(default="")):
    token = #get token from request headers if frontend passes token in
authorization headers

    deepsec_provider.set_end_user_identity(token)
    conn = oracledb.connect(
                                    user="hr_db",
                                    password="<database password>",
                                    dsn="<your connect string>",
```

```
                                    wallet_location="path to wallet file",
                                    wallet_password="<wallet password>",
                                    extra_auth_params={
                                        "end_user_sec_params": {
                                            "spi_type": 'azure_tokens',
                                            "auth_flow": 'client_credentials',
                                            "client_id":"<client_id>",
                                            "client_credential":"<secret>",
                                            "authority":"https://
login.microsoftonline.com/<tenant_id>",
                                            "scopes":"<scopes>"
                                        }
                                    }
                                )

    data = mygetdata(conn)
    return data
```

The plug-in acquires the database-access token and attaches the security context payload automatically.

For more information about the python-oracledb driver's support for Oracle Deep Data Security, see the python-oracledb documentation.

9.2.4 Configure .NET Applications

The ODP.NET driver supports Oracle Deep Data Security (Deep Sec) by propagating the end-user security context payload from your .NET application to the database.

For details, see the Oracle Data Provider for .NET documentation.

9.2.5 Advanced: Implement a Security Context Provider

If your authentication and authorization workflow requires custom logic, or if there's no out-of-the-box (OOTB) security context provider available for your application framework, you can build your own provider.

A custom provider bridges your application's security context and the Oracle client driver, enabling automatic propagation of the end-user security context payload to the database without modifying your application code.

The instructions in this section are primarily for the Oracle JDBC driver and Java-based application frameworks. If you are using a different client driver (such as python-oracledb or ODP.NET), apply the same principles using the equivalent provider interface and registration mechanism in your driver. See the respective driver documentation for API details.

Topics:

- Note for Other Client Drivers

9.2.5.1 Understand the Security Context Payload

Before building a provider, understand the components of the end-user security context payload that the database expects. Every payload must include the following mandatory components and may include optional components.

- **End-user identity (mandatory):** The end user's name as asserted in the IAM access token. For end users managed locally, this is the name of the end user created in the database.
- **Database access token (mandatory):** Authorizes the application's connection to the database. Obtain this token from your IAM system using one of the following flows:
  - *Client credentials flow:* The application authenticates as itself.
  - *On-behalf-of (OBO) flow:* The application exchanges the end-user's token for a database-access token.
- **Data roles (optional):** A list of additional data roles that the application can enable for the end-user security context, beyond those mapped to application roles in IAM and those enabled by default for the application identity.
- **Context attributes (optional):** A dictionary of application-defined key-value pairs to include in the security context. Used when application logic or data grants rely on custom end-user context attributes.

9.2.5.2 Understand the Driver's Provider Interface

To propagate the end-user security context payload, the Oracle JDBC driver defines a Service Provider Interface (SPI) named `EndUserSecurityContextProvider`. Your custom provider must implement this interface and its `getEndUserSecurityContext()` method.

The JDBC driver invokes `getEndUserSecurityContext()` before every database operation on the connection. Your implementation must return an `EndUserSecurityContext` object containing the end-user identity, database-access token, and any optional data roles or attributes. The driver piggybacks this payload to the database on the next round-trip. If the method returns null, the driver proceeds without attaching a security context payload.

> **Note**
>
> For other client drivers, use the equivalent provider interface. For example, the python-oracledb driver's `end_user_sec_provider` plug-in uses a configuration-driven mechanism with middleware hooks. See the respective driver documentation for the specific interface contract.

9.2.5.3 Build the Security Context Provider

Your provider implementation must perform three tasks each time the driver calls `getEndUserSecurityContext()`. The following steps describe these tasks for a JDBC provider. Adapt the approach to your client driver as needed.

1. **Extract the end-user identity from your framework's security context.**

Every application framework maintains a request-scoped or thread-local security context that holds the authenticated user's identity after the framework's authentication filter has processed the request. Your provider must read the end-user identity from this context.

Examples of framework security context mechanisms:

- **Jakarta EE / Servlet containers:** `HttpServletRequest.getUserPrincipal()` or the JAX-RS `SecurityContext`
- **Helidon:** `io.helidon.security.SecurityContext`

Other frameworks provide equivalent mechanisms. Consult your framework's security documentation.

Extract either the raw OAuth 2.0 access token (for IAM-managed users) or the user name and lookup key (for users managed in the application's own user store). If your framework stores the token as a parsed object (such as a JWT), you may need to retrieve the original serialized token string as the database expects the raw token.

2. **Acquire the database-access token from your IAM.**

   Your provider must obtain a token that authorizes the application to access the database resource. This token is separate from the end user's token.

   Use an OAuth 2.0 client library available in your framework or ecosystem to request the token from your IAM's token endpoint. The specific library depends on your environment:

   - **Microsoft Entra ID:** Use MSAL for Java (`com.microsoft.azure:msal4j`) for both client credentials and OBO flows.
   - **OCI IAM:** Use the OCI SDK for Java or a standard HTTP client to call the OCI IAM token endpoint with client credentials.

   When requesting the token, use the client ID, client secret, and scopes from your application's IAM registration.

   Database-access tokens are typically valid for a limited duration (for example, one hour). Cache the token and reuse it until it nears expiry to avoid requesting a new token on every database call. For OBO tokens, use the end-user token as the cache key, because each user produces a distinct OBO token.

3. **Construct and return the `EndUserSecurityContext`.**

   Use the JDBC driver's builder methods to assemble the security context object from the components gathered in previous steps.

   For IAM-managed users (token-based identity):

```
EndUserSecurityContext securityContext =
    EndUserSecurityContext.createWithToken(
        databaseAccessToken, endUserToken)
      .withDataRoles(Set.of(dataRoles))               // optional
      .withAttributes(contextAttributes);             // optional
return securityContext;
```

   For users managed in the application's own user store (user name and lookup key):

```
EndUserSecurityContext securityContext =
    EndUserSecurityContext.createWithUsername(
        databaseAccessToken, endUserName, key)
      .withAttributes(contextAttributes);             // optional
return securityContext;
```

If no end-user identity is available for the current request (for example, a health-check endpoint that does not authenticate users), return `null`. The driver proceeds without attaching a security context payload, and the connection operates with its default privileges.

9.2.5.4 Register the Provider with the Driver

The JDBC driver discovers your provider through the standard Java `ServiceLoader` mechanism.

To register your provider:

1. Create a file named `oracle.jdbc.spi.EndUserSecurityContextProvider` in your provider JAR's `META-INF/services/` directory.
2. Add a single line containing the fully qualified class name of your provider implementation. For example:

```
# META-INF/services/oracle.jdbc.spi.EndUserSecurityContextProvider
com.example.myapp.MySecurityContextProvider
```

When a new JDBC connection is created (that is, when `java.sql.Driver.connect(String, Properties)` is called), the driver scans the class path for implementations of `EndUserSecurityContextProvider` and loads the one matching the provider name configured in the connection properties.

9.2.5.5 Configure the Driver to Use Your Provider

Set the `oracle.jdbc.provider.endUserSecurityContext` connection property to the name of your provider.

This name is the value returned by your provider's `getName()` method. Set this property in your connection pool's data source configuration.

For example, in a HikariCP-based configuration:

```
# Activate the custom provider
spring.datasource.hikari.data-source-properties.\
  oracle.jdbc.provider.endUserSecurityContext = my-custom-provider
```

Or programmatically when configuring the data source:

```
Properties props = new Properties();
props.setProperty(
    "oracle.jdbc.provider.endUserSecurityContext",
    "my-custom-provider"
);
dataSource.setDataSourceProperties(props);
```

9.2.5.6 Support Privilege Elevation (Optional)

Some application operations require temporary elevated privileges. For example, reading all employees' salary data to generate a summary report. To support this, your provider can accept additional data roles injected at runtime for the duration of a specific code block.

The recommended approach is to pass elevated data roles through your framework's existing security context mechanism, keeping the provider decoupled from application code. The OOTB Spring Boot provider uses this pattern:

- Application code temporarily adds data roles (with a distinguishing prefix) to the framework's security context before the database call.
- The provider reads the prefixed roles from the security context and includes them in the `EndUserSecurityContext` payload.
- After the method returns, the application restores the original security context.

To implement this pattern in your framework, use the equivalent of a method interceptor or decorator that wraps the target method, augments the security context with additional data roles, invokes the method, and restores the original context in a `finally` block.

For the Spring Boot reference implementation of this pattern (using `GrantedAuthority` and the `@RunWithDataRoles` annotation), see the Oracle JDBC Extensions source code on GitHub.

9.2.5.7 Note for Other Client Drivers

If your application uses python-oracledb or ODP.NET instead of JDBC, the same architectural pattern applies: implement a component that extracts the end-user identity, acquires the database-access token, constructs the security context payload, and hooks into the driver's connection life cycle. The specific interfaces, registration mechanisms, and configuration properties differ by driver.

For the python-oracledb driver, see the python-oracledb documentation.

For the ODP.NET driver, see the Oracle Data Provider for .NET documentation.

9.3 Configure a SQL Client for Interactive Logon (Direct Logon)

If your IAM-managed users connect directly to the database using a SQL client (such as SQL*Plus) and browser-based OAuth sign-in, configure the client machine for token-based authentication.

After an end user successfully authenticates, the database automatically establishes an end-user security context for the user based on the identity provider configuration (see Set Up IAM Integration for Direct Logon) and the `roles` or `groups` claim present in the end-user token.

For `sqlnet.ora` authentication parameters and `tnsnames.ora` connection alias configuration, see the following sections in *Oracle AI Database Security Guide*:

- **For Microsoft Entra ID:** Enabling Clients to Directly Retrieve Entra ID Tokens.
- **For OCI IAM:** Configuring a Client Connection for SQL*Plus That Uses an IAM Token.

> **Note**
>
> You must configure the database server for TLS before configuring the client. The client-side trust store requires the server certificate to establish trust. For TLS configuration, see Configuring Transport Layer Security Encryption in *Oracle AI Database Security Guide*.

Part IV
User and Role Administration

Learn how to administer application users, application identities, and data roles in Oracle Deep Data Security (Deep Sec). Each chapter covers a functional area and documents the SQL grammar, parameters, usage guidelines, and examples.

Deep Sec separates application users and data roles from traditional database users and roles. Unlike traditional database user accounts, which own schemas and often grant excessive privileges when shared across users, Deep Sec uses a decoupled framework. This allows you to enforce fine-grained data authorization without granting end users direct ownership of database objects.

You can manage application users and roles in two ways:

- **Externally:** Use an identity and access management (IAM) system to map data roles in the database to application roles in IAM.
- **Locally:** Manage users and roles directly within the database. This method is ideal for simple applications, demonstrations, or development and testing environments.

The primary entities used in Deep Sec user and role administration are as follows:

- **End user**

  A user of an application who does not own database schemas or database objects. An end user can be:

  – A user whose identity is managed in IAM. Typically, they connect to the database through the application. They can also log in directly using token-based authentication.

  – A user created and managed in the database using the `CREATE END USER` statement. End users managed in the database can log in directly using password authentication. Additionally, they can be mapped by user name to users managed in the application's own user store, so that those users can connect through the application's trust, without requiring direct password authentication on the database server.

- **Application identity:**

  A registered identity for an application. Enables data roles and common privileges for all users connecting through that application.

- **Data role**

  A role in the database used for fine-grained access to data. You can grant data privileges (through data grants) and standard database roles to a data role. A data role can be:

  – Mapped to an application role in IAM using the `MAPPED TO` clause. The database automatically enables it when the end user’s token includes the corresponding role claim.

  – Managed locally in the database. A data role managed locally in the database can be granted to end users, application identities, or other data roles (that are managed locally).

Topics:

- Configure Application Identities
- Configure Data Roles
- Grant and Revoke Data Roles

> **See Also**
>
> User and Role Management in the Database in *Part I: Oracle Deep Data Security Fundamentals*.

10

Configure Local End Users

Use the SQL statements in this chapter to create, alter, and drop local end users.

A local end user is an application user created and managed directly in the database. Unlike traditional database users, these end users do not own schemas or database objects. However, they can log in directly using password authentication and receive fine-grained data grants.

Additionally, you can map local end users by name to users managed in the application's own user store. This allows the application to connect on behalf of the end user through proxy or trust, without requiring direct password authentication on the database server.

Topics:

10.1 Create End User

Use the `CREATE END USER` command to create a local end user in the database.

You can configure the local end user with a password for direct database login, associate it with a database schema for name resolution, and assign an account validity period. Query the `DBA_END_USERS` data dictionary view to review existing local end users.

Required privilege

The `CREATE END USER` system privilege.

Syntax

```
CREATE END USER [ IF NOT EXISTS ] end_user
  [ IDENTIFIED BY password ]
  [ PROFILE profile ]
  [ PASSWORD EXPIRE ]
  [ ACCOUNT { LOCK | UNLOCK } ]
  [ SCHEMA schema ]
  [ START TIME timestamp ] [ END TIME timestamp ];
```

Parameters

Parameter	Description
`end_user`	The name of the end user to be created.

Parameter	Description
password	The password for the end user, enabling direct database login. The password uses a SHA-512 verifier, and the Oracle client must be compatible with Oracle Database 12c or later. If not specified, the authentication type is set to NONE.
profile	The name of the Oracle profile to assign. Profiles limit database resource usage. If omitted, the DEFAULT profile is assigned.
PASSWORD EXPIRE	A setting that expires the end user's password immediately, forcing the user or administrator to set a new password before the next login.
ACCOUNT LOCK	A setting that locks the account and disables access.
ACCOUNT UNLOCK	A setting that unlocks the account and enables access.
schema	The existing database schema to associate with this end user for name resolution. Optional. If not specified, no schema is associated.
START TIME	The time stamp from which the end-user account becomes effective, in TIMESTAMP WITH TIME ZONE format. Optional.
END TIME	The time stamp on which the end-user account becomes ineffective, in TIMESTAMP WITH TIME ZONE format. Optional.

Usage notes and restrictions

- IF NOT EXISTS behavior: If the end user does not exist, it is created. If it already exists, an error is raised unless IF NOT EXISTS is specified, in which case the statement is a no-op.
- When specifying time values with TO_TIMESTAMP, Oracle uses the session time zone. To specify a different time zone, use TO_TIMESTAMP_TZ.

For syntax diagrams and additional details, see CREATE END USER in *Oracle AI Database SQL Language Reference*.

Example 10-1 Create end user with password

Create a local end user named Emma who can log in with a password, effective from 2025-03-01 19:30:00 UTC:

```
CREATE END USER emma
  IDENTIFIED BY <password>
  START TIME TO_TIMESTAMP_TZ('2025-03-01 19:30:00 +00:00',
                             'YYYY-MM-DD HH24:MI:SS TZH:TZM');
```

10.2 Alter End User

Use the ALTER END USER command to modify the properties of an existing local end user.

You can use this command to update the password, schema association, account validity period, profile assignment, and account lock status.

A local end user can change their own password using the ALTER END USER command. When the user does not hold the ALTER END USER system privilege, they must provide the old password. A local end user can also use the PASSWORD command to change their password.

Required privilege

The `ALTER END USER` system privilege, except when a local end user is changing their own password.

Syntax

```
ALTER END USER [ IF EXISTS ] end_user
  [ IDENTIFIED BY password [ REPLACE old_password ] ]
  [ PROFILE profile ]
  [ PASSWORD EXPIRE ]
  [ ACCOUNT { LOCK | UNLOCK } ]
  [ { SCHEMA schema | NO SCHEMA } ]
  [ { START TIME timestamp | NO START TIME } ]
  [ { END TIME timestamp | NO END TIME } ];
```

Parameters

Parameter	Description
`end_user`	The name of the end user to be altered.
`password`	The new password. If not specified, the current password remains unchanged.
`old_password`	The old password. Required when a local end user who does not hold the `ALTER END USER` privilege is changing their own password. Validated before the new password is set; failed attempts increment the login failure counter (ORA-28008).
`profile`	The new profile to assign. If not specified, the current profile is retained.
`PASSWORD EXPIRE`	A setting that expires the current password, forcing a reset at next login.
`ACCOUNT LOCK`	A setting that locks the account and disables access.
`ACCOUNT UNLOCK`	A setting that unlocks the account and enables access.
`SCHEMA schema`	A setting that associates the end user with the specified database schema. If not specified, the current association is retained.
`NO SCHEMA`	A setting that removes the current schema association.
`START TIME`	A setting that sets or replaces the account effective time stamp (`TIMESTAMP WITH TIME ZONE` format).
`NO START TIME`	A setting that removes the configured start time.
`END TIME`	A setting that sets or replaces the account expiry time stamp (`TIMESTAMP WITH TIME ZONE` format).
`NO END TIME`	A setting that removes the configured end time.

Usage notes and restrictions

- When `IF EXISTS` is specified:
  - If the end user does not exist, the statement is a no-op. No error is raised.
  - If the end user exists, it is altered.
- When `IF EXISTS` is omitted:
  - If the end user does not exist, an error is raised.

  - If the end user exists, it is altered.
- An end user who holds the `ALTER END USER` privilege can change passwords for other end users without providing the old password. If an incorrect old password is provided, the change still succeeds.

For syntax diagrams and additional details, see ALTER END USER in *Oracle AI Database SQL Language Reference*.

Example 10-2 Unlock account

Unlock the account for end user Emma and associate her with the HR schema.

```
ALTER END USER emma ACCOUNT UNLOCK SCHEMA hr;
```

10.3 Drop End User

Use the `DROP END USER` command to remove an existing local end user from the database.

Required privilege

The `DROP END USER` system privilege.

Syntax

```
DROP END USER [ IF EXISTS ] end_user;
```

Parameters

Parameter	Description
`end_user`	The name of the end user to be dropped.

Usage notes and restrictions

- When `IF EXISTS` is specified:
  - If the end user does not exist, the statement is a no-op. No error is raised.
  - If the end user exists, it is dropped.
- When `IF EXISTS` is omitted:
  - If the end user does not exist, an error is raised.
  - If the end user exists, it is dropped.
- You cannot drop an end user who has an active session.

For syntax diagrams and additional details, see DROP END USER in *Oracle AI Database SQL Language Reference*.

Example 10-3 Drop end user

Drop an end user, Emma.

```
DROP END USER emma;
```

11

Configure Application Identities

Use the SQL statements in this chapter to create and drop application identities.

An application identity represents an application in the database. Although application users do not require their own database identities, some use cases require the application itself to have a registered identity in the database. This allows the database to ensure that users connecting through the application can enable only the data roles and end-user contexts that belong to that specific application.

Query the `DBA_APPLICATION_IDENTITIES` data dictionary view to see existing application identities.

Topics:

11.1 Create Application Identity

Use the `CREATE APPLICATION IDENTITY` command to register an application as an identity in the database. After creation, you can grant data roles (that are managed locally in the database) to the application identity so that all sessions associated with that application can enable those roles.

Required privilege

The `CREATE APPLICATION IDENTITY` system privilege.

Syntax

```
CREATE [ OR REPLACE ] APPLICATION IDENTITY
  [ IF NOT EXISTS ] app_identity
  MAPPED TO 'identifier_string';
```

Parameters

Parameter	Description
`app_identity`	The name of the application identity to be created.
`identifier_string`	The identifier string for the external identity provider mapping. Supported prefixes are: `AZURE_CLIENT_ID=<id>` for Microsoft Entra ID, or `IAM_OAUTH_CLIENT_ID=<id>` for Oracle Cloud Infrastructure Identity and Access Management (OCI IAM). The `identifier_string` parameter must be fewer than 1024 characters. The database raises `ORA-28303` if this limit is exceeded.

Usage notes and restrictions

- When `OR REPLACE` is specified:

  - If the application identity already exists, the `identifier_string` is replaced.
  - If the application identity does not exist, it is created.
- When `OR REPLACE` is omitted:
  - If the application identity already exists, an error is raised.
  - If the application identity does not exist, it is created.
- When `IF NOT EXISTS` is specified:
  - If the application identity already exists, the statement is a no-op. No error is raised.
  - If the application identity does not exist, it is created.
- When `IF NOT EXISTS` is omitted:
  - If the application identity already exists, an error is raised.
  - If the application identity does not exist, it is created.
- `OR REPLACE` and `IF NOT EXISTS` are mutually exclusive in the same statement. Using both raises the ORA-11541 error.
- Each application identity must have a unique identifier string. Creating a new identity with an identifier string already belonging to another application identity raises an error. The identifier string comparison is case-insensitive.
- The same application cannot have more than one application identity mapping in the database.

For syntax diagrams and additional details, see CREATE APPLICATION IDENTITY in *Oracle AI Database SQL Language Reference*.

Example 11-1 Create an application identity using Microsoft Entra ID

Create an application identity for an HCM application using the `app_id` (v1 tokens) or `azp` (v2 tokens) claim from the Entra ID token.

```
CREATE APPLICATION IDENTITY hcm_app
  MAPPED TO 'AZURE_CLIENT_ID=2edc9c9f-8e1e-4ade-8a4a-cc286ed1b899';
```

Example 11-2 Create an application identity using OCI IAM

Create an application identity for an HCM application using the `client_id` claim from the OCI IAM token.

```
CREATE APPLICATION IDENTITY hcm_app
  MAPPED TO 'IAM_OAUTH_CLIENT_ID=e83a43ac80d94637bb1958b06929ac32';
```

Example 11-3 Grant data role to application identity

Grant a data role `hcm_role` that is managed locally to the application identity, so the HCM application can enable it during its sessions for all application users.

```
GRANT DATA ROLE hcm_role TO hcm_app;
```

11.2 Drop Application Identity

Use the `DROP APPLICATION IDENTITY` command to remove an application identity from the database.

Required privilege

The `DROP APPLICATION IDENTITY` system privilege.

Syntax

```
DROP APPLICATION IDENTITY [ IF EXISTS ] app_identity;
```

Parameters

Parameter	Description
`app_identity`	The name of the application identity to be dropped.

Usage notes and restrictions

- When `IF EXISTS` is specified:
  - If the application identity does not exist, the statement is a no-op. No error is raised.
  - If the application identity exists, it is dropped.
- When `IF EXISTS` is omitted:
  - If the application identity does not exist, an error is raised.
  - If the application identity exists, it is dropped.

For syntax diagrams and additional details, see DROP APPLICATION IDENTITY in *Oracle AI Database SQL Language Reference*.

Example 11-4 Drop application identity

Drop the application identity for the HCM application.

```
DROP APPLICATION IDENTITY hcm_app;
```

12

Configure Data Roles

Use the SQL statements in this chapter to create and drop data roles.

You define a data role in the database specifically for fine-grained data grants. Oracle Deep Data Security (Deep Sec) supports two types of data roles:

- **Data roles that are externally mapped:** Database representations of external IAM roles (created using the `MAPPED TO` clause). The database automatically enables these roles based on the user's token claims when an end-user security context is established.
- **Data roles that are locally managed:** Data roles created and managed entirely within the database. You can grant these to local end users, application identities, or other data roles (that are managed locally in the database). They do not map to external IAM roles.

Query the `DBA_DATA_ROLES` data dictionary view to review existing data roles and their properties.

Topics:

12.1 Create Data Role

Use the `CREATE DATA ROLE` command to create a data role in the database.

Required privilege

The `CREATE DATA ROLE` system privilege.

Syntax

```
CREATE [ OR REPLACE ] DATA ROLE [ IF NOT EXISTS ] data_role_name
  { external_role_clause | local_role_clause };

external_role_clause ::=
  MAPPED TO 'identifier_string'

local_role_clause ::=
  [ { ENABLED | DISABLED } ]
```

Parameters

Parameter	Description
`data_role_name`	The name of the data role to be created.

Parameter	Description
`identifier_string`	The external provider mapping string (for externally mapped roles only). The supported formats are: • `AZURE_ROLE=<role_name>` or `AZURE_APP=<aud>:AZURE_ROLE=<role_name>` for Microsoft Entra ID. • `IAM_OAUTH_GROUP=<group_name>` for Oracle Cloud Infrastructure Identity and Access Management (OCI IAM). The `identifier_string` parameter must be fewer than 1024 characters. The database raises `ORA-28303` if this limit is exceeded.
`ENABLED | DISABLED`	The option that specifies whether a data role (that is managed locally in the database) is enabled or disabled on creation. The default value is `ENABLED` if this option is omitted. After a data role is granted to an application identity, you cannot set it to the `ENABLED` or `DISABLED` state.

Usage notes and restrictions

- When `IF NOT EXISTS` is specified:
  - If the data role exists, the statement is a no-op. No error is raised.
  - If the data role does not exist, it is created.
- When `IF NOT EXISTS` is omitted:
  - If the data role already exists, an error is raised.
  - If the data role does not exist, it is created.
- When `OR REPLACE` is specified:
  - If the data role already exists, the options are replaced.
  - If the data role does not exist, it is created.
- When `OR REPLACE` is omitted:
  - If the data role already exists, an error is raised.
  - If the data role does not exist, it is created.
- `OR REPLACE` and `IF NOT EXISTS` are mutually exclusive. Using both in the same statement raises the ORA-11541 error.
- A data role can map to at most one external application role. Attempting to create a data role with an external identifier string already in use raises an error. The identifier string comparison is case-insensitive.
- You cannot replace a data role that is externally mapped with a data role that is locally managed, or vice versa. The database raises an error if you attempt such a replacement.
- After you grant a data role that is managed locally to an application identity, only that application can enable it. You cannot grant that role to other end users or data roles. After a data role is granted to an application identity, you cannot set it to the `ENABLED` or `DISABLED` state. The database raises `ORA-52539` if you attempt to do so.

For syntax diagrams and additional details, see CREATE DATA ROLE in *Oracle AI Database SQL Language Reference*.

Example 12-1 Create a data role for Microsoft Entra ID role

Create a data role `employee_role` that maps to a Microsoft Entra ID role called `employee`.

For Microsoft Entra ID, the mapping must use one of the following formats.

- Using the role name only:

```
CREATE DATA ROLE employee_role MAPPED TO 'AZURE_ROLE=employee';
```

- Using the application audience and role name:

```
CREATE DATA ROLE employee_role MAPPED TO
  'AZURE_APP=https://supremo.onmicrosoft.com/2edc9c9f-8e1e-4ade-8a4a-
cc286ed1b899:AZURE_ROLE=employee';
```

The database first checks for the `AZURE_APP` prefix. If it is not present, the database matches using `AZURE_ROLE` alone. The optional `AZURE_APP` prefix uses the end-user token's audience (`aud`) claim to distinguish between application roles that share the same name across different application registrations.

Example 12-2 Create a data role for OCI IAM group

Create a data role `employee_role` for an OCI IAM group called `employee`.

```
CREATE DATA ROLE employee_role MAPPED TO 'IAM_OAUTH_GROUP=employee';
```

An application identifier prefix is not required for OCI IAM because groups are shared across applications within the same identity domain, and the database can only be configured with a single domain.

Example 12-3 Create data role that is managed locally in the database

Create a data role, `it_support_role`, with default settings.

```
CREATE DATA ROLE it_support_role;
```

Example 12-4 Create a data role as disabled

Create a data role (that is managed locally in the database) in the disabled state.

