Quick answers to common problems

Oracle Database 12c Security Cookbook

Secure your Oracle Database 12c with this valuable Oracle support resource, featuring more than 100 solutions to the challenges of protecting your data

Zoran Pavlović  Maja Veselica

www.allitebooks.com
Secure your Oracle Database 12c with this valuable Oracle support resource, featuring more than 100 solutions to the challenges of protecting your data

Zoran Pavlović
Maja Veselica

BIRMINGHAM - MUMBAI
About the Authors

Zoran Pavlović has worked on various complex database environments including RAC, ASM, Data Guard, GoldenGate, and so on. Areas of his expertise are security, performance/SQL tuning and high availabilty/disaster recovery of Oracle database. He has been working as an instructor for Oracle University since 2010 and during that time he has trained more than 200 students in Europe. In the last couple of years, Zoran has also been working on projects for Oracle Consulting. He is an Oracle ACE and he has been featured speaker/author at many conferences/magazines. He was actively engaged in beta testing Oracle Database 12c. Currently, Zoran is working as an Oracle Technical Architect in Parallel d.o.o. Belgrade.

I would like to take this opportunity to acknowledge some important people in my life who continuously inspire and support me. First, I want to say thank you to my parents Milenko and Stanojka Pavlovic, for everything they taught me, and for all the support they gave me during all these years. Second, I would like to say thank you to my family and my good friends, who helped me become a better person and a better professional. I am very thankful to our excellent team of technical reviewers: Arup Nanda, Gokhan Atil, Dmitri Levin, Osama Mustafa, and Kenneth Roth for their great suggestions and a very helpful feedback. I am also very thankful to Maja Veselica (it was a pleasure writing this book with you), all the editors, and everyone involved in this book.
Maja Veselica, MSc in software engineering, is currently working for Parallel d.o.o., Belgrade, as an Oracle Database consultant (security, performance tuning, and so on). She has been working as an instructor for Oracle University since 2010. In the last couple of years, she has also been working for Oracle Consulting. Also, Maja is a member of Oracle ACE Program and has more than 20 Oracle certificates. She enjoys (beta) testing Oracle products and participating in other Oracle-related activities.

This is the first book I’ve written, and because of that, it will always be special to me. I would like to thank my entire family and friends for their patience and support. I am especially grateful to my parents, Mirko and Sanja Veselica, who informally reviewed most parts of the book, and to my uncle Dušan, aunt Zora, and my best friend Mirjana Marković for very creative suggestions.

I am very thankful to the technical reviewers: Arup Nanda, Gokhan Atil, Dmitri Levin, Osama Mustafa, and Kenneth Roth for spending their spare time reviewing this book and for providing us with very valuable feedback (corrections, suggestions, ideas, and opinions). Also, this book couldn’t have been written without the Packt Publishing team - thank you all!

Zoran, I always enjoy working with you. Hopefully someday, we’ll write another book together.
About the Reviewers

Gokhan Atil is an Oracle ACE Director and DBA team lead at Bilyoner.com in Istanbul, Turkey. He has more than 15 years of experience in the IT industry, working with Oracle, PostgreSQL, Microsoft SQL Server, MySQL, and NoSQL databases. He has a strong background in software development and UNIX systems. Gokhan is an Oracle Certified Professional (OCP), and he specializes in high availability solutions, performance tuning, and monitoring tools.

Gokhan is a founding member and current vice president of Turkish Oracle User Group (TROUG). He’s also a member of Independent Oracle User Group (IOUG). Gokhan has presented at various conferences, and he is a coauthor of Expert Oracle Enterprise Manager 12c book.

Gokhan shares his experience of working with Oracle products by blogging at www.gokhanatil.com since 2008 and on Twitter with the handle @gokhanatil.

Dmitri Levin has been working as a database administrator for more than 20 years. His areas of interest include the database design, replication, and performance tuning. Dmitri has spoken at several national and international conferences.

He is currently working as senior database architect and administrator at alphabroder co.

Dmitri has an MS degree in Mathematics from St. Petersburg University, Russia, Oracle Database 11g OCA, and MS SQL Server 2012 certified DBA.

He can be reached at d_levin@hotmail.com.
Osama Mustafa (Oracle ACE Director) has progressive experience in the Oracle products community. He recently served as an Oracle DBA team leader and is certified in Oracle products, such as Fusion middleware, and is a database professional, Oracle Certified Implementation Specialist, and certified Solaris System Administrator. He loves to share his learning with the Oracle community, so when he is not delivering an Oracle-related session, he spends a lot of his time participating in OTN (Oracle Technology Network) discussion forums.

Osama Mustafa is a popular speaker at many Oracle conferences around the world. He is also the President and Director of JAOUG (Jordan Amman Oracle User Group, which is the first group in Jordan). He worked as an Oracle database developer and Oracle database administrator, and now he is a Fusion middleware security specialist and certified in multiple oracle products.

In addition to this, Osama is a volunteer in Oracle User Group, an author for Oracle penetration testing books, and a reviewer for Oracle books such as Oracle Data Guard 11gR2 Administration Beginner’s Guide and Oracle 11g Anti-hacker’s Cookbook. He also organizes RAC Attack around the world, publishes online articles on his blog https://osamamustafa.blogspot.com, and his articles are published in Oracle Magazine and OTech magazine. Osama Mustafa is active on Twitter as @osamaoracle and his blog.

First and foremost, I would like to thank my parents and my family for allowing me to follow my ambitions throughout my childhood and for standing beside me throughout my career. Special thanks to the girl who changed my life for the better and taught me a lot of things in life.

They have all been the inspiration and motivation for continuing to improve my knowledge and move my career forward and having the patience with me for having taken yet another challenge, which decreases the amount of time I can spend with them, and I hope that one day they can understand why I spent so much time in front of my computer.

Thanks to my friends and Oracle community friends around the world who support me and guide me to be the person I am today.
Arup Nanda has been an Oracle DBA for more than 20 years with experience spanning all aspects from modeling to performance tuning and Exadata. He gives speeches frequently; he has authored about 500 articles and coauthored 5 books. He also blogs at arup.blogspot.com and mentors new and seasoned DBAs. He won the Oracle’s DBA of the Year in 2003 and Enterprise Architect of the Year awards in 2012. He is also an Oracle ACE director and a member of Oak Table Network.


To my beautiful wife Anu and son Anish for putting up with me during the review of this book.

Kenneth Roth is an Oracle Certified Professional with over 25 years of IT experience primarily focused on Oracle database products. Ken has worked in a variety of IT-related roles and industries, including financial services, transportation, pharmaceutical, manufacturing, and the public sector. Based in Chicago, he currently enjoys the freedom, variety, and challenges associated with being an independent technology consultant.
Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at customercare@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Instant updates on new Packt books

Get notified! Find out when new books are published by following @PacktEnterprise on Twitter or the Packt Enterprise Facebook page.
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>1</td>
</tr>
<tr>
<td>Chapter 1: Basic Database Security</td>
<td>7</td>
</tr>
<tr>
<td>Introduction</td>
<td>7</td>
</tr>
<tr>
<td>Creating a password profile</td>
<td>8</td>
</tr>
<tr>
<td>Getting ready</td>
<td>8</td>
</tr>
<tr>
<td>How to do it…</td>
<td>8</td>
</tr>
<tr>
<td>How it works…</td>
<td>9</td>
</tr>
<tr>
<td>There's more…</td>
<td>9</td>
</tr>
<tr>
<td>See also</td>
<td>10</td>
</tr>
<tr>
<td>Creating password-authenticated users</td>
<td>10</td>
</tr>
<tr>
<td>Getting ready</td>
<td>10</td>
</tr>
<tr>
<td>How to do it…</td>
<td>11</td>
</tr>
<tr>
<td>How it works…</td>
<td>11</td>
</tr>
<tr>
<td>There's more…</td>
<td>12</td>
</tr>
<tr>
<td>How to create a user using EM Express</td>
<td>12</td>
</tr>
<tr>
<td>See also</td>
<td>16</td>
</tr>
<tr>
<td>Changing a user's password</td>
<td>16</td>
</tr>
<tr>
<td>Getting ready</td>
<td>17</td>
</tr>
<tr>
<td>How to do it…</td>
<td>17</td>
</tr>
<tr>
<td>How it works…</td>
<td>18</td>
</tr>
<tr>
<td>There's more…</td>
<td>18</td>
</tr>
<tr>
<td>See also</td>
<td>18</td>
</tr>
<tr>
<td>Creating a user with the same credentials on another database</td>
<td>19</td>
</tr>
<tr>
<td>Getting ready</td>
<td>19</td>
</tr>
<tr>
<td>How to do it…</td>
<td>19</td>
</tr>
<tr>
<td>How it works…</td>
<td>20</td>
</tr>
<tr>
<td>There's more…</td>
<td>20</td>
</tr>
<tr>
<td>See also</td>
<td>21</td>
</tr>
<tr>
<td>Locking a user account</td>
<td>21</td>
</tr>
<tr>
<td>Getting ready</td>
<td>21</td>
</tr>
<tr>
<td>How to do it…</td>
<td>22</td>
</tr>
<tr>
<td>How it works…</td>
<td>22</td>
</tr>
<tr>
<td>See also</td>
<td>23</td>
</tr>
<tr>
<td>Expiring a user's password</td>
<td>23</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Getting ready</td>
<td>24</td>
</tr>
<tr>
<td>How to do it…</td>
<td>24</td>
</tr>
<tr>
<td>How it works…</td>
<td>24</td>
</tr>
<tr>
<td>See also</td>
<td>24</td>
</tr>
<tr>
<td><strong>Creating and using OS-authenticated users</strong></td>
<td>25</td>
</tr>
<tr>
<td>Getting ready</td>
<td>25</td>
</tr>
<tr>
<td>How to do it…</td>
<td>25</td>
</tr>
<tr>
<td>How it works…</td>
<td>26</td>
</tr>
<tr>
<td>There's more…</td>
<td>26</td>
</tr>
<tr>
<td><strong>Creating and using proxy users</strong></td>
<td>27</td>
</tr>
<tr>
<td>Getting ready</td>
<td>27</td>
</tr>
<tr>
<td>How to do it…</td>
<td>27</td>
</tr>
<tr>
<td>How it works…</td>
<td>28</td>
</tr>
<tr>
<td>There's more…</td>
<td>29</td>
</tr>
<tr>
<td><strong>Creating and using database roles</strong></td>
<td>30</td>
</tr>
<tr>
<td>Getting ready</td>
<td>30</td>
</tr>
<tr>
<td>How to do it…</td>
<td>30</td>
</tr>
<tr>
<td>How it works…</td>
<td>31</td>
</tr>
<tr>
<td>There's more…</td>
<td>32</td>
</tr>
<tr>
<td>See also</td>
<td>33</td>
</tr>
<tr>
<td><strong>The sysbackup privilege – how, when, and why span /should</strong></td>
<td>33</td>
</tr>
<tr>
<td>/spanspanyou/span use it?</td>
<td>33</td>
</tr>
<tr>
<td>Getting ready</td>
<td>33</td>
</tr>
<tr>
<td>How to do it…</td>
<td>33</td>
</tr>
<tr>
<td>Database authentication</td>
<td>33</td>
</tr>
<tr>
<td>OS authentication</td>
<td>35</td>
</tr>
<tr>
<td>How it works…</td>
<td>35</td>
</tr>
<tr>
<td>There's more…</td>
<td>38</td>
</tr>
<tr>
<td>See also</td>
<td>38</td>
</tr>
<tr>
<td><strong>The syskm privilege – how, when, and why span /should</strong></td>
<td>38</td>
</tr>
<tr>
<td>/spanspanyou/span use it?</td>
<td>38</td>
</tr>
<tr>
<td>Getting ready</td>
<td>39</td>
</tr>
<tr>
<td>How to do it…</td>
<td>39</td>
</tr>
<tr>
<td>Database authentication</td>
<td>39</td>
</tr>
<tr>
<td>OS authentication</td>
<td>40</td>
</tr>
<tr>
<td>How it works…</td>
<td>40</td>
</tr>
<tr>
<td>There's more…</td>
<td>41</td>
</tr>
<tr>
<td>See also</td>
<td>41</td>
</tr>
<tr>
<td><strong>The sysdg privilege – how, when, and why span /should</strong></td>
<td>41</td>
</tr>
<tr>
<td>/spanspanyou/span use it?</td>
<td>41</td>
</tr>
<tr>
<td>Getting ready</td>
<td>41</td>
</tr>
<tr>
<td>How to do it…</td>
<td>41</td>
</tr>
<tr>
<td>Database authentication</td>
<td>41</td>
</tr>
<tr>
<td>OS authentication</td>
<td>41</td>
</tr>
<tr>
<td>How it works…</td>
<td>41</td>
</tr>
<tr>
<td>There's more…</td>
<td>41</td>
</tr>
<tr>
<td>See also</td>
<td>41</td>
</tr>
</tbody>
</table>
### Chapter 2: Security Considerations in Multitenant Environment