```
CREATE DATA ROLE temp_role DISABLED;
```

12.2 Drop Data Role

Use the `DROP DATA ROLE` command to remove a data role from the database.

Required privilege

The `DROP DATA ROLE` system privilege.

Syntax

```
DROP DATA ROLE [ IF EXISTS ] data_role;
```

Parameters

Parameter	Description
`data_role`	The name of the data role to be dropped.

Usage notes and restrictions

- When `IF EXISTS` is specified:
  - If the data role does not exist, the statement is a no-op. No error is raised.
  - If the data role exists, it is dropped.
- When `IF EXISTS` is omitted:
  - If the data role does not exist, an error is raised.
  - If the data role exists, it is dropped.

For syntax diagrams and additional details, see DROP DATA ROLE in *Oracle AI Database SQL Language Reference*.

Example 12-5 Drop data role

Drop an existing data role, `employee_role`.

```
DROP DATA ROLE employee_role;
```

13

Grant and Revoke Data Roles

Use the SQL statements in this chapter to grant and revoke data roles.

Granting a data role allows specific end users, application identities, or other data roles to use its associated fine-grained data privileges. Revoking a data role removes this access.

Before you grant data roles, review the role-grant restrictions detailed in the following table.

Type of Role	Can Be Granted To
Database role	Data role
Data role that is managed locally in the database	• Local end user • Another data role that is managed locally in the database • Application identity
Data role that is mapped to an application role in IAM	None You cannot grant data roles that are mapped to external application roles to local end users or data roles. Instead, you must enable these data roles for end users through IAM.

Query the `DBA_DATA_ROLE_GRANTS` data dictionary view to review existing data role grants, including their start and end times.

Topics:

13.1 Grant Data Role

Use the `GRANT DATA ROLE` command to grant one or more data roles that are managed locally in the database to local end users, other data roles (that are managed locally), or application identities.

The `GRANT DATA ROLE` command supports two distinct use cases:

- Granting a data role to a local end user or another data role (managed locally). The grantee inherits the privileges associated with the granted data role.
- Granting a data role to an application identity. The application can then enable the data role in its database session.

Required privilege

The `GRANT ANY DATA ROLE` system privilege.

Syntax

```
GRANT DATA ROLE [ IF EXISTS ] data_role_list
  TO grantee_list
  [ START TIME timestamp ] [ END TIME timestamp ];

data_role_list ::= data_role [, data_role ]...

grantee_list  ::= grantee [, grantee ]...

grantee       ::= { end_user | application_identity | data_role }
```

Parameters

Parameter	Description
`data_role_list`	A comma-separated list of data roles to be granted. Only data roles that are managed locally in the database are permitted; data roles that are externally mapped cannot be granted through this command.
`grantee_list`	A comma-separated list of grantees. Each grantee can be a local end user, an application identity, or a data role that is managed locally.
`START TIME`	The time stamp from which the grant becomes effective, in `TIMESTAMP WITH TIME ZONE` format. Optional. Not applicable when granting to an application identity.
`END TIME`	The time stamp on which the grant becomes ineffective, in `TIMESTAMP WITH TIME ZONE` format. Optional. Not applicable when granting to an application identity.

Usage notes and restrictions

- When `IF EXISTS` is specified:
  - If any data role or grantee in the statement does not exist, the grant is skipped for those non-existing entries and no error is raised.
  - If any data role grant already exists, its effective dates are updated.
- When `IF EXISTS` is omitted:
  - If any data role or grantee in the statement does not exist, an error is raised.
  - If the data role grant already exists, its effective dates are updated.
- A single `GRANT DATA ROLE` statement cannot mix application identities and local end users or data roles in the same grantee list. If a mixture is specified, an error is raised. This restriction exists because data roles granted to application identities operate differently from those granted to local end users or data roles.
- A data role granted to a local end user or data role cannot be granted to an application identity, and vice versa.
- `START TIME` and `END TIME` do not apply when granting a data role to an application identity. Application-managed data roles are enabled and disabled dynamically by the application at runtime.

For syntax diagrams and additional details, see GRANT DATA ROLE in *Oracle AI Database SQL Language Reference*.

Example 13-1 Grant data role to end user

Grant a data role `manager_role` to a local end user, Marvin.

```
GRANT DATA ROLE manager_role TO marvin;
```

Example 13-2 Grant one data role to another

Grant a data role `employee_role` to another data role `manager_role`, so `manager_role` inherits the privileges of `employee_role`.

```
GRANT DATA ROLE employee_role TO manager_role;
```

Example 13-3 Grant multiple data roles to multiple end users

Grant multiple data roles to multiple local end users in a single statement.

```
GRANT DATA ROLE hr_rep_role, it_support_role
  TO emma, jdoe;
```

Example 13-4 Grant data role to application identity

Grant a data role to an application identity, `hcm_app`.

```
GRANT DATA ROLE hcm_role TO hcm_app;
```

13.2 Revoke Data Role

Use the `REVOKE DATA ROLE` command to remove one or more data role grants from specified grantees (revokees).

Required privilege

The `GRANT ANY DATA ROLE` system privilege.

Syntax

```
REVOKE DATA ROLE [ IF EXISTS ] data_role_list
  FROM revokee_list;

data_role_list ::= data_role [, data_role ]...

revokee_list   ::= revokee [, revokee ]...

revokee        ::= { end_user | application_identity | data_role }
```

Parameters

Parameter	Description
`data_role_list`	A comma-separated list of data roles to be revoked. Only data roles that are managed locally in the database are permitted.

Parameter	Description
`revokee_list`	A comma-separated list of local end users, application identities, or other data roles from which the data roles are to be revoked.

Usage notes and restrictions

- When `IF EXISTS` is specified:
  - If any data role or revokee does not exist, the revoke is skipped for those non-existing entries and no error is raised.
  - If the data role exists, it is revoked.
- When `IF EXISTS` is omitted:
  - If any data role or revokee does not exist, an error is raised.
  - If the data role exists, it is revoked.

For syntax diagrams and additional details, see REVOKE DATA ROLE in *Oracle AI Database SQL Language Reference*.

Example 13-5 Revoke a data role

Revoke a data role, `manager_role`, from a local end user, Marvin.

```
REVOKE DATA ROLE manager_role FROM marvin;
```

13.3 Grant Database Role to Data Role

Use the standard `GRANT` statement to grant one or more database roles to data roles. This allows data roles to carry traditional Oracle AI Database privileges in addition to Oracle Deep Data Security (Deep Sec) data grants.

> **Note**
>
> In the Deep Sec framework, you cannot grant database roles directly to local end users. Instead, you must first assign them to data roles that are managed locally in the database. You can then grant those data roles to local end users, allowing them to inherit the combined privileges of both the data role and the underlying database role.

Syntax

```
GRANT database_role_list
  TO grantee_list;

database_role_list ::= database_role [, database_role ]...

grantee_list       ::= data_role [, data_role ]...
```

Parameters

Parameter	Description
`database_role_list`	A comma-separated list of existing database roles to be granted.
`grantee_list`	A comma-separated list of data roles to which the database roles are granted. Only data roles are permitted as grantees in this context.

Example 13-6 Grant database role to data role

Grant a database role `select_catalog_role` to a data role `hr_rep_role`.

```
GRANT select_catalog_role TO hr_rep_role;
```

Part V

Data Access Control Configuration

Learn how to configure data access control in Oracle Deep Data Security (Deep Sec). This part of the guide explains how to establish end-user security contexts, define and manage end-user context attributes, create data grants for fine-grained access control, and enforce mandatory data privileges.

Topics:

14

Configure the Security Context

The end-user security context is the mechanism through which the database enforces per-user security policies at runtime. When an end-user security context is active, it replaces the security domain of the shared connection pool user account with the individual end user's identity, roles, and context attributes, ensuring that every SQL operation is authorized against the actual requesting user, and not the application's database user account.

Learn how an end-user security context is established in Oracle AI Database with the integrated Oracle Deep Data Security (Deep Sec) feature.

This chapter outlines the database and application configurations required for establishing end-user security contexts. It also details the two supported establishment methods (token-based for IAM-managed users and local for database-managed users), and explains how the database server creates, caches, and attaches these security contexts at runtime.

Topics:

> **See also**
>
> End-User Security Context in *Part I: Oracle Deep Data Security Fundamentals.*

14.1 About the End-User Security Context

In a typical scenario, the application connects to the database using a shared, highly privileged database user account (the connection pool user account). All queries run under this account's identity, regardless of which end user is actually making the request.

This broad-access model creates a significant security risk; for example, a SQL injection attack or a compromised AI-generated query can run with the same elevated privileges as the application itself.

The end-user security context solves this problem. When active, it replaces the database session's broad security domain (the connection pool user account and its roles) with the end user's identity, data roles, and context attributes. Consequently, the database authorizes all executed SQL based on who the end user actually is, rather than the connection pool user account. This is conceptually similar to the security context switch that occurs when a definer-rights PL/SQL procedure executes. As a result, the application's database user account requires only basic privileges, such as `CREATE SESSION` and `CREATE END USER SECURITY CONTEXT`, effectively mitigating the risk of attacks like SQL injection.

How the security context is established

When the application initiates a database operation, the Oracle client driver (JDBC, Python, or ODP.NET) invokes a callback, enabling the application code to populate an `EndUserSecurityContext` object. Through this object, the application provides the following details to the database:

- **End-user identity:** The end user's name as asserted in the IAM access token. For end users managed locally, this is the name of the end user created in the database.
- **Data roles (optional):** Additional data roles that the application can enable for the end-user security context, beyond those mapped to application roles in IAM and those enabled by default for the application identity.
- **End-user context attributes (optional):** A dictionary of application-defined key-value pairs to include in the security context. Used when data grants or application logic rely on custom end-user context attributes.
- **Database-access token:** An on-behalf-of (OBO) token or an OAuth client-credential token that the application obtains from IAM to authorize its access to the database. This token ensures the database accepts requests only from trusted applications. Only an authorized application (with its application secret) can obtain this token from IAM.

Once populated, the client driver attaches the `EndUserSecurityContext` object to the database connection as metadata before sending the request to the database. The driver then forwards this metadata with every subsequent SQL request, ensuring that long-running connection-pool connections always operate under the correct, current user context. When the database receives the payload, it validates the access tokens and establishes the end-user security context. The database creates, attaches, and destroys security contexts automatically; no custom session-management code is required in the application.

The direct logon option

While the application architecture described above is the most common use case, Oracle AI Database also supports a direct logon option. If your users (such as data analysts or developers) need to connect directly to the database using a SQL client, they can bypass the application entirely.

Depending on how you manage your environment, these users will authenticate in one of two ways:

- **IAM-managed users:** They provide their IAM access token during authentication. The database automatically extracts their identity and roles from the token.
- **Locally managed users:** They connect using the local end-user credentials (end-user name and password), and the database resolves their identity and roles locally.

In either direct setup, no application or its database-access token is involved. The database natively establishes the correct end-user security context from the user's login.

> **Note**
>
> End-user security context creation fails if the security context name passed to the `setEndUserSecurityContext` API contains non-ASCII characters.

14.2 Prerequisites for Establishing a Token-Based Security Context

Before the database can attach a token-based end-user security context to a session, you must configure your database and application environments.

These requirements apply when you manage end users through an IAM system, such as Microsoft Entra ID or OCI IAM. Review the deployment scenario that matches your environment.

IAM-managed users connecting through an application

Complete the following steps to enable security context establishment for this scenario.

1. **Configure the database for application sessions**
   For detailed Oracle AI Database configuration instructions, see Set Up IAM Integration for Application-Mediated Connections.
2. **Configure the application**
   In your application, use an Oracle client driver (such as JDBC, Python, or ODP.NET) to build and transmit the `EndUserSecurityContext` object on each database call. The following JDBC example illustrates a complete security context payload attachment:

```
System.out.println("Attaching End-User Security Context");

final String USER_TOKEN     = getUserToken();
final String DB_ACCESS_TOKEN = getDbaccessToken();

Map<String, OracleJsonObject> ctxAttrs = new HashMap<>();
OracleJsonObject attrs = new OracleJsonFactory().createObject();
    attrs.put("service_center_id", 52);
    attrs.put("region_id", "EMEA");
    ctxAttrs.put("HR.HCM", attrs);
    try (
      Connection connection = DriverManager.getConnection(sslURL, props)) {
        EndUserSecurityContext securityContext =
            EndUserSecurityContext.createWithToken(DB_ACCESS_TOKEN,
USER_TOKEN)
              .withDataRoles(Set.of("hcm_role"))
              .withAttributes(ctxAttrs);
            connection.unwrap(OracleConnection.class)
               .setEndUserSecurityContext(securityContext);

    query(connection);
}
```

For additional application configuration details, such as connection properties and settings, see Update Application Configuration with IAM Details.

IAM-managed users connecting directly (no application)

If your IAM-managed users (such as data analysts or developers) connect directly to the database using a SQL client and their own IAM access tokens, complete the following tasks.

- **Configure the database**

For detailed Oracle AI Database configuration instructions, see Set Up IAM Integration for Direct Logon.

- **Create end-user context definitions (optional)**
  Additionally, if your data grants rely on custom end-user context attributes, make sure you create the corresponding end-user context definitions. See Configure End-User Contexts and Attributes.

> **Note**
>
> In this scenario:
>
> - **No connection pool user account is required:** Because your users connect directly, you don't need a shared database user account.
> - **Connection strings:** For direct logon, you use the same `tnsnames.ora` connection string format as the standard token-based database authentication. See Configure Oracle Deep Data Security for Direct Logon with End Users in IAM.

14.3 Prerequisites for Establishing a Local Security Context

For environments bypassing centralized IAM in favor of local user management, perform the following configuration tasks. These tasks ensure the database can successfully bind a local end-user security context to an active session.

Review the deployment scenario that matches your environment.

Local application users connecting through an application

If your application manages users in its own user store and those users connect through the application, complete the following steps to establish a security context for their sessions. The application user is identified by a user name and a security context lookup key supplied by the application.

1. **Configure the database for application sessions**
   For detailed Oracle AI Database configuration instructions, see Set Up Local Authentication for Application-Mediated Connections.
2. **Configure the application**
   In your application, use an Oracle client driver (such as JDBC, Python, or ODP.NET) to build and transmit the `EndUserSecurityContext` object (with user name and lookup key) on each database call. The following JDBC example illustrates a local security context payload attachment:

```
EndUserSecurityContext securityContext =
    EndUserSecurityContext.createWithUsername(DB_ACCESS_TOKEN, username,
lookup_key)
        .withAttributes(ctxAttrs);
```

> **Note**
>
> Data roles cannot be provided in an end-user security context payload with end user's name and lookup key.

For additional application configuration details, such as connection properties and settings, see Update Application Configuration with IAM Details.

Local end users connecting directly (password authentication)

For environments where local end users establish direct database sessions through a SQL client and credential-based authentication, perform the following configuration tasks.

1. **Configure the database**
   For detailed Oracle AI Database configuration instructions, see Set Up Local Authentication for Direct Logon.
2. **Create end-user context definitions (optional)**
   Additionally, if your data grants rely on custom end-user context attributes, make sure you create the corresponding end-user context definitions. See Configure End-User Contexts and Attributes.

14.4 How the Database Server Manages an End-User Security Context

When a client driver sends an `EndUserSecurityContext` payload during a SQL operation, the database server automatically executes the steps described in this section.

Use the information provided here for reference and diagnostic purposes; no administrator action is required during this process.

- Scenario 1: Token-Based (IAM-Managed) End User
- Scenario 2: Locally Managed End User

14.4.1 Scenario 1: Token-Based (IAM-Managed) End User

If your end users authenticate using OAuth 2.0 access tokens issued by IAM (such as Microsoft Entra ID or OCI IAM), the database must validate these tokens to ensure that the request originates from a trusted source.

This scenario has the following two variants, depending on how an end user connects to the database:

- **Scenario 1a — Through an application:** The end user accesses the database through an application, and the application supplies both the end-user token and the database-access token through client drivers.
- **Scenario 1b — Direct logon (no application):** The end user connects directly to the database using their end-user token, with no application involved (often used by data analysts or developers).

Scenario 1a — Through an application

This flow triggers when your application sends the `EndUserSecurityContext` payload using a client driver. The payload contains an IAM-issued end-user token and a database-access token that identifies the application. Behind the scenes, the database server performs the following steps:

1. **Extracts the payload:** The server pulls the end-user token, the database-access token, the application roles, and any end-user context attributes from the encoded payload.
2. **Validates the database-access token:** The server verifies the database-access token (which must be an on-behalf-of (OBO) token or a client credentials token). It checks the

token's signature and expiration, and validates the `aud` (audience) claim against the database's initialization parameter, `identity_provider_config` (for Entra ID) or `identity_provider_oauth_config` (for OCI IAM). If validation fails, the server rejects the attachment attempt.

3. **Validates the end-user token:** The server checks the end-user token's signature and expiration, then extracts the user name. It looks for the `upn` claim for Entra ID, or the `sub` claim for OCI IAM.
4. **Resolves data roles:** The server determines which data roles to activate by combining two sources:
   - *IAM roles:* The server extracts the role claims from the end-user token (the `roles` claim for Entra ID, or a configured `group` claim for OCI IAM) and finds the corresponding data roles in the database. For each claim value, the server looks for a data role with a `MAPPED TO` clause matching the claim. Matching data roles are enabled. For details on role resolution across different IAM systems, see examples in Configure Data Roles.
   - *Application data roles:* The server compares the additional or common data roles requested by the application against the data roles granted to the application identity. Granted data roles that are enabled (the default option) are always included in the security context (even if not requested), while granted data roles that are disabled on creation are only included if explicitly requested, allowing privilege elevation. Requests for ungranted data roles are ignored. See Grant and Revoke Data Roles for examples of granting data roles to an application identity.

   Note

   - In a data role definition (for example, `CREATE DATA ROLE manager_role MAPPED TO 'AZURE_ROLE=manager'`), the comparison between the role name in the IAM token and the role name specified after the prefix (for example, `manager` in `AZURE_ROLE=manager`) is case-insensitive.
   - With Microsoft Entra ID claims, your application roles appear in the `roles` claim. If you need to use group information, you must explicitly configure your IAM token to include it.
   - With OCI IAM claims, your group information appears in a custom claim, provided you have configured your IAM domain to populate it.
5. **Manages the security context cache:** To optimize performance, the server does the following actions:
   - The server uniquely identifies an end-user security context by the combination of its end-user token, database-access token, and data role fields.
   - If an exact matching security context (same end-user token, database-access token, and data roles) is already attached to the database session, the server leaves it attached. If the tokens or data roles have changed, the server detaches the existing security context and attaches the new one.
   - If a matching security context exists but isn't currently attached, the server reattaches the security context to the database session.
   - If an existing security context has a stale role-graph (for example, a data role was modified), the server detaches it and reattaches a refreshed security context.
   - If no match exists, the server creates a new security context, assigns the extracted user name as the external user identity, enables the resolved set of data roles, and attaches the security context with the provided end-user context attributes.

6. **Handles security context attach, detach, and cleanup:** The database automatically manages the detachment and reattachment of an end-user security context to a database session through the following mechanisms:
   - *Connection release:* When your application code releases the connection, the security context is detached from the database session.
   - *Security context replacement:* When your application reuses a connection, the database seamlessly replaces the previous security context with the new one. If the incoming request does not use an end-user security context, the server simply detaches the old one.
   - *Garbage collection:* The database automatically garbage-collects any end-user security contexts that remain inactive (not attached to any database session) after a timeout duration of one hour.

Scenario 1b — Direct logon (no application)

If your users require direct SQL access without an application, they can log in using their end-user token. When a user presents an end-user token for direct logon, the database first looks for a standard schema mapped to that user. If no schema match is found, the server executes the direct-logon flow. It performs the following tasks:

1. **Matches data roles:** The server retrieves all the data roles whose `MAPPED TO` clause matches the role claims inside the end-user token.
2. **Retrieves database roles:** For every matching data role, the server retrieves all standard database roles granted to it.
3. **Evaluates privileges and establishes security context:** If any of those database roles contain the `CREATE SESSION` privilege, the server permits the direct logon and establishes the end-user security context with those data roles active.
   With direct logon, the end-user security context is destroyed when the database session closes.

> **Note**
>
> **Oracle Deep Data Security (Deep Sec) session behavior**
>
> - Current user state: After the security context attaches successfully, querying `sys_context('userenv', 'current_user')` returns `XS$NULL`. All of your other `USERENV` attributes remain entirely unchanged.
> - End-user identity: You can identify the current user by checking `ORA_END_USER_CONTEXT.username`, which reflects the `upn` claim if you use Microsoft Entra ID, or the `sub` claim if you use OCI IAM.
> - View active roles: You can query the `V$END_USER_DATA_ROLE` view to list the specific data roles currently active (derived from the end-user token's role claims).
> - View all end-user security contexts: A DBA can view the list of end-user security contexts in the database using the `DBA_END_USER_SECURITY_CONTEXTS` view.

14.4.2 Scenario 2: Locally Managed End User

When end users are managed locally (with no IAM system involved), the database server resolves the end-user identity and data roles entirely within the database.

This scenario has the following two variants, depending on how an end user connects to the database:

- **Scenario 2a — Through an application:** An end user from the application's own user store accesses the database through the application. The application supplies the `EndUserSecurityContext` payload through client drivers, identifying the end user through a user name and a security context lookup key, rather than an IAM token. Additionally, it provides a database-access token.
- **Scenario 2b — Direct logon (password authentication):** The local end user connects directly to the database using password authentication.

Scenario 2a — Through an application

This flow triggers when your application sends the `EndUserSecurityContext` payload using a client driver. The payload contains an end-user name, a security context lookup key, and a database-access token. Behind the scenes, the database server performs the following steps:

1. **Verifies the local application user:** The server checks if a local end user matching the user name in the `EndUserSecurityContext` payload exists in the database. If no match is found, the server raises an error.
2. **Searches for an existing security context:** If the end user is present, the server looks for an existing end-user security context that matches the provided security context lookup key.
3. **Creates a new security context (if necessary):** If no matching security context exists, the server creates a new one using the local end user's identity and their explicitly granted data roles.
4. **Attaches the security context:** Finally, the server attaches the security context (along with any provided end-user context attributes) to the active session.
5. **Handles security context attach, detach, and cleanup:** The database automatically manages the detachment and reattachment of an end-user security context to a database session through the following mechanisms:
   - *Connection release:* When your application code releases the connection, the security context is detached from the database session.
   - *Security context replacement:* When your application reuses a connection, the database seamlessly replaces the previous security context with the new one. If the incoming request does not use an end-user security context, the server simply detaches the old one.
   - *Garbage collection:* The database automatically garbage-collects any end-user security contexts that remain inactive (not attached to any database session) after a timeout duration of one hour.

Scenario 2b — Direct logon (password authentication)

If your end users created in the database log in directly using a SQL client, the database server performs the following tasks:

1. **Validates identity:** The server verifies the end user using credential-based password authentication.

2. **Identifies data roles:** The server identifies the end user's assigned data roles.
3. **Retrieves database roles:** For each identified data role, the server retrieves all standard database roles granted to it.
4. **Evaluates privileges and establishes security context:** If any of those database roles contain the `CREATE SESSION` privilege, the server permits the direct logon and establishes the end-user security context with those data roles active.
   With direct logon, the end-user security context is destroyed when the database session closes.

15

Configure End-User Contexts and Attributes

Use the SQL statements in this chapter to create, drop, read, and modify end-user context definitions and their attributes. These context definitions serve as templates that the database uses at runtime to enforce fine-grained authorization policies through data grants.

Before configuring your environment, it is important to distinguish between two related, similarly named concepts:

- **End-user security context** *(runtime values within a database session)*: The session-level object the database creates automatically when an application sends an `EndUserSecurityContext` payload. It holds the active end user's identity, their enabled data roles, and all live attribute values. It is created, attached, and destroyed automatically by the database server.
- **End-user context** *(database-side definition / template)*: A database schema object you create with the `CREATE END USER CONTEXT` command that defines a set of attributes, their data types, their default values, and optional PL/SQL routines to populate them. At runtime, the database uses it as a template for JSON-style name-value pairs and instantiates it on first use in the current end-user security context — either when the application payload is attached or when an attribute is required for authorization checks.

The relationship between the two is this: if your data grants or application logic reference custom context attributes (for example, `ORA_END_USER_CONTEXT.hr.hcm_context.emp_id`), you must first create the corresponding end-user context definition in the database before those attributes are available within a runtime end-user security context.

> **Note**
>
> To query `ORA_END_USER_CONTEXT` directly or to use it in data grant predicates, you must set the database instance's `COMPATIBLE` initialization parameter to `20.0` or greater.

Topics:

15.1 About Default and Custom Contexts

Discover how the database uses both default and custom end-user contexts to manage session attributes and enforce fine-grained access control for applications.

Context attributes extend the end-user security context beyond identity and roles. They are JSON name-value pairs organized by named namespaces that an application can use in data

grant predicates, SQL queries, and application logic. The application passes these context attributes to the database as part of the `EndUserSecurityContext` payload.

Requirement for a context definition in the database

Before an application can pass context attributes at runtime, you must create a corresponding `END USER CONTEXT` schema object in the database using the `CREATE END USER CONTEXT` command. This definition specifies the end-user context name, its allowed attributes, their data types, and optional PL/SQL handler logic.

If the application sends a context attribute that does not have a corresponding end-user context definition, the attribute is ignored. Data grant predicates that reference the attribute through the `ORA_END_USER_CONTEXT` function do not receive its value.

Oracle AI Database supports two categories of end-user contexts:

- Default (System-Defined) Contexts
- Custom (User-Defined) Contexts

15.1.1 Default (System-Defined) Contexts

Two predefined, system-managed end-user contexts are always available in every end-user security context and do not require explicit instantiation. You cannot modify attributes in these end-user contexts directly; they are initialized by the database server based on the user identity propagated from IAM.

- **`USER.DEFAULT:`** Contains system attributes that identify the end user, such as `USERNAME`, `LOGON_END_USER`, and `CURRENT_END_USER`. It also includes all predefined database session attributes from the `SYS_CONTEXT` namespace `USERENV`, such as `AUTHENTICATED_IDENTITY` and `DB_NAME`. For ease of use, `USER.DEFAULT` attributes are elevated to the top level of the `ORA_END_USER_CONTEXT` namespace, so you can reference them directly without specifying `USER.DEFAULT` in the path. For example, `ORA_END_USER_CONTEXT.username` returns the end user's name.
- **`USER.TOKEN:`** Contains claims extracted directly from the end user's OAuth 2.0 access token. Only the following claims are available: `iss [issuer]`, `sub [subject]`, and `aud [audience]`. You must prefix these attributes with the full path in the function, for example, `ORA_END_USER_CONTEXT.USER.TOKEN.iss`.
  To view the full end-user token content, query `ORA_END_USER_CONTEXT.USER.TOKEN` directly. The following examples show the end-user token output for supported identity providers:
  - Microsoft Entra ID:

    ```
    SELECT ORA_END_USER_CONTEXT.USER.TOKEN FROM DUAL;

    TOKEN
    --------------------------------------------------------------------------------
    --------
    {"iss":"https://sts.windows.net/443e3044-e82d-410a-8b0a-57498722241d/",
     "sub":"tgc1Nc73wj6E6xKO0r1wMFN95PyNB_nnfCwj9URUxx0",
     "aud":"https://supremo.onmicrosoft.com/2edc9c9f-8e1e-4ade-8a4a-
    cc286ed1b899"}
    ```

  - OCI IAM:

    ```
    SELECT ORA_END_USER_CONTEXT.USER.TOKEN FROM DUAL;
    ```

```
TOKEN
--------------------------------------------------------------------------------
--------
{"iss":"https://identity.oraclecloud.com/",
 "sub":"deep_sec_employee",
 "aud":"OracleDB"}
```

15.1.2 Custom (User-Defined) Contexts

In addition to system contexts, you can create your own end-user contexts using JSON schema definitions.

Custom contexts allow different applications to define and manage their own attributes without conflict. For example:

- An HCM application may define an end-user context `hr.hcm_context` containing attributes `emp_id` and `org_id`.
- A CRM application may define an end-user context `ct.crm_context` containing attributes `territory_id` and `customer_id`.

You can configure each attribute in a custom context with either a static default value or a lazy-loading event handler (PL/SQL callback), which is invoked the first time the attribute is read. You must create custom contexts in the database before you can use them at runtime. Instantiation occurs automatically on first reference and does not require an explicit command.

Size limits

When defining custom contexts and attributes, ensure you stay within the following system limits:

- Maximum context name length: 128 characters
- Maximum attribute name length: 128 characters
- Maximum JSON schema payload length in the `CREATE END USER CONTEXT` command: 4,000 characters

15.2 Create a Custom End-User Context

Use the `CREATE END USER CONTEXT` command to define a new, custom end-user context object in the database.

The context definition specifies a set of named attributes, their data types, default values, and optional PL/SQL event handlers. The definition acts as a template that the database instantiates at runtime within each end-user security context. You can assign each attribute a default value or associate it with a PL/SQL handler function for lazy loading on first read.

Required privilege

- The `CREATE END USER CONTEXT` system privilege to create an end-user context in the executing user's own schema.
- The `CREATE ANY END USER CONTEXT` system privilege to create an end-user context in any schema.

Syntax

```
CREATE [OR REPLACE] END USER CONTEXT [IF NOT EXISTS] [schema.]name
    USING JSON SCHEMA '{JSON_Schema}';

JSON_Schema ::= "type": "object", properties
properties  ::= "properties": { attribute [, attribute]... }
attribute   ::= attr_name : {
                    "type": attr_type,
                    property_name_value
                }
attr_type   ::= {"integer" | "string" | "object" | "null"}
property_name_value ::=
    {"o:onFirstRead" : "hdlowner.package.function" |
     "default" : value}
```

Parameters

Parameter	Description
`schema`	The schema (owner) name of the end-user context definition. If omitted, the current schema is used.
`name`	The name of the end-user context definition.
`attr_name`	The name of the attribute within the context.
`attr_type`	The data type of the attribute. Supported types are `integer`, `string`, and `object`. Use `object` for nested attributes.
`value`	The default value of the attribute. A JSON number for integer attributes; a double-quoted JSON string for string attributes.
`hdlowner`	The schema that owns the PL/SQL handler function for the attribute.
`package`	The PL/SQL package that contains the handler function.
`function`	The PL/SQL function name that is executed when the specified event (`o:onFirstRead`) occurs on the attribute.

Usage notes and restrictions

- When `IF NOT EXISTS` is specified:
  - If the context exists, the statement is a no-op. No error is raised.
  - If the context does not exist, it is created.
- When `IF NOT EXISTS` is omitted:
  - If the context already exists, an error is raised.
  - If the context does not exist, it is created.
- When `OR REPLACE` is specified:
  - If the context already exists, its attribute-value settings are replaced.
  - If the context does not exist, it is created.
- When `OR REPLACE` is omitted:
  - If the context already exists, an error is raised.

  - If the context does not exist, it is created.
- `OR REPLACE` and `IF NOT EXISTS` cannot be combined in the same statement. If combined, an error is raised.
- Event handlers:
  - Each attribute can specify either a default value or an event handler, not both.
  - The `o:onFirstRead` event causes the handler function to be called the first time the attribute is read during a session. The handler function must then set the attribute value.
  - A context can have at most one unique PL/SQL handler function. If multiple attributes reference event handlers, they must all reference the same `schema.package.function`; otherwise, an error is raised.

> **Note**
>
> - Context instantiation is transparent. When an end user or application references a custom context for the first time, the database automatically instantiates it from the definition. No explicit instantiation command is required.
> - Context instances are session-specific. They are only visible and modifiable within the current session.

For syntax diagrams and additional details, see CREATE END USER CONTEXT in *Oracle AI Database SQL Language Reference*.

Example 15-1 Create a custom context with lazy-loaded and static attributes

The following example creates an end-user context `hr.hcm_context` with two attributes. The `emp_id` attribute is loaded lazily through a PL/SQL callback. The `org_id` attribute has a static default value of `1`.

```
CREATE END USER CONTEXT hr.hcm_context USING JSON SCHEMA '{
    "type": "object",
    "properties": {
        "emp_id": {
            "type": "integer",
            "o:onFirstRead": "hr.hcm_core.init_user_context"
        },
        "org_id": {
            "type": "integer",
            "default": 1
        }
    }
}';
```

When `emp_id` is first accessed, the database invokes `hr.hcm_core.init_user_context` to set the value. The `org_id` attribute is always initialized with the value `1`.

Example 15-2 Replace an existing context definition

To replace the definition of an existing end-user context, use `OR REPLACE`.

```
CREATE OR REPLACE END USER CONTEXT hr.hcm_context USING JSON SCHEMA '{
    "type": "object",
    "properties": {
        "emp_id": {
            "type": "integer",
            "o:onFirstRead": "hr.hcm_core.init_user_context"
        },
        "org_id": {
            "type": "integer",
            "default": 1
        },
        "region_id": {
            "type": "string",
            "default": "EMEA"
        }
    }
}';
```

Example 15-3 Create a context only if it does not already exist

```
CREATE END USER CONTEXT IF NOT EXISTS hr.hcm_context USING JSON SCHEMA '{
    "type": "object",
    "properties": {
        "org_id": {
            "type": "integer",
            "default": 1
        }
    }
}';
```

15.3 Drop a Custom End-User Context

Use the `DROP END USER CONTEXT` command to remove a custom end-user context definition from the database.

After a context is dropped, the definition is no longer available for instantiation in future sessions. Attribute values stored in existing sessions are unaffected until the session is detached or recreated.

Required privilege

- To drop an end-user context in the executing user's own schema, no privilege is required.
- The `DROP ANY END USER CONTEXT` system privilege to drop an end-user context in any schema.

Syntax

```
DROP END USER CONTEXT [IF EXISTS] [schema.]name;
```

Parameters

Parameter	Description
`schema`	The schema (owner) name of the end-user context to be dropped. If omitted, the current schema is used.
`name`	The name of the end-user context definition to drop.

Usage notes and restrictions

- When `IF EXISTS` is specified:
  - If the context does not exist, the statement is a no-op. No error is raised.
  - If the context exists, it is dropped.
- When `IF EXISTS` is omitted:
  - If the context does not exist, an error is raised.
  - If the context exists, it is dropped.

For syntax diagrams and additional details, see DROP END USER CONTEXT in *Oracle AI Database SQL Language Reference*.

Example 15-4 Drop an end-user context

```
DROP END USER CONTEXT hr.hcm_context;

-- Drop only if the context exists (no error if absent):
DROP END USER CONTEXT IF EXISTS hr.hcm_context;
```

15.4 Read End-User Context Attributes

Use the `ORA_END_USER_CONTEXT` function to retrieve *end-user context attributes* from the current *end-user security context*.

This function returns a JSON object containing all end-user context attributes for the current security context. You can use dot notation to navigate the JSON hierarchy. Depending on the provided field parameters, the function can return a single attribute value, all attributes within a specific end-user context, or all attributes across all instantiated end-user contexts.

In addition to the `ORA_END_USER_CONTEXT` function, the `SYS.END_USER_CONTEXT` view exposes all instantiated end-user contexts and their attributes in a table format, with each row representing one context's content as a JSON document. This view is the primary interface for querying or updating end-user context attributes using standard DML syntax.

Required privilege

- The `SELECT ANY END USER CONTEXT` system privilege to read attributes from any end-user context.
- The `SELECT` privilege on the `SYS.END_USER_CONTEXT` view for a specific end-user context, granted through a data grant. For example:

  ```
  CREATE DATA GRANT [schema.]grant_name AS
    SELECT
    ON SYS.END_USER_CONTEXT
  ```

```
WHERE owner = 'HR' AND name = 'HCM_CONTEXT'
TO <end user or data role>;
```

> **Note**
>
> To create data grants on the SYS.END_USER_CONTEXT view, the data grant creator must have the ADMINISTER ANY DATA GRANT system privilege at the system level. In addition, they must have all the other required privileges listed in Create Data Grants.

Syntax

```
ORA_END_USER_CONTEXT[.schema[.name[.attribute]]]
```

Parameters

Parameter	Description
schema	The owner of the end-user context. Omit for USER.DEFAULT system attributes (which are elevated to the top level).
name	The name of the end-user context. If omitted, all end-user contexts under the schema are returned.
attribute	The specific attribute within the end-user context. If omitted, the entire end-user context is returned as a JSON object.

Usage notes and restrictions

- The ORA_END_USER_CONTEXT function does not take function arguments. The dot-path notation is possible because the function returns a JSON type, and the JSON field navigation syntax applies.
- If you call SELECT ORA_END_USER_CONTEXT without a dot-path, it returns a complete JSON document containing all predefined system attributes and all instantiated custom contexts with their attributes.
- Attributes of the USER.DEFAULT context are elevated to the top level of the JSON return value. To access them, use ORA_END_USER_CONTEXT.username rather than ORA_END_USER_CONTEXT.USER.DEFAULT.username.
- Attributes of the USER.TOKEN context require the full prefix: ORA_END_USER_CONTEXT.USER.TOKEN.iss.
- Custom context attributes require both schema and name in the path: ORA_END_USER_CONTEXT.hr.hcm_context.emp_id.
- You must have defined an end-user context (using the CREATE END USER CONTEXT command) before its attributes can be referenced. Instantiation occurs automatically on first reference.
- The SYS.END_USER_CONTEXT view provides an alternative table-based interface. You can use the standard SELECT with a WHERE predicate on the OWNER and NAME columns to filter to a specific context. The CONTEXT column contains the JSON payload for that end-user context.

For syntax diagrams and additional details, see ORA_END_USER_CONTEXT in *Oracle AI Database SQL Language Reference*.

Example 15-5 Return all available context attributes for the current session

```
SELECT ORA_END_USER_CONTEXT FROM dual;
```

If `hr.hcm_context` has been instantiated with `emp_id = 3` and `org_id = 5`, the above statement returns a JSON document similar to the following example:

```
{
    "USERNAME": "SCOTT",
    "LOGON_END_USER": "SCOTT",
    "CURRENT_END_USER": "SCOTT",
    "USER": {
        "TOKEN": {
            "TOKEN_ID": "...",
            ...
        }
    },
    "HR": {
        "HCM_CONTEXT": {
            "emp_id": 3,
            "org_id": 5
        }
    }
}
```

Example 15-6 Return all attributes for a specific context

Return all attributes for `hr.hcm_context`.

```
SELECT ORA_END_USER_CONTEXT.hr.hcm_context FROM dual;
```

Output:

```
{
    "emp_id": 3,
    "org_id": 5
}
```

Example 15-7 Return all contexts under a schema

Return all contexts under the `hr` schema.

```
SELECT ORA_END_USER_CONTEXT.hr FROM dual;
```

Example 15-8 Return a specific attribute value

Return the `emp_id` attribute from `hr.hcm_context`.

```
SELECT ORA_END_USER_CONTEXT.hr.hcm_context.emp_id FROM dual;
```

Example 15-9 Access system-level attributes

```
SELECT ORA_END_USER_CONTEXT.username FROM dual;
SELECT ORA_END_USER_CONTEXT.DB_NAME FROM dual;
SELECT ORA_END_USER_CONTEXT.USER.TOKEN.iss FROM dual;
```

Example 15-10 Query a specific attribute using the `SYS.END_USER_CONTEXT` view

Use the `SYS.END_USER_CONTEXT` view to query a specific attribute using the standard table syntax.

```
SELECT t.CONTEXT.org_id
FROM   SYS.END_USER_CONTEXT t
WHERE  owner = 'HR' AND name = 'HCM_CONTEXT';
```

This is equivalent to the following command:

```
SELECT ORA_END_USER_CONTEXT.hr.hcm_context.org_id FROM dual;
```

Example 15-11 Use a context attribute in a query predicate

Use a subquery against the `SYS.END_USER_CONTEXT` view in a predicate.

```
SELECT * FROM hr.employees
WHERE  emp_id = (
    SELECT t.CONTEXT.emp_id
    FROM   SYS.END_USER_CONTEXT t
    WHERE  owner = 'HR' AND name = 'HCM_CONTEXT'
);
```

Example 15-12 Use a context attribute in a data grant predicate

```
CREATE DATA GRANT hcm_dg AS
  SELECT ON hr.employees
  WHERE emp_id = ORA_END_USER_CONTEXT.hr.hcm_context.emp_id
  TO john;
```

15.5 Modify Custom End-User Context Attributes

After a custom context is defined and instantiated in an end-user security context, you can update individual attribute values using a standard `UPDATE` statement on the `SYS.END_USER_CONTEXT` view. This allows application code or PL/SQL handlers to set dynamic attribute values during a session.

Required privilege

- The `UPDATE ANY END USER CONTEXT` system privilege, or

- The `UPDATE` privilege on the `SYS.END_USER_CONTEXT` view for the target end-user context, granted through a data grant. For example:

```
CREATE DATA GRANT [schema.]grant_name AS
  UPDATE
  ON SYS.END_USER_CONTEXT
  WHERE owner = 'HR' AND name = 'HCM_CONTEXT'
  TO <end user or data role>;
```

> **Note**
>
> To create data grants on the `SYS.END_USER_CONTEXT` view, the data grant creator must have the `ADMINISTER ANY DATA GRANT` system privilege at the system level. In addition, they must have all the other required privileges listed in Create Data Grants.

Syntax

```
UPDATE SYS.END_USER_CONTEXT t
SET    t.CONTEXT.attribute = value
WHERE  OWNER = owner AND NAME = name;
```

Parameters

Parameter	Description
`owner`	The schema name (owner) of the custom context to update.
`name`	The name of the custom context to update.
`attribute`	The attribute within the custom context whose value is being changed.
`value`	The new value to assign to the attribute. Must conform to the attribute's declared data type.

Usage notes and restrictions

- You can enforce fine-grained access to specific end-user contexts through data grants. The following example grants `hcm_admin_role` the ability to instantiate and update `hr.hcm_context`.

```
CREATE DATA GRANT update_hcm_attr AS
  SELECT, UPDATE
  ON SYS.END_USER_CONTEXT
  WHERE owner = 'HR' AND name = 'HCM_CONTEXT'
  TO hcm_admin_role;
```

- There is no physical table behind `SYS.END_USER_CONTEXT`. The `UPDATE` statement is rewritten internally to operate on the in-memory end-user context cache of the current security context.
- End-user context instances are session-specific. An update made in one connection is not visible to another connection that shares the same end-user security context ID.

- If two parallel connections share the same security context ID and both make updates, the end-user context attribute values stored at detachment follow the *last-write-wins* rule: the end-user context belonging to the connection that detaches last persists.
- For data grants on `SYS.END_USER_CONTEXT`, only row-level `SELECT` and `UPDATE` privileges are supported. Other privileges and column-level privileges are not permitted. The `SELECT` privilege allows instantiation of an end-user context at runtime; `UPDATE` allows attribute modification.
- Authorization for PL/SQL handler functions follows the standard invoker or definer rights model. When the application identity instantiates the end-user context, it becomes the invoker of any `o:onFirstRead` handler. The handler must have `UPDATE` access on the context (through a data grant or `UPDATE ANY END USER CONTEXT` system privilege) to successfully write attribute values.
- The updated attribute value must match the attribute type defined in the end-user context schema. If the value does not match the defined type, an error is raised.
- Bind variables are not supported. You cannot use bind variables from either PL/SQL or a JDBC client.

For syntax diagrams and additional details, see MODIFY END USER CONTEXT in *Oracle AI Database SQL Language Reference*.

Example 15-13 Set an end-user context attribute for the current security context

Set the `emp_id` attribute in `hr.hcm_context` to `3` for the current end-user security context.

```
UPDATE SYS.END_USER_CONTEXT t
SET    t.CONTEXT.emp_id = 3
WHERE  owner = 'HR' AND name = 'HCM_CONTEXT';
```

Example 15-14 Implement a PL/SQL callback to populate an attribute value at runtime

If a PL/SQL callback is specified, create and implement the packages and procedures as needed to populate the attribute value at runtime.

```
CREATE OR REPLACE PACKAGE BODY hr.hcm_core AS
  PROCEDURE init_user_context IS
   sql_stmt VARCHAR2(1000);
  BEGIN
   -- application logic...
   sql_stmt := 'UPDATE END_USER_CONTEXT t SET t.CONTEXT.emp_id = 100 WHERE
owner = ''HR'' and name = ''HCM_CONTEXT''';
   EXECUTE IMMEDIATE sql_stmt;
  END;
END;
/
```

16

Configure Data Grants

Use the SQL statements in this chapter to manage data authorization policies of Oracle Deep Data Security (Deep Sec). These statements allow you to create and drop data grants, execute access check functions, and enforce mandatory access control.

Topics:

16.1 About Data Grants

Data grants are the central policy mechanism in Oracle Deep Data Security (Deep Sec) for controlling fine-grained access at the row, column, and cell levels.

A data grant authorizes access to specific rows and columns within a table, view, or materialized view by specifying privileges, an optional predicate to filter rows, and the grantee (end user or data role).

Through multiple data grants on the same object, a user can hold different privileges on different row sets and column values (cells); for example, `SELECT` on some rows, `UPDATE` on specific column values within a subset of those rows, and `SELECT`, `UPDATE`, `DELETE` on another set of rows. All data grants are additive: the effective privilege is the union of all applicable grants.

You define data grants using the `CREATE DATA GRANT` statement and remove them with `DROP DATA GRANT`. You can grant them only to Deep Sec end users or data roles; standard database users and roles cannot be grantees of a data grant.

You can query the `DBA_DATA_GRANTS` data dictionary view to review the existing data grants.

Data manipulation language (DML) privilege semantics

The combination of row and column grants produces a fine-grained authorization model that applies to all CRUD operations. Each DML operation is evaluated as follows:

- **`SELECT`:** This is a row and column-level operation. It returns only rows where a row-level or authorized column-level predicate evaluates to `TRUE`.
  - For returned rows, unauthorized column values (cell values) are `NULL`.
  - DML statements with `SELECT` clauses (such as `INSERT ... AS SELECT`, `CREATE TABLE AS SELECT`, and `UPDATE ... AS SELECT`) also use `NULL` for unauthorized column values.

  - The `SELECT` privilege is also checked for `UPDATE`, `INSERT`, and `DELETE` statements with `RETURNING INTO` clauses.
  - If the `sql92_security` parameter is enabled, the `SELECT` privilege is also evaluated for `UPDATE` and `DELETE` statements.
- **`UPDATE:`** This is a cell-level operation. It updates only cells that satisfy either a row-level or column-level predicate.
  - When multiple cells in a single row are targeted, all cells must be individually authorized; if any target cell is unauthorized, the entire row update is silently skipped.
  - Additionally, a post-update check ensures the updated data still satisfies the authorized predicates (equivalent to a `WITH CHECK OPTION` in views). If it doesn't, the row is not updated.
- **`INSERT:`** This is a row or cell-level operation. It inserts records only if the user has a row-level privilege or column-level privileges for all columns being populated. Any unspecified columns are populated with their default values. If data to be inserted does not satisfy the authorized predicates, an error is raised.
- **`DELETE:`** This is a row-level operation enforcing row-level delete privilege. Unlike other operations, `DELETE` privileges cannot be combined with column-level grants.

Reference table: `hr.employees`

All examples in this chapter use the following `hr.employees` table.

```
EMPLOYEE_ID  FIRST_NAME  LAST_NAME  EMAIL       MANAGER     SSN
SALARY  PHONE
-----------  ----------  ---------  ----------  ----------  -----------
------  --------
100          Victoria    Williams   vwilliams               219-09-9999
13000   555-0100
200          Marvin      Anderson   manderson   vwilliams   457-55-5462
12030   555-0200
300          Chris       Evans      cevans      vwilliams   321-12-4567
6900    555-0300
400          Emma        Baker      ebaker      manderson   733-02-9821
8200    555-0400
500          Taylor      Mills      tmills      manderson   558-76-1243
9000    555-0500
```

16.2 Create Data Grants

Use the `CREATE DATA GRANT` statement to create a named policy that authorizes Oracle Deep Data Security (Deep Sec) end users or data roles to perform CRUD operations on targeted rows and columns of a table or view.

Required privilege

- The `CREATE DATA GRANT` system privilege to create data grants in the user's own schema.
- The `CREATE ANY DATA GRANT` system privilege to create data grants in any schema (can also be granted at the schema level).
- The `ADMINISTER ANY DATA GRANT` system privilege to create data grants on objects in any schema (can also be granted at the schema level for non-SYS schema).

For example, to create a data grant in schema A that targets a table in schema B:

```
CREATE DATA GRANT A.datagrant
   AS SELECT
   ON B.table1
   WHERE <predicate> TO grantee;
```

You must have the `CREATE DATA GRANT` privilege (or `CREATE ANY DATA GRANT`) on schema A, and you must have the `ADMINISTER ANY DATA GRANT` privilege at the system level or on schema B.

> **Note**
>
> - Data grants are not supported for objects owned by `SYS`, with the exception of the `SYS.END_USER_CONTEXT` view.
> - To create data grants on the `SYS.END_USER_CONTEXT` view, the data grant creator must have the `ADMINISTER ANY DATA GRANT` system privilege at the system level.

Syntax

```
CREATE [ OR REPLACE ] DATA GRANT [ IF NOT EXISTS ]
  [schema. ] grant_name AS
  privilege_list
  ON [ schema. ] object
  [ WHERE predicate ]
  TO user_role_list
  [ START TIME timestamp ] [ END TIME timestamp ];

privilege_list::= privilege_item [, privilege_item]…

privilege_item::= privilege [ ( [ALL COLUMNS EXCEPT] column_list ) ]

privilege::= SELECT | UPDATE | DELETE | INSERT

column_list::= column [, column ]…

user_role_list::= user_role [, user_role ]…

user_role::= end_user | data_role
```

Parameters

Parameter/Keyword	Description
`schema`	The schema name of the data grant being created. Must be a valid database user. If omitted, the current schema is used. Error `ORA-52553` is raised if the schema (owner) of the data grant is invalid, that is: • The owner is not a valid database user, or • A Deep Sec user (whose schema is `XS$NULL`) tries to create a data grant without specifying a schema name.

Parameter/Keyword	Description
grant_name	The name of the data grant object. If the data grant already exists and neither OR REPLACE nor IF NOT EXISTS is specified, ORA-52550 is raised.
privilege_list	One or more CRUD operations (SELECT, UPDATE, INSERT, DELETE) granted by this data grant. A privilege cannot appear more than once in the list; duplicate privileges raise ORA-01711.
ALL COLUMNS EXCEPT	A setting that grants the specified privilege on all columns of the target object except those listed in column_list. Future columns added to the object are also granted to the grantees.
column_list	A list of columns to be specified for SELECT, UPDATE, and INSERT only. If omitted, the privilege is granted on all columns in the table or view. Column-level privileges cannot be specified for DELETE; if specified, ORA-52502 is raised. If the same column appears more than once in the list, ORA-00957 is raised.
object	The target table, view, or materialized view. If the schema is omitted, the current schema is assumed. If the object doesn't exist, an error is raised. There can be only one object per ON clause.
predicate	A SQL WHERE clause that identifies the rows to which the data grant applies. It represents a subset of rows within a table or view. If omitted, the grantee can access all rows. Supports subqueries and references to the end-user context (ORA_END_USER_CONTEXT). Predicates in data grants have the following limitations: • They cannot exceed 4,000 characters. • GROUP BY extension operators (GROUPING SETS, CUBE, ROLLUP) are not supported. • If cyclic predicates (referencing the same object as the ON clause) are used, ORA-52561 is raised at query-execution time. **Note** The owner (schema) of the data grant must have sufficient privileges on any objects referenced in the predicate; for example, SELECT on tables and views, and EXECUTE on PL/SQL functions or SQL macros. These privileges are enforced at runtime, not at DDL creation time. However, the grant owner isn't required to have privileges on the target object specified in the ON clause.
user_role_list	One or more Deep Sec end users or data roles receiving the privilege. Standard database users and roles cannot be grantees. An end user or a data role cannot appear more than once in the list; if it does, ORA-52501 is raised. If a grantee does not exist or is not a Deep Sec end user or data role, ORA-52551 is raised.
end_user	The name of an end user created in the database. See Create End User.
data_role	The name of a data role. See Create Data Role.

Parameter/Keyword	Description
START TIME	The time from which the data grant is effective. Must not be later than END TIME; if it is, ORA-52503 is raised. If omitted, the data grant takes effect immediately.
END TIME	The time at which the data grant becomes inactive. If omitted, the grant remains in effect indefinitely.

Usage notes and restrictions

- When IF NOT EXISTS is specified:
  - If the data grant exists, the statement is a no-op. No error is raised.
  - If the data grant does not exist in the schema, it is created.
- When IF NOT EXISTS is omitted:
  - If the data grant already exists, ORA-52550 is raised.
  - If the data grant does not exist in the schema, it is created.
- When OR REPLACE is specified:
  - If the data grant already exists, its definition is replaced without dropping the grant.
  - If the data grant does not exist, it is created.
- When OR REPLACE is omitted:
  - If the data grant already exists, an error is raised.
  - If the data grant does not exist, it is created.
- With OR REPLACE:
  - You can update the privilege_list, user_role_list, predicate, START TIME, and END TIME of an existing data grant.
  - You cannot change the target object (the ON clause object). If you attempt to change, ORA-52556 is raised.
- OR REPLACE and IF NOT EXISTS cannot be combined in the same statement. If combined, ORA-11541 is raised.
- WITH GRANT OPTION is not supported for data grants.
- The predicate is validated for syntax and object existence at CREATE DATA GRANT time. Cyclic reference checks (for example, where the predicate references the same table as the ON clause) are performed at query-execution time.
- You can specify START TIME and END TIME independently of each other. If START TIME is not specified, the data grant is effective immediately. If END TIME is not specified, the data grant is effective until it is dropped.
- You cannot create data grants on views that reference remote objects through database links. You also cannot reference remote objects in a data grant predicate. If you attempt either, the database raises ORA-52554.
- For data grants on the SYS.END_USER_CONTEXT view, only row-level SELECT and UPDATE privileges are supported. Other privileges and column-level privileges are not permitted.

For syntax diagrams and additional details, see CREATE DATA GRANT in *Oracle AI Database SQL Language Reference*.

Example 16-1 Grant full SELECT access on a table without a predicate

The following data grant provides the data role `employee_role` full access to the `hr.employees` table.

```
CREATE OR REPLACE DATA GRANT hr.employees_full_grant
  AS SELECT
  ON hr.employees
  TO employee_role;
```

Example 16-2 Grant row-level SELECT access to an employee's own record

The following data grant allows employees to view only their own record in `hr.employees`. The predicate compares the employee's email against the current end user's name from the end-user security context.

```
CREATE OR REPLACE DATA GRANT hr.employees_own_record
    AS SELECT
    ON hr.employees
    WHERE email = ORA_END_USER_CONTEXT.username
    TO employee_role;
```

When Emma, who has the `employee_role` data role, queries the table, only her own row is returned.