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>45</td>
</tr>
<tr>
<td>Creating a common user</td>
<td>47</td>
</tr>
<tr>
<td>Getting ready</td>
<td>41</td>
</tr>
<tr>
<td>How to do it…</td>
<td>42</td>
</tr>
<tr>
<td>Database authentication</td>
<td>42</td>
</tr>
<tr>
<td>OS authentication</td>
<td>42</td>
</tr>
<tr>
<td>How it works…</td>
<td>43</td>
</tr>
<tr>
<td>There’s more…</td>
<td>44</td>
</tr>
<tr>
<td>Creating a local user</td>
<td>52</td>
</tr>
<tr>
<td>Getting ready</td>
<td>52</td>
</tr>
<tr>
<td>How to do it…</td>
<td>52</td>
</tr>
<tr>
<td>How it works…</td>
<td>52</td>
</tr>
<tr>
<td>Rules/guidelines for creating and managing local users</td>
<td>53</td>
</tr>
<tr>
<td>There’s more…</td>
<td>53</td>
</tr>
<tr>
<td>Creating a common role</td>
<td>54</td>
</tr>
<tr>
<td>Getting ready</td>
<td>55</td>
</tr>
<tr>
<td>How to do it…</td>
<td>55</td>
</tr>
<tr>
<td>How it works…</td>
<td>55</td>
</tr>
<tr>
<td>There’s more…</td>
<td>56</td>
</tr>
<tr>
<td>How to create a common role using OEM 12c</td>
<td>57</td>
</tr>
<tr>
<td>Creating a local role</td>
<td>58</td>
</tr>
<tr>
<td>Getting ready</td>
<td>58</td>
</tr>
<tr>
<td>How to do it…</td>
<td>59</td>
</tr>
<tr>
<td>How it works…</td>
<td>59</td>
</tr>
<tr>
<td>There’s more…</td>
<td>60</td>
</tr>
<tr>
<td>How to create a local role using OEM 12c</td>
<td>60</td>
</tr>
<tr>
<td>Granting privileges and roles commonly</td>
<td>60</td>
</tr>
<tr>
<td>Getting ready</td>
<td>60</td>
</tr>
<tr>
<td>How to do it…</td>
<td>61</td>
</tr>
<tr>
<td>How it works…</td>
<td>62</td>
</tr>
<tr>
<td>Chapter 5: Data Redaction</td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td></td>
</tr>
<tr>
<td>Getting ready</td>
<td>99</td>
</tr>
<tr>
<td>How to do it…</td>
<td>100</td>
</tr>
<tr>
<td>There's more…</td>
<td>102</td>
</tr>
<tr>
<td>See also</td>
<td>102</td>
</tr>
<tr>
<td>Creating column-level policies</td>
<td>103</td>
</tr>
<tr>
<td>Getting ready</td>
<td>103</td>
</tr>
<tr>
<td>How to do it…</td>
<td>103</td>
</tr>
<tr>
<td>How it works…</td>
<td>106</td>
</tr>
<tr>
<td>Creating a driving context</td>
<td>106</td>
</tr>
<tr>
<td>Getting ready</td>
<td>106</td>
</tr>
<tr>
<td>How to do it…</td>
<td>107</td>
</tr>
<tr>
<td>Creating policy groups</td>
<td>107</td>
</tr>
<tr>
<td>Getting ready</td>
<td>107</td>
</tr>
<tr>
<td>How to do it…</td>
<td>108</td>
</tr>
<tr>
<td>Setting context as a driving context</td>
<td>108</td>
</tr>
<tr>
<td>Getting ready</td>
<td>108</td>
</tr>
<tr>
<td>How to do it…</td>
<td>109</td>
</tr>
<tr>
<td>Adding policy to a group</td>
<td>109</td>
</tr>
<tr>
<td>Getting ready</td>
<td>109</td>
</tr>
<tr>
<td>How to do it…</td>
<td>110</td>
</tr>
<tr>
<td>Exempting users from VPD policies</td>
<td>114</td>
</tr>
<tr>
<td>Getting ready</td>
<td>114</td>
</tr>
<tr>
<td>How to do it…</td>
<td>115</td>
</tr>
<tr>
<td>Chapter 5: Data Redaction</td>
<td>116</td>
</tr>
<tr>
<td>Introduction</td>
<td>116</td>
</tr>
<tr>
<td>Creating a redaction policy when using full redaction</td>
<td>119</td>
</tr>
<tr>
<td>Getting ready</td>
<td>119</td>
</tr>
<tr>
<td>How to do it…</td>
<td>119</td>
</tr>
<tr>
<td>How it works…</td>
<td>122</td>
</tr>
<tr>
<td>There's more…</td>
<td>124</td>
</tr>
<tr>
<td>How to change the default value</td>
<td>125</td>
</tr>
<tr>
<td>See also</td>
<td>127</td>
</tr>
<tr>
<td>Creating a redaction policy when using partial redaction</td>
<td>128</td>
</tr>
<tr>
<td>How to do it…</td>
<td>128</td>
</tr>
<tr>
<td>How it works…</td>
<td>131</td>
</tr>
<tr>
<td>There's more…</td>
<td>133</td>
</tr>
<tr>
<td>Creating a redaction policy when using random redaction</td>
<td>133</td>
</tr>
<tr>
<td>Getting ready</td>
<td>133</td>
</tr>
<tr>
<td>How to do it…</td>
<td>134</td>
</tr>
</tbody>
</table>
Chapter 6: Transparent Sensitive Data Protection

Introduction 163
Creating a sensitive type 164
  Getting ready 165
  How to do it… 165
  How it works… 165
  There’s more… 166
Determining sensitive columns 166
  Getting ready 166
  How to do it… 167
  How it works… 168
Creating transparent sensitive data protection policy 168
  Getting ready 169
  How to do it… 169

Creating a redaction policy when using regular expression redaction 137
  Getting ready 137
  How to do it… 137
  How it works… 140
Using Oracle Enterprise Manager Cloud Control 12c to manage redaction policies 140
  Getting ready 140
  How to do it… 140
Changing the function parameters for a specified column 150
  Getting ready 151
  How to do it… 151
Add a column to the redaction policy 152
  Getting ready 152
  How to do it… 153
  How it works… 154
  See also 154
Enabling, disabling, and dropping redaction policy 154
  Getting ready 154
  How to do it… 155
  See also 160
Exempting users from data redaction policies 161
  Getting ready 161
  How to do it… 161
  How it works… 162
How it works…
See also
Associating transparent sensitive data protection policy with sensitive type
Getting ready
How to do it…
There's more…
See also
Enabling, disabling, and dropping policy
Getting ready
How to do it…
How it works…
There's more…
Altering transparent sensitive data protection policy
Getting ready
How to do it…
How it works…
See also
Chapter 7: Privilege Analysis
Introduction
Creating database analysis policy
Getting ready
How to do it…
How it works…
There's more…
See also
Creating role analysis policy
Getting ready
How to do it…
There's more…
See also
Creating context analysis policy
Getting ready
How to do it…
There's more…
See also
Creating combined analysis policy
Getting ready
How to do it…
Enabling Unified Auditing mode
  Getting ready
  How to do it…
  How it works…
    Predefined unified audit policies
  There's more…
  See also

Configuring whether loss of audit data is acceptable
  Getting ready
  How to do it…
  How it works…
  There's more…
  See also

Which roles do you need to have to be able to create audit policies and to view audit data?
  Getting ready
  How to do it…
  How it works…
  There's more…

Auditing RMAN operations
  Getting ready
  How to do it…
  How it works…
  See also

Auditing Data Pump operations
  Getting ready
  How to do it…
  See also

Auditing Database Vault operations
  Getting ready
  How to do it…
  How it works…
  There's more…
  See also

Creating audit policies to audit privileges, actions and roles under specified conditions
  Getting ready
  How to do it…
  How it works…
  See also

Enabling audit policy
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>343</td>
</tr>
<tr>
<td>Exploring and using built-in contexts</td>
<td>344</td>
</tr>
<tr>
<td>Getting ready</td>
<td>344</td>
</tr>
<tr>
<td>How to do it...</td>
<td>345</td>
</tr>
<tr>
<td>How it works...</td>
<td>346</td>
</tr>
<tr>
<td>There's more...</td>
<td>347</td>
</tr>
<tr>
<td>See also</td>
<td>347</td>
</tr>
<tr>
<td>Creating an application context</td>
<td>348</td>
</tr>
<tr>
<td>Getting ready</td>
<td>348</td>
</tr>
<tr>
<td>How to do it...</td>
<td>348</td>
</tr>
<tr>
<td>How it works...</td>
<td>349</td>
</tr>
<tr>
<td>Setting application context attributes</td>
<td>349</td>
</tr>
<tr>
<td>Getting ready</td>
<td>349</td>
</tr>
<tr>
<td>How to do it...</td>
<td>349</td>
</tr>
<tr>
<td>How it works...</td>
<td>351</td>
</tr>
<tr>
<td>There's more...</td>
<td>351</td>
</tr>
<tr>
<td>See also</td>
<td>351</td>
</tr>
<tr>
<td>Using an application context</td>
<td>351</td>
</tr>
<tr>
<td>Getting ready</td>
<td>352</td>
</tr>
<tr>
<td>How to do it...</td>
<td>352</td>
</tr>
<tr>
<td>How it works...</td>
<td>353</td>
</tr>
<tr>
<td>See also</td>
<td>353</td>
</tr>
</tbody>
</table>

Index 354
This book covers most of the Oracle Database 12c Security features and solutions that exist in Oracle Database 12c. Oracle Database 12c Security Cookbook will help you better understand database security challenges. It will guide you through the process of implementing appropriate security mechanisms, helping you to ensure that you are taking proactive steps to keep your data safe. Because the book features solutions for common security problems in the new Oracle Database 12c, it will make you confident about securing your database from a range of different threats and problems.

What this book covers

Chapter 1, Basic Database Security, introduces you to the different authentication methods supported by Oracle Database 12c and also provides a brief overview about creating and using database roles.

Chapter 2, Security Considerations in Multitenant Environment, focuses on some of the security considerations concerning common and local: users, roles, and privileges.

Chapter 3, PL/SQL Security, helps you understand the differences and usages of definer and invoker rights procedures as well as usages of code-based access control. It gives required information about authorization.

Chapter 4, Virtual Private Database, introduces you to the Oracle Virtual Private Database, which is a security feature introduced in Oracle Database 8i, which enables you to have a more granular control over security of your data.

Chapter 5, Data Redaction, introduces you to the new security feature Oracle Data Redaction, which helps you mask (hide/redact) some (sensitive) data from end users in a production environment.

Chapter 6, Transparent Sensitive Data Protection, teaches you ways to create classes of sensitive data and helps you gain more centralized control over how sensitive data is protected.

Chapter 7, Privilege Analysis, it shows how to create and enable privilege analysis policies. It also covers how to generate reports and revoke both used and unused Object/System privileges.
Chapter 8, *Transparent Data Encryption*, explains key concepts and tasks such as: two-key architecture, key management, message authentication code (MAC), salt, encrypting columns in a table, encrypting a tablespace, creating an encrypted RMAN backup, and so on.

Chapter 9, *Database Vault*, covers basic concepts of Oracle Database Vault. It teaches you how to create and appropriately use realms, rules, rule sets, command rules, factors, and secure application roles.

Chapter 10, *Unified Auditing*, introduces a new auditing architecture.

Chapter 11, *Additional Topics*, covers more advanced topics and teaches you how to perform everyday administration tasks in Database Vault environment.

Chapter 12, *Appendix - Application Contexts*, will cover how to retrieve values from built-in contexts and to create, set, and use an application context.

**What you need for this book**

Software required (with version)- Oracle Database 12c, Enterprise Manager Cloud Control 12c R4, Oracle Enterprise Manager Database Express 12c

Hardware specifications- OS required- Linux (Preferred Oracle Linux 6.5)

**Who this book is for**

This book is for DBAs, developers, and architects who are keen to know more about security in Oracle Database 12c. This book is best suited for beginners and intermediate-level database security practitioners. Basic knowledge of Oracle Database is expected, but no prior experience of securing a database is required.

**Sections**

In this book, you will find several headings that appear frequently (Getting ready, How to do it, How it works, There’s more, and See also).
To give clear instructions on how to complete a recipe, we use these sections as follows:

**Getting ready**
This section tells you what to expect in the recipe and describes how to set up any software or any preliminary settings required for the recipe.

**How to do it…**
This section contains the steps required to follow the recipe.

**How it works…**
This section usually consists of a detailed explanation of what happened in the previous section.

**There's more…**
This section consists of additional information about the recipe in order to make the reader more knowledgeable about the recipe.

**See also**
This section provides helpful links to other useful information for the recipe.

**Conventions**
In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Create a local user (for example, mike)."

Any command-line input or output is written as follows:

```
c##zoran@DB1> create user c##maja identified by oracle1 container=all;
```
New terms and important words are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "Click on the Create button."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

For this book we have outlined the shortcuts for the Mac OX platform if you are using the Windows version you can find the relevant shortcuts on the WebStorm help page https://www.jetbrains.com/webstorm/help/keyboard-shortcuts-by-category.html.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail feedback@packtpub.com, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for this book from your account at http://www.packtpub.com. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you.
You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the SUPPORT tab at the top.
3. Click on Code Downloads & Errata.
4. Enter the name of the book in the Search box.
5. Select the book for which you’re looking to download the code files.
6. Choose from the drop-down menu where you purchased this book from.
7. Click on Code Download.

You can also download the code files by clicking on the Code Files button on the book’s webpage at the Packt Publishing website. This page can be accessed by entering the book’s name in the Search box. Please note that you need to be logged in to your Packt account.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the book is also hosted on GitHub at https://github.com/PacktPublishing/Oracle-Database-12c-Security-Cookbook. We also have other code bundles from our rich catalog of books and videos available at https://github.com/PacktPublishing/. Check them out!

**Errata**

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting http://www.packtpub.com/submit-errata, selecting your book, clicking on the Errata Submission Form link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.
To view the previously submitted errata, go to https://www.packtpub.com/books/content/support and enter the name of the book in the search field. The required information will appear under the Errata section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at questions@packtpub.com, and we will do our best to address the problem.
Introduction

Authentication is a very important process, whose purpose is to determine whether someone or something is, in fact, who or what it claims to be.

In this chapter, you'll learn basic stuff about some of the different authentication methods supported by Oracle Database 12c. Also, a brief overview about creating and using database roles will be given.
There are three new administrative privileges introduced in Oracle Database 12c (sysbackup, syskm, and sysdg). Their purpose is to enable better separation of duties and they are designed in such a way to also enable implementation of the least privilege principle. Although it may seem that implementation of this principle in systems is easy or straightforward, usually it’s quite tricky.

For all recipes in this chapter, you will use non-CDB 12c. We assume that the database is up and running and each user has at least the create session privilege.

In this set of recipes, you will learn to perform, mostly basic, user administration tasks.

## Creating a password profile

You can use a profile to implement your password policy.

### Getting ready

To complete this recipe, you’ll need an existing user who has create profile privilege (such as an OS-authenticated user who has database administrators (dba) role, for example, ops$zoran). Also, you’ll need an unlocked user account named scott.

Make sure that the resource_limit parameter is set to true.

### How to do it…

1. Connect to the database as a user who has create profile privilege:

   ```sql
   sqlplus /
   ```

2. Create a password profile:

   ```sql
   create profile userprofile limit
   failed_login_attempts 4
   password_lock_time 2
   password_life_time 180;
   ```
3. Alter the user to use a newly created password profile:

   ```sql
   alter user scott profile userprofile;
   ```

4. Alter the default password profile:

   ```sql
   alter profile default limit
   failed_login_attempts 4;
   ```

How it works…

In step 1, you used OS authentication to connect to the database.

In step 2, you created a password profile with the name `userprofile` that has the following restrictions:

- The system allows four login attempts before locking a user account (`failed_login_attempts`)
- After locking a user account, it will remain locked for two days (`password_lock_time`)
- A password for the user can remain unchanged for 180 days – after which the password will expire, and the user will have to change the password for his next login (`password_life_time`)

In step 3, we assigned a newly created password profile to the user `scott`. If we don’t assign a password profile to the user, that user uses the default password profile.

In step 4, we altered the default password profile with the `failed_login_attempts` restriction.

There's more…

You can create different password profiles for different users in the database. There are a lot of restrictions that can be applied to a password profile.

In Oracle Database 12c, there are three password verify functions, out of which, two are new and improved:

- `verify_function_11G` (carried over)
- `ora12c_verify_function` (new)
- `ora12c_strong_verify_function` (new)
If password complexity checking is not enabled, and you want to use it, you should run the `utlpwdmg.sql` script provided by Oracle. It's located in `$ORACLE_HOME/rdbms/admin`. The `ora12c_verify_function` function is the default function that the `utlpwdmg.sql` script uses. If you want, you can customize password verify functions.

Password complexity checking, even when enabled, doesn't apply to `sys` user.

If you want to choose which verify function will be used in the default profile, you can achieve that by using the following statement:

```
alter profile default limit password_verify_function
ora12c_strong_verify_function;
```

In subsequent recipes, it is assumed that default values are set for the default profile and the password verify function is not used.

### See also

- Creating password-authenticated users
- Locking a user account
- Creating and using OS-authenticated users

### Creating password-authenticated users

In this task, you will create several users.

### Getting ready

To complete this recipe, you'll need an existing user who has `create user` privilege (you may use the OS-authenticated user who has the DBA role).

You'll use Oracle Enterprise Manager Database Express 12c (EM Express). To learn more about it (for example, how to configure an HTTPS port for EM Express and how to start it), see the third chapter of the official Oracle guide – *Oracle Database 2 Day DBA, 12c Release 1*. 
How to do it...

1. Connect to the database as a user who has create user privilege:

   `$ sqlplus /`

2. Create a password-authenticated user (for example, username: jessica, password: oracle_1) as follows:

   `SQL> create user jessica identified by oracle_1;`

3. Create a password-authenticated user with a more complex password:

   `SQL> create user tom identified by "Qax7UnP!123*";`

4. Create a user that uses a specific password profile:

   `SQL> create user m Ike identified by test1 profile userprofile;`

5. Create a user and force it to change password upon the first login:

   `SQL> create user john identified by password1 password expire;`

6. Create a user richard, whose default tablespace is users, temporary tablespace is temp, and who has their quota set to unlimited on the users tablespace:

   `SQL> create user richard identified by oracle_2 default tablespace users temporary tablespace temp quota unlimited on users;`

How it works...

In step 1, you used OS authentication to connect to the database.

In step 2, you created a password-authenticated user jessica with simpler password.

In step 3, you created a password-authenticated user tom with more complex password. In this case (because a password contains special characters), you are using quotation marks (") to enclose the password.
Both of these users are using the default password profile.

In step 4, you created a password-authenticated user with the assigned password profile userprofile.

In step 5, you created user john. This user has to change his password at the first database login.

In step 6, you created the user richard. In the create user statement, quota unlimited on users means that you want to let the user allocate space in the tablespace without bound. The quota clause lets you define the maximum amount of space the user can allocate in the tablespace. You can have multiple quota clauses for multiple tablespaces within one create user statement. The unlimited tablespace system privilege enables users to have an unlimited quota on all tablespaces in the database.

If you grant unlimited tablespace system privilege to a user and afterwards you revoke it, all explicitly granted quotas will also be revoked.

There's more...

You can also create users using Oracle Enterprise Manager Cloud Control 12c or Oracle Enterprise Manager Database Express 12c (EM Express). Oracle Enterprise Manager Database Control is no longer available in Oracle Database 12c.

How to create a user using EM Express

1. Start EM Express and log in to it using the user that has either EM_EXPRESS_BASIC or EM_EXPRESS_ALL role (you can use sys or system users, but that isn't recommended):
2. Select Users from the Security drop-down menu:
3. Click on the **Create User** tab:

![Image of Oracle Enterprise Manager with Create User tab highlighted]

4. Enter user details in the pop-up dialog (for example, username: **ted**, password: **oracle_123**, here you can also choose the authentication method, password profile, lock account, expire password) leave the default values and click on the **Next** button (see image here) as follows:
5. In this step, you can choose default tablespace and temporary tablespace from the drop-down lists. Leave the default values, as shown in the following screenshot:
6. In this step, you can grant privileges to user ted by selecting them in the left pane and moving them to the right pane (use > button). If you want to revoke privileges, do the opposite (select them in right pane and use < button). When you are satisfied with the list of privileges in the right pane (the ones you are going to grant to user ted), click on the OK button as follows:

![Create User window]

7. A pop-up window confirmation should appear with the following message: SQL statement has been processed successfully.

Click on the OK button to close the window.

See also

- Creating and using OS-authenticated users

Changing a user's password

Changing a user's password is easy. You will practice it by changing passwords for several users in this recipe.
Basic Database Security

Getting ready
To complete this recipe, you’ll need an existing user who has `alter user` privilege (you may use OS-authenticated user who has the DBA role) and other existing users (for example, jessica and tom).

How to do it…

1. Connect to the database as a user who has `alter user` privilege:
   
   ```bash
   $ sqlplus /
   
   ``

2. Change the password for user jessica:
   
   ```sql
   SQL> password jessica;
   
   ``

3. Enter a new password (for example, oracle_2) on a command line (note that typing will not be visible in the command line):
   
   ```
   New password:
   ``

4. Retype the new password (for example, oracle_2) on the command line (note that typing will not be visible in the command line):
   
   ```
   Retype new password:
   ``

5. Connect to the database as any user (for example, tom, to change their own password):
   
   ```bash
   $ sqlplus tom"Qax7UnP!123*"
   
   ``

6. Change the password using the following code:
   
   ```sql
   SQL> password
   
   ``

7. Enter the old password (for example, Qax7UnP!123*) on the command line (note that typing will not be visible on the command line):
   
   ```
   Old password:
   ```
8. Enter the new password (for example, oracle_123) on the command line (note that typing will not be visible on the command line):

   **New password:**

9. Retype the new password (for example, oracle_123) on the command line (note that typing will not be visible on the command line):

   **Retype new password:**

---

**How it works...**

In step 1, you used OS authentication to connect to the database.

In steps 2 through 4, a privileged user changed jessica's password, where in steps 6 through 9, the user tom changed his own password.

---

**There's more...**

There is another way to change the user's password using the `alter user` statement as follows:

   SQL> alter user jessica identified by oracle_2;

   This approach is not recommended because password remains in the command-line history.

---

**See also**

- *Creating and using OS-authenticated users*
Creating a user with the same credentials on another database

This recipe explains a way to create a user with the same credentials on another database.

Getting ready

To complete this recipe, you’ll need:

- An existing user who has dba role in the first database (you can use an OS-authenticated user)
- An existing user in the first database (for example, jessica)
- An existing (for example, password-authenticated) user, who has create user privilege, in the second database (for example, zoran)

How to do it…

1. Connect to the first database as a user who has a DBA role:

   $ sqlplus /

2. Find a Data Definition Language (DDL) statement (ddl) that is used for user creation (for example, user jessica):

   SQL> select dbms_metadata.get_ddl('USER', 'JESSICA') from dual;

3. Connect to the second database as a user who has create user privilege:

   $ sqlplus zoran@orcl2

4. Create a user using the value you found in step 2:

   SQL> create user "JESSICA" identified by values 'S:D82E6EF961F2EA7A878BCDDBC7E5C542BC148C4759D19A720A96BBF65658;H:F297A50FD538EF4AB119EB0278C9E72D;C50B19C9A52EC2';
How it works…

In step 1, you used OS authentication to connect to the database.

In step 2, you found a DDL statement that has been used for user creation. This DDL statement may contain default and temporary tablespace assignments (note that even if you haven’t explicitly assigned these tablespaces during user creation, the system will assign them implicitly using default values for the database). For instance, output in step 2 may look like this:

```
SQL> select dbms_metadata.get_ddl('USER', 'JESSICA') from dual;
```

```
CREATE USER "JESSICA" IDENTIFIED BY VALUES
'S:D82E6EF961F2EA7A878B0DCCB7E5C542BC148C4759D19A720A96BBF65658;H:F297A50FD538E4AB1 19EB0278C9E72D;C50B1E9C9AA52EC2'
DEFAULT TABLESPACE "USERS" TEMPORARY TABLESPACE "TEMP"
```

However, we used only the first part of this DDL in step 4 to create a user on the second database (and let the database decide about default tablespaces).