```
EMMA> SELECT * FROM hr.employees;

EMPLOYEE_ID FIRST_NAME LAST_NAME EMAIL   MANAGER    SSN         SALARY PHONE
----------- ---------- --------- ------- ---------- ----------- ------
--------
400         Emma       Baker     ebaker  manderson  733-02-9821  8200
555-0400

1 row selected.
```

Example 16-3 Grant cell-level UPDATE access to an employee's own phone number

The following data grant allows employees to view their record and update only their own phone number.

```
CREATE OR REPLACE DATA GRANT hr.employee_update_record
    AS SELECT, UPDATE (phone)
    ON hr.employees
    WHERE email = ORA_END_USER_CONTEXT.username
    TO employee_role;
```

Example 16-4 Grant hierarchical access with column masking for managers

The following data grant allows managers to view their direct reports' records, excluding the social security number (SSN) column, and to update their direct reports' salary.

```
CREATE OR REPLACE DATA GRANT hr.manager_direct_reports
    AS SELECT (ALL COLUMNS EXCEPT ssn), UPDATE (salary)
    ON hr.employees
```

```
    WHERE manager = ORA_END_USER_CONTEXT.username
    TO manager_role;
```

Marvin, who holds both `employee_role` (own-record) and `manager_role` (direct reports) data roles, sees the following result:

```
MARVIN> SELECT * FROM hr.employees;

EMPLOYEE_ID FIRST_NAME LAST_NAME EMAIL      MANAGER    SSN         SALARY
PHONE
----------- ---------- --------- ---------- ---------- ----------- ------
--------
200         Marvin     Anderson  manderson  vwilliams  457-55-5462 12030
555-0200
400         Emma       Baker     ebaker     manderson              8200
555-0400
500         Taylor     Mills     tmills     manderson              9000
555-0500

3 rows selected.
```

Emma and Taylor's SSN values are returned as `NULL` because Marvin does not hold the `SELECT` privilege for the `SSN` column in the `manager_direct_reports` data grant.

Example 16-5 Grant `INSERT` access with column exclusions

The following data grant allows employees to insert values for their own record into all columns except `SSN` and `SALARY`.

```
CREATE OR REPLACE DATA GRANT hr.employees_own_record
    AS SELECT, UPDATE (first_name, phone),
       INSERT (ALL COLUMNS EXCEPT ssn, salary)
    ON hr.employees
    WHERE email = ORA_END_USER_CONTEXT.username
    TO employee_role;
```

Example 16-6 Create a time-bound data grant

The following data grant is active only within a specified time window.

```
CREATE OR REPLACE DATA GRANT hr.temp_access_grant
    AS SELECT
    ON hr.employees
    WHERE email = ORA_END_USER_CONTEXT.username
    TO employee_role
    START TIME TO_TIMESTAMP('2026-03-01 19:30:00',
                            'YYYY-MM-DD HH24:MI:SS')
    END TIME TO_TIMESTAMP_TZ('2026-09-01 19:30:00',
                            'YYYY-MM-DD HH24:MI:SS TZH:TZM');
```

Example 16-7 Avoid cyclic references in data grant predicates

The following command is valid at DDL time but raises an error at query-execution time due to a cyclic reference in the predicate (the predicate queries the same table as the `ON` clause).

```
CREATE OR REPLACE DATA GRANT hr.loop_grant
    AS SELECT
    ON hr.employees
    WHERE employee_id IN (
        SELECT employee_id FROM hr.employees
    )
    TO employee_role;

-- At query time, this raises:
-- ORA-52561: Invalid predicate in data grant.
```

Example 16-8 Grant access to a specific end user

The following data grant targets a specific end user instead of a data role.

```
CREATE OR REPLACE DATA GRANT hr.marvin_emp_access
    AS SELECT
    ON hr.employees
    TO marvin_user;
```

Example 16-9 Grant access to multiple grantees

The following data grant includes multiple grantees.

```
CREATE OR REPLACE DATA GRANT hr.admin_access
    AS SELECT, UPDATE (salary)
    ON hr.employees
    TO employee_role, manager_role;
```

Example 16-10 Create a data grant on a view

The following data grant's target object is a view.

```
CREATE VIEW hr.employees_view AS
    SELECT * FROM hr.employees;

CREATE OR REPLACE DATA GRANT hr.employees_view_grant
    AS SELECT
    ON hr.employees_view
    TO employee_role;
```

Example 16-11 Control access to end-user context attributes using a data grant

The following data grant allows `hcm_admin_role` to access attributes in the end-user context named `hr.hcm_context`.

```
CREATE DATA GRANT update_hcm_attr
    AS SELECT, UPDATE
    ON SYS.END_USER_CONTEXT
```

```
WHERE OWNER = 'HR' AND NAME = 'HCM_CONTEXT'
TO hcm_admin_role;
```

16.3 Drop Data Grants

Use the `DROP DATA GRANT` command to remove a named data grant and all associated data privileges from the database.

After a data grant is dropped, the end users or data roles of the grant no longer hold the data privileges it provided, and they cannot exercise the corresponding data access on the target object.

Required privilege

- Users can drop data grants in their own schema without any specific privilege.
- The `DROP ANY DATA GRANT` system privilege to drop a data grant in another schema (can be granted at the system level or for a specific schema).

Syntax

```
DROP DATA GRANT [IF EXISTS] [schema.]grant_name;
```

Parameters

Parameter	Description
`schema`	The schema name of the data grant. Must be a valid database user. If omitted, the current schema is used. Error `ORA-52553` is raised if the schema (owner) of the data grant is invalid, that is: • The owner is not a valid database user, or • An Oracle Deep Data Security user (whose schema is `XS$NULL`) tries to drop a data grant without specifying a schema name.
`grant_name`	The name of the data grant to drop.

Usage notes and restrictions

- When `IF EXISTS` is specified:
  - If the data grant does not exist, the statement is a no-op. No error is raised.
  - If the data grant exists, it is dropped.
- When `IF EXISTS` is omitted:
  - If the data grant does not exist, `ORA-52552` is raised.
  - If the data grant exists, it is dropped.

For syntax diagrams and additional details, see DROP DATA GRANT in *Oracle AI Database SQL Language Reference*.

Example 16-12 Drop a data grant in own schema

```
DROP DATA GRANT IF EXISTS manager_direct_reports;
```

Example 16-13 Drop a data grant in another schema

```
DROP DATA GRANT IF EXISTS hr.manager_direct_reports;
```

16.4 Use Access Check Functions

Oracle Deep Data Security (Deep Sec) provides two SQL functions that allow applications and users to programmatically inspect data authorization at runtime. These functions are particularly useful for building dynamic user interfaces that reflect a user's actual privileges, or for handling unauthorized `NULL` values in query results.

Topics:

16.4.1 Use the `ORA_IS_COLUMN_AUTHORIZED` Function

The `ORA_IS_COLUMN_AUTHORIZED` function differentiates between a genuine `NULL` value stored in the database and a `NULL` returned due to insufficient Deep Sec user privileges.

This is important because Deep Sec returns `NULL` for unauthorized column cells, which is otherwise indistinguishable from a stored `NULL`.

Return value

- Returns `TRUE` if the current Deep Sec user is authorized to access the column value for the current row, or if the column is not protected by a data grant.
- Returns `FALSE` if the user is not authorized to access the column value.

Syntax

```
ORA_IS_COLUMN_AUTHORIZED(column_reference [, privilege])
  RETURN BOOLEAN

privilege ::= SELECT | INSERT | UPDATE
```

Parameters

Parameter	Description
`column_reference`	A reference to a specific column in a table or view that requires an authorization check for a given row. This parameter does not accept expressions. It can optionally be qualified with schema and object name, for example, `schema.object.column`. If the column name is ambiguous (same column name in multiple objects in the same query), the column reference must be qualified; otherwise `ORA-00918` is raised.
`privilege`	The privilege to check authorization for. You can specify `SELECT`, `INSERT`, or `UPDATE`; `SELECT` is the default. If `DELETE` is specified along with a column reference, `ORA-52540` is raised. If `NULL` is passed, `ORA-01760` is raised.

For syntax diagrams and additional details, see ORA_IS_COLUMN_AUTHORIZED in *Oracle AI Database SQL Language Reference*.

Example 16-14 Display a masked value for unauthorized column values

Marvin, who is both an employee and a manager, uses `ORA_IS_COLUMN_AUTHORIZED` to display a placeholder value (`000-00-0000`) instead of `NULL` for SSN values he is not authorized to view.

```
MARVIN> SELECT first_name, last_name,
            DECODE(
                ORA_IS_COLUMN_AUTHORIZED(ssn),
                FALSE, '000-00-0000',
                TRUE, ssn
            ) AS ssn,
            email,
            manager
        FROM hr.employees;
```

For rows where Marvin is not authorized to see the SSN (his direct reports Emma and Taylor), the function returns `FALSE` and the `DECODE` expression substitutes `000-00-0000`. For his own row, the function returns `TRUE` and the actual value is shown.

```
FIRST_NAME LAST_NAME  SSN          EMAIL       MANAGER
---------- ---------- ------------ ----------- ----------
Marvin     Anderson   457-55-5462  manderson   vwilliams
Emma       Baker      000-00-0000  ebaker      manderson
Taylor     Mills      000-00-0000  tmills      manderson

3 rows selected.
```

16.4.2 Use the `ORA_CHECK_DATA_PRIVILEGE` Function

The `ORA_CHECK_DATA_PRIVILEGE` function allows applications and users to determine which privileges a Deep Sec user holds for specific rows or column values.

This is essential for enforcing security in APIs and controlling dynamic UI elements (for example, enabling or disabling an **Edit** button) based on a user's access rights.

Return value

- Returns `TRUE` for row and column values where the privilege is granted on the specified object.
- If the privilege is not granted to the Deep Sec user, it returns `FALSE`.

Syntax

```
ORA_CHECK_DATA_PRIVILEGE(object, privilege [, column])
  RETURN BOOLEAN
```

Parameters

Parameter	Description
`object`	A schema-qualified object name or object alias.

Parameter	Description
privilege	The CRUD privilege to check. You can specify SELECT, INSERT, UPDATE, or DELETE. DELETE cannot be combined with the column parameter; if combined, ORA-52540 is raised. If NULL is passed, ORA-01760 is raised.
column	Optional. When specified, checks whether the privilege is authorized for the specified column value in the current row, in addition to the row-level check.

For syntax diagrams and additional details, see ORA_CHECK_DATA_PRIVILEGE in *Oracle AI Database SQL Language Reference*.

Example 16-15 Check row-level and column-level privileges

The following query determines which operations Marvin (who holds both employee_role and manager_role data roles) can perform on hr.employees. It checks the row-level SELECT access and the column-level UPDATE access for the phone column.

```
MARVIN> SELECT first_name, last_name,
               manager,
               ORA_CHECK_DATA_PRIVILEGE(emp, 'SELECT')         AS can_view,
               ORA_CHECK_DATA_PRIVILEGE(emp, 'UPDATE', phone)  AS
can_update_phone
       FROM   hr.employees emp;

FIRST_NAME LAST_NAME  MANAGER    CAN_VIEW  CAN_UPDATE_PHONE
---------- ---------- ---------- --------- ---------------
Marvin     Anderson   vwilliams  TRUE      TRUE
Emma       Baker      manderson  TRUE      FALSE
Taylor     Mills      manderson  TRUE      FALSE

3 rows selected.
```

The result confirms that Marvin can view all three rows (his own record and his direct reports) but can only update the phone column for his own row. His direct reports' phone columns are visible, but he cannot update them.

16.5 Enforce Mandatory Data Privileges

Use the SET USE DATA GRANTS ONLY command to enable mandatory access control (MAC) on a specific table or view.

By design, Oracle Deep Data Security (Deep Sec) applies data grants additively on top of standard database privileges. This means that a Deep Sec user who holds both a data grant (restricting access only to their row) and a direct database privilege (such as SELECT granted by the table owner) can bypass the row restriction by accessing the table through a view that runs with the view owner's privileges.

When you enable MAC on an object, Deep Sec users can access the object only through data grants, regardless of any database privileges they hold. This ensures consistent policy enforcement across all access paths, including views defined on the protected object.

> **See Also**
>
> Mandatory Access Control in *Part I: Oracle Deep Data Security Fundamentals*.

Required privilege

- The `SET USE DATA GRANTS ONLY` system privilege (can be granted at the system level or for a specific schema).

Syntax

```
SET USE DATA GRANTS ONLY
    ON [schema.]object [ENABLED | DISABLED];
```

Parameters

Parameter	Description
`schema.object`	The table or view on which to enable or disable MAC.
`ENABLED | DISABLED`	The option that specifies whether `USE DATA GRANTS ONLY` should be enabled or disabled for the specified object. The default value is `ENABLED` if this option is omitted.

Usage notes and restrictions

- MAC enforcement is applicable only to Deep Sec users. Standard database users are not affected by this setting.
- If `USE DATA GRANTS ONLY` is enabled for an object and no data grants exist for that object, Deep Sec users cannot access the object at all.
- MAC propagates through view hierarchies. If `USE DATA GRANTS ONLY` is enabled on an intermediate view (not the base table), a Deep Sec user accessing an outer view that builds on it requires a data grant on the intermediate view as well.
- MAC is enforced regardless of whether the user accesses the object directly or through a view, preventing access-path bypass.

Example 16-16 Enable MAC on a base table

Consider a scenario where the `employee_role` data role has been granted unrestricted access to `hr.employees_view` (a view over `hr.employees`), but is restricted to own-record access on the base table `hr.employees`. Without MAC, Emma (an employee) can see all the rows by querying through the view.

```
-- Grant broad access to the view
CREATE OR REPLACE DATA GRANT hr.employees_view_grant
    AS SELECT
    ON hr.employees_view
    TO employee_role;

-- Grant restricted access to the base table (own record only)
CREATE OR REPLACE DATA GRANT hr.employees_own_record
    AS SELECT
    ON hr.employees
    WHERE email = ORA_END_USER_CONTEXT.username
```

```
    TO employee_role;

-- Without MAC, Emma sees 1 row via the table but all rows via the view.
EMMA> SELECT first_name FROM hr.employees;       -- 1 row
EMMA> SELECT first_name FROM hr.employees_view;   -- 5 rows
```

Enable MAC on `hr.employees` to enforce consistent access.

```
SET USE DATA GRANTS ONLY ON hr.employees ENABLED;

-- With MAC enabled, Emma now sees only her own row through both paths:
EMMA> SELECT first_name FROM hr.employees;

FIRST_NAME
----------
Emma

1 row selected.


EMMA> SELECT first_name FROM hr.employees_view;

FIRST_NAME
----------
Emma

1 row selected.
```

Example 16-17 Enable MAC on an intermediate view

Consider a view `hr.employees_personal_info` defined on `hr.employees_view`, which is itself defined on `hr.employees`. If MAC is enabled on `hr.employees_view` (not the base table), then a Deep Sec user accessing `hr.employees_personal_info` requires a data grant on both `hr.employees_personal_info` and `hr.employees_view`.

```
-- Create view hierarchy
CREATE VIEW hr.employees_view AS
    SELECT * FROM hr.employees;

CREATE VIEW hr.employees_personal_info AS
    SELECT first_name, last_name, phone, email FROM hr.employees_view;

-- Grant access to the outer view
CREATE OR REPLACE DATA GRANT hr.employees_personal_info_grant
    AS SELECT
    ON hr.employees_personal_info
    WHERE email = ORA_END_USER_CONTEXT.username
    TO employee_role;

-- Enable MAC on the intermediate view
SET USE DATA GRANTS ONLY ON hr.employees_view ENABLED;

-- End user queries the employees_personal_info view but sees no data
-- A data grant on hr.employees_view is also required
SELECT first_name FROM hr.employees_personal_info;
```

```
FIRST_NAME
----------
0 rows selected.

-- Grant access to the intermediate view as well:
CREATE OR REPLACE DATA GRANT hr.employees_view_grant
    AS SELECT (first_name, last_name, phone, email)
    ON hr.employees_view
    WHERE email = ORA_END_USER_CONTEXT.username
    TO employee_role;

-- End user can now access their own row
SELECT first_name FROM hr.employees_personal_info;

FIRST_NAME
----------
Emma

1 row selected.
```

Example 16-18 Disable MAC on a base table

```
SET USE DATA GRANTS ONLY ON hr.employees DISABLED;
```

16.6 Data Grant Behavior on Dropping Users or Objects

Learn how existing data grants are affected when standard database users, local end users, data roles, tables, views, or columns are dropped. Understanding this behavior is important for maintaining a consistent and predictable security configuration as your schema evolves.

Topics:

- Impact of Dropping Database Users
- Impact of Dropping End Users or Data Roles
- Impact of Dropping Tables and Views
- Impact of Dropping or Renaming Columns
- Impact of Creating or Replacing Views

16.6.1 Impact of Dropping Database Users

When you drop a standard database user using `DROP USER`, the database automatically drops the following data grants as a cascading effect.

- All data grants created in the schema of the dropped user (that is, all grants for which the dropped user is the schema or owner).
- All data grants created on objects owned by the dropped user.

> **Note**
>
> The cascade removal of data grants occurs automatically as part of the standard `DROP USER` cascade semantics. No separate cleanup is required.

16.6.2 Impact of Dropping End Users or Data Roles

When you drop a local end user or a data role, the following changes are made to existing data grants.

- The dropped end user or data role is removed from the grantee list of existing data grants.
- If the dropped end user or data role was the only grantee in a data grant, the entire data grant is dropped. This behavior ensures that orphaned data grants (grants with no remaining grantees) do not persist after a grantee is removed.

16.6.3 Impact of Dropping Tables and Views

When you drop a table or view, all data grants that reference the dropped object in their `ON` clause are automatically dropped as well.

- This applies regardless of which schema owns the data grants.
- All data grants on the dropped object are removed, including those created by different administrators in different schemas.

> **Note**
>
> Dropping a table with associated data grants bypasses the recycle bin and permanently deletes all related objects. Neither the table nor its data grants can be recovered. This prevents security vulnerabilities that could occur if a table were restored without its associated data grants.

16.6.4 Impact of Dropping or Renaming Columns

When you drop or rename a column in a table, any column-level data grant that references that column becomes invalid and returns an error at run time.

For example, consider the following data grant:

```
CREATE OR REPLACE DATA GRANT employees_own_record AS
    SELECT,
    UPDATE (first_name, phone),
    INSERT (ALL COLUMNS EXCEPT ssn, salary)
    ON hr.employees
    WHERE email = ORA_END_USER_CONTEXT.username
    TO employee_role;
```

After dropping the `phone` column:

```
ALTER TABLE hr.employees DROP COLUMN phone;
```

The column-level data grant for the phone column becomes invalid and returns an error at run time.

You can query the DBA_DATA_GRANTS view to identify data grants with invalid columns and replace the data grants as needed. Adding the column back to the table definition also resolves the invalid data grant issue.

16.6.5 Impact of Creating or Replacing Views

When a view is replaced using CREATE OR REPLACE VIEW and a column referenced by a data grant with column-level privileges is no longer present in the new view definition, the data grant becomes invalid and returns an error at run time.

For example, consider the following view and data grant:

```
CREATE OR REPLACE VIEW hr.employees_view AS
  SELECT * FROM hr.employees;

CREATE OR REPLACE DATA GRANT employees_own_view_record AS
  SELECT, UPDATE (first_name, phone),
  INSERT (ALL COLUMNS EXCEPT ssn, salary)
  ON hr.employees_view
  WHERE email = ORA_END_USER_CONTEXT.username
  TO employee_role;
```

After replacing the view with a definition that removes the phone column:

```
CREATE OR REPLACE VIEW hr.employees_view AS
  SELECT employee_id, first_name, last_name, ssn, salary, email, manager
  FROM hr.employees;
```

The column-level data grant for the phone column becomes invalid and returns an error at run time.

You can query the DBA_DATA_GRANTS view to identify data grants with invalid columns and replace the data grants as needed. Adding the column back to the view definition also resolves the invalid data grant issue.

Part VI
Diagnostics Reference

Monitor, audit, and troubleshoot Oracle Deep Data Security (Deep Sec) operations. This part describes how to configure audit policies to track Deep Sec administrative actions and end-user database activity. It also provides troubleshooting guidelines and data dictionary views to diagnose and resolve issues.

Topics:

17

Audit Oracle Deep Data Security Operations

Use Oracle Unified Auditing to monitor and audit Oracle Deep Data Security (Deep Sec) configuration changes and end-user database activity.

Auditing provides centralized visibility into exactly who performed an operation and when it occurred. For operations performed in an end-user security context, the audit trail also captures the end-user identity and security context identifier. These details assist you in managing security governance, addressing compliance requirements, and troubleshooting issues.

You can audit actions used to configure Deep Sec through the standard Oracle Unified Auditing framework. Create custom audit policies to track these configuration changes. Additionally, you can query the `UNIFIED_AUDIT_TRAIL` view to track both configuration changes made by administrators and database activity by end users in an end-user security context.

Topics:

17.1 Auditable Actions

Oracle Unified Auditing can capture Oracle Deep Data Security (Deep Sec) configuration and security context lifecycle operations. Create custom audit policies with the actions listed below to track policy administration, role assignments, data grant changes, and end-user security context management.

Configuration actions

Database administrators (DBAs) use the following actions to configure Deep Sec. You can include these actions in a unified audit policy:

- `CREATE END USER`
- `ALTER END USER`
- `DROP END USER`
- `CREATE APPLICATION IDENTITY`
- `DROP APPLICATION IDENTITY`
- `CREATE DATA ROLE`
- `DROP DATA ROLE`
- `GRANT DATA ROLE`
- `REVOKE DATA ROLE`
- `CREATE DATA GRANT`
- `DROP DATA GRANT`

- `CREATE END USER CONTEXT`
- `DROP END USER CONTEXT`

Security context lifecycle actions

The following action tracks the creation of end-user security contexts. This action occurs automatically during session establishment and can be used to trace database activity back to specific end-user sessions.

Because this action may generate a large number of audit records, it is not audited when you configure an audit policy with `ACTIONS ALL`. To audit this action, you must specify it explicitly in your audit policy.

- `CREATE END USER SECURITY CONTEXT`

17.2 Create an Audit Policy

Define a custom unified audit policy to audit the commands used to configure Oracle Deep Data Security (Deep Sec), such as creating end users, managing data grants, and assigning data roles.

The following example creates a policy that audits a subset of Deep Sec configuration actions:

1. Create an audit policy.

```
CREATE AUDIT POLICY deep_sec_config_pol
  ACTIONS
    CREATE END USER,
    ALTER END USER,
    DROP END USER,
    CREATE DATA GRANT,
    DROP DATA GRANT;
```

2. Enable the policy.

```
AUDIT POLICY deep_sec_config_pol;
```

   When these actions are audited, the `UNIFIED_AUDIT_TRAIL` records the SQL text, object name, and object type for each operation in the standard audit trail columns. See UNIFIED_AUDIT_TRAIL in *Oracle AI Database Reference*.

You can extend this policy to include additional Deep Sec configuration actions as needed, such as data role and application identity operations.

For more information on creating and managing unified audit policies, see Creating Custom Unified Audit Policies in *Oracle AI Database Security Guide*.

17.3 Audit End-User Activity

Just as you audit regular database users, you can configure audit policies for operations such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE` to capture the actions performed by Oracle Deep Data Security (Deep Sec) end users.

To help you identify the end user who performed a specific action, two new columns are introduced to the `UNIFIED_AUDIT_TRAIL` view:

- `END_USER_NAME`: The name of the end user whose security context was active during the operation.
- `END_USER_SECURITY_CONTEXT_ID`: The identifier of the end-user security context associated with the operation.

These columns are populated when a Deep Sec end user connects to the database and performs actions in an end-user security context. They enable you to trace database activity back to the specific end user, even when multiple users share the same database connection through a connection pool.

> **Note**
>
> Deep Sec configuration actions (such as `CREATE END USER` or `CREATE DATA GRANT`) are performed by database administrators, not end users. These actions do not populate the `END_USER_NAME` or `END_USER_SECURITY_CONTEXT_ID` columns in the audit trail.

You can use the `END_USER_NAME` column to filter for operations performed in an end-user security context.

Failed access attempts in the audit trail are particularly valuable to review. They typically indicate misconfigured data grants, missing role assignments, or incorrect security context attributes. They can also point to malicious attempts by end users to access data outside their authorization.

The following query returns failed operations performed in an end-user security context:

```
SELECT SQL_TEXT,
       END_USER_NAME,
       END_USER_SECURITY_CONTEXT_ID,
       ACTION_NAME,
       OBJECT_NAME,
       RETURN_CODE
  FROM UNIFIED_AUDIT_TRAIL
 WHERE END_USER_NAME IS NOT NULL
   AND RETURN_CODE <> 0;
```

For a complete description of the `UNIFIED_AUDIT_TRAIL` view and its columns, see UNIFIED_AUDIT_TRAIL in *Oracle AI Database Reference*.

18

Troubleshoot Oracle Deep Data Security

Troubleshooting Oracle Deep Data Security (Deep Sec) requires a systematic review of your system configuration, user-privilege setup, data grant enforcement, and diagnostic logging. This chapter provides structured workflows to help you resolve common setup and operational issues.

Each section addresses a specific symptom, identifies probable causes, and provides a step-by-step resolution procedure. This chapter also includes instructions for generating and analyzing trace files.

Topics:

18.1 General Troubleshooting Methodology

Follow this systematic approach when diagnosing Oracle Deep Data Security (Deep Sec) issues.

Prerequisite: Deep Sec is available on Oracle AI Database version 23.26.2 and later. Confirm that your database instance meets this minimum requirement before you begin.

1. **Define the problem clearly**

   Identify the impacted users, schemas, or applications, and gather all observable symptoms.

2. **Isolate error messages**

   Note specific error codes (for example, ORA-xxxxx errors) returned in application logs or SQL*Plus output. Consult the Oracle Database Error Messages reference for initial guidance and troubleshooting steps.

3. **Review session configuration**

   Examine the end-user security contexts, data roles, and context attributes to ensure the user session is correctly configured. See End-User Security Context Issues and Context Attribute Definition or Usage Issues.

4. **Validate user and role privileges**

   Check data grants and data role assignments to verify they align with the expected level of database access. See Access and Privilege Issues.

5. **Generate and review audit logs**

Use Oracle Unified Auditing to track Deep Sec operations and identify misconfigurations.

- Set up audit policies to capture operations such as end user and data role management, data grant changes, and security context activity.
- Query the `UNIFIED_AUDIT_TRAIL` dictionary view for Deep Sec operations. Focus on failed access attempts as these typically indicate misconfigured data grants, missing role assignments, or incorrect security context attributes.

See Audit Oracle Deep Data Security Operations.

6. **Use trace files to troubleshoot**

   Generate Deep Sec trace files at either the system or session level. Ensure that you set the tracing level to *high*. This increases the granularity of diagnostic information recorded for Deep Sec operations during your session. See Enable Diagnostic Tracing.

7. **Consult the knowledge base**

   Review official Oracle documentation and My Oracle Support (MOS) for known issues, workarounds, or applicable patches.

8. **Escalate to Oracle Support**

   For persistent or critical issues, compile reproducible test cases, trace files, and audit logs before logging a Service Request (SR) with Oracle Support. See Escalate to Oracle Support.

> **Tip**
>
> Always test configuration changes in a non-production environment before applying them to production systems. Document every change and follow your organization's change management protocols.

18.2 End-User Connection Issues

Troubleshoot end-user login issues for both locally managed accounts and externally managed accounts (through an IAM system).

Topics:

18.2.1 Local End User Cannot Connect to the Database

If a locally managed end user cannot connect to the database, the account may lack the required privilege or may not be active.

Issue description

A locally managed end user (created with the `CREATE END USER` statement and authenticated by password) cannot connect to the database despite providing the correct credentials.

Probable causes

- The local end user does not have the `CREATE SESSION` privilege, which is required to log in to the database.

- The end user's account status is not active, or the `START_TIME` and `END_TIME` assigned to the end user do not permit login at the current time.

Resolution procedure

1. Create a database role and grant it the `CREATE SESSION` privilege.
2. Create a data role, and grant the database role created in *Step 1* to the data role.
3. Grant this data role to the local end user.
4. Query the `DBA_END_USERS` view. Verify that the `ACCOUNT_STATUS` is active and the `START_TIME` and `END_TIME` values permit the current login time.
5. Attempt the connection again with the end user's credentials.

18.2.2 External End User Cannot Connect Through IAM

If an externally managed end user cannot connect through an IAM provider, the IAM configuration or token claims may not align with the identity provider settings in the database.

Issue description

An externally managed end user (whose identity is maintained in an IAM system, such as Microsoft Entra ID or OCI IAM) cannot connect to the database or cannot establish an end-user security context through the application.

Probable causes

- The end user is not properly configured or does not have the required application roles assigned in the IAM system.
- The identity provider configuration in the database does not match the claims in the OAuth 2.0 access token (audience mismatch, incorrect application URI, or wrong domain URL).

Resolution procedure

1. Verify that the end user exists and is properly configured in the IAM system. Confirm that the user has the required application roles assigned. See Configure Microsoft Entra ID for Application-Mediated Access and Configure OCI IAM for Application-Mediated Access.
2. Verify the identity provider configuration in the database. See Configure the Database for IAM Integration.
3. If the issue persists, enable diagnostic tracing and examine the trace output. See Enable Diagnostic Tracing.

18.3 Access and Privilege Issues

Troubleshoot issues with data grant enforcement, role propagation, and user access to secured database objects.

Topics:

- End User Denied Access to Secured Data
- Data Grants Not Enforced
- Data Role Not Effective in End-User Session

18.3.1 End User Denied Access to Secured Data

If an end user cannot access data they are expected to be authorized for, the user may not be provisioned correctly, or the required data grants or data roles may not be in effect.

Issue description

An end user cannot view or modify data they should have access to. Queries against tables or views protected by Oracle Deep Data Security (Deep Sec) return no rows, return fewer rows than expected, or the database raises an access-related error.

Probable causes

- The end user has not been created as a local end user in the database or as an external end user in the IAM system, or a data role in the database has not been mapped to the end user's IAM application role.
- No data grant exists on the target object for the affected end user or data role.
- The data grant exists but specifies an incorrect privilege, predicate, or grantee.
- The required data roles are not enabled in the current end-user security context.
- The data grant predicate references context attributes whose current values do not satisfy the predicate condition.

Resolution procedure

1. Confirm that the end user is correctly provisioned. Query the `DBA_END_USERS` view for local end users, or verify provisioning in the IAM system for external end users.
2. Query the `DBA_DATA_GRANTS` view and verify that a data grant exists for the target object, grants the required privilege (`SELECT`, `UPDATE`, `INSERT`, or `DELETE`), and names the correct end user or data role as the grantee.
3. If data roles mediate the grant, query `DBA_DATA_ROLES` and `DBA_DATA_ROLE_GRANTS` to confirm that the expected data grants, data roles, and database roles are granted to the data role.
4. Verify that the correct data roles are enabled in the session by querying `V$END_USER_DATA_ROLE` from within the affected session, or by querying `DBA_END_USER_SECURITY_CONTEXT_DATA_ROLES` as a database administrator (DBA) after obtaining the context identifier (`ID`) from `DBA_END_USER_SECURITY_CONTEXTS`.
5. If the data grant includes a `WHERE` predicate that references context attributes, query `END_USER_CONTEXT` from within the session (or `DBA_END_USER_SECURITY_CONTEXT_ATTRIBUTES` as a DBA) and confirm that the attribute values satisfy the predicate logic.
6. If the issue persists, enable diagnostic tracing and examine the trace output. See Enable Diagnostic Tracing.

18.3.2 Data Grants Not Enforced

If data grants do not restrict or permit access as expected, the grant configuration may be incorrect.

Issue description

Data grants defined on a table or view do not restrict or permit data access as expected. End users see more or fewer rows than the data grant predicate specifies, end users see actual values for more or fewer columns than the data grant specifies, or the data grant appears to have no effect on query results.

Probable causes

The data grant targets the wrong object or column(s), specifies an incorrect privilege or predicate, or names the wrong grantee.

Resolution procedure

1. Query the `DBA_DATA_GRANTS` view and verify that each data grant targets the correct object and column(s), specifies the intended privilege, includes the correct `WHERE` predicate, and names the correct grantee.
2. If the issue persists, enable diagnostic tracing and examine the trace output. See Enable Diagnostic Tracing.

18.3.3 Data Role Not Effective in End-User Session

If a granted data role does not take effect during a session, the grant may be outside its valid time window, an intermediate data role may not be enabled, or the IAM role mapping may be incorrect.

Issue description

A data role that is directly or indirectly granted to an end user does not take effect during the session. The end user does not receive the data access associated with that data role.

Probable causes

- The data role grant has a `START_TIME` or `END_TIME` constraint, and the current time falls outside the valid window.
- The data role is granted indirectly through other data roles, and one or more intermediate data roles are not enabled in the session.
- For externally managed end users, the data role in the database is not correctly mapped to the IAM application role.

Resolution procedure

1. Query the `DBA_DATA_ROLES` and `DBA_DATA_ROLE_GRANTS` views. Confirm that the `START_TIME` and `END_TIME` values are properly set and that the current time falls within the valid window.
2. If the data role is granted indirectly (through other data roles), obtain the end user's context ID from `DBA_END_USER_SECURITY_CONTEXTS` and query `DBA_END_USER_SECURITY_CONTEXT_DATA_ROLES` to verify that the intermediate data roles are enabled in the session.

3. For externally managed end users, verify that the IAM application role is correctly mapped to the data role in the database. See Data Role in the Database Is Not Mapped to IAM Application Role.
4. If the issue persists, enable diagnostic tracing and examine the trace output. See Enable Diagnostic Tracing.

18.4 Context Attribute Definition or Usage Issues

Troubleshoot issues related to creating, validating, and using end-user context attributes.

Topics:

- Cannot Create a Context Attribute Definition Object
- Context Attribute Cannot Be Used in Session

18.4.1 Cannot Create a Context Attribute Definition Object

If the `CREATE END USER CONTEXT` statement fails, the executing user may lack the required privilege or the JSON schema payload may be invalid.

Issue description

The `CREATE END USER CONTEXT` statement fails when the administrator attempts to create a custom end-user context attribute definition.

Probable causes

- The executing user does not have the `CREATE END USER CONTEXT` privilege.
- The JSON schema payload provided in the statement is invalid or does not conform to the required schema format.

Resolution procedure

1. Verify that the `CREATE END USER CONTEXT` privilege has been granted to the executing user.
2. Ensure that the JSON schema payload conforms to valid JSON syntax.

18.4.2 Context Attribute Cannot Be Used in Session

If a defined context attribute cannot be read or updated within a session, the attribute definition may be misconfigured, the required privileges may not be granted, or the associated PL/SQL callbacks may be invalid.

Issue description

A context attribute that has been defined cannot be read or updated within an end-user session. Queries or data grant predicates that reference the attribute fail or return unexpected results.

Probable causes

- The context attribute definition was not created properly or does not exist.
- The required `SELECT` (and `UPDATE`, if applicable) privilege on the specific end-user context has not been granted to the end user through a data grant or as a system privilege.

- PL/SQL callbacks used for loading the attribute are invalid, or the `EXECUTE` privilege on the PL/SQL package has not been granted.

Resolution procedure

1. Confirm that the context attribute definition exists and is correctly configured by querying the `DBA_END_USER_CONTEXT_DEFINITIONS` view.
2. Verify that the `SELECT` privilege (and the `UPDATE` privilege, if the attribute must be writable) on the specific end-user context has been granted to the end user through a data grant or as a system privilege.
3. If PL/SQL callbacks are used to load the attribute value, verify the following:
   - The PL/SQL package and procedure referenced in the context definition are valid.
   - The `EXECUTE` privilege on the PL/SQL package has been granted to the appropriate user or role.
4. If the issue persists, enable diagnostic tracing and examine the trace output. See Enable Diagnostic Tracing.

18.5 End-User Security Context Issues

Troubleshoot issues with establishing end-user security contexts.

Topics:

18.5.1 End-User Security Context Not Attached to Session

If the database session does not reflect the end user's identity, the end-user security context may not have been established.

Issue description

The following query returns an empty JSON object (`{}`) instead of the expected end-user identity (the `upn` claim for Entra ID tokens, or the `sub` claim for OCI IAM tokens):

```
SELECT ORA_END_USER_CONTEXT.username FROM dual;
```

Probable cause

A valid end-user security context has not been established. The session operates under standard database privileges and bypasses Oracle Deep Data Security (Deep Sec) enforcement.

Resolution procedure

1. Review the database and application configuration. See Configure the Database and Application.

2. If the issue persists, enable diagnostic tracing and examine the trace output. See Enable Diagnostic Tracing.

18.5.2 Invalid Database-Access Token

If the database rejects a database-access token during security context payload attachment, the token's audience claim may not match the database's identity provider configuration.

Issue description

When the application attempts to attach an end-user security context payload to the database session, the database raises the following error:

```
Invalid database-access token in the end-user security context: Failed to
verify claims
```

Probable cause

The audience claim in the database-access token does not match the corresponding field in the database's identity provider configuration.

Resolution procedure

1. Decode the database-access token (for example, using a JSON Web Token (JWT) decoder) and inspect the audience (`aud`) claim.
2. For Entra ID `v1` tokens, verify that the token's `aud` claim matches the `application_id_uri` field in the database's `IDENTITY_PROVIDER_CONFIG`. For `v2` tokens, verify that it matches the `app_id` field.
3. For OCI IAM identity domain tokens:
   - Confirm that a credential named `OCI_IAM_DOMAIN_DB_CRED$` has been created using the `DBMS_CREDENTIAL.CREATE_CREDENTIAL` procedure.
   - Confirm that the `app_id` field in `IDENTITY_PROVIDER_OAUTH_CONFIG` matches the token's `resource_app_id` claim.
   - Confirm that the `domain_url` in `IDENTITY_PROVIDER_OAUTH_CONFIG` matches the token's `tenant_iss` claim.
4. Correct any mismatched configuration values in the database's identity provider settings, and retry the `setEndUserSecurityContext` call from the application.

18.5.3 Data Role in the Database Is Not Mapped to IAM Application Role

If an IAM application role present in the access token does not activate the expected data role in an end-user security context, the role mapping in the database may be missing or incorrectly formatted.

Issue description

The IAM access token contains a valid application role, but after the application calls `setEndUserSecurityContext`, the corresponding data role does not appear in the `V$END_USER_DATA_ROLE` view. Consequently, the end user does not have the expected data access privileges.

Probable cause

There is no data role in the database that is mapped to the application role in the IAM token.

> **Note**
>
> The database does not raise an error when a role mapping is missing. Instead, it silently ignores the unmapped role and logs a warning in the database trace file (if tracing is enabled).

Resolution procedure

1. Decode the access token to identify the exact application roles present in the token's claims.
2. Query the `DBA_DATA_ROLES` view for the data role that was expected to be enabled, and check its `MAPPED_TO` value.
3. Depending on your token provider, ensure the `MAPPED_TO` string matches the correct format.
   - For Microsoft Entra ID tokens, the mapping must use one of the following formats:

     ```
     'AZURE_APP=<access_token_aud_claim>:AZURE_ROLE=<azure_role>'
     ```

     or

     ```
     'AZURE_ROLE=<azure_role>'
     ```

     Context: The database first checks for the `AZURE_APP` prefix. If it is not present, it defaults to checking `AZURE_ROLE`. The optional `AZURE_APP` prefix uses the token's audience claim to distinguish between identically named application roles defined in different application registrations.
   - For Oracle Cloud Infrastructure (OCI) IAM tokens, the mapping must use the following format:

     ```
     'IAM_OAUTH_GROUP=<oci_iam_group_name>'
     ```

     Context: In an OCI IAM identity domain use case, an application identifier prefix is not needed because the group is shared across applications within the same domain, and the database can only configure a single domain.
4. Alter the data role definition to correct any missing or mismatched `MAPPED_TO` values.
5. Establish a new session (or detach and re-attach the end-user security context) and query `V$END_USER_DATA_ROLE` to verify that the data role is now active.

18.5.4 Data Role Granted to an Application Is Not Enabled

If a data role granted to an application does not take effect after the end-user security context is established, the application identity mapping or the default enablement setting on the role grant may be incorrect.

Issue description

A data role granted to an application does not appear in the `V$END_USER_DATA_ROLE` view after the end-user security context is established. End users connecting through the application do not inherit the data role's privileges.

Probable causes

- The application name in the database is mapped to an incorrect IAM client ID.
- The data role granted to the application is not configured to be enabled by default.

Resolution procedure

1. Decode the database-access token to locate the application identity claim. The specific claim depends on your token type:
   - Entra ID `v1` tokens: Look for the `app_id` claim.
   - Entra ID `v2` tokens: Look for the `azp` claim.
   - OCI IAM client credentials tokens: Look for the `client_id` claim.
2. Query the `DBA_APPLICATION_IDENTITIES` view to inspect the `MAPPED_TO` column for the row corresponding to your application.
3. Validate and correct the mapping format based on your token provider.
   - For Entra ID tokens (`v1` and `v2`), verify the mapping matches the following format, where `<ENTRA_ID_APP_ID>` is the value of the `app_id` or `azp` claim:

     ```
     'AZURE_CLIENT_ID=<ENTRA_ID_APP_ID>'
     Example: 'AZURE_CLIENT_ID=2edc9c9f-8e1e-4ade-8a4a-cc286ed1b899'
     ```

   - For OCI IAM tokens, verify the mapping matches the following format, where `<OCI_IAM_CLIENT_ID>` is the value of the `client_id` claim:

     ```
     'IAM_OAUTH_CLIENT_ID=<OCI_IAM_CLIENT_ID>'
     Example: 'IAM_OAUTH_CLIENT_ID=e83a43ac80d94637bb1958b06929ac32'
     ```

4. Alter the application identity definition to correct any misconfigurations in the `MAPPED_TO` value.
5. Verify the data role grant configuration. Ensure the data role granted to the application is enabled by default. If it is not enabled by default, you must either alter the role grant to enable it or explicitly pass it in the `EndUserSecurityContext` payload's optional data roles list.
6. Establish a new session (or detach and re-attach the end-user security context) and query `V$END_USER_DATA_ROLE` to verify the data role is now enabled.

18.5.5 ORA-00406 Error When Querying End-User Context

If querying `ORA_END_USER_CONTEXT` raises a `COMPATIBLE` parameter error, the database instance may be running at a compatibility level that does not support this feature.

Issue description

After an end-user security context has been established, running the following query raises an error:

```
SQL> SELECT ORA_END_USER_CONTEXT.username FROM dual;
ERROR at line 1:
ORA-00406: COMPATIBLE parameter needs to be 20.0 or greater
```

Probable cause

The database instance's `COMPATIBLE` initialization parameter is set to a value lower than 20.0.

Resolution procedure

1. Check the current value of the `COMPATIBLE` initialization parameter:

   ```
   SHOW PARAMETER compatible;
   ```

2. If the value is below 20.0, update the initialization parameter to 20.0 or higher. Consult the *Oracle AI Database Upgrade Guide* for the procedure to modify this parameter.

18.6 Enable Diagnostic Tracing

Oracle Deep Data Security (Deep Sec) generates trace files at the system or session level to capture detailed diagnostic information regarding data grant enforcement, query rewrites, data definition language (DDL) operations, and security context establishment.

You can use trace files to capture detailed operations when standard troubleshooting does not resolve an issue, or when requested by Oracle Support. Set the tracing level to *high* to ensure that the maximum level of diagnostic information is recorded.

> **Note**
>
> - Tracing generates substantial output that increases storage consumption and may degrade database performance. To minimize the impact, enable tracing for the minimum time necessary, reproduce the issue immediately, and disable tracing as soon as you capture the required data.
> - Trace files capture relevant data (such as query text, applied predicates, and rewritten queries) *only* on the first execution of a query. After the first successful execution, the cursor is cached and subsequent executions bypass the query rewrite phase.

Prerequisites:

- Connect to the pluggable database (PDB) as a database user or a Deep Sec user.

- Ensure you have the `ALTER SYSTEM` privilege to enable system-level tracing, or the `ALTER SESSION` privilege to enable session-level tracing.

Topics:

18.6.1 Trace Data Grants and Query Rewrites

Diagnose issues with data grant enforcement, predicate application, and query rewrite behavior.

1. Enable tracing.
   Run the following commands to enable data security tracing at the system level (all sessions):

   ```
   ALTER SYSTEM SET events '10053 TRACE NAME CONTEXT FOREVER';
   ALTER SYSTEM SET events 'TRACE[XSXDS] disk=high';
   ALTER SYSTEM SET events 'TRACE[XSVPD] disk=high';
   ALTER SYSTEM SET events 'TRACE[DATA_GRANTS] disk=high';
   ```

   Run the following commands to enable data security tracing at the session level (current session only):

   ```
   ALTER SESSION SET events '10053 TRACE NAME CONTEXT FOREVER';
   ALTER SESSION SET events 'TRACE[XSXDS] disk=high';
   ALTER SESSION SET events 'TRACE[XSVPD] disk=high';
   ALTER SESSION SET events 'TRACE[DATA_GRANTS] disk=high';
   ```

2. Reproduce the issue.
   After enabling the trace, immediately execute the queries or operations that are causing the issue.
3. Locate and examine the trace file.
   Run the following query to locate the trace file for your session:

   ```
   SELECT VALUE FROM V$DIAG_INFO WHERE NAME = 'Default Trace File';
   ```

   Open the trace file and inspect it for the generated query text, applied predicates, rewritten query, and any errors related to Deep Sec logic. Share this file with Oracle Support if requested.
4. Disable tracing after troubleshooting is complete.
   At the system level:

   ```
   ALTER SYSTEM SET events '10053 TRACE NAME CONTEXT OFF';
   ALTER SYSTEM SET events 'TRACE[XSXDS] OFF';
   ALTER SYSTEM SET events 'TRACE[XSVPD] OFF';
   ALTER SYSTEM SET events 'TRACE[DATA_GRANTS] OFF';
   ```

At the session level:

```
ALTER SESSION SET events '10053 TRACE NAME CONTEXT OFF';
ALTER SESSION SET events 'TRACE[XSXDS] OFF';
ALTER SESSION SET events 'TRACE[XSVPD] OFF';
ALTER SESSION SET events 'TRACE[DATA_GRANTS] OFF';
```

> **Note**
>
> Session-level tracing is automatically disabled when the database session ends.

18.6.2 Trace End-User and Data Role DDLs

Diagnose issues with DDL statements for end users and data roles (for example, `CREATE END USER` or `CREATE DATA ROLE`).

1. Enable tracing.
   At the system level (all sessions):

   ```
   ALTER SYSTEM SET events 'TRACE[END_USER_AND_DATA_ROLE] disk=high';
   ```

   At the session level (current session only):

   ```
   ALTER SESSION SET events 'TRACE[END_USER_AND_DATA_ROLE] disk=high';
   ```

2. Reproduce the issue.
   After enabling the trace, immediately execute the queries or operations that are causing the issue.
3. Locate and examine the trace file.
   Run the following query to locate the trace file for your session:

   ```
   SELECT VALUE FROM V$DIAG_INFO WHERE NAME = 'Default Trace File';
   ```

   Open the trace file and inspect it for end user and data role creation or modification details, privilege assignments, and any related errors. Share this file with Oracle Support if requested.
4. Disable tracing after troubleshooting is complete.
   At the system level:

   ```
   ALTER SYSTEM SET events 'TRACE[END_USER_AND_DATA_ROLE] OFF';
   ```

   At the session level:

   ```
   ALTER SESSION SET events 'TRACE[END_USER_AND_DATA_ROLE] OFF';
   ```

> **Note**
>
> Session-level tracing is automatically disabled when the database session ends.

18.6.3 Trace End-User Context Attributes

Diagnose issues with context attribute loading, evaluation, and PL/SQL callback execution.

1. Enable tracing.
   At the system level (all sessions):

   ```
   ALTER SYSTEM SET events 'TRACE[END_USER_CONTEXT] disk=high';
   ```

   At the session level (current session only):

   ```
   ALTER SESSION SET events 'TRACE[END_USER_CONTEXT] disk=high';
   ```

2. Reproduce the issue.
   After enabling the trace, immediately execute the queries or operations that are causing the issue.
3. Locate and examine the trace file.
   Run the following query to locate the trace file for your session:

   ```
   SELECT VALUE FROM V$DIAG_INFO WHERE NAME = 'Default Trace File';
   ```

   Open the trace file and inspect it for context attribute loading details, attribute evaluation results, PL/SQL callback execution, and any related errors. Share this file with Oracle Support if requested.
4. Disable tracing after troubleshooting is complete.
   At the system level:

   ```
   ALTER SYSTEM SET events 'TRACE[END_USER_CONTEXT] OFF';
   ```

   At the session level:

   ```
   ALTER SESSION SET events 'TRACE[END_USER_CONTEXT] OFF';
   ```

> **Note**
>
> Session-level tracing is automatically disabled when the database session ends.

18.6.4 Trace End-User Security Contexts

Diagnose issues with end-user security context establishment, token validation, and data role enablement during context attachment.

1. Enable tracing.
   At the system level (all sessions):

   ```
   ALTER SYSTEM SET events 'TRACE[END_USER_SECURITY_CONTEXT] DISK=HIGHEST';
   ALTER SYSTEM SET events 'TRACE[DBIAM] DISK=HIGHEST';
   ALTER SYSTEM SET events 'TRACE[TOKEN] DISK=HIGHEST';
   ALTER SYSTEM SET events 'TRACE[XSSESSION] DISK=HIGHEST';
   ```

```
ALTER SYSTEM SET events 'TRACE[END_USER_CONTEXT] DISK=LOW'; -- Context
attributes in end-user security context
```

At the session level (current session only):

```
ALTER SESSION SET events 'TRACE[END_USER_SECURITY_CONTEXT] DISK=HIGHEST';
ALTER SESSION SET events 'TRACE[DBIAM] DISK=HIGHEST';
ALTER SESSION SET events 'TRACE[TOKEN] DISK=HIGHEST';
ALTER SESSION SET events 'TRACE[XSSESSION] DISK=HIGHEST';
ALTER SESSION SET events 'TRACE[END_USER_CONTEXT] DISK=LOW'; -- Context
attributes in end-user security context
```

2. Reproduce the issue.
   After enabling the trace, immediately execute the queries or operations that are causing the issue.
3. Locate and examine the trace file.
   Run the following query to locate the trace file for your session:

```
SELECT VALUE FROM V$DIAG_INFO WHERE NAME = 'Default Trace File';
```

   Open the trace file and inspect it for context establishment details, token validation results, data role enablement details, and any related errors. Share this file with Oracle Support if requested.
4. Disable tracing after troubleshooting is complete.
   At the system level:

```
ALTER SYSTEM SET events 'TRACE[END_USER_SECURITY_CONTEXT] OFF';
ALTER SYSTEM SET events 'TRACE[DBIAM] OFF';
ALTER SYSTEM SET events 'TRACE[TOKEN] OFF';
ALTER SYSTEM SET events 'TRACE[XSSESSION] OFF';
ALTER SYSTEM SET events 'TRACE[END_USER_CONTEXT] OFF';
```

   At the session level:

```
ALTER SESSION SET events 'TRACE[END_USER_SECURITY_CONTEXT] OFF';
ALTER SESSION SET events 'TRACE[DBIAM] OFF';
ALTER SESSION SET events 'TRACE[TOKEN] OFF';
ALTER SESSION SET events 'TRACE[XSSESSION] OFF';
ALTER SESSION SET events 'TRACE[END_USER_CONTEXT] OFF';
```

> **Note**
>
> When enabling session-level tracing for application-mediated access scenarios, enable the trace on the connection pool user account's database session.

18.7 Escalate to Oracle Support

If troubleshooting does not resolve the issue, escalate to Oracle Support.

Gather the following information before opening a Service Request (SR):

- A detailed description of the symptoms, affected users, schemas, and objects.
- The Oracle AI Database version and patch level.
- Steps to reproduce the issue in a controlled environment.
- Trace files generated by enabling the relevant tracing category.
- Outputs from relevant diagnostic views: `DBA_DATA_GRANTS`, `DBA_DATA_ROLE_GRANTS`, `DBA_DATA_ROLES`, `DBA_END_USERS`, `DBA_APPLICATION_IDENTITIES`, `DBA_END_USER_SECURITY_CONTEXTS`, and `DBA_END_USER_SECURITY_CONTEXT_DATA_ROLES`.
- Relevant `UNIFIED_AUDIT_TRAIL` entries for Oracle Deep Data Security (Deep Sec) operations and access attempts.

After you have collected this information, submit your SR through My Oracle Support (MOS) and attach all diagnostic data and trace files.

19

Oracle Deep Data Security Data Dictionary Views

Review the data dictionary views introduced by Oracle Deep Data Security (Deep Sec) in Oracle AI Database.

These data dictionary views expose metadata about end users, application identities, data roles, data grants, end-user context definitions, and end-user security contexts. Administrators and application developers can use these views to inspect, audit, and manage the fine-grained access control configuration that governs row-level and column-level data authorization within the database.

The views described in this chapter are available in `DBA_`, `ALL_`, and `USER_` forms, following the standard Oracle data dictionary privilege conventions. Dynamic performance views (`V$` views) are also included for monitoring the state of active end-user security contexts at runtime.

Topics:

19.1 Data Authorization Views

These views help you validate data role definitions, data grant assignments, and the resulting row-level and column-level authorization rules.

Summary of views

View or Object Name	Description
DBA_DATA_ROLES	Displays all data roles defined in the database.
DBA_DATA_ROLE_GRANTS	Displays all data role grants and database role grants to data roles.
DBA_DATA_GRANTS	Displays all fine-grained data grants in the database.
ALL_DATA_GRANTS	Displays all fine-grained data grants accessible to the current user.
USER_DATA_GRANTS	Displays all fine-grained data grants owned by the current user.

19.1.1 DBA_DATA_ROLES

Displays all data roles defined in the database. Data roles can be locally defined or externally mapped to IAM attributes (for example, Entra ID roles). Data roles that are externally mapped activate automatically when the corresponding claim is present in the end-user token.

Column	Datatype	NULL	Description
DATA_ROLE	VARCHAR2(128)		Name of the data role
MAPPED_TO	VARCHAR2(4000)		External name (from IAM) that a data role maps to.
ENABLED_BY_DEFAULT	BOOLEAN		Indicates whether the data role is enabled by default (TRUE) or not (FALSE)

19.1.2 DBA_DATA_ROLE_GRANTS

Displays all data role grants and database role grants assigned to data roles, end users, or application identities. Each row represents a single grant with optional temporal validity constraints.

Column	Datatype	NULL	Description
DATA_ROLE	VARCHAR2(128)		Name of the granted role
ROLE_TYPE	VARCHAR2(13)		Type of granted role: • DATA ROLE • DATABASE ROLE
GRANTEE	VARCHAR2(128)		Name of the grantee
GRANTEE_TYPE	VARCHAR2(20)		Type of grantee: • APPLICATION IDENTITY • DATA ROLE • END USER
START_TIME	VARCHAR2(28)		Start time from which the grant is valid
END_TIME	VARCHAR2(28)		End time until which the grant is valid

19.1.3 DBA_DATA_GRANTS

Displays all Oracle Deep Data Security (Deep Sec) data grants in the database, created using the CREATE DATA GRANT statement. Because predicated grants contain unique attributes, such as grant names and active time windows (start or end times), they are not recorded in the standard DBA_TAB_PRIVS view. Instead, Deep Sec data grants appear exclusively in this dedicated dictionary view.

Note the distinction between the two ownership columns: OBJECT_OWNER identifies the schema of the target object being protected, whereas OWNER identifies the schema that owns the data grant itself.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(128)		Owner of the data grant
GRANT_NAME	VARCHAR2(128)		Name of the data grant

Column	Datatype	NULL	Description
PRIVILEGE	VARCHAR2(128)		Type of privilege granted
COLUMN_NAME	VARCHAR2(128)		Name of the column to which the data grant applies Null if the granted privilege is not a column privilege
GRANTED_WITH_ALL_COLUMNS_EXCEPT	VARCHAR2(128)		Name of the column to which the data grant applies, if granted using the ALL COLUMNS EXCEPT clause Null if the granted privilege is not a column privilege or if the privilege was not granted using the ALL COLUMNS EXCEPT clause
OBJECT_OWNER	VARCHAR2(128)		Owner of the object to which the data grant applies
OBJECT_NAME	VARCHAR2(128)	NOT NULL	Name of the object to which the data grant applies
OBJECT_TYPE	VARCHAR2(24)		Type of object to which the data grant applies (such as TABLE, VIEW)
PREDICATE	VARCHAR2(4000)		Predicate used for the data grant
GRANTEE	VARCHAR2(128)		Name of the grantee
GRANTEE_TYPE	VARCHAR2(30)		Type of grantee: • DATA ROLE • END USER
USE_DATA_GRANTS_ONLY	BOOLEAN		Indicates whether USE DATA GRANTS ONLY is enabled for the object to which the data grant applies (TRUE) or not (FALSE)
START_TIME	TIMESTAMP(6) WITH TIME ZONE		Start time for the data grant, if specified
END_TIME	TIMESTAMP(6) WITH TIME ZONE		End time for the data grant, if specified
INVALID_COLUMN_NAME	VARCHAR2(128)		Displays the column name if the data grant on the column is no longer valid due to, for example, a column name change or a dropped column Null if the granted privilege is not a column privilege or if the column is valid

19.1.4 ALL_DATA_GRANTS

Displays all Oracle Deep Data Security (Deep Sec) data grants that are accessible to the current user.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(128)		Owner of the data grant
GRANT_NAME	VARCHAR2(128)		Name of the data grant
PRIVILEGE	VARCHAR2(128)		Type of privilege granted

Column	Datatype	NULL	Description
COLUMN_NAME	VARCHAR2(128)		Name of the column to which the data grant applies Null if the granted privilege is not a column privilege
GRANTED_WITH_ALL_COLUMNS_EXCEPT	VARCHAR2(128)		Name of the column to which the data grant applies, if granted using the ALL COLUMNS EXCEPT clause Null if the granted privilege is not a column privilege or if the privilege was not granted using the ALL COLUMNS EXCEPT clause
OBJECT_OWNER	VARCHAR2(128)		Owner of the object to which the data grant applies
OBJECT_NAME	VARCHAR2(128)	NOT NULL	Name of the object to which the data grant applies
OBJECT_TYPE	VARCHAR2(24)		Type of object to which the data grant applies (such as TABLE, VIEW)
PREDICATE	VARCHAR2(4000)		Predicate used for the data grant
GRANTEE	VARCHAR2(128)		Name of the grantee
GRANTEE_TYPE	VARCHAR2(30)		Type of grantee: • DATA ROLE • END USER
USE_DATA_GRANTS_ONLY	BOOLEAN		Indicates whether USE DATA GRANTS ONLY is enabled for the object to which the data grant applies (TRUE) or not (FALSE)
START_TIME	TIMESTAMP(6) WITH TIME ZONE		Start time for the data grant, if specified
END_TIME	TIMESTAMP(6) WITH TIME ZONE		End time for the data grant, if specified
INVALID_COLUMN_NAME	VARCHAR2(128)		Displays the column name if the data grant on the column is no longer valid due to, for example, a column name change or a dropped column Null if the granted privilege is not a column privilege or if the column is valid

19.1.5 USER_DATA_GRANTS

Displays all Oracle Deep Data Security (Deep Sec) data grants that are owned by the current user.

Column	Datatype	NULL	Description
GRANT_NAME	VARCHAR2(128)		Name of the data grant
PRIVILEGE	VARCHAR2(128)		Type of privilege granted
COLUMN_NAME	VARCHAR2(128)		Name of the column to which the data grant applies Null if the granted privilege is not a column privilege

Column	Datatype	NULL	Description
GRANTED_WITH_ALL_COLUMNS_EXCEPT	VARCHAR2(128)		Name of the column to which the data grant applies, if granted using the ALL COLUMNS EXCEPT clause Null if the granted privilege is not a column privilege or if the privilege was not granted using the ALL COLUMNS EXCEPT clause
OBJECT_OWNER	VARCHAR2(128)		Owner of the object to which the data grant applies
OBJECT_NAME	VARCHAR2(128)	NOT NULL	Name of the object to which the data grant applies
OBJECT_TYPE	VARCHAR2(24)		Type of object to which the data grant applies (such as TABLE, VIEW)
PREDICATE	VARCHAR2(4000)		Predicate used for the data grant
GRANTEE	VARCHAR2(128)		Name of the grantee
GRANTEE_TYPE	VARCHAR2(30)		Type of grantee: • DATA ROLE • END USER
USE_DATA_GRANTS_ONLY	BOOLEAN		Indicates whether USE DATA GRANTS ONLY is enabled for the object to which the data grant applies (TRUE) or not (FALSE)
START_TIME	TIMESTAMP(6) WITH TIME ZONE		Start time for the data grant, if specified
END_TIME	TIMESTAMP(6) WITH TIME ZONE		End time for the data grant, if specified
INVALID_COLUMN_NAME	VARCHAR2(128)		Displays the column name if the data grant on the column is no longer valid due to, for example, a column name change or a dropped column Null if the granted privilege is not a column privilege or if the column is valid

19.2 Identity Views

These views help you validate application identity and end-user configurations.

Summary of views

View or Object Name	Description
DBA_END_USERS	Displays all local end users defined in the database.
USER_END_USERS	Displays the current end user's own account information.
DBA_APPLICATION_IDENTITIES	Displays all application identities defined in the database.

19.2.1 DBA_END_USERS

Displays all local end users defined in the database. Use this view to query account status, authentication type, password expiry, profile associations, and effective time ranges for each end user.

Column	Datatype	NULL	Description
USERNAME	VARCHAR2(128)		Name of the end user
USER_ID	NUMBER	NOT NULL	Unique identifier number for the end user
ACCOUNT_STATUS	VARCHAR2(32)	NOT NULL	Account status of the end user. Valid values: OPEN, EXPIRED, EXPIRED(GRACE), LOCKED, LOCKED(TIMED), EXPIRED & LOCKED, EXPIRED(GRACE) & LOCKED, EXPIRED & LOCKED(TIMED), OPEN & IN ROLLOVER, EXPIRED & IN ROLLOVER, LOCKED & IN ROLLOVER, EXPIRED & LOCKED & IN ROLLOVER, EXPIRED & LOCKED(TIMED) & IN ROLLOVER
LOCK_DATE	DATE		Date on which the end-user account was locked This column is populated only when ACCOUNT_STATUS = LOCKED.
EXPIRY_DATE	DATE		Date of expiration for the end-user account
CREATED_DATE	DATE		Creation date for the end-user account
PROFILE	VARCHAR2(128)		Name of the database profile associated with the end user
AUTHENTICATION_TYPE	VARCHAR2(8)		Authentication mechanism for the end user: • NONE - Not configured with a password • PASSWORD - Configured with a password
PASSWORD_CHANGE_DATE	DATE		Date on which the end user's password was last set This column is populated only when AUTHENTICATION_TYPE = PASSWORD
SCHEMA	VARCHAR2(128)		Schema for object name resolution associated with the end user
START_TIME	VARCHAR2(28)		Effective start time for the end user
END_TIME	VARCHAR2(28)		Effective end time for the end user
MANDATORY_PROFILE_VIOLATION	BOOLEAN		Indicates whether the end user account password violates the mandatory profile's password complexity requirements (TRUE) or not (FALSE)

19.2.2 USER_END_USERS

Displays the current end user's own account information. This view is restricted to the session's authenticated end user and exposes a subset of the columns available in DBA_END_USERS.

Column	Datatype	NULL	Description
USERNAME	VARCHAR2(128)		Name of the end user

Column	Datatype	NULL	Description
USER_ID	NUMBER	NOT NULL	Unique identifier number for the end user
ACCOUNT_STATUS	VARCHAR2(32)	NOT NULL	Account status of the current end user. Valid values: OPEN, EXPIRED, EXPIRED(GRACE), LOCKED, LOCKED(TIMED), EXPIRED & LOCKED, EXPIRED(GRACE) & LOCKED, EXPIRED & LOCKED(TIMED), OPEN & IN ROLLOVER, EXPIRED & IN ROLLOVER, LOCKED & IN ROLLOVER, EXPIRED & LOCKED & IN ROLLOVER, EXPIRED & LOCKED(TIMED) & IN ROLLOVER
LOCK_DATE	DATE		Date on which the end-user account was locked This column is populated only when ACCOUNT_STATUS = LOCKED.
EXPIRY_DATE	DATE		Date of expiration for the end-user account
CREATED_DATE	DATE		Creation date for the end-user account
AUTHENTICATION_TYPE	VARCHAR2(8)		Authentication mechanism for the end user: • NONE - Not configured with a password • PASSWORD - Configured with a password
PASSWORD_CHANGE_DATE	DATE		Date on which the end user's password was last set This column is populated only when AUTHENTICATION_TYPE = PASSWORD.
MANDATORY_PROFILE_VIOLATION	BOOLEAN		Indicates whether the end user account password violates the mandatory profile's password complexity requirements (TRUE) or not (FALSE)

19.2.3 DBA_APPLICATION_IDENTITIES

Displays all application identities defined in the database. Application identities represent service principals or middleware applications.

Column	Datatype	NULL	Description
APPLICATION_NAME	VARCHAR2(128)		Name of the application identity
APPLICATION_ID	NUMBER	NOT NULL	Unique identifier number for the application identity
MAPPED_TO	VARCHAR2(4000)		External identifier string for the application identity

19.3 End-User Context Views

These views help you validate end-user context definitions.

Summary of views

View or Object Name	Description
DBA_END_USER_CONTEXT_DEFINITIONS	Displays all end-user contexts defined in the database.
ALL_END_USER_CONTEXT_DEFINITIONS	Displays all end-user contexts that are accessible to the current user.
USER_END_USER_CONTEXT_DEFINITIONS	Displays all end-user contexts owned by the current user.

19.3.1 DBA_END_USER_CONTEXT_DEFINITIONS

Displays all end-user contexts defined in the database. Each context can define a JSON schema and an associated handler procedure for dynamic attribute resolution.

Column	Datatype	NULL	Description
CONTEXT_OWNER	VARCHAR2(128)	NOT NULL	Owner of the end-user context
CONTEXT_NAME	VARCHAR2(128)	NOT NULL	Name of the end-user context
HANDLER_OWNER	VARCHAR2(128)		Schema of the context-handler procedure
HANDLER_PACKAGE	VARCHAR2(128)		Package containing the context-handler procedure
HANDLER_PROCEDURE	VARCHAR2(128)		Name of the context-handler procedure
HANDLER_STATUS	VARCHAR2(7)		Indicates whether the context-handler procedure is VALID or INVALID
JSON_SCHEMA	JSON		Schema definition for the JSON payload

19.3.2 ALL_END_USER_CONTEXT_DEFINITIONS

Displays all end-user contexts that are accessible to the current user.

Column	Datatype	NULL	Description
CONTEXT_OWNER	VARCHAR2(128)	NOT NULL	Owner of the end-user context
CONTEXT_NAME	VARCHAR2(128)	NOT NULL	Name of the end-user context
HANDLER_OWNER	VARCHAR2(128)		Schema of the context-handler procedure
HANDLER_PACKAGE	VARCHAR2(128)		Package containing the context-handler procedure
HANDLER_PROCEDURE	VARCHAR2(128)		Name of the context-handler procedure
HANDLER_STATUS	VARCHAR2(7)		Indicates whether the context-handler procedure is VALID or INVALID
JSON_SCHEMA	JSON		Schema definition for the JSON payload

19.3.3 USER_END_USER_CONTEXT_DEFINITIONS

Displays all end-user contexts owned by the current user.

Column	Datatype	NULL	Description
CONTEXT_NAME	VARCHAR2(128)	NOT NULL	Name of the end-user context
HANDLER_OWNER	VARCHAR2(128)		Schema of the context-handler procedure
HANDLER_PACKAGE	VARCHAR2(128)		Package containing the context-handler procedure
HANDLER_PROCEDURE	VARCHAR2(128)		Name of the context-handler procedure
HANDLER_STATUS	VARCHAR2(7)		Indicates whether the context-handler procedure is VALID or INVALID
JSON_SCHEMA	JSON		Schema definition for the JSON payload

19.4 End-User Security Context Views

These views help you inspect the active end-user security context and validate the end-user identity and resolved data roles.

Summary of views

View or Object Name	Description
DBA_END_USER_SECURITY_CONTEXTS	Displays all end-user security contexts currently maintained in the database.
DBA_END_USER_SECURITY_CONTEXT_DATA_ROLES	Displays data roles enabled in end-user security contexts.
DBA_END_USER_SECURITY_CONTEXT_ATTRIBUTES	Displays the end-user context attributes as of the last saved state for each end-user security context.
END_USER_CONTEXT	Displays the end-user context attributes in the current end-user security context.
V$END_USER_DATA_ROLE	Displays data roles in the end-user security context for the current request.

19.4.1 DBA_END_USER_SECURITY_CONTEXTS

Displays all end-user security contexts currently maintained in the database. Each security context represents an active or cached end-user session with associated metadata, such as creation time, last access time, and inactivity timeout.

Column	Datatype	NULL	Description
USERNAME	VARCHAR2(128)	NOT NULL	User name for the end-user security context
CONTEXT_ID	RAW(16)	NOT NULL	Unique end-user security context identifier
LOOKUP_KEY	VARCHAR2(1024)		Lookup key associated with the end-user security context
CREATED_TIME	TIMESTAMP(6) WITH TIME ZONE		Creation time of the end-user security context

Column	Datatype	NULL	Description
LAST_ACCESS_TIME	TIMESTAMP(6) WITH TIME ZONE		Most recent access time of the end-user security context
INACTIVE_TIMEOUT_MINUTES	NUMBER(6)		End-user security context inactivity timeout value (in minutes)
USER_TYPE	VARCHAR2(8)		User type of the end-user security context: • EXTERNAL • LOCAL
SECURITY_CONTEXT_TYPE	VARCHAR2(19)		Type of the end-user security context: • DIRECT_LOGON • THROUGH_APPLICATION

19.4.2 DBA_END_USER_SECURITY_CONTEXT_DATA_ROLES

Displays the data roles enabled within each end-user security context. Join on the CONTEXT_ID column with DBA_END_USER_SECURITY_CONTEXTS to correlate roles with specific end-user sessions.

Column	Datatype	NULL	Description
USERNAME	VARCHAR2(128)	NOT NULL	User name for the end-user security context
CONTEXT_ID	RAW(16)	NOT NULL	Unique end-user security context identifier
DATA_ROLE	VARCHAR2(128)	NOT NULL	Name of the enabled data role

19.4.3 DBA_END_USER_SECURITY_CONTEXT_ATTRIBUTES

Displays the end-user context attributes as of the last saved state for each end-user security context.

Column	Datatype	NULL	Description
USERNAME	VARCHAR2(128)	NOT NULL	User name for the end-user security context
CONTEXT_ID	RAW(16)	NOT NULL	Unique identifier of the end-user security context
CONTEXT	JSON		End-user context attributes as of the last saved state, in JSON format

19.4.4 END_USER_CONTEXT

Displays the end-user context attributes in the current end-user security context.

Column	Datatype	NULL	Description
OWNER	VARCHAR2(128)		Owner of the end-user context
NAME	VARCHAR2(128)		Name of the end-user context
CONTEXT	JSON		Context attributes for the current request, in JSON format

19.4.5 V$END_USER_DATA_ROLE

Displays the data roles activated in the end-user security context for the current database request. This dynamic performance view is useful for runtime diagnostics and verifying that expected data roles are active within a session.

Column	Datatype	Description
ROLE_NAME	VARCHAR2(4000)	Name of the activated data role in the current request
CON_ID	NUMBER	The ID of the container to which the data pertains. Possible values include: • 0: This value is used for rows containing data that pertain to the entire CDB; this value is also used for rows in non-CDBs • 1: This value is used for rows containing data that pertain to only the root • *n*: Where *n* is the applicable container ID for the rows containing data

A

Sample Scripts for Configuring Oracle Deep Data Security

Use the SQL scripts in this appendix to automate the three quick-start scenarios in Part II of this guide.

The scripts are grouped by scenario:

- Section A.1 provides the scripts for Configure Oracle Deep Data Security for Direct Logon with Local End Users.
- Section A.2 provides the scripts for Configure Oracle Deep Data Security for Direct Logon with End Users in IAM.
- Section A.3 provides the scripts for Configure Oracle Deep Data Security for a Sample Application.

Each section contains a demo script that creates the demo environment and a cleanup script that removes all objects the demo script created. The corresponding log files capture the output of running each script.

See the respective chapter for the full configuration walkthrough and an explanation of the scenario.

Before you begin

- You must have the `SYS` password for the target pluggable database (PDB).
- For the scenarios in Section A.2 and Section A.3, you must have completed the Microsoft Entra ID configuration steps described in the corresponding chapter.

About the scripts

Each demo script connects as `SYS` to create a helper administrative user named `DEEPSEC_DBA`, which then performs the DBA-level steps the corresponding chapter (in Part II) performs interactively as a named DBA user. The cleanup scripts also connect as `SYS` to drop `DEEPSEC_DBA`. Supply the `SYS` password as the first argument to every script.

A.1 Scripts for Direct Logon with Local End Users

Run these scripts to automate the configuration described in Configure Oracle Deep Data Security for Direct Logon with Local End Users.

Topics:

- How to Run the Scripts
- Scripts
- Log Files

A.1.1 How to Run the Scripts

Run these scripts as SYS from a shell on the database host, in the following order.

1. `01_local_demo.sql`: Sets up and runs the demo scenario. The script reads these parameters in order from either command-line arguments or interactive prompts:
   - `syspasswd`: SYS password for the target PDB.
   - `dbapasswd`: Password for the DEEPSEC_DBA administrative user that the script creates.
   - `passwd`: Password for the `manderson` and `ebaker` end users.
   - `target_pdb`: Service name of the target PDB.
2. `01_local_cleanup.sql`: Removes all objects the demo script created. Run this to reset the environment. The script reads these parameters in order from either command-line arguments or interactive prompts:
   - `syspasswd`: SYS password for the target PDB.
   - `target_pdb`: Service name of the target PDB.

A.1.2 Scripts

Review the source for both scripts used in the scenario.

File	Purpose
01_local_demo.sql	Sets up and runs the demo scenario.
01_local_cleanup.sql	Removes all objects created by the demo script.

A.1.2.1 01_local_demo.sql

The source for the `01_local_demo.sql` script.

```
SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 200
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100

COL EMPLOYEE_ID FORMAT 9999
COL FIRST_NAME  FORMAT A12
COL LAST_NAME   FORMAT A12
COL EMAIL       FORMAT A24
COL MANAGER     FORMAT A24
COL SSN         FORMAT A12
COL SALARY      FORMAT 9999999
COL PHONE       FORMAT A12

define syspasswd=&1
define dbapasswd=&2
define passwd=&3
define target_pdb=&4
```

```
spool 01_local_demo.log

-----------------------------------------------------------------------
--  HR Security Demo - Local End User Scenario
-----------------------------------------------------------------------
-- This script creates the demo environment:
--  * the administrative user
--  * the HR schema and employee table
--  * sample employee rows
--  * local end users for the SQL*Plus demo
--  * the session role, data roles, and role assignments
--  * the data grants that enforce row and column security
--  * row and column filtering for local end users
--  * end user connects directly to the database using their own identity

-----------------------------------------------------------------------
-- 1. Connect as SYS to the target PDB
-----------------------------------------------------------------------

connect sys/&syspasswd@&target_pdb as sysdba

-----------------------------------------------------------------------
-- 2. Create the administration user
-----------------------------------------------------------------------

-- This local user is used to manage the demo environment.
create user DEEPSEC_DBA identified by &dbapasswd;
grant DBA to DEEPSEC_DBA;

-----------------------------------------------------------------------
-- 3. Connect as the administration user
-----------------------------------------------------------------------

connect DEEPSEC_DBA/&dbapasswd@&target_pdb

-----------------------------------------------------------------------
-- 4. Create tablespace and HR schema
-----------------------------------------------------------------------

-- Create the tablespace used by the HR demo schema.
-- Oracle Managed Files (OMF) controls the datafile location.
CREATE TABLESPACE users DATAFILE SIZE 100M AUTOEXTEND ON;

-- Create the HR schema that owns the employee table.
-- NO AUTHENTICATION means this schema cannot be logged into directly;
-- it is accessed only through object grants.
CREATE USER hr NO AUTHENTICATION
  DEFAULT TABLESPACE users
  QUOTA UNLIMITED ON users;

-----------------------------------------------------------------------
-- 5. Create the employee table
-----------------------------------------------------------------------

-- Create the data model used by all demo scenarios.
```

```
CREATE TABLE hr.employees (
    employee_id NUMBER PRIMARY KEY,
    first_name  VARCHAR2(50),
    last_name   VARCHAR2(50),
    email       VARCHAR2(128),
    manager     VARCHAR2(128),
    ssn         VARCHAR2(20),
    salary      NUMBER(10,2),
    phone       VARCHAR2(20)
);

-----------------------------------------------------------------------
-- 6. Load sample data
-----------------------------------------------------------------------

INSERT INTO hr.employees VALUES
(100,'Victoria','Williams','vwilliams',NULL,'219-09-9999',13000,'555-0100');
INSERT INTO hr.employees VALUES
(200,'Marvin','Anderson','manderson','vwilliams','457-55-5462',12030,'555-0200
');
INSERT INTO hr.employees VALUES
(300,'Chris','Evans','cevans','vwilliams','321-12-4567',6900,'555-0300');
INSERT INTO hr.employees VALUES
(400,'Emma','Baker','ebaker','manderson','733-02-9821',8200,'555-0400');
INSERT INTO hr.employees VALUES
(500,'Taylor','Mills','tmills','manderson','558-76-1243',9000,'555-0500');
COMMIT;

-----------------------------------------------------------------------
-- 7. Create local end user identities
-----------------------------------------------------------------------

-- Local end users whose names are matching the email column above
CREATE END USER "manderson" IDENTIFIED BY &passwd;
CREATE END USER "ebaker" IDENTIFIED BY &passwd;

-----------------------------------------------------------------------
-- 8. Create data roles
-----------------------------------------------------------------------

-- Employees can see only their own row.
CREATE OR REPLACE DATA ROLE employee_role;

-- Managers can see their direct reports (with SSN masked).
CREATE OR REPLACE DATA ROLE manager_role;

-----------------------------------------------------------------------
-- 9. Create the session role
-----------------------------------------------------------------------

-- A standard database role that carries the CREATE SESSION privilege.
-- Granting this role to a data role allows end users who hold that
-- data role to establish a database session.
CREATE ROLE db_role;
GRANT CREATE SESSION TO db_role;
```

```
-------------------------------------------------------------------------
-- 10. Grant create session privilege to data roles
-------------------------------------------------------------------------

GRANT db_role TO employee_role;
GRANT db_role TO manager_role;

-------------------------------------------------------------------------
-- 11. Assign data roles to end users
-------------------------------------------------------------------------

-- Marvin holds both roles: he can see his own record as an employee,
-- and see his direct reports (Emma, Taylor) as a manager.
GRANT DATA ROLE manager_role TO "manderson";
GRANT DATA ROLE employee_role TO "manderson";

-- Emma holds only the employee role: she can see only her own record.
GRANT DATA ROLE employee_role TO "ebaker";

-------------------------------------------------------------------------
-- 12. Create data grants
-------------------------------------------------------------------------

-- Employee access rule:
-- A user can see only the row where the email matches their identity.
CREATE OR REPLACE DATA GRANT hr.employees_own_record
  AS SELECT
  ON hr.employees
  WHERE email = ORA_END_USER_CONTEXT.username
  TO employee_role;

-- Manager access rule:
-- A manager can see rows where the manager column matches their identity.
-- The ALL COLUMNS EXCEPT clause masks SSN for those rows.
CREATE OR REPLACE DATA GRANT hr.manager_direct_reports
  AS SELECT (ALL COLUMNS EXCEPT ssn)
  ON hr.employees
  WHERE manager = ORA_END_USER_CONTEXT.username
  TO manager_role;

-------------------------------------------------------------------------
-- 13. Query as Marvin Anderson (employee and manager)
-------------------------------------------------------------------------

-- Marvin holds both employee_role and manager_role.
connect "manderson"/&passwd@&target_pdb

-- Confirm the active end user identity.
SELECT ORA_END_USER_CONTEXT.username FROM dual;

-- Expected results:
--   employee_role:  Marvin's own row, including his SSN and salary.
--   manager_role:   Emma Baker and Taylor Mills (direct reports),
--                   with their SSN column masked.
SELECT * FROM hr.employees;
```

```
-------------------------------------------------------------------------
-- 14. Query as Emma Baker (employee only)
-------------------------------------------------------------------------

-- Emma holds only employee_role.
connect "ebaker"/&passwd@&target_pdb

-- Confirm the active end user identity.
SELECT ORA_END_USER_CONTEXT.username FROM dual;

-- Expected result: Emma's own row only.
-- No other rows are visible; SSN and salary are accessible
-- because employee_role grants full column access to her own record.
SELECT * FROM hr.employees;

spool off
```

A.1.2.2 01_local_cleanup.sql

The source for the `01_local_cleanup.sql` script.

```
SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 200
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100

define syspasswd=&1
define target_pdb=&2

spool 01_local_cleanup.log

-- This script removes all objects created by the demo so the
-- environment can be reused cleanly.