There's more…

There is another way to accomplish the task.

You can only reveal the hash value of user's password (you cannot reveal the actual password).

This way requires select on the sys.user$ table:

1. Connect to the first database as a user who has the select privilege on the sys.user$ table (for example, user who has the sysdba privilege):

   ```
   $ sqlplus / as sysdba
   ```

2. Find the hash value of a user's password (for example, user jessica):

   ```
   SQL> select spare4 
   from user$ 
   where name='JESSICA';
   ```
3. Connect to the second database as a user who has `create user` privilege:

   `$ sqlplus zoran@orcl2`

4. Create a user with the same username (for example, `jessica`) using the hash value of the password that you have found in step 2:

   ```sql
   SQL> create user jessica identified by values
   'S: 2724193130FC67E7E23E3E44E33AF143F7A6C36489792B
      5856133DCB331D; H: 184895E50EA2FBCC2311ED76A3E5CF35;
      T: BECCD5FC6F6E62BC34DF1C826AEE899EC6A6025FA0D5071659DA
      7DD1ABB37763483B5C821E5A34C1184A56BE4B1C92CED79639D11101D
      61B86ACBE60A30F19CC277D5753F7D8756DC1B7705C0ACE81F3' ;
   ```

### See also

- *Creating and using OS-authenticated users*

### Locking a user account

In this recipe, you'll learn to lock and unlock user accounts.

### Getting ready

To complete this recipe, you'll need an existing (for example, OS-authenticated) user who has `alter user` privilege (you may use user who has a DBA role) and another existing user (for example, `mike`).
How to do it…

1. Connect to the database as a user who has `alter user` privilege:
   
   ```
   $ sqlplus /
   ```

2. Lock the account of user `mike`:
   
   ```
   SQL> alter user mike account lock;
   ```

3. Unlock the account of user `mike`:
   
   ```
   SQL> alter user mike account unlock;
   ```

How it works…

In step 1, you used OS authentication to connect to the database.

In step 2, you locked the account of user `mike`. This means that user `mike` cannot connect to the database:

   ```
   SQL> alter user mike account lock;
   User altered
   SQL> connect mike/welcome1
   ERROR:ORA-28000: the account is locked
   ```
However, objects in mike’s schema are available, so users can access them (considering that they have necessary privileges):

```
SQL> select a, b from mike.table1;
A   B
---  ---
1    3
2    4
4    9
```

It is recommended that you lock the accounts of users that own your application objects (application schemas).

In step 3, you unlocked the account of user mike. Now user mike can successfully connect to the database:

```
SQL> alter user mike account unlock;
User altered.
SQL> conn mike/welcome1
Connected.
```

See also

- Creating and using OS-authenticated users

**Expanding a user's password**

The expiration of user's password is a very easy task.
Getting ready

To complete this recipe, you’ll need an existing (for example, OS-authenticated) user who has the alter user privilege (you may use user who has a DBA role) and another existing user (for example, mike).

How to do it…

1. Connect to the database as a user who has the alter user privilege:

   $ sqlplus /

2. Mike’s password expires with the following command:

   SQL> alter user mike password expire;

How it works…

In step 1, you used OS authentication to connect to the database.

In step 2, you expired password for the user mike. This means that the password is no longer valid and user mike must change his password after the next login:

   SQL> alter user mike password expire;

   User altered.

   SQL> conn mike/welcome1
   ERROR: ORA-28001: the password has expired
   Changing password for mike
   New password:
   Retype new password:
   Password changed
   Connected.

See also

- Creating and using OS-authenticated users
Basic Database Security

Creating and using OS-authenticated users
In this recipe, you'll learn about OS-authenticated users.

Getting ready
To complete this recipe, you'll need an existing user who has a dba role, for example,
johndba. It is assumed that you are working on Linux.

How to do it…
1. Connect to the database as a user who has a DBA role:
$ sqlplus johndba

2. Find the prefix for operating system authentication:
SQL> show parameter os_authent_prefix
NAME
TYPE
VALUE
---------------------------------os_authent_prefix
string
ops$

3. Create an OS-authenticated user:
SQL> create user ops$zoran identified externally;

4. Grant this user the create session privilege:
SQL> grant create session to ops$zoran;

5. Log in to the operating system as the user zoran:
$ su - zoran

6. Connect to the database without entering a user name or password:
$ sqlplus /

[ 25 ]


How it works…

In OS authentication, database delegates user authentication to the operating system. This means that in order for OS authentication to work, user must exist as the user of the operating system. In database, these users are created with a prefix that is defined in the os_authent_prefix parameter (default is ops$). If an OS-authenticated user has the create session privilege, he or she can connect to the database using the following syntax:

```
SQL> connect /
Connected.
SQL> show user
USER is "OPS$ZORAN"
```

Note that you cannot grant a sysdba, sysoper, sysbackup, sysdg, or syskm privilege to users that are identified externally, using a grant statement:

```
SQL> grant sysdba to ops$zoran;
grant sysdba to ops$zoran
ERROR at line 1: ORA-01997: GRANT failed: user 'OPS$ZORAN' identified externally
```

If you want to connect as sysdba using OS authentication, you have to add OS user zoran to OS group DBA:

```
[root@db121 ~]# usermod -a -G dba zoran
[root@db121 ~]# su - zoran
[ zoran@db121 ~]$ sqlplus / as sysdba
```

There's more…

You can change the os_authent_prefix parameter with custom value (or you can leave it blank if you want OS-authenticated database users to have the same name as OS users).
Creating and using proxy users
In this recipe, you’ll learn about proxy users.

Getting ready
To complete this recipe, you’ll need an existing (for example, OS-authenticated) user who has a DBA role and another existing user (for example, mike).

How to do it…

1. Connect to the database as a user who has a DBA role:

   ```
   $ sqlplus /
   ```

2. Create a proxy user named appserver:

   ```
   SQL> create user appserver identified by oracle_1;
   ```

3. Grant create session to the user appserver:

   ```
   SQL> grant create session to appserver;
   ```

4. Alter the user to connect through the proxy user:

   ```
   SQL> alter user mike grant connect through appserver;
   ```

5. Connect to the database through proxy user:

   ```
   SQL> connect appserver[mike]
   ```

6. Enter a password for the appserver user (for example, oracle_1):

   ```
   Enter password:
   ```

7. To revoke connection through the proxy user, first connect to the database as a user who has altered user privilege:

   ```
   $ sqlplus /
   ```
8. Revoke connection through the proxy user appserver from user mike:

   SQL> alter user mike revoke connect through appserver;

How it works...

Proxy authentication is best-suited type of authentication for three-tiered environments. The middle tier is represented as a proxy user in the database and this user can authenticate end-users in such a way that these end users can be audited by the database. In the second step, you created a user appserver (to be the proxy user). In the third step, you granted this user only the create session privilege.

It is recommended that you grant only the create session privilege to proxy users.

In step 4, you authorized user mike to connect through proxy user appserver. This means that the user appserver can connect to the database on behalf of user mike:

   SQL> connect appserver[mike]

Enter password:
Connected.

   SQL> show user
   USER is "MIKE"

   SQL> select sys_context('USERENV','PROXY_USER') from dual;

   SYS_CONTEXT('USERENV','PROXY_USER')  -----------------------------------
       APPSERVER

To see proxy users, you can query the proxy_users view:

   SQL> select * from proxy_users;

   -----------------------------------
   PROXY     CLI   ENT   AUT     FLAGS
   -----------------------------------
   APPSERVER   MIKE  NO   PROXY MAY ACTIVATE ALL CLIENT ROLES

In the last step, you revoked authorization from user mike to connect through proxy user appserver. This means that the user appserver can no longer connect to the database on behalf of user mike.
There's more…

You can control which roles the proxy user can activate for user. By default, all user roles are activated. If you want the proxy user to activate only particular roles (or no roles) for a user, you can do that by adding the `WITH ROLES <role1, role2, .., roleN>` (or `WITH NO ROLES`) clause at the end of the `alter user` statement.

For instance, if the user `mike` has many roles (including `usr_role`), and you want him to have only `usr_role` when he is connected through proxy user `appserver`, statement will look like this:

```sql
SQL> alter user mike grant connect through appserver with roles usr_role;
User altered.

SQL> connect appserver[mike]
Enter password:
Connected.

SQL> select * from session_roles;
ROLE
----------
USR_ROLE

SQL> connect mike
Enter password:
Connected.

SQL> select count(*) from session_roles;
COUNT(*)
--------
25
```

You can request reauthentication of a user to the database. This means that during proxy authentication, a user's password must be provided. This is done by using the `authentication required` clause at the end of `alter user` statement:

```sql
SQL> alter user mike grant connect through appserver authentication required;
User altered.
```
Creating and using database roles

In this recipe, you'll learn the basics about database roles. Roles group together related system and/or object privileges and they can be granted to users and other roles. They simplify privilege management (for example, rather than granting the same set of privileges to many users, you can grant those privileges to a role and then grant that role to users that need those privileges).

Getting ready

For this recipe, you will need an existing (for example, OS-authenticated) user that has a dba role and another three existing users (for example, mike, tom, and jessica). It is assumed that sample schemas are installed.

How to do it…

1. Connect to the database as a user who has a dba role:

   $ sqlplus /

2. Create the role usr_role:

   SQL> create role usr_role;

3. Grant system privilege to usr_role:

   SQL> grant create session to usr_role;

4. Grant object privileges to usr_role:

   SQL> grant select, insert on hr.employees to usr_role;

5. Create another role as follows:

   SQL> create role mgr_role;

6. Grant usr_role to mgr_role:

   SQL> grant usr_role to mgr_role;
7. Grant system privileges to `mgr_role`:

   ```sql
   SQL> grant create table to mgr_role;
   ```

8. Grant object privileges to `mgr_role`:

   ```sql
   SQL> grant update, delete on hr.employees to mgr_role;
   ```

9. Grant `usr_role` to user (`mike`):

   ```sql
   SQL> grant usr_role to mike;
   ```

10. Grant `mgr_role` to user (`tom`):

    ```sql
    SQL> grant mgr_role to tom
    ```

How it works…

In the first step, you used OS authentication to connect to the database. In steps 2 and 3, you granted system privileges and object privileges, respectively, to the role `usr_role`. In the next steps, you practiced using database roles; you granted the following:

- A role to another role
- System and object privileges to role
- Roles to users

You revoke privileges and roles by using a `revoke` statement. For example:

   ```sql
   SQL> revoke usr_role from mike;
   ```

   Circular granting of roles is not allowed.

   ```sql
   SQL> grant role1 to role2;
   Grant succeeded.
   
   SQL> grant role2 to role1;
   grant role2 to role1
   * ERROR at line 1:ORA-01934: circular role grant detected
   ```
There's more...

You should be careful about granting privileges to the PUBLIC role because then every database user can use these privileges.

Suppose that user mike grants object privilege to user jessica with a grant option and user jessica grants that privilege to user tom. If user mike revokes that privilege from jessica, it will be automatically revoked from tom.

Revoking a system privilege will not cascade.

```sql
SQL> grant select on hr.employees to jessica with grant option;
Grant succeeded.

SQL> connect jessica
Enter password:
Connected.

SQL> grant select on hr.employees to tom;
Grant succeeded.

SQL> connect tom/oracle_123
Connected.

SQL> select count(*) from hr.employees;
COUNT(*)
--------
  107

SQL> connect mike/welcome1
Connected.

SQL> revoke select on hr.employees from jessica;
Revoke succeeded.

SQL> connect tom/oracle_123
Connected.

SQL> select count(*) from hr.employees;
```
ERROR at line 1:
ORA-00942: table or view does not exist

You cannot revoke object privileges you didn’t grant.

See also

- If you want to learn more about roles, see the official Oracle documentation—Oracle Database Security Guide 12c Release 1 (refer Chapter 4, Configuring Privilege and Role Authorization, of this documentation).

The sysbackup privilege – how, when, and why should you use it?

It is recommended that you use the **sysbackup** administrative privilege instead of the **sysdba** administrative privilege to perform operations related to backup and recovery tasks.

Getting ready

For this recipe, you’ll need:

- An existing database user (for example, **tom**) and a password file in 12c format, if you want to complete it using a password-authenticated user
- An existing OS user (for example, **john**), who belongs to the **backupdba** OS group, in order to connect to the database using OS authentication

How to do it…

Instructions are given in the **Database authentication** and **OS authentication** sections.
Database authentication

The instructions for database authentication are as follows:

1. Connect to the database as **sysdba** (or another user that can grant the **sysbackup** privilege):

   ```sql
   sqlplus / as sysdba
   ```

2. Grant the **sysbackup** privilege to user **tom**:

   ```sql
   grant sysbackup to tom;
   ```

3. Verify that there is an entry in the password file that grants user **tom** the **sysbackup** administrative privilege. Select data from the **v$pwfile_users** view:

   ```sql
   select * from v$pwfile_users;
   ```

   The following table is the result of the preceding command:

<table>
<thead>
<tr>
<th>Username</th>
<th>sysdb</th>
<th>sysop</th>
<th>sysas</th>
<th>sysba</th>
<th>sysdg</th>
<th>syskm</th>
<th>con_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>sys</td>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>0</td>
</tr>
<tr>
<td>sysdg</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>0</td>
</tr>
<tr>
<td>sysbackup</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>0</td>
</tr>
<tr>
<td>syskm</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>0</td>
</tr>
<tr>
<td>tom</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>0</td>
</tr>
</tbody>
</table>

4. Test the connection using **RMAN**:

   ```sql
   rman target '"
tom\|oracle_123 as sysbackup"'
   ```
OS authentication

The instructions for OS authentication are as follows:

1. Verify that the OS user (for example, john) is a member of the backupdba OS group:
   
   $ id john

2. Connect to the database using the sysbackup privilege (SQL*Plus or RMAN):
   
   $> sqlplus / as sysbackup
   $> rman target "'/ as sysbackup"

How it works...

You can use either Oracle Recovery Manager (RMAN) or SQL*Plus to perform the operations. When you connect to the database as sysbackup, you are connected as a predefined user sysbackup. If you want to check this, run the following statement:

   SQL> select user from dual;

Otherwise, the following statement:

   SQL> show user

Using the sysbackup privilege, you can connect to the database even when it is not open. This privilege enables better separation of duties and the implementation of the least privilege principle.

From a security perspective, it is recommended that you implement the least privilege principle. The least privilege principle is an important security concept that requires that users are given only those privileges they need to perform their job.
To view the list of privileges a user can exercise when connected to the database using `sysbackup` privilege, you can create a user (for example, `tom`) and grant the user only `sysbackup` privileges. The next step is to connect to the database as user `tom`, using the `sysbackup` privilege and the `execute` statement:

```sql
select * from session_privs;
```

These privileges are shown in the following table:

<table>
<thead>
<tr>
<th>Privileges (output from the previous statement)</th>
<th>select any transaction</th>
<th>select any dictionary</th>
<th>resumable</th>
<th>alter database</th>
<th>audit any</th>
<th>create any cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysbackup</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>create any directory</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>create any table</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>alter session</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This is how you can check `enabled` roles:

```sql
SQL> select * from session_roles;
```

<table>
<thead>
<tr>
<th>ROLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT_CATALOG_ROLE</td>
</tr>
<tr>
<td>HS_ADMIN_SELECT_ROLE</td>
</tr>
</tbody>
</table>

`HS_ADMIN_SELECT_ROLE` is granted to `SELECT_CATALOG_ROLE`.  

[ 36 ]
If you want to view the roles and privileges granted to `sysbackup`, you can query `DBA_ROLE_PRIVS` and `DBA_SYS_PRIVS`:

```
SQL> select * from dba_role_privs where grantee='SYSBACKUP';
SQL> select * from dba_sys_privs where grantee='SYSBACKUP';
```

Also, this new administrative privilege enables you to select, insert, delete, execute, and perform operations:

<table>
<thead>
<tr>
<th>SELECT</th>
<th>PERFORM operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>X$ tables</td>
<td>STARTUP, SHUTDOWN</td>
</tr>
<tr>
<td>V$ and GV$ views</td>
<td>CREATE PFILE, CREATE SPFILE</td>
</tr>
<tr>
<td>APPQOSSYS.WLM_CLASSIFIER_PLAN</td>
<td>CREATE CONTROLFILE</td>
</tr>
<tr>
<td>SYSTEM.LOGSTDBY$PARAMETERS</td>
<td>FLASHBACK DATABASE</td>
</tr>
<tr>
<td><strong>INSERT/DELETE</strong></td>
<td></td>
</tr>
<tr>
<td>SYS.APPLY$_SOURCE_SCHEMA</td>
<td>CREATE/DROP RESTORE POINT (including GUARANTEED restore points)</td>
</tr>
<tr>
<td>SYSTEM.LOGSTDBY$PARAMETERS</td>
<td></td>
</tr>
<tr>
<td><strong>EXECUTE</strong></td>
<td></td>
</tr>
<tr>
<td>SYS.DBMS_BACKUP_RESTORE</td>
<td>SYS.DBMS_DATAPUMP</td>
</tr>
<tr>
<td>SYS.DBMS_RCVMAN</td>
<td>SYS.DBMS_IR</td>
</tr>
<tr>
<td>SYS.DBMS_PIPE</td>
<td>SYS.SYS_ERROR</td>
</tr>
<tr>
<td>SYS.DBMS_TTS</td>
<td>SYS.DBMS_TDB</td>
</tr>
<tr>
<td>SYS.DBMS_PLUGTS</td>
<td>SYS.DBMS_PLUGTSP</td>
</tr>
</tbody>
</table>

It is important for you to remember that:
When using the `sysbackup` privilege, you can’t view application data.
There's more…

You can't drop user \texttt{sysbackup}. In a multitenant environment, you can restrict a user to be able to perform backups only for the PDB it can connect to. You can accomplish that by creating a local user in the PDB and granting the \texttt{sysbackup} privilege to the user. When you are connected to the database as the \texttt{sysbackup}, you are connected as \texttt{sysbackup} user to \texttt{SYS} schema:

\begin{verbatim}
SQL> connect / as sysbackup
Connected.

SQL> show user
USER is "SYSBACKUP"

SQL> select sys_context( 'userenv', 'current_schema' ) from dual;
SYS_CONTEXT('USERENV','CURRENT_SCHEMA')
---------------------------------------
SYS
\end{verbatim}

See also

- Creating password-authenticated users
- Creating and using OS-authenticated users

The syskm privilege – how, when, and why should you use it?

It is recommended that you use the \texttt{syskm} administrative privilege instead of the \texttt{sysdba} administrative privilege to perform operations related to managing the transparent data encryption (TDE) keystore.
Getting ready

For this recipe, you'll need:

- An existing database user (for example, jessica) and a password file in the 12c format, if you want to complete it using a password-authenticated user
- An existing OS user (for example, bob), who belongs to the kmdba OS group, in order to connect to the database using OS authentication

How to do it…

Instructions are split into sections for database authentication and OS authentication.

Database authentication

The instructions for database authentication are as follows:

1. Connect to the database as sysdba (or another user that can grant the syskm privilege):

   sqlplus / as sysdba

2. Grant the syskm privilege to user jessica:

   grant syskm to jessica;

3. Connect user jessica to the database as syskm:

   SQL> connect jessica/oracle_1 as syskm

4. View privileges:

   SQL> select * from user_tab_privs;
   SQL> select * from session_privs;
OS authentication

The instructions for OS authentication are as follows:

1. Verify that an OS user (for example, bob) is a member of the kmdba OS group.

   $ id bob

2. Connect to the database using syskm privilege:

   $ sqlplus / as syskm

How it works…

When you connect to the database as syskm, you are connected as a predefined user, syskm. Using the syskm privilege, you can connect to the database even when it is not open.

In most circumstances when using TDE, you don’t have to have syskm administrative privilege. For a more detailed discussion about TDE operations and which privileges users need, see recipes in Chapter 8, Transparent Data Encryption.

In the Database authentication section after completing step 3, you can perform operations related to managing the TDE keystore. Step 4 is not necessary and its sole purpose is to show you which privileges you can use when connected as syskm. These privileges are:

- ADMINISTER KEY MANAGEMENT
- CREATE SESSION
- SELECT on V$ (and GV$) views:

  - SYS.V$ENCRYPTED_TABLESPACES
  - SYS.V$ENCRYPTION_WALLET
  - SYS.V$WALLET
  - SYS.V$ENCRYPTION KEYS
  - SYS.V$CLIENT_SECRETS
  - SYS.DBA_ENCRYPTION_KEY_USAGE
  - SYS.DATABASE_KEY_INFO

It is important for you to remember that:

When using syskm privilege, you can't view the application data.
There's more…
You can't drop user `syskm`.
When you are connected to the database as `syskm`, you are connected as the `syskm` user to `SYS` schema:

```
SQL> connect / as syskm
Connected.

SQL> show user
USER is "SYSKM"

SQL> select sys_context('userenv', 'current_schema') from dual;
SYS_CONTEXT('USERENV','CURRENT_SCHEMA')
---------------------------------------
SYS
```

See also

- Creating password-authenticated users
- Creating and using OS-authenticated users
- Chapter 8, Transparent Data Encryption

The sysdg privilege – how, when, and why should you use it?

It is recommended that you use the `sysdg` administrative privilege instead of `sysdba` administrative privilege to perform operations related to data guard tasks.

Getting ready

For this recipe, you'll need:

- An existing database user (for example, `mike`) and a password file in the 12c format if you want to complete it using a password-authenticated user
- An existing OS user (for example, `kelly`), who belongs to the `dgdba` OS group in order to connect to the database using OS authentication
How to do it…

Instructions are split into sections for database authentication and OS authentication.

Database authentication

The instructions for database authentication are as follows:

1. Connect to the database as `sysdb` (or another user who can grant the `sysd` privilege):

   ```sql
   sqlplus / as sysdb
   ```

2. Grant `SYSD` privilege to user `mike`:

   ```sql
   SQL> grant sysd to mike;
   ```

3. Exit SQL*Plus, connect `mike` using the `dgmgrl` command-line interface:

   ```sql
   SQL> exit
   $ dgmgrl
   DGMGRL> connect mike/test_1
   ```

OS authentication

The instructions for OS authentication are as follows:

1. Verify that the OS user (for example, `kelly`) is a member of the `dgdba` OS group:

   ```bash
   $ id kelly
   ```

2. Connect using the `dgmgrl` utility and OS authentication:

   ```bash
   $ dgmgrl
   DGMGRL> connect /
   ```
How it works…

When you connect to the database as sysdg, you are connected as a predefined user, sysdg. Using the sysdg privilege, you can connect to the database even when it is not open.

After completing step 2 successfully in the Database authentication section, user mike, as expected, can grant/revoke sysdg privilege to/from another existing user. If you want to try it out, type the statements given here.

After you connect to the database using the sysdg administrative privilege, you can perform the following operations:

<table>
<thead>
<tr>
<th>Operations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STARTUP, SHUTDOWN</td>
<td>CREATE SESSION</td>
</tr>
<tr>
<td>ALTER SESSION</td>
<td>SELECT ANY DICTIONARY</td>
</tr>
<tr>
<td>ALTER DATABASE</td>
<td>FLASHBACK DATABASE</td>
</tr>
<tr>
<td>ALTER SYSTEM</td>
<td>EXECUTE SYS.DBMS_DRS</td>
</tr>
<tr>
<td>CREATE/DROP RESTORE POINT (including GUARANTEED restore points)</td>
<td>SELECT X$ tables, V$ and GV$ views</td>
</tr>
<tr>
<td>DELETE APPQOSSYS.WLM_CLASSIFIER_PLAN</td>
<td>SELECT APPQOSSYS.WLM_CLASSIFIER_PLAN</td>
</tr>
</tbody>
</table>

It is important for you to remember that:

When using the sysdg administrative privilege, you can't view application data.
There's more...

You can't drop user sysdg. 
When you are connected to the database as sysdg, you are connected as sysdg user to the SYS schema:

```
SQL> connect / as sysdg  
Connected.

SQL> show user
USER is "SYSDG"

SQL> select sys_context( 'userenv', 'current_schema' ) from dual;
SYS_CONTEXT('USERENV','CURRENT_SCHEMA')
------------------------------------------------------------------
SYS
```

See also

- Creating password-authenticated users
- Creating and using OS-authenticated users
Security Considerations in Multitenant Environment

In this chapter, we will cover the following tasks:

- Creating a common user
- Creating a local user
- Creating a common role
- Creating a local role
- Granting privileges commonly
- Granting privileges locally
- Granting common and local roles
- The effects of plugging/unplugging operations on users, roles, and privileges

Introduction

The Oracle multitenant environment is a new architecture of Oracle Database, introduced in version 12c (12.1.0.1). It brings major changes to the way Oracle Database administrators think about the concept of databases and how they work (in a multitenant environment). One of the most significant changes is that many databases (up to 252) can share one database instance.

This chapter is focused on some of the security considerations concerning common and local users, roles, and privileges. The prerequisite for understanding recipes in this chapter is to have at least basic knowledge of fundamental multitenant concepts, such as what is a container database (CDB), pluggable database (PDB), root container, and seed.
Figure 1 shows the traditional architecture of Oracle Database.

![Figure 1 – A traditional architecture](image)

Figure 2 shows the separation of the data dictionary in a multitenant architecture:

![Figure 2 – Data Dictionary separation](image)
Figure 3 shows a multitenant architecture. To learn more about it, see the Oracle official guide, *Oracle Database Concepts, 12c Release 1 (12.1), Part VI Multitenant Architecture*.

![Multitenant Architecture Diagram]

For all recipes in this chapter, you can use Oracle Database 12c Enterprise Edition with the multitenant option. All of the concepts presented in this chapter also apply to the single-tenant architecture (one CDB and one PDB), which exists in all editions of Oracle Database 12c. Also, for all recipes in this chapter, it is assumed that a container database (cdb1) is up and running. Also, the EM Cloud Control version should be 12.1.0.3+.

The default prompt in SQL*Plus is SQL>. In this chapter, the glogin.sql script (located under $ORACLE_HOME/sqlplus/admin) is changed so that the prompt reflects the connected user and the current container. The only purpose is to make it easier to follow who is doing what and where. You don’t have to change the prompt.

### Creating a common user

A **common user** is a user created in the root container, which has the same identity across all containers. The main purpose of a common user is to perform “infrastructure” administrative tasks, such as starting up a CDB, plugging and unplugging PDBs, and opening PDBs. There are two types of common users: Oracle-supplied (for example, **SYS** and **SYSTEM**) and user-created common users.
Getting ready
To complete this recipe, you’ll need an existing common user who has create user privilege granted commonly.

How to do it…

1. Connect to the root container as a common user who has create user privilege granted commonly (for example, c##zoran or system user):

```
SQL> connect c##zoran@cdb1
```

2. Create a common user (for example, c##maja):

```
c##zoran@cdb1> create user c##maja identified by oracle1 container=all;
```

How it works…

C##maja is actually not a single user, but each container has a user named C##maja and the passwords must be the same.
Rules/guidelines for creating and managing common users

There are a few rules you should be aware of:

- The name of a common user must be unique across all containers. In version 12.1.0.1, it must begin with c## or C## unless you change the internal parameter common_user_prefix (which you shouldn't do on a production system without approval from Oracle Support) and, in version 12.1.0.2, it is best practice to use a prefix (default value c## or C##). However, you can choose it by changing the value of the common_user_prefix parameter (this naming convention doesn't apply to Oracle-supplied users in either version).
- A common user can have different privileges in different containers.
- The schemas for a common user may contain different objects in different containers.

The column oracle_maintained (in DBA_USERS) provides information as to whether a user is created and maintained by Oracle-supplied scripts:

```
c##zoran@cDB1> select username, oracle_maintained from dba_users where username='SYSTEM' or username='C##ZORAN';
```

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM</td>
<td>Y</td>
</tr>
<tr>
<td>C##ZORAN</td>
<td>N</td>
</tr>
</tbody>
</table>

There's more…

You can also create common users by using Oracle Enterprise Manager Cloud Control (OEM) 12c.

How to create a common user using OEM 12c

1. Start OEM 12c and log in using user SYSMAN or SYSTEM.

2. From the Databases page, select the root database in which you want to create a common user. The database home page appears.
3. From the **Administration** menu, select **Security** (a drop-down menu) and then **Users** (see Figure 5):

![Figure 5]

4. If prompted, log in to the root as a common user who has a **create user** privilege (for example, `c##zoran`; see Figure 6):

![Figure 6]
5. Click on the **Create** button (see Figure 7):

![Figure 7](attachment:image1)

6. To create a common user, it is enough to fill out the following fields on the **General** tab: **Name** (for example, `c#john`), **Enter Password**, and **Confirm Password** (see Figure 8) and then click on the **OK** button:

![Figure 8](attachment:image2)
Creating a local user

A local user is a user that is created and that exists in only one PDB. A local user can't be created in the root container.

Getting ready

A pluggable database (in our case, pdb1) should be open. You'll need an existing user (either common or local) who has create user privilege in that pluggable database.

How to do it…

1. Connect to PDB (for example, pdb1) as a common user or local user who has create user privilege in that PDB (for example, c##zoran or system user):

   SQL> connect c##zoran@pdb1

2. Create a local user (for example, mike):

   c##zoran@PDB1> create user mike identified by pa3t5brii container=current;

How it works…

Figure 9
Rules/guidelines for creating and managing local users

There are a few rules you should be aware of:

- The name of a local user must be unique within its pluggable database and it must not begin with c## or C##
- A local user cannot be created in the root
- A local user exists in one and only one PDB and owns a schema in that PDB

There's more…

You can also create local users by using *Oracle Enterprise Manager Cloud Control (OEM) 12c*.

How to create a local user using OEM 12c

You can follow the steps given in the *How to create a common user using OEM 12c* section, except that, in Step 2, you should connect to the pluggable database (for example, pdb1) instead of the root. Also, you can connect to PDB as a local user who has a local create user privilege. If you want to switch container, you should click on **Container Switcher** and a drop-down menu will open (see Figure 10):
In Figure 11, it is shown that the common user you created in the previous recipe is created in the pluggable database (for example, c##john is created in pdb1; the common user is created in all pluggable databases that reside in the CDB and will be created in all future PDBs). By clicking on the Create button shown in Figure 11, you can create (only) a local user:

![Figure 11](image)

Creating a common role

**Common roles** are roles created in the root container and they exist in all containers. These roles can have a different set of privileges in different containers and they can be granted to either common or local users or roles.
Getting ready
To complete this recipe, you’ll need an existing common user who has `create role` privilege granted commonly.

How to do it…

1. Connect to the root container as a common user who has `create role` privilege granted commonly (for example, `c##zoran` or system user):

   ```sql
   SQL> connect c##zoran@cdb1
   ```

2. Create a common role (for example, `c##role1`):

   ```sql
   SQL> create role c##role1 container=all;
   ```

How it works…
When you create a common role, that role exists in all containers in that database (including a root container and existing and future pluggable databases).

Figure 12
Security Considerations in Multitenant Environment

```sql
c##zoran@CDB1> select * from dba_roles where role='C##ROLE1';

ROLE                  PASSWORD AUTHENTICAT   COM  O
----------------      --------  -----------  ---  -
C##ROLE1              NO        NONE         YES  N

```

```sql
c##zoran@CDB1> connect c##zoran/oracle@pdb1
Connected.

c##zoran@PDB1> select * from dba_roles where role='C##ROLE1';

ROLE                  PASSWORD AUTHENTICAT   COM  O
----------------      --------  -----------  ---  -
C##ROLE1              NO        NONE         YES  N

```

```sql
c##zoran@PDB1> connect c##zoran/oracle@pdb2
Connected.

c##zoran@PDB2> select * from dba_roles where role='C##ROLE1';

ROLE                  PASSWORD AUTHENTICAT   COM  O
----------------      --------  -----------  ---  -
C##ROLE1              NO        NONE         YES  N

```

There's more…

You can also create common roles by using Oracle Enterprise Manager Cloud Control (OEM) 12c.
How to create a common role using OEM 12c

You should connect to the root (CDB$ROOT) as a common user who has `create role` privilege granted commonly (for example, c##zoran or system user). From the Administration menu, select Security (drop-down menu) and then Roles (see Figure 13):
On the **Roles** page, click on the **Create** button and the **Create Role** page appears (Figure 14):

![Create Role Page](image)

Figure 14

On the **Create Role** page, you name the role on the **General** tab (for example, `c##role2`). Also, you may grant other roles and privileges to `c##role2` (using the tabs **Roles**, **System Privileges**, and **Object Privileges**). After choosing the options and granting privileges to the role, click on the **OK** button to create it.

### Creating a local role

**Local roles** are roles created in PDB and they exist only in that PDB. These roles can be granted *only locally* to either common or local users or roles.

### Getting ready

For this recipe, a pluggable database (in our case, `pdb1`) should be open. You’ll need an existing user (either common or local) who has `create role` privilege in that pluggable database.
Security Considerations in Multitenant Environment

How to do it…

1. Connect to PDB (for example, pdb1) as a common or local user who has create role privilege in that PDB (for example, c##maja):

   SQL> connect c##maja@pdb1

2. Create a local role (for example, local_role1):

   c##maja@PDB1> create role local_role1 container=current;

How it works…

When you create a local role, that role exists only in the pluggable database in which it is created. Local roles cannot be created in the root container. These roles are traditional roles.

![Diagram showing a CDB with a root container and two PDBs, with a local role indicated.](image)

**Figure 15**

```
c##maja@CDB1> select * from dba_roles where rol e='LOCAL_ROLE1';
no rows selected
```

```
c##maja@CDB1> connect c##maja/oracle@pdb1
Connected.
```

```
c##maja@pdb1> select * from dba_roles where role='LOCAL_ROLE1';
no rows selected
```
Security Considerations in Multitenant Environment

<table>
<thead>
<tr>
<th>ROLE</th>
<th>PASSWORD</th>
<th>AUTHENTICAT</th>
<th>COM</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCAL_ROLE1</td>
<td>NO</td>
<td>NONE</td>
<td>NO</td>
<td>N</td>
</tr>
</tbody>
</table>

```
c##maja@pdb1> connect c##maja/oracle@pdb2
Connected.

c##maja@pdb2> select * from dba_roles where role='LOCAL_ROLE1';
no rows selected
```

There's more...
You can also create local roles by using Oracle Enterprise Manager Cloud Control (OEM) 12c.

How to create a local role using OEM 12c

You should connect to PDB (for example, pdb1) as a common or local user who has create role privilege in that PDB (for example, c##maja). All the remaining steps are done in the same way as in the How to create a common role using OEM 12c section.

Granting privileges and roles commonly

The common privilege is a privilege that can be exercised across all containers in a container database. Depending only on the way it is granted, a privilege becomes common or local. When you grant a privilege commonly (across all containers) it becomes a common privilege. Only common users or roles can have common privileges. Only common role can be granted commonly.

Getting ready

For this recipe, you will need to connect to the root container as an existing common user who is able to grant a specific privilege or existing role (in our case, create session, select any table, c##role1, c##role2) to another existing common user (c##john). If you want to try out examples in the How it works section, you should open pdb1 and pdb2.
You will use the following:

- Common users `c##maja` and `c##zoran` with the dba role granted commonly
- Common user `c##john`
- Common roles `c##role1` and `c##role2`

**How to do it…**

1. You should connect to the root container as a common user who can grant these privileges and roles (for example, `c##maja` or system user):

   ```sql
   SQL> connect c##maja@cdb1
   ```

2. Grant a privilege (for example, `create session`) to a common user (for example, `c##john`) commonly:

   ```sql
   c##maja@cDB1> grant create session to c##john container=all;
   ```

3. Grant a privilege (for example, `select any table`) to a common role (for example, `c##role1`) commonly:

   ```sql
   c##maja@cDB1> grant select any table to c##role1 container=all;
   ```

4. Grant a common role (for example, `c##role1`) to a common role (for example, `c##role2`) commonly:

   ```sql
   c##maja@cDB1> grant c##role1 to c##role2 container=all;
   ```

5. Grant a common role (for example, `c##role2`) to a common user (for example, `c##john`) commonly:

   ```sql
   c##maja@cDB1> grant c##role2 to c##john container=all;
   ```
How it works…

You can grant privileges or common roles commonly only to a common user. You need to connect to the root container as a common user who is able to grant a specific privilege or role.

In Step 2, system privilege, create session is granted to the common user c##john commonly by adding a container=all clause to the grant statement. This means that the user c##john can connect (create session) to the root or any pluggable database in this container database (including all pluggable databases that will be plugged in in the future).

Note that the container = all clause is NOT optional even though you are connected to the root. Unlike during the creation of common users and roles (if you omit container=all, the user or role will be created in all containers commonly), if you omit this clause during the privilege or role grant, the privilege or role will be granted locally and it can be exercised only in root container.

```
SQL> connect c##john/oracle@pdb1
Connected.
```

```
c##john@CDB1> connect c##john/oracle@pdb1
```

Figure 16
Connected.

c##ohn@PDB1> connect c##ohn/oracle@pdb2
Connected.

c##ohn@PDB2>

In step 3, system privilege `select any table` is granted to the common role `c##role1` commonly. This means that the role `c##role1` contains the `select any table` privilege in all containers (root and pluggable databases):

```
c##zoran@CDB1> select * from role_sys_privs where role='C##ROLE1';
ROLE                     PRIVILEGE                     ADM  COM
-----------------         ------------------            ---  ---
C##ROLE1                 SELECT ANY TABLE               NO  YES
```

c##zoran@CDB1> connect c##zoran/oracle@pdb1
Connected.

c##zoran@PDB1> select * from role_sys_privs where role='C##ROLE1';
ROLE                     PRIVILEGE                     ADM  COM
-----------------         ------------------            ---  ---
C##ROLE1                 SELECT ANY TABLE               NO  YES

c##zoran@PDB1> connect c##zoran/oracle@pdb2
Connected.

c##zoran@PDB2> select * from role_sys_privs where role='C##ROLE1';
ROLE                     PRIVILEGE                     ADM  COM
-----------------         ------------------            ---  ---
C##ROLE1                 SELECT ANY TABLE               NO  YES
In Step 4, the common role `c##role1` is granted to another common role `c##role2` commonly. This means that the role `c##role2` has granted the role `c##role1` in all containers:

```
c##zoran@CDB1> select * from role_role_privs where role='C##ROLE2';
ROLE               GRANTED_ROLE
-----------------        ----------------------       ---- ---
C##ROLE2               C##ROLE1                     NO   YES
```

```
c##zoran@CDB1> connect c##zoran/oracle@pdb1
Connected.
```

```
c##zoran@PDB1> select * from role_role_privs where role='C##ROLE2';
ROLE               GRANTED_ROLE
-----------------        ----------------------       ---- ---
C##ROLE2               C##ROLE1                     NO   YES
```

```
c##zoran@PDB1> connect c##zoran/oracle@pdb2
Connected.
```

```
c##zoran@PDB2> select * from role_role_privs where role='C##ROLE2';
ROLE               GRANTED_ROLE
-----------------        ----------------------       ---- ---
C##ROLE2               C##ROLE1                     NO   YES
```

In step 5, the common role `c##role2` is granted to the common user `c##john` commonly. This means that the user `c##john` has `c##role2` in all containers.
Consequently, the user `c##john` can use the `select any table` privilege in all containers in this container database:

```
c##john@CDB1> select count(*) from c##zoran.t1;
COUNT(*)  
----------
     4

C##john@CDB1> connect c##john/oracle@pdb1
Connected.

C##john@PDB1> select count(*) from hr.employees;
COUNT(*)  
----------
   107

C##john@PDB1> connect c##john/oracle@pdb2
Connected.

C##john@PDB2> select count(*) from sh.sales;
COUNT(*)  
----------
918843
```

Granting privileges and roles locally

A local privilege is a privilege than can be exercised only in a container in which it is granted. Depending only on the way it is granted, a privilege becomes common or local. When you grant privilege locally (in the current container), it becomes a local privilege. Both common and local users or roles can have local privileges.
Getting ready

For this recipe, you’ll need an existing user (c##maja) who can grant some privileges (for example, create procedure, create table, create view, and create synonym) and roles (c##role1, c##role2, c##role3, c##role4, and local_role1) in a specific container (root or PDB; in our case, pdb1) to existing users and roles (c##john, mike, local_role1, c##role1, c##role3, and c##role4).

How to do it…

1. You should connect to the container (root or pluggable database) in which you want to grant the privilege as a common or local user who can grant that privilege (for example, c##maja):

   SQL> connect c##maja@pdb1

2. Grant a privilege (for example, create synonym) to a common user (for example, c##john) locally:

   c##maja@PDB1> grant create synonym to c##john container=current;

3. Grant a privilege (for example, create view) to a local user (for example, mike) locally:

   c##maja@PDB1> grant create view to mike container=current;

4. Grant a privilege (for example, create table) to a common role (for example, c##role1) locally:

   c##maja@PDB1> grant create table to c##role1 container=current;

5. Grant a privilege (for example, create procedure) to a local role (for example, local_role1) locally:

   c##maja@PDB1> grant create procedure to local_role1 container=current;
6. Grant a common role (for example, c##role2) to another common role (for example, c##role3) locally:

   c##maja@PDB1> grant c##role2 to c##role3 container=current;

7. Grant a common role (for example, c##role3) to a local role (for example, local_role1) locally:

   c##maja@PDB1> grant c##role3 to local_role1 container=current;

8. Grant a local role (for example, local_role1) to a common role (for example, c##role4) locally:

   c##maja@PDB1> grant local_role1 to c##role4 container=current;

9. Grant a common role (for example, c##role4) to a common user (for example, c##john) locally:

   c##maja@PDB1> grant c##role4 to c##john container=current;

How it works…

In the previous section, we have seen different types of local grants. Local grants are valid only in the current container even though the granted user (or role) is common. Consequently, common users and common roles can have a different set of privileges in different containers. Steps 3, 5, 7, and 8 can’t be done in the root container because there are no local users and local roles in the root container.

Effects of plugging/unplugging operations on users, roles, and privileges

The purpose of this recipe is to show what is going to happen to users, roles, and privileges when you unplug a pluggable database from one container database (cdb1) and plug it into some other container database (cdb2).
Getting ready

To complete this recipe, you will need the following:

- Two container databases (cdb1 and cdb2)
- One pluggable database (pdb1) in the container database cdb1
- Local user mike in the pluggable database pdb1 with the local create session privilege
- The common user c##john with the create session common privilege and create synonym local privilege on the pluggable database pdb1

How to do it…

1. Connect to the root container of cdb1 as user sys:
   ```
   SQL> connect sys@cdb1 as sysdba
   ```

2. Unplug pdb1 by creating an XML metadata file:
   ```
   SQL> alter pluggable database pdb1 unplug into '/u02/oradata/pdb1.xml';
   ```

3. Drop pdb1 and keep the datafiles:
   ```
   SQL> drop pluggable database pdb1 keep datafiles;
   ```

4. Connect to the root container of cdb2 as user sys:
   ```
   SQL> connect sys@cdb2 as sysdba
   ```

5. Create (plug) pdb1 to cdb2 by using the previously created metadata file:
   ```
   SQL> create pluggable database pdb1 using '/u02/oradata/pdb1.xml' nocopy;
   ```
How it works…

By completing the previous steps, you unplugged pdb1 from cdb1 and plugged it into cdb2. After this operation, all local users and roles (in pdb1) are migrated with the pdb1 database.

The following is how you try to connect to pdb1 as a local user:

```
SQL> connect mike@pdb1
```

All local privileges are migrated even if they are granted to common users/roles. However, if you try to connect to pdb1 as a previously created common user, c##john, you’ll get an error, as follows:

```
SQL> connect c##john@pdb1
ERROR: 
ORA-28000: the account is locked 
Warning: You are no longer connected to ORACLE.
```

This happened because, after migration, common users are migrated in a pluggable database as locked accounts. You can continue to use objects in these users' schemas, or you can create these users in a root container of a new CDB. To do this, we first need to close pdb1:

```
sys@cDB2> alter pluggable database pdb1 close;
Pluggable database altered.

sys@cDB2> create user c##john identified by oracle container=all;
User created.

sys@cDB2> alter pluggable database pdb1 open;
Pluggable database altered.
```
If we try to connect to pdb1 as the user c##john, we will get the following error:

```
SQL> conn c##john/oracle@pdb1
ERROR: 
ORA-01045: user C##JOHN lacks CREATE SESSION privilege; logon denied
Warning: You are no longer connected to ORACLE.
```

Even though c##john had the create session common privilege in cdb1, he cannot connect to the migrated PDB. This is because common privileges are not migrated! So, we need to give the create session privilege (either common or local) to the user c##john, as follows:

```
sys@CDB2> grant create session to c##john container=all;
Grant succeeded.
```

In the earlier recipe (Granting privileges and roles locally), we granted a create synonym local privilege to a user, c##john. Let's try this privilege on the migrated pdb2:

```
c##john@PDB1> create synonym emp for hr.employees;
Synonym created.
```

This proves that local privileges are always migrated.
In this chapter, we will cover the following tasks:

- Creating and using definer's rights procedures
- Creating and using invoker's rights procedures
- Using code-based access control
- Restricting access to program units by using accessible by

**Introduction**

In this section, you will learn the definitions of concepts that will be used in the rest of the chapter.

**Definer** is the owner of a procedure.

**Invoker** is a user who uses (invokes) a procedure, but is not the definer of the procedure.

**Definer's rights procedure** is a procedure (or a program unit) that executes with the privileges of its definer.

**Invoker's rights procedure** is a procedure (or a program unit) that executes with the privileges of the invoker.

Another difference between definer's and invoker's rights procedures is that invoker's rights procedures are not bound to the schema in which they are located.
**Code base access control** is a new feature, introduced in Oracle Database 12c. It enables you to grant database roles to PL/SQL functions, procedures, or packages. You can use it with definer’s and invoker’s rights procedures.

The purpose of the **accessible by** clause is to limit the calling set of program units to be those in the **accessible by** clause and the unit itself.

For all the recipes in this chapter, you will use non-CDB 12c. We assume that the database is up and running.

### Creating and using definer's rights procedures

In this recipe, you'll learn to create and use definer's rights procedures.

**Getting ready**

To complete this recipe, you’ll use a user who has a DBA role.

**How to do it…**

1. Connect to the database as a user with the DBA role (for example, zoran)

   ```sql
   SQL> connect zoran
   ```

2. Create two users (**procowner** and **procuser**) and grant them appropriate privileges:

   ```sql
   SQL> create user procowner identified by oracle1;
   SQL> create user procuser identified by oracle2;
   SQL> grant create session, create procedure to procowner;
   SQL> grant create session to procuser;
   ```
3. Create a table called `zoran.tbl` and grant users privileges on this table:

```sql
SQL> create table zoran.tbl(a number, b varchar2(40));
SQL> insert into zoran.tbl values(1, 'old_value');
SQL> commit;
SQL> grant select on zoran.tbl to procuser;
SQL> grant update on zoran.tbl to procowner;
```

4. Connect as a user, `procowner`, create a procedure to update table `zoran.tbl`, and grant `execute` on this procedure to user `procuser`:

```sql
SQL> connect procowner/oracle1
CREATE OR REPLACE PROCEDURE UpdateTbl (x IN number, y IN varchar2)
  AUTHID DEFINER
  AS
  BEGIN
    UPDATE ZORAN.TBL
    SET b = y
    WHERE a = x;
  END;
/
SQL> grant execute on UpdateTbl to procuser;
```

5. Connect as user `procuser` and try to directly update table `zoran.tbl`:

```sql
SQL> connect procuser/oracle2
SQL> UPDATE ZORAN.TBL SET B = 'value1' WHERE A = 1;
UPDATE ZORAN.TBL SET B = 'value1' WHERE A = 1
  * ERROR at line 1:
  ORA-01031: insufficient privileges
```

6. When the previous step fails, update table by using the `UpdateTbl` procedure:

```sql
SQL> EXEC procowner.UpdateTbl(1, 'new_value');
PL/SQL procedure successfully completed.
```

7. Check whether the table is updated:

```sql
SQL> select * from zoran.tbl;
  A   B
---------- ----------------------------------------
 1  new_value
```
How it works…

Definer's rights procedures are executed by using privileges that are granted to the owner of the procedure. In our example, we have two users: procowner - a user who is the owner of the procedure and has privilege to update table zoran.tbl and procuser - a user who just executes the procedure. In step 4, procuser creates procedure by using the AUTHID DEFINER clause, which means that this procedure will be definer's rights procedure. This is a default behavior (we can omit the AUTHID DEFINER clause). In step 5, procuser tries to update table zoran.tbl directly, but it gets an error:

```
SQL> UPDATE ZORAN_TBL SET B = 'value1' WHERE A = 1;
UPDATE ZORAN_TBL SET B = 'value1' WHERE A = 1
    *
ERROR at line 1:
ORA-01031: insufficient privileges
```

This is the expected behavior, considering that procuser doesn't have an update privilege on zoran.tbl. When procuser executes the procedure in step 6, the table is updated because the privilege of the definer is applied.

Creating and using invoker's right procedures

In this recipe, you'll learn to create and use invoker's rights procedures. They can be useful when creating PL/SQL procedures in a highly privileged schema (because in this case, it is more secure to grant specific privileges to the invoker). Also, when there is no SQL code in the PL/SQL procedure and the procedure is available to other users, invoker's rights procedure will be executed more efficiently. There are no changes in the values of current schema and currently enabled roles during the execution (these changes are not necessary because without SQL in PL/SQL code, privilege checking is not performed).

Getting ready

To complete this recipe, you'll use a user who has the DBA role.
How to do it…

1. Connect to the database as a user with the DBA role (for example, zoran):

   SQL> connect zoran

2. Create two users (procuser1, procuser2) and grant them privileges:

   SQL> create user procuser1 identified by oracle1;
   SQL> create user procuser2 identified by oracle2;
   SQL> grant create session to procuser1;
   SQL> grant create session to procuser2;

3. Create the table table1 and grant select and update privileges on that table to procuser1 and only select privilege to procuser2:

   SQL> create table table1(a number, b varchar2(30));
   SQL> insert into zoran.table1 values(1, 'old_value');
   SQL> commit;
   SQL> grant select on zoran.table1 to procuser1;
   SQL> grant update on zoran.table1 to procuser1;
   SQL> grant select on zoran.table1 to procuser2;

4. Create an invoker's rights procedure to update table1:

   CREATE OR REPLACE PROCEDURE UpdateTable1 (x IN number, y IN varchar2)
   AUTHID CURRENT_USER
   AS
   BEGIN
       UPDATE ZORAN.TABLE1
       SET b = y
       WHERE a = x;
   END;
/

5. Grant execute on that procedure to procuser1 and procuser2:

   SQL> grant execute on zoran.UpdateTable1 to procuser1;
   SQL> grant execute on zoran.UpdateTable1 to procuser2;
6. Connect as user `procuser1` and execute the procedure `UpdateTable1`:

```
SQL> connect procuser1
SQL> EXEC zoran.UpdateTable1(1, 'new_value');
PL/SQL procedure successfully completed.
SQL> commit;
```

7. Check whether the table is updated:

```
SQL> select * from zoran.table1;
A          B
---------- ----------------------------------------
1          new_value
```

8. Connect as the user `procuser2` and try to execute the procedure `UpdateTable1`:

```
SQL> connect procuser2
SQL> EXEC zoran.UpdateTable1(1, 'newer_value');
BEGIN zoran.UpdateTable1(1, 'newer_value'); END;
*  
ERROR at line 1:
ORA-01031: insufficient privileges
ORA-06512: at "ZORAN.UPDATETABLE1", line 5
ORA-06512: at line 1
```

How it works…

Invoker's rights procedures are executed by using privileges that are granted to the user that executes the procedure. In step 4, the user `zoran` creates an invoker's rights procedure by specifying the `AUTHID CURRENT_USER` clause. When `procuser1` executes that procedure in step 6, he or she succeeds because `update` privilege is granted to `procuser1`, but when `procuser2` tries to execute it in step 8, he or she gets an error because `procuser2` lacks the `update` privilege on table1.
There's more…

Let's consider this security problem.

1. Connect as a user who has a DBA role (for example, zoran). Create a new user `maluser` and grant him the privileges `create session` and `create procedure`.

   SQL> create user maluser identified by oracle1;
   SQL> grant create session, create procedure to maluser;

2. Connect as the user `maluser` and create the following “malicious” procedure with the purpose of granting him the DBA role:

   SQL> connect maluser/oracle1
   create or replace procedure mal_proc
        authid current_user
        as
        begin
            execute immediate 'grant dba to maluser';
        end;
    /

3. Connect as a user who has a DBA role (for example, zoran) and execute the procedure you created in the previous step:

   SQL> connect zoran
   SQL> EXEC maluser.mal_proc;
   PL/SQL procedure successfully completed.

4. Connect as `maluser` and check whether the DBA role is granted:

   SQL> connect maluser
   SQL> select * from session_roles where role= 'DBA';
In this example, we've seen that a low-privileged user can trick the DBA user to grant him the DBA role, by tricking the DBA user (in this case, zoran) to execute an invoker's rights procedure that was created by low-privileged user (in this case, maluser). The user zoran can avoid this scenario by examining code that he is executing using his own privileges and specifying users whose procedures he wants to execute using his own privileges. The latter can be done by granting the INHERIT PRIVILEGE privilege to these users. Remember that this privilege is granted by default to public user, meaning that zoran can execute procedures from all users in the database. The first thing zoran can do is to revoke this privilege from the public user and then grant it only to users whose invoker's rights procedures he wants to execute. Let's try this:

1. Connect as a user zoran and revoke inherit privileges from public user:

   ```sql
   SQL> connect zoran
   SQL> revoke inherit privileges on user zoran from public;
   ```

   ![Figure 1](image-url)
2. Try to execute `mal_proc`:

```
SQL> EXEC maluser.mal_proc;
BEGIN maluser.mal_proc; END;
*
ERROR at line 1:
ORA-06598: insufficient INHERIT PRIVILEGES privilege
ORA-06512: at "MALUSER.MAL_PROC", line 1
ORA-06512: at line 1
```

Figure 2
3. Grant inherit rights privileges to maluser:

```sql
SQL> grant inherit privileges on user zoran to maluser;
```

Figure 3

4. Try again to execute `mal_proc`:

```sql
SQL> EXEC maluser.mal_proc;
PL/SQL procedure successfully completed.
```

Figure 4
Users who have the `inherit any privileges` system privilege are exempted from this rule, meaning that procedures from these users can be executed by all the users in the database. For example, we have the following steps:

1. Connect as a user who has a DBA role (for example, *zoran*), create two users, and grant them the following privileges:

   ```sql
   SQL> connect zoran
   SQL> create user super_user identified by oracle1;
   SQL> create user regular_user identified by oracle2;
   SQL> grant create session, create procedure to super_user;
   SQL> grant create session, create procedure to regular_user;
   ```

2. Grant `inherit any privileges` only to `super_user`:

   ```sql
   SQL> grant inherit any privileges to super_user;
   ```

3. Connect as `regular_user` and create the following procedure:

   ```sql
   SQL> connect regular_user
   create or replace procedure reg_proc
     authid current_user
     as
     begin
       execute immediate 'grant dba to regular_user';
     end;
   /  
   ```

4. Connect as `super_user` and create the following procedure:

   ```sql
   SQL> connect super_user
   create or replace procedure sup_proc
     authid current_user
     as
     begin
       execute immediate 'grant dba to super_user';
     end;
   /  
   ```
5. Connect as user zoran and try to execute `reg_proc` from `regular_user`
   (observe an error because `regular_user` doesn't have the `inherit
   privileges` privilege on user `zoran`):

   SQL> connect zoran/oracle_4U
   Connected.
   SQL> EXEC regular_user.reg_proc;
   BEGIN regular_user.reg_proc; END;
   *
   ERROR at line 1:
   ORA-06598: insufficient INHERIT PRIVILEGES privilege
   ORA-06512: at "REGULAR_USER.REG_PROC", line 1
   ORA-06512: at line 1

6. Try to execute `sup_proc` from `super_user` (this succeeds because, even though
   `super_user` doesn't have `inherit privileges` privilege on the user `zoran`, he
   has inherited any `privileges system` privilege, which can be interpreted as
   `inherit privileges` on all users of the database):

   SQL> EXEC super_user.sup_proc;
   PL/SQL procedure successfully completed.

Using code-based access control

In this recipe, you'll use code base access control with invoker's rights procedure.

Getting ready

To complete this recipe, you'll use a user who has a DBA role.

How to do it…

1. Connect to the database as a user with a DBA role (for example, `zoran`), create
   `proc_user`, and grant him the `create session` privilege:

   SQL> create user proc_user identified by oracle1;
   SQL> grant create session to proc_user;
2. Create table `tbl1` and insert test data:

```
SQL> create table tbl1(a number, b varchar2(30));
SQL> insert into tbl1 values (1, 'old_value');
SQL> commit;
```

3. Create the invoker's rights procedure `UpdateTbl1` and grant `execute` on that procedure to `proc_user`:

```
CREATE OR REPLACE PROCEDURE UpdateTbl1 (x IN number,
y IN varchar2)
AUTHID CURRENT_USER
AS
BEGIN
    UPDATE ZORAN.TBL1
    SET b = y
    WHERE a = x;
END;
/
SQL> grant execute on zoran.UpdateTbl1 to proc_user;
```

4. Create the role `proc_role` and grant `update` on `tbl1` to `proc_role`:

```
SQL> create role proc_role;
SQL> grant update on zoran.tbl1 to proc_role;
```

5. Grant `proc_role` to the procedure `UpdateTbl1`:

```
SQL> grant proc_role to procedure zoran.UpdateTbl1;
```

6. Connect as a user `proc_user`:

```
SQL> connect proc_user
```

7. Try to directly update the table:

```
SQL> update zoran.tbl1 set b = 'value1' where a = 1;
update zoran.tbl1 set b = 'value1' where a = 1
*ERROR at line 1:
ORA-00942: table or view does not exist
```

8. Execute the procedure `UpdateTbl1`:

```
SQL> execute zoran.UpdateTbl1(1, 'new_value');
PL/SQL procedure successfully completed.
```
9. Connect as the user zoran and verify whether the table is updated:

```sql
SQL> connect zoran
SQL> select * from tbl1;
A          B
---------- ----------------------------------------
1          new_value
```

**How it works…**

Code-based access control allows us to grant a role to a PL/SQL procedure, function, or package. It works with both definer’s rights and invoker’s rights procedures. The scenario in this example shows one use of this feature. The invoker’s rights procedure in step 3 created by zoran (UpdateTbl1), is used to update the table tbl1. Execute on this procedure is granted to the user proc_user. This is an invoker’s rights procedure, meaning that it is executed by using privileges granted to invoker (in our case, proc_user). However, proc_user doesn't have update privilege on this table, but he can still execute it successfully because procedure itself contains update privilege on tbl1 table, granted through the role proc_role in step 5.

**There's more…**

Remember that, in some cases, privileges granted to users via roles are not active during the PL/SQL calls. Let’s try this:

1. Connect as a user who has a DBA role (for example, zoran), create the user plsusr, and grant him the create session and create procedure privileges:

   ```sql
   SQL> create user plsusr identified by oracle1;
   SQL> grant create session, create procedure to plsusr;
   ```

2. Create the role plsrole1 and grant the create table privilege to it:

   ```sql
   SQL> create role plsrole1;
   SQL> grant create table to plsrole1;
   ```

3. Grant plsrole1 to the user plsusr:

   ```sql
   SQL> grant plsrole1 to plsusr;
   ```
4. Connect as **plsusr** and create the procedure **cr_table**:

   ```sql
   SQL> connect plsusr
   create or replace procedure cr_table
     authid definer
     as
     begin
       execute immediate 'create table test2(a int)';
     end;
   /
   ```

5. Create the table **test1** to check whether the **plsusr** user has a **create table** privilege:

   ```sql
   SQL> create table test1(a int);
   Table created.
   ```

6. Execute the **cr_table** procedure and observe the **insufficient privileges** error. Even though the user **plsusr** has a **create table** privilege, that privilege is granted via role and roles are not active during this PL/SQL call resulting in the **insufficient privileges** error.

   ```sql
   SQL> exec cr_table;
   BEGIN cr_table; END;
   *
   ERROR at line 1:
   ORA-01031: insufficient privileges
   ORA-06512: at "PLSUSR.CR_TABLE", line 5
   ORA-06512: at line 1
   ```

7. Connect as a user who has the DBA role and grant the **create table** privilege directly to the user **plsusr**:

   ```sql
   SQL> connect zoran
   SQL> grant create table to plsusr;
   ```

8. Connect as the user **plsusr** and try to execute the procedure **cr_table** again. This time, the **create table** privilege is granted directly; thus, it is active during the PL/SQL call, resulting in successful completion:

   ```sql
   SQL> connect plsusr/oracle1
   SQL> exec cr_table;
   PL/SQL procedure successfully completed.
   ```
Restricting access to program units by using accessible by

In this recipe, you'll learn about the effects of using the `accessible by` clause.

Getting ready

To complete this recipe, you'll use a user who has the `create procedure` privilege.

How to do it…

1. Connect as a user who has the `create procedure` privilege (for example, `zoran`):

   ```sql
   SQL> connect zoran
   ```

2. Create the `protected_pkg` package that is only accessible by `public_pkg`:

   ```sql
   CREATE OR REPLACE PACKAGE protected_pkg
       ACCESSIBLE BY (public_pkg)
   IS
     PROCEDURE protected_proc;
   END;
   /
   CREATE OR REPLACE PACKAGE BODY protected_pkg
   IS
     PROCEDURE protected_proc
     IS
       BEGIN
         DBMS_OUTPUT.PUT_LINE ('This is a Protected Procedure that can only be accessed from Public Package');
       END;
     END;
   /
   ```