-------------------------------------------------------------------------
-- 1. Connect as SYS
-------------------------------------------------------------------------

connect sys/&syspasswd@&target_pdb as sysdba

-------------------------------------------------------------------------
-- 2. Remove local end users
-------------------------------------------------------------------------

DROP END USER "manderson";
DROP END USER "ebaker";

-------------------------------------------------------------------------
-- 3. Remove the HR schema
-------------------------------------------------------------------------

DROP USER hr CASCADE;
```

```
-----------------------------------------------------------------------
-- 4. Remove data roles
-----------------------------------------------------------------------

DROP DATA ROLE employee_role;
DROP DATA ROLE manager_role;

-----------------------------------------------------------------------
-- 5. Remove the database role
-----------------------------------------------------------------------

DROP ROLE db_role;

-----------------------------------------------------------------------
-- 6. Remove the administration user and tablespace
-----------------------------------------------------------------------

DROP USER DEEPSEC_DBA CASCADE;
DROP TABLESPACE users INCLUDING CONTENTS AND DATAFILES;

spool off
```

A.1.3 Log Files

Review the output generated by running each script.

File	Purpose
01_local_demo.log	Output generated by running `01_local_demo.sql`.
01_local_cleanup.log	Output generated by running `01_local_cleanup.sql`.

A.1.3.1 01_local_demo.log

The output generated by running `01_local_demo.sql`.

```
SQL>
SQL> ------------------------------------------------------------------------
SQL> --  HR Security Demo - Local End User Scenario
SQL> ------------------------------------------------------------------------
SQL> -- This script creates the demo environment:
SQL> --  * the administrative user
SQL> --  * the HR schema and employee table
SQL> --  * sample employee rows
SQL> --  * local end users for the SQL*Plus demo
SQL> --  * the session role, data roles, and role assignments
SQL> --  * the data grants that enforce row and column security
SQL> --  * row and column filtering for local end users
SQL> --  * end user connects directly to the database using their own identity
SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 1. Connect as SYS to the target PDB
SQL> ------------------------------------------------------------------------
SQL>
```

```
SQL> connect sys/&syspasswd@&target_pdb as sysdba
Connected.
SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 2. Create the administration user
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- This local user is used to manage the demo environment.
SQL> create user DEEPSEC_DBA identified by &dbapasswd;
old   1: create user DEEPSEC_DBA identified by &dbapasswd
new   1: create user DEEPSEC_DBA identified by dba_pwd

User created.

SQL> grant DBA to DEEPSEC_DBA;

Grant succeeded.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 3. Connect as the administration user
SQL> ------------------------------------------------------------------------
SQL>
SQL> connect DEEPSEC_DBA/&dbapasswd@&target_pdb
Connected.
SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 4. Create tablespace and HR schema
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- Create the tablespace used by the HR demo schema.
SQL> -- Oracle Managed Files (OMF) controls the datafile location.
SQL> CREATE TABLESPACE users DATAFILE SIZE 100M AUTOEXTEND ON;

Tablespace created.

SQL>
SQL> -- Create the HR schema that owns the employee table.
SQL> -- NO AUTHENTICATION means this schema cannot be logged into directly;
SQL> -- it is accessed only through object grants.
SQL> CREATE USER hr NO AUTHENTICATION
  2    DEFAULT TABLESPACE users
  3    QUOTA UNLIMITED ON users;

User created.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 5. Create the employee table
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- Create the data model used by all demo scenarios.
SQL> CREATE TABLE hr.employees (
  2      employee_id NUMBER PRIMARY KEY,
  3      first_name  VARCHAR2(50),
  4      last_name   VARCHAR2(50),
```

```
  5      email       VARCHAR2(128),
  6      manager     VARCHAR2(128),
  7      ssn         VARCHAR2(20),
  8      salary      NUMBER(10,2),
  9      phone       VARCHAR2(20)
 10  );

Table created.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 6. Load sample data
SQL> ------------------------------------------------------------------------
SQL>
SQL> INSERT INTO hr.employees VALUES
(100,'Victoria','Williams','vwilliams',NULL,'219-09-9999',13000,'555-0100');

1 row created.

SQL> INSERT INTO hr.employees VALUES
(200,'Marvin','Anderson','manderson','vwilliams','457-55-5462',12030,'555-0200
');

1 row created.

SQL> INSERT INTO hr.employees VALUES
(300,'Chris','Evans','cevans','vwilliams','321-12-4567',6900,'555-0300');

1 row created.

SQL> INSERT INTO hr.employees VALUES
(400,'Emma','Baker','ebaker','manderson','733-02-9821',8200,'555-0400');

1 row created.

SQL> INSERT INTO hr.employees VALUES
(500,'Taylor','Mills','tmills','manderson','558-76-1243',9000,'555-0500');

1 row created.

SQL> COMMIT;

Commit complete.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 7. Create local end user identities
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- Local end users whose names are matching the email column above
SQL> CREATE END USER "manderson" IDENTIFIED BY &passwd;
old   1: CREATE END USER "manderson" IDENTIFIED BY &passwd
new   1: CREATE END USER "manderson" IDENTIFIED BY pwd

End user created.
```

```
SQL> CREATE END USER "ebaker" IDENTIFIED BY &passwd;
old   1: CREATE END USER "ebaker" IDENTIFIED BY &passwd
new   1: CREATE END USER "ebaker" IDENTIFIED BY pwd

End user created.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 8. Create data roles
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- Employees can see only their own row.
SQL> CREATE OR REPLACE DATA ROLE employee_role;

Data role created.

SQL>
SQL> -- Managers can see their direct reports (with SSN masked).
SQL> CREATE OR REPLACE DATA ROLE manager_role;

Data role created.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 9. Create the session role
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- A standard database role that carries the CREATE SESSION privilege.
SQL> -- Granting this role to a data role allows end users who hold that
SQL> -- data role to establish a database session.
SQL> CREATE ROLE db_role;

Role created.

SQL> GRANT CREATE SESSION TO db_role;

Grant succeeded.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 10. Grant create session privilege to data roles
SQL> ------------------------------------------------------------------------
SQL>
SQL> GRANT db_role TO employee_role;

Grant succeeded.

SQL> GRANT db_role TO manager_role;

Grant succeeded.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 11. Assign data roles to end users
SQL> ------------------------------------------------------------------------
SQL>
```

```
SQL> -- Marvin holds both roles: he can see his own record as an employee,
SQL> -- and see his direct reports (Emma, Taylor) as a manager.
SQL> GRANT DATA ROLE manager_role TO "manderson";

Grant succeeded.

SQL> GRANT DATA ROLE employee_role TO "manderson";

Grant succeeded.

SQL>
SQL> -- Emma holds only the employee role: she can see only her own record.
SQL> GRANT DATA ROLE employee_role TO "ebaker";

Grant succeeded.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 12. Create data grants
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- Employee access rule:
SQL> -- A user can see only the row where the email matches their identity.
SQL> CREATE OR REPLACE DATA GRANT hr.employees_own_record
  2    AS SELECT
  3    ON hr.employees
  4    WHERE email = ORA_END_USER_CONTEXT.username
  5    TO employee_role;

Data grant created.

SQL>
SQL> -- Manager access rule:
SQL> -- A manager can see rows where the manager column matches their
identity.
SQL> -- The ALL COLUMNS EXCEPT clause masks SSN for those rows.
SQL> CREATE OR REPLACE DATA GRANT hr.manager_direct_reports
  2    AS SELECT (ALL COLUMNS EXCEPT ssn)
  3    ON hr.employees
  4    WHERE manager = ORA_END_USER_CONTEXT.username
  5    TO manager_role;

Data grant created.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 13. Query as Marvin Anderson (employee and manager)
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- Marvin holds both employee_role and manager_role.
SQL> connect "manderson"/&passwd@&target_pdb
Connected.
SQL>
SQL> -- Confirm the active end user identity.
SQL> SELECT ORA_END_USER_CONTEXT.username FROM dual;
```

```
USERNAME
--------------------------------------------------------------------------------
--
"manderson"

1 row selected.

SQL>
SQL> -- Expected results:
SQL> --   employee_role:  Marvin's own row, including his SSN and salary.
SQL> --   manager_role:   Emma Baker and Taylor Mills (direct reports),
SQL> --                   with their SSN column masked.
SQL> SELECT * FROM hr.employees;

EMPLOYEE_ID FIRST_NAME   LAST_NAME    EMAIL
MANAGER                  SSN            SALARY PHONE
----------- ------------ ------------ ----------------------
----------------------- ------------ -------- ------------
        200 Marvin       Anderson     manderson
vwilliams                457-55-5462     12030 555-0200
        400 Emma         Baker        ebaker
manderson                                 8200 555-0400
        500 Taylor       Mills        tmills
manderson                                 9000 555-0500

3 rows selected.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 14. Query as Emma Baker (employee only)
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- Emma holds only employee_role.
SQL> connect "ebaker"/&passwd@&target_pdb
Connected.
SQL>
SQL> -- Confirm the active end user identity.
SQL> SELECT ORA_END_USER_CONTEXT.username FROM dual;

USERNAME
--------------------------------------------------------------------------------
--
"ebaker"

1 row selected.

SQL>
SQL> -- Expected result: Emma's own row only.
SQL> -- No other rows are visible; SSN and salary are accessible
SQL> -- because employee_role grants full column access to her own record.
SQL> SELECT * FROM hr.employees;

EMPLOYEE_ID FIRST_NAME   LAST_NAME    EMAIL
MANAGER                  SSN            SALARY PHONE
----------- ------------ ------------ ----------------------
----------------------- ------------ -------- ------------
```

```
        400 Emma         Baker        ebaker
manderson                733-02-9821      8200 555-0400

1 row selected.

SQL>
SQL> spool off
```

A.1.3.2 01_local_cleanup.log

The output generated by running `01_local_cleanup.sql`.

```
SQL>
SQL> -- This script removes all objects created by the demo so the
SQL> -- environment can be reused cleanly.
SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 1. Connect as SYS
SQL> ------------------------------------------------------------------------
SQL>
SQL> connect sys/&syspasswd@&target_pdb as sysdba
Connected.
SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 2. Remove local end users
SQL> ------------------------------------------------------------------------
SQL>
SQL> DROP END USER "manderson";

End user dropped.

SQL> DROP END USER "ebaker";

End user dropped.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 3. Remove the HR schema
SQL> ------------------------------------------------------------------------
SQL>
SQL> DROP USER hr CASCADE;

User dropped.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 4. Remove data roles
SQL> ------------------------------------------------------------------------
SQL>
SQL> DROP DATA ROLE employee_role;

Data role dropped.

SQL> DROP DATA ROLE manager_role;
```

```
Data role dropped.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 5. Remove the database role
SQL> ------------------------------------------------------------------------
SQL>
SQL> DROP ROLE db_role;

Role dropped.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 6. Remove the administration user and tablespace
SQL> ------------------------------------------------------------------------
SQL>
SQL> DROP USER DEEPSEC_DBA CASCADE;

User dropped.

SQL> DROP TABLESPACE users INCLUDING CONTENTS AND DATAFILES;

Tablespace dropped.

SQL>
SQL> spool off
```

A.2 Scripts for Direct Logon with End Users in IAM

Run these scripts to automate the configuration described in Configure Oracle Deep Data Security for Direct Logon with End Users in IAM.

These scripts perform database configuration only. End-user validation occurs interactively when a user signs in to Microsoft Entra ID from a SQL*Plus client, as described in the chapter.

Topics:

A.2.1 How to Run the Scripts

Run these scripts as `SYS` from a shell on the database host, in the following order.

1. `02_entra_demo.sql`: Sets up the demo environment. The script reads these parameters in order from either command-line arguments or interactive prompts:
   - `syspasswd`: `SYS` password for the target PDB.
   - `dbapasswd`: Password for the `DEEPSEC_DBA` administrative user that the script creates.
   - `target_pdb`: Service name of the target PDB.
   - `TENANT_ID`: Directory (tenant) ID of your Microsoft Entra ID tenant.

- `DB_APP_ID`: Application (client) ID of the `OracleDB_Resource` application registration in Entra ID. See Register the Database Resource.
- `DB_APP_ID_URI`: Application ID URI of the `OracleDB_Resource` application registration in Entra ID.
- `ENTRA_DOMAIN`: Your Microsoft Entra ID domain (for example, `supremo.onmicrosoft.com`). Used to construct sample employee email addresses.

2. `02_entra_cleanup.sql`: Removes all objects the demo script created. Run this to reset the environment. The script reads these parameters in order from either command-line arguments or interactive prompts:
   - `syspasswd`: `SYS` password for the target PDB.
   - `target_pdb`: Service name of the target PDB.

A.2.2 Scripts

Review the source for both scripts used in the scenario.

File	Purpose
02_entra_demo.sql	Sets up the demo environment.
02_entra_cleanup.sql	Removes all objects created by the demo script.

A.2.2.1 02_entra_demo.sql

The source for the `02_entra_demo.sql` script.

```
SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 200
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100

define syspasswd=&1
define dbapasswd=&2
define target_pdb=&3
define TENANT_ID=&4
define DB_APP_ID=&5
define DB_APP_ID_URI=&6
define ENTRA_DOMAIN=&7

spool 02_entra_demo.log

-----------------------------------------------------------------------
--  HR Security Demo - Direct Logon with IAM Users
-----------------------------------------------------------------------
-- This script creates the demo environment:
-- * Configure the database to accept external identities
-- * Create data roles, which map to Entra ID roles
-- * Create sample HR application data
-- * Define data grants to restrict access to Marvin and Emma based on their
roles
```

```
-----------------------------------------------------------------------
-- 1. Connect as SYS to the target PDB
-----------------------------------------------------------------------

connect sys/&syspasswd@&target_pdb as sysdba

-----------------------------------------------------------------------
-- 2. Create the administration user
-----------------------------------------------------------------------

-- This local user is used to manage the demo environment.
create user DEEPSEC_DBA identified by &dbapasswd;
grant DBA to DEEPSEC_DBA;

-----------------------------------------------------------------------
-- 3. Connect as the administration user
-----------------------------------------------------------------------

connect DEEPSEC_DBA/&dbapasswd@&target_pdb

-----------------------------------------------------------------------
-- 4. Configure the identity provider
-----------------------------------------------------------------------

-- Enable Microsoft Entra ID as the identity provider for this PDB.
-- SCOPE = BOTH applies the setting to both the CDB and the PDB.
-- The database will use these values to validate incoming Entra tokens.
ALTER SYSTEM SET IDENTITY_PROVIDER_TYPE = AZURE_AD SCOPE = BOTH;

-- Supply the Entra tenant details so the database can validate tokens.
--   application_id_uri: the Application ID URI of the EmployeeRecordsDB
--                       Entra app registration  [DB_APP_ID_URI]
--   tenant_id:          the Directory (tenant) ID of your Entra tenant
[TENANT_ID]
--   app_id:             the Application (client) ID of EmployeeRecordsDB
[DB_APP_ID]
ALTER SYSTEM SET IDENTITY_PROVIDER_CONFIG = '{
  "application_id_uri": "&DB_APP_ID_URI",
  "tenant_id": "&TENANT_ID",
  "app_id": "&DB_APP_ID"
}' SCOPE = BOTH;

-----------------------------------------------------------------------
-- 5. Create data roles
-----------------------------------------------------------------------

-- Employees can see only their own row.
CREATE DATA ROLE employee_role MAPPED TO 'AZURE_ROLE=EMPLOYEE';

-- Managers can see their direct reports (with SSN masked).
CREATE DATA ROLE manager_role MAPPED TO 'AZURE_ROLE=MANAGER';

-----------------------------------------------------------------------
-- 6. Create the session role
-----------------------------------------------------------------------
```

```
-- A standard database role that carries the CREATE SESSION privilege.
-- Granting this role to a data role allows end users who hold that
-- data role to establish a database session.
CREATE ROLE db_role;
GRANT CREATE SESSION TO db_role;

-------------------------------------------------------------------------
-- 7. Grant create session privilege to data roles
-------------------------------------------------------------------------

GRANT db_role TO employee_role;
GRANT db_role TO manager_role;

-------------------------------------------------------------------------
-- 8. Create tablespace and HR schema
-------------------------------------------------------------------------

-- Create the tablespace used by the HR demo schema.
-- Oracle Managed Files (OMF) controls the datafile location.
CREATE TABLESPACE users DATAFILE SIZE 100M AUTOEXTEND ON;

-- Create the HR schema that owns the employee table.
-- NO AUTHENTICATION means this schema cannot be logged into directly;
-- it is accessed only through object grants.
CREATE USER hr NO AUTHENTICATION
  DEFAULT TABLESPACE users
  QUOTA UNLIMITED ON users;

-------------------------------------------------------------------------
-- 9. Create the employee table
-------------------------------------------------------------------------

-- Create the data model
CREATE TABLE hr.employees (
    employee_id NUMBER PRIMARY KEY,
    first_name  VARCHAR2(50),
    last_name   VARCHAR2(50),
    email       VARCHAR2(128),
    manager     VARCHAR2(128),
    ssn         VARCHAR2(20),
    salary      NUMBER(10,2),
    phone       VARCHAR2(20)
);

-------------------------------------------------------------------------
-- 10. Load sample data
-------------------------------------------------------------------------

-- Email and manager values use UPN format (firstname@ENTRA_DOMAIN)
-- The Entra-issued JWT and passes as ORA_END_USER_CONTEXT.username.
INSERT INTO hr.employees VALUES
(100,'Victoria','Williams','victoria@&ENTRA_DOMAIN',NULL,'219-09-9999',13000,'
555-0100');
INSERT INTO hr.employees VALUES
(200,'Marvin','Anderson','marvin@&ENTRA_DOMAIN','victoria@&ENTRA_DOMAIN','457-
```

```
55-5462',12030,'555-0200');
INSERT INTO hr.employees VALUES
(300,'Chris','Evans','chris@&ENTRA_DOMAIN','victoria@&ENTRA_DOMAIN','321-12-45
67',6900,'555-0300');
INSERT INTO hr.employees VALUES
(400,'Emma','Baker','emma@&ENTRA_DOMAIN','marvin@&ENTRA_DOMAIN','733-02-9821',
8200,'555-0400');
INSERT INTO hr.employees VALUES
(500,'Taylor','Mills','taylor@&ENTRA_DOMAIN','marvin@&ENTRA_DOMAIN','558-76-12
43',9000,'555-0500');
COMMIT;

-----------------------------------------------------------------------
-- 11. Create data grants
-----------------------------------------------------------------------

-- Employee access rule:
-- A user can see only the row where the email matches their identity.
CREATE OR REPLACE DATA GRANT hr.employees_own_record
  AS SELECT
  ON hr.employees
  WHERE email = ORA_END_USER_CONTEXT.username
  TO employee_role;

-- Manager access rule:
-- A manager can see rows where the manager column matches their identity.
-- The ALL COLUMNS EXCEPT clause masks SSN for those rows.
CREATE OR REPLACE DATA GRANT hr.manager_direct_reports
  AS SELECT (ALL COLUMNS EXCEPT ssn)
  ON hr.employees
  WHERE manager = ORA_END_USER_CONTEXT.username
  TO manager_role;

spool off
```

A.2.2.2 02_entra_cleanup.sql

The source for the `02_entra_cleanup.sql` script.

```
SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 200
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100

define syspasswd=&1
define target_pdb=&2

spool 02_entra_cleanup.log

-- This script removes all objects created by the demo so the
-- environment can be reused cleanly.
```

```
-----------------------------------------------------------------------
-- 1. Connect as SYS
-----------------------------------------------------------------------

connect sys/&syspasswd@&target_pdb as sysdba

-----------------------------------------------------------------------
-- 2. Remove the HR schema
-----------------------------------------------------------------------

DROP USER hr CASCADE;

-----------------------------------------------------------------------
-- 3. Remove data roles
-----------------------------------------------------------------------

DROP DATA ROLE employee_role;
DROP DATA ROLE manager_role;

-----------------------------------------------------------------------
-- 4. Remove the database role
-----------------------------------------------------------------------

DROP ROLE db_role;

-----------------------------------------------------------------------
-- 5. Remove the administration user and tablespace
-----------------------------------------------------------------------

DROP USER DEEPSEC_DBA CASCADE;
DROP TABLESPACE users INCLUDING CONTENTS AND DATAFILES;

spool off
```

A.2.3 Log Files

Review the output generated by running each script.

File	Purpose
02_entra_demo.log	Output generated by running `02_entra_demo.sql`.
02_entra_cleanup.log	Output generated by running `02_entra_cleanup.sql`.

A.2.3.1 02_entra_demo.log

The output generated by running `02_entra_demo.sql`.

```
SQL>
SQL> ------------------------------------------------------------------------
SQL> --  HR Security Demo - Direct Logon with IAM Users
SQL> ------------------------------------------------------------------------
SQL> -- This script creates the demo environment:
SQL> -- * Configure the database to accept external identities
SQL> -- * Create data roles, which map to Entra ID roles
```

```
SQL> -- * Create sample HR application data
SQL> -- * Define data grants to restrict access to Marvin and Emma based on
their roles
SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 1. Connect as SYS to the target PDB
SQL> ------------------------------------------------------------------------
SQL>
SQL> connect sys/&syspasswd@&target_pdb as sysdba
Connected.
SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 2. Create the administration user
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- This local user is used to manage the demo environment.
SQL> create user DEEPSEC_DBA identified by &dbapasswd;
old   1: create user DEEPSEC_DBA identified by &dbapasswd
new   1: create user DEEPSEC_DBA identified by dba_pwd

User created.

SQL> grant DBA to DEEPSEC_DBA;

Grant succeeded.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 3. Connect as the administration user
SQL> ------------------------------------------------------------------------
SQL>
SQL> connect DEEPSEC_DBA/&dbapasswd@&target_pdb
Connected.
SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 4. Configure the identity provider
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- Enable Microsoft Entra ID as the identity provider for this PDB.
SQL> -- SCOPE = BOTH applies the setting to both the CDB and the PDB.
SQL> -- The database will use these values to validate incoming Entra tokens.
SQL> ALTER SYSTEM SET IDENTITY_PROVIDER_TYPE = AZURE_AD SCOPE = BOTH;

System altered.

SQL>
SQL> -- Supply the Entra tenant details so the database can validate tokens.
SQL> --   application_id_uri: the Application ID URI of the EmployeeRecordsDB
SQL> --                       Entra app registration  [DB_APP_ID_URI]
SQL> --   tenant_id:          the Directory (tenant) ID of your Entra tenant
[TENANT_ID]
SQL> --   app_id:             the Application (client) ID of
EmployeeRecordsDB [DB_APP_ID]
SQL> ALTER SYSTEM SET IDENTITY_PROVIDER_CONFIG = '{
  2    "application_id_uri": "&DB_APP_ID_URI",
  3    "tenant_id": "&TENANT_ID",
```

```
  4    "app_id": "&DB_APP_ID"
  5  }' SCOPE = BOTH;
old   2:   "application_id_uri": "&DB_APP_ID_URI",
new   2:   "application_id_uri": "<DB_APP_ID_URI>",
old   3:   "tenant_id": "&TENANT_ID",
new   3:   "tenant_id": "<TENANT_ID>",
old   4:   "app_id": "&DB_APP_ID"
new   4:   "app_id": "<DB_APP_ID>"

System altered.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 5. Create data roles
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- Employees can see only their own row.
SQL> CREATE DATA ROLE employee_role MAPPED TO 'AZURE_ROLE=EMPLOYEE';

Data role created.

SQL>
SQL> -- Managers can see their direct reports (with SSN masked).
SQL> CREATE DATA ROLE manager_role MAPPED TO 'AZURE_ROLE=MANAGER';

Data role created.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 6. Create the session role
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- A standard database role that carries the CREATE SESSION privilege.
SQL> -- Granting this role to a data role allows end users who hold that
SQL> -- data role to establish a database session.
SQL> CREATE ROLE db_role;

Role created.

SQL> GRANT CREATE SESSION TO db_role;

Grant succeeded.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 7. Grant create session privilege to data roles
SQL> ------------------------------------------------------------------------
SQL>
SQL> GRANT db_role TO employee_role;

Grant succeeded.

SQL> GRANT db_role TO manager_role;

Grant succeeded.
```

```
SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 8. Create tablespace and HR schema
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- Create the tablespace used by the HR demo schema.
SQL> -- Oracle Managed Files (OMF) controls the datafile location.
SQL> CREATE TABLESPACE users DATAFILE SIZE 100M AUTOEXTEND ON;

Tablespace created.

SQL>
SQL> -- Create the HR schema that owns the employee table.
SQL> -- NO AUTHENTICATION means this schema cannot be logged into directly;
SQL> -- it is accessed only through object grants.
SQL> CREATE USER hr NO AUTHENTICATION
  2    DEFAULT TABLESPACE users
  3    QUOTA UNLIMITED ON users;

User created.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 9. Create the employee table
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- Create the data model
SQL> CREATE TABLE hr.employees (
  2      employee_id NUMBER PRIMARY KEY,
  3      first_name  VARCHAR2(50),
  4      last_name   VARCHAR2(50),
  5      email       VARCHAR2(128),
  6      manager     VARCHAR2(128),
  7      ssn         VARCHAR2(20),
  8      salary      NUMBER(10,2),
  9      phone       VARCHAR2(20)
 10  );

Table created.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 10. Load sample data
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- Email and manager values use UPN format (firstname@ENTRA_DOMAIN)
SQL> -- The Entra-issued JWT and passes as ORA_END_USER_CONTEXT.username.
SQL> INSERT INTO hr.employees VALUES
(100,'Victoria','Williams','victoria@&ENTRA_DOMAIN',NULL,'219-09-9999',13000,'
555-0100');
old   1: INSERT INTO hr.employees VALUES
(100,'Victoria','Williams','victoria@&ENTRA_DOMAIN',NULL,'219-09-9999',13000,'
555-0100')
new   1: INSERT INTO hr.employees VALUES
(100,'Victoria','Williams','victoria@<ENTRA_DOMAIN>',NULL,'219-09-9999',13000,
'555-0100')
```

```
1 row created.

SQL> INSERT INTO hr.employees VALUES
(200,'Marvin','Anderson','marvin@&ENTRA_DOMAIN','victoria@&ENTRA_DOMAIN','457-
55-5462',12030,'555-0200');
old   1: INSERT INTO hr.employees VALUES
(200,'Marvin','Anderson','marvin@&ENTRA_DOMAIN','victoria@&ENTRA_DOMAIN','457-
55-5462',12030,'555-0200')
new   1: INSERT INTO hr.employees VALUES
(200,'Marvin','Anderson','marvin@<ENTRA_DOMAIN>','victoria@<ENTRA_DOMAIN>','45
7-55-5462',12030,'555-0200')

1 row created.

SQL> INSERT INTO hr.employees VALUES
(300,'Chris','Evans','chris@&ENTRA_DOMAIN','victoria@&ENTRA_DOMAIN','321-12-45
67',6900,'555-0300');
old   1: INSERT INTO hr.employees VALUES
(300,'Chris','Evans','chris@&ENTRA_DOMAIN','victoria@&ENTRA_DOMAIN','321-12-45
67',6900,'555-0300')
new   1: INSERT INTO hr.employees VALUES
(300,'Chris','Evans','chris@<ENTRA_DOMAIN>','victoria@<ENTRA_DOMAIN>','321-12-
4567',6900,'555-0300')

1 row created.

SQL> INSERT INTO hr.employees VALUES
(400,'Emma','Baker','emma@&ENTRA_DOMAIN','marvin@&ENTRA_DOMAIN','733-02-9821',
8200,'555-0400');
old   1: INSERT INTO hr.employees VALUES
(400,'Emma','Baker','emma@&ENTRA_DOMAIN','marvin@&ENTRA_DOMAIN','733-02-9821',
8200,'555-0400')
new   1: INSERT INTO hr.employees VALUES
(400,'Emma','Baker','emma@<ENTRA_DOMAIN>','marvin@<ENTRA_DOMAIN>','733-02-9821
',8200,'555-0400')

1 row created.

SQL> INSERT INTO hr.employees VALUES
(500,'Taylor','Mills','taylor@&ENTRA_DOMAIN','marvin@&ENTRA_DOMAIN','558-76-12
43',9000,'555-0500');
old   1: INSERT INTO hr.employees VALUES
(500,'Taylor','Mills','taylor@&ENTRA_DOMAIN','marvin@&ENTRA_DOMAIN','558-76-12
43',9000,'555-0500')
new   1: INSERT INTO hr.employees VALUES
(500,'Taylor','Mills','taylor@<ENTRA_DOMAIN>','marvin@<ENTRA_DOMAIN>','558-76-
1243',9000,'555-0500')

1 row created.

SQL> COMMIT;

Commit complete.

SQL>
```

```
SQL> ------------------------------------------------------------------------
SQL> -- 11. Create data grants
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- Employee access rule:
SQL> -- A user can see only the row where the email matches their identity.
SQL> CREATE OR REPLACE DATA GRANT hr.employees_own_record
  2    AS SELECT
  3    ON hr.employees
  4    WHERE email = ORA_END_USER_CONTEXT.username
  5    TO employee_role;

Data grant created.

SQL>
SQL> -- Manager access rule:
SQL> -- A manager can see rows where the manager column matches their
identity.
SQL> -- The ALL COLUMNS EXCEPT clause masks SSN for those rows.
SQL> CREATE OR REPLACE DATA GRANT hr.manager_direct_reports
  2    AS SELECT (ALL COLUMNS EXCEPT ssn)
  3    ON hr.employees
  4    WHERE manager = ORA_END_USER_CONTEXT.username
  5    TO manager_role;

Data grant created.

SQL>
SQL> spool off
```

A.2.3.2 02_entra_cleanup.log

The output generated by running `02_entra_cleanup.sql`.

```
SQL>
SQL> -- This script removes all objects created by the demo so the
SQL> -- environment can be reused cleanly.
SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 1. Connect as SYS
SQL> ------------------------------------------------------------------------
SQL>
SQL> connect sys/&syspasswd@&target_pdb as sysdba
Connected.
SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 2. Remove the HR schema
SQL> ------------------------------------------------------------------------
SQL>
SQL> DROP USER hr CASCADE;

User dropped.

SQL>
SQL> ------------------------------------------------------------------------
```

```
SQL> -- 3. Remove data roles
SQL> ------------------------------------------------------------------------
SQL>
SQL> DROP DATA ROLE employee_role;

Data role dropped.

SQL> DROP DATA ROLE manager_role;

Data role dropped.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 4. Remove the database role
SQL> ------------------------------------------------------------------------
SQL>
SQL> DROP ROLE db_role;

Role dropped.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 5. Remove the administration user and tablespace
SQL> ------------------------------------------------------------------------
SQL>
SQL> DROP USER DEEPSEC_DBA CASCADE;

User dropped.

SQL> DROP TABLESPACE users INCLUDING CONTENTS AND DATAFILES;

Tablespace dropped.

SQL>
SQL> spool off
```

A.3 Scripts for the Employee Records Application

Run these scripts to automate the configuration described in Configure Oracle Deep Data Security for a Sample Application.

These scripts perform database configuration only. End-user validation occurs when the Spring Boot application is called through its REST endpoints, as described in the chapter.

Topics:

- How to Run the Scripts
- Scripts
- Log Files

A.3.1 How to Run the Scripts

Run these scripts as `SYS` from a shell on the database host, in the following order.

1. `03_application_demo.sql`: Sets up the demo environment. The script reads these parameters in order from either command-line arguments or interactive prompts:
   - `syspasswd`: `SYS` password for the target PDB.
   - `dbapasswd`: Password for the `DEEPSEC_DBA` administrative user that the script creates.
   - `target_pdb`: Service name of the target PDB.
   - `TENANT_ID`: Directory (tenant) ID of your Microsoft Entra ID tenant.
   - `DB_APP_ID`: Application (client) ID of the `EmployeeRecordsDB` application registration in Entra ID. See Register the Database Resource.
   - `DB_APP_ID_URI`: Application ID URI of the `EmployeeRecordsDB` application registration in Entra ID.
   - `ENTRA_DOMAIN`: Your Microsoft Entra ID domain (for example, `supremo.onmicrosoft.com`). Used to construct sample employee email addresses.
   - `EMP_RECORDS_APP_ID`: Application (client) ID of the `EmployeeRecordsAPI` application registration in Entra ID. See Register the Spring Boot Application.
2. `03_application_cleanup.sql`: Removes all objects the demo script created. Run this to reset the environment. The script reads these parameters in order from either command-line arguments or interactive prompts:
   - `syspasswd`: `SYS` password for the target PDB.
   - `target_pdb`: Service name of the target PDB.

A.3.2 Scripts

Review the source for both scripts used in the scenario.

File	Purpose
03_application_demo.sql	Sets up the demo environment.
03_application_cleanup.sql	Removes all objects created by the demo script.

A.3.2.1 03_application_demo.sql

The source for the `03_application_demo.sql` script.

```
SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 200
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100

define syspasswd=&1
define dbapasswd=&2
define target_pdb=&3
```

```
define TENANT_ID=&4
define DB_APP_ID=&5
define DB_APP_ID_URI=&6
define ENTRA_DOMAIN=&7
define EMP_RECORDS_APP_ID=&8

spool 03_application_demo.log

-----------------------------------------------------------------------
--  HR Security Demo - Spring Boot application
-----------------------------------------------------------------------
-- This script creates the demo environment:
-- * Configure the database to accept external identities from Microsoft
--   Entra ID
-- * Create sample employee records data
-- * Set up the connection pool user account and application identity for the
--   application
-- * Create data roles, which map to Entra ID application roles
-- * Define data grants that limit a user with employee role to his or her
--   own employee record, and enable privilege elevation for the
--   salary-summary endpoint

-----------------------------------------------------------------------
-- 1. Connect as SYS to the target PDB
-----------------------------------------------------------------------

connect sys/&syspasswd@&target_pdb as sysdba

-----------------------------------------------------------------------
-- 2. Create the administration user
-----------------------------------------------------------------------

-- This local user is used to manage the demo environment.
create user DEEPSEC_DBA identified by &dbapasswd;
grant DBA to DEEPSEC_DBA;

-----------------------------------------------------------------------
-- 3. Connect as the administration user
-----------------------------------------------------------------------

connect DEEPSEC_DBA/&dbapasswd@&target_pdb

-----------------------------------------------------------------------
-- 4. Configure the identity provider
-----------------------------------------------------------------------

-- Enable Microsoft Entra ID as the identity provider for this PDB.
-- SCOPE = BOTH applies the setting to both the CDB and the PDB.
-- The database will use these values to validate incoming Entra tokens.
ALTER SYSTEM SET IDENTITY_PROVIDER_TYPE = AZURE_AD SCOPE = BOTH;

-- Supply the Entra tenant details so the database can validate tokens.
--   application_id_uri: the Application ID URI of the EmployeeRecordsDB
--                       Entra app registration  [DB_APP_ID_URI]
--   tenant_id:          the Directory (tenant) ID of your Entra tenant
[TENANT_ID]
```

```
--   app_id:            the Application (client) ID of EmployeeRecordsDB
[DB_APP_ID]
ALTER SYSTEM SET IDENTITY_PROVIDER_CONFIG = '{
  "application_id_uri": "&DB_APP_ID_URI",
  "tenant_id": "&TENANT_ID",
  "app_id": "&DB_APP_ID"
}' SCOPE = BOTH;

-----------------------------------------------------------------------
-- 5. Create tablespace and HR schema
-----------------------------------------------------------------------

-- Create the tablespace used by the HR demo schema.
-- Oracle Managed Files (OMF) controls the datafile location.
CREATE TABLESPACE users DATAFILE SIZE 100M AUTOEXTEND ON;

-- Create the HR schema that owns the employee table.
-- NO AUTHENTICATION means this schema cannot be logged into directly;
-- it is accessed only through object grants.
CREATE USER hr NO AUTHENTICATION
  DEFAULT TABLESPACE users
  QUOTA UNLIMITED ON users;

-----------------------------------------------------------------------
-- 6. Create the employee table
-----------------------------------------------------------------------

-- Create the data model used by all demo scenarios.
CREATE TABLE hr.employees (
    employee_id NUMBER PRIMARY KEY,
    first_name  VARCHAR2(50),
    last_name   VARCHAR2(50),
    email       VARCHAR2(128),
    manager     VARCHAR2(128),
    ssn         VARCHAR2(20),
    salary      NUMBER(10,2),
    phone       VARCHAR2(20)
);

-----------------------------------------------------------------------
-- 7. Load sample data
-----------------------------------------------------------------------

-- Email and manager values use UPN format (firstname@ENTRA_DOMAIN).
-- This matches the identity the Spring Boot application receives from
-- the Entra-issued JWT and passes as ORA_END_USER_CONTEXT.username.
INSERT INTO hr.employees VALUES
(100,'Victoria','Williams','victoria@&ENTRA_DOMAIN',NULL,'219-09-9999',13000,'
555-0100');
INSERT INTO hr.employees VALUES
(200,'Marvin','Anderson','marvin@&ENTRA_DOMAIN','victoria@&ENTRA_DOMAIN','457-
55-5462',12030,'555-0200');
INSERT INTO hr.employees VALUES
(300,'Chris','Evans','chris@&ENTRA_DOMAIN','victoria@&ENTRA_DOMAIN','321-12-45
67',6900,'555-0300');
INSERT INTO hr.employees VALUES
```

```
(400,'Emma','Baker','emma@&ENTRA_DOMAIN','marvin@&ENTRA_DOMAIN','733-02-9821',
8200,'555-0400');
INSERT INTO hr.employees VALUES
(500,'Taylor','Mills','taylor@&ENTRA_DOMAIN','marvin@&ENTRA_DOMAIN','558-76-12
43',9000,'555-0500');
COMMIT;

-----------------------------------------------------------------------
-- 8. Create the connection pool user
-----------------------------------------------------------------------

-- The Spring Boot application authenticates to the database using this
-- account and its own Entra database-access token (AZURE_CLIENT_ID).
-- The Oracle JDBC Spring Boot Provider obtains this token automatically
-- from Entra using the application's client credentials.
CREATE USER hr_app_user IDENTIFIED GLOBALLY AS
    'AZURE_CLIENT_ID=&EMP_RECORDS_APP_ID';

GRANT CREATE SESSION TO hr_app_user;

-- Allows the application to attach an end-user security context payload
-- (the end user's Entra JWT) to each session on behalf of an end user.
GRANT CREATE END USER SECURITY CONTEXT TO hr_app_user;

-----------------------------------------------------------------------
-- 9. Create the application identity
-----------------------------------------------------------------------

-- Maps the Spring Boot application's Entra client ID to a named
-- principal in the database. Data roles granted to this identity
-- can only be activated when the application includes them in the
-- end-user security context payload.
CREATE OR REPLACE APPLICATION IDENTITY hr_app
    MAPPED TO 'AZURE_CLIENT_ID=&EMP_RECORDS_APP_ID';

-----------------------------------------------------------------------
-- 10. Create data roles
-----------------------------------------------------------------------

-- Employees can see only their own row.
CREATE DATA ROLE employee_role MAPPED TO 'AZURE_ROLE=EMPLOYEE';

-- compensation_analyst is created DISABLED
-- Granting it to hr_app means only the application can activate it.
-- End users cannot enable this role on their own.
--
-- At runtime, the @RunWithDataRoles({"COMPENSATION_ANALYST"}) annotation
-- on EmployeeService.getSalarySummary() instructs the Oracle JDBC Spring
-- Boot Provider to temporarily include this role in the end-user context
-- payload for that method call only. The role is deactivated as soon as
-- the method returns.
CREATE OR REPLACE DATA ROLE compensation_analyst DISABLED;
GRANT DATA ROLE compensation_analyst TO hr_app;

-----------------------------------------------------------------------
-- 11. Create data grants
```

```
-----------------------------------------------------------------------

-- Employee access rule:
-- A user can see only the row where the email matches their identity.
-- Used by the Spring Boot GET /api/employees endpoint.
CREATE OR REPLACE DATA GRANT hr.employees_own_record
  AS SELECT
  ON hr.employees
  WHERE email = ORA_END_USER_CONTEXT.username
  TO employee_role;

-- Analyst access rule:
-- Exposes the salary column across all rows when the role is active.
-- Used by the Spring Boot GET /api/employees/salary-summary endpoint
-- through the @RunWithDataRoles({"COMPENSATION_ANALYST"}) annotation.
CREATE OR REPLACE DATA GRANT hr.employees_salary_summary
  AS SELECT (salary)
  ON hr.employees
  WHERE 1 = 1
  TO compensation_analyst;

spool off
```

A.3.2.2 03_application_cleanup.sql

The source for the `03_application_cleanup.sql` script.

```
SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 200
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100

define syspasswd=&1
define target_pdb=&2

spool 03_application_cleanup.log

-- This script removes all objects created by the demo so the
-- environment can be reused cleanly.

-----------------------------------------------------------------------
-- 1. Connect as SYS
-----------------------------------------------------------------------

connect sys/&syspasswd@&target_pdb as sysdba

-----------------------------------------------------------------------
-- 2. Remove application objects
-----------------------------------------------------------------------

DROP USER hr_app_user CASCADE;
DROP APPLICATION IDENTITY hr_app;
```

```
-----------------------------------------------------------------------
-- 3. Remove the HR schema
-----------------------------------------------------------------------

DROP USER hr CASCADE;

-----------------------------------------------------------------------
-- 4. Remove data roles
-----------------------------------------------------------------------

DROP DATA ROLE employee_role;
DROP DATA ROLE compensation_analyst;

-----------------------------------------------------------------------
-- 5. Remove the administration user and tablespace
-----------------------------------------------------------------------

DROP USER DEEPSEC_DBA CASCADE;
DROP TABLESPACE users INCLUDING CONTENTS AND DATAFILES;

spool off
```

A.3.3 Log Files

Review the output generated by running each script.

File	Purpose
03_application_demo.log	Output generated by running `03_application_demo.sql`.
03_application_cleanup.log	Output generated by running `03_application_cleanup.sql`.

A.3.3.1 03_application_demo.log

The output generated by running `03_application_demo.sql`.

```
SQL>
SQL> ------------------------------------------------------------------------
SQL> --  HR Security Demo - Spring Boot application
SQL> ------------------------------------------------------------------------
SQL> -- This script creates the demo environment:
SQL> -- * Configure the database to accept external identities from Microsoft
SQL> --   Entra ID
SQL> -- * Create sample employee records data
SQL> -- * Set up the connection pool user account and application identity
for the
SQL> --   application
SQL> -- * Create data roles, which map to Entra ID application roles
SQL> -- * Define data grants that limit a user with employee role to his or
her
SQL> --   own employee record, and enable privilege elevation for the
SQL> --   salary-summary endpoint
SQL>
```

```
SQL> ------------------------------------------------------------------------
SQL> -- 1. Connect as SYS to the target PDB
SQL> ------------------------------------------------------------------------
SQL>
SQL> connect sys/&syspasswd@&target_pdb as sysdba
Connected.
SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 2. Create the administration user
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- This local user is used to manage the demo environment.
SQL> create user DEEPSEC_DBA identified by &dbapasswd;
old   1: create user DEEPSEC_DBA identified by &dbapasswd
new   1: create user DEEPSEC_DBA identified by dba_pwd

User created.

SQL> grant DBA to DEEPSEC_DBA;

Grant succeeded.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 3. Connect as the administration user
SQL> ------------------------------------------------------------------------
SQL>
SQL> connect DEEPSEC_DBA/&dbapasswd@&target_pdb
Connected.
SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 4. Configure the identity provider
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- Enable Microsoft Entra ID as the identity provider for this PDB.
SQL> -- SCOPE = BOTH applies the setting to both the CDB and the PDB.
SQL> -- The database will use these values to validate incoming Entra tokens.
SQL> ALTER SYSTEM SET IDENTITY_PROVIDER_TYPE = AZURE_AD SCOPE = BOTH;

System altered.

SQL>
SQL> -- Supply the Entra tenant details so the database can validate tokens.
SQL> --   application_id_uri: the Application ID URI of the EmployeeRecordsDB
SQL> --                       Entra app registration  [DB_APP_ID_URI]
SQL> --   tenant_id:          the Directory (tenant) ID of your Entra tenant
[TENANT_ID]
SQL> --   app_id:             the Application (client) ID of
EmployeeRecordsDB [DB_APP_ID]
SQL> ALTER SYSTEM SET IDENTITY_PROVIDER_CONFIG = '{
  2    "application_id_uri": "&DB_APP_ID_URI",
  3    "tenant_id": "&TENANT_ID",
  4    "app_id": "&DB_APP_ID"
  5  }' SCOPE = BOTH;
old   2:   "application_id_uri": "&DB_APP_ID_URI",
new   2:   "application_id_uri": "<DB_APP_ID_URI>",
```

```
old   3:   "tenant_id": "&TENANT_ID",
new   3:   "tenant_id": "<TENANT_ID>",
old   4:   "app_id": "&DB_APP_ID"
new   4:   "app_id": "<DB_APP_ID>"

System altered.

SQL>
SQL> -----------------------------------------------------------------------
SQL> -- 5. Create tablespace and HR schema
SQL> -----------------------------------------------------------------------
SQL>
SQL> -- Create the tablespace used by the HR demo schema.
SQL> -- Oracle Managed Files (OMF) controls the datafile location.
SQL> CREATE TABLESPACE users DATAFILE SIZE 100M AUTOEXTEND ON;

Tablespace created.

SQL>
SQL> -- Create the HR schema that owns the employee table.
SQL> -- NO AUTHENTICATION means this schema cannot be logged into directly;
SQL> -- it is accessed only through object grants.
SQL> CREATE USER hr NO AUTHENTICATION
  2    DEFAULT TABLESPACE users
  3    QUOTA UNLIMITED ON users;

User created.

SQL>
SQL> -----------------------------------------------------------------------
SQL> -- 6. Create the employee table
SQL> -----------------------------------------------------------------------
SQL>
SQL> -- Create the data model used by all demo scenarios.
SQL> CREATE TABLE hr.employees (
  2      employee_id NUMBER PRIMARY KEY,
  3      first_name  VARCHAR2(50),
  4      last_name   VARCHAR2(50),
  5      email       VARCHAR2(128),
  6      manager     VARCHAR2(128),
  7      ssn         VARCHAR2(20),
  8      salary      NUMBER(10,2),
  9      phone       VARCHAR2(20)
 10  );

Table created.

SQL>
SQL> -----------------------------------------------------------------------
SQL> -- 7. Load sample data
SQL> -----------------------------------------------------------------------
SQL>
SQL> -- Email and manager values use UPN format (firstname@ENTRA_DOMAIN).
SQL> -- This matches the identity the Spring Boot application receives from
SQL> -- the Entra-issued JWT and passes as ORA_END_USER_CONTEXT.username.
SQL> INSERT INTO hr.employees VALUES
```

```
(100,'Victoria','Williams','victoria@&ENTRA_DOMAIN',NULL,'219-09-9999',13000,'
555-0100');
old   1: INSERT INTO hr.employees VALUES
(100,'Victoria','Williams','victoria@&ENTRA_DOMAIN',NULL,'219-09-9999',13000,'
555-0100')
new   1: INSERT INTO hr.employees VALUES
(100,'Victoria','Williams','victoria@<ENTRA_DOMAIN>',NULL,'219-09-9999',13000,
'555-0100')

1 row created.

SQL> INSERT INTO hr.employees VALUES
(200,'Marvin','Anderson','marvin@&ENTRA_DOMAIN','victoria@&ENTRA_DOMAIN','457-
55-5462',12030,'555-0200');
old   1: INSERT INTO hr.employees VALUES
(200,'Marvin','Anderson','marvin@&ENTRA_DOMAIN','victoria@&ENTRA_DOMAIN','457-
55-5462',12030,'555-0200')
new   1: INSERT INTO hr.employees VALUES
(200,'Marvin','Anderson','marvin@<ENTRA_DOMAIN>','victoria@<ENTRA_DOMAIN>','45
7-55-5462',12030,'555-0200')

1 row created.

SQL> INSERT INTO hr.employees VALUES
(300,'Chris','Evans','chris@&ENTRA_DOMAIN','victoria@&ENTRA_DOMAIN','321-12-45
67',6900,'555-0300');
old   1: INSERT INTO hr.employees VALUES
(300,'Chris','Evans','chris@&ENTRA_DOMAIN','victoria@&ENTRA_DOMAIN','321-12-45
67',6900,'555-0300')
new   1: INSERT INTO hr.employees VALUES
(300,'Chris','Evans','chris@<ENTRA_DOMAIN>','victoria@<ENTRA_DOMAIN>','321-12-
4567',6900,'555-0300')

1 row created.

SQL> INSERT INTO hr.employees VALUES
(400,'Emma','Baker','emma@&ENTRA_DOMAIN','marvin@&ENTRA_DOMAIN','733-02-9821',
8200,'555-0400');
old   1: INSERT INTO hr.employees VALUES
(400,'Emma','Baker','emma@&ENTRA_DOMAIN','marvin@&ENTRA_DOMAIN','733-02-9821',
8200,'555-0400')
new   1: INSERT INTO hr.employees VALUES
(400,'Emma','Baker','emma@<ENTRA_DOMAIN>','marvin@<ENTRA_DOMAIN>','733-02-9821
',8200,'555-0400')

1 row created.

SQL> INSERT INTO hr.employees VALUES
(500,'Taylor','Mills','taylor@&ENTRA_DOMAIN','marvin@&ENTRA_DOMAIN','558-76-12
43',9000,'555-0500');
old   1: INSERT INTO hr.employees VALUES
(500,'Taylor','Mills','taylor@&ENTRA_DOMAIN','marvin@&ENTRA_DOMAIN','558-76-12
43',9000,'555-0500')
new   1: INSERT INTO hr.employees VALUES
(500,'Taylor','Mills','taylor@<ENTRA_DOMAIN>','marvin@<ENTRA_DOMAIN>','558-76-
1243',9000,'555-0500')
```

```
1 row created.

SQL> COMMIT;

Commit complete.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 8. Create the connection pool user
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- The Spring Boot application authenticates to the database using this
SQL> -- account and its own Entra database-access token (AZURE_CLIENT_ID).
SQL> -- The Oracle JDBC Spring Boot Provider obtains this token automatically
SQL> -- from Entra using the application's client credentials.
SQL> CREATE USER hr_app_user IDENTIFIED GLOBALLY AS
  2      'AZURE_CLIENT_ID=&EMP_RECORDS_APP_ID';
old   2:     'AZURE_CLIENT_ID=&EMP_RECORDS_APP_ID'
new   2:     'AZURE_CLIENT_ID=<EMP_RECORDS_APP_ID>'

User created.

SQL>
SQL> GRANT CREATE SESSION TO hr_app_user;

Grant succeeded.

SQL>
SQL> -- Allows the application to attach an end-user security context payload
SQL> -- (the end user's Entra JWT) to each session on behalf of an end user.
SQL> GRANT CREATE END USER SECURITY CONTEXT TO hr_app_user;

Grant succeeded.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 9. Create the application identity
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- Maps the Spring Boot application's Entra client ID to a named
SQL> -- principal in the database. Data roles granted to this identity
SQL> -- can only be activated when the application includes them in the
SQL> -- end-user security context payload.
SQL> CREATE OR REPLACE APPLICATION IDENTITY hr_app
  2      MAPPED TO 'AZURE_CLIENT_ID=&EMP_RECORDS_APP_ID';
old   2:     MAPPED TO 'AZURE_CLIENT_ID=&EMP_RECORDS_APP_ID'
new   2:     MAPPED TO 'AZURE_CLIENT_ID=<EMP_RECORDS_APP_ID>'

Application identity created.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 10. Create data roles
SQL> ------------------------------------------------------------------------
SQL>
```

```
SQL> -- Employees can see only their own row.
SQL> CREATE DATA ROLE employee_role MAPPED TO 'AZURE_ROLE=EMPLOYEE';

Data role created.

SQL>
SQL> -- compensation_analyst is created DISABLED
SQL> -- Granting it to hr_app means only the application can activate it.
SQL> -- End users cannot enable this role on their own.
SQL> --
SQL> -- At runtime, the @RunWithDataRoles({"COMPENSATION_ANALYST"}) annotation
SQL> -- on EmployeeService.getSalarySummary() instructs the Oracle JDBC Spring
SQL> -- Boot Provider to temporarily include this role in the end-user context
SQL> -- payload for that method call only. The role is deactivated as soon as
SQL> -- the method returns.
SQL> CREATE OR REPLACE DATA ROLE compensation_analyst DISABLED;

Data role created.

SQL> GRANT DATA ROLE compensation_analyst TO hr_app;

Grant succeeded.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 11. Create data grants
SQL> ------------------------------------------------------------------------
SQL>
SQL> -- Employee access rule:
SQL> -- A user can see only the row where the email matches their identity.
SQL> -- Used by the Spring Boot GET /api/employees endpoint.
SQL> CREATE OR REPLACE DATA GRANT hr.employees_own_record
  2    AS SELECT
  3    ON hr.employees
  4    WHERE email = ORA_END_USER_CONTEXT.username
  5    TO employee_role;

Data grant created.

SQL>
SQL> -- Analyst access rule:
SQL> -- Exposes the salary column across all rows when the role is active.
SQL> -- Used by the Spring Boot GET /api/employees/salary-summary endpoint
SQL> -- through the @RunWithDataRoles({"COMPENSATION_ANALYST"}) annotation.
SQL> CREATE OR REPLACE DATA GRANT hr.employees_salary_summary
  2    AS SELECT (salary)
  3    ON hr.employees
  4    WHERE 1 = 1
  5    TO compensation_analyst;

Data grant created.

SQL>
SQL> spool off
```

A.3.3.2 03_application_cleanup.log

The output generated by running `03_application_cleanup.sql`.

```
SQL>
SQL> -- This script removes all objects created by the demo so the
SQL> -- environment can be reused cleanly.
SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 1. Connect as SYS
SQL> ------------------------------------------------------------------------
SQL>
SQL> connect sys/&syspasswd@&target_pdb as sysdba
Connected.
SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 2. Remove application objects
SQL> ------------------------------------------------------------------------
SQL>
SQL> DROP USER hr_app_user CASCADE;

User dropped.

SQL> DROP APPLICATION IDENTITY hr_app;

Application identity dropped.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 3. Remove the HR schema
SQL> ------------------------------------------------------------------------
SQL>
SQL> DROP USER hr CASCADE;

User dropped.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 4. Remove data roles
SQL> ------------------------------------------------------------------------
SQL>
SQL> DROP DATA ROLE employee_role;

Data role dropped.

SQL> DROP DATA ROLE compensation_analyst;

Data role dropped.

SQL>
SQL> ------------------------------------------------------------------------
SQL> -- 5. Remove the administration user and tablespace
SQL> ------------------------------------------------------------------------
SQL>
SQL> DROP USER DEEPSEC_DBA CASCADE;
```

```
User dropped.

SQL> DROP TABLESPACE users INCLUDING CONTENTS AND DATAFILES;

Tablespace dropped.

SQL>
SQL> spool off
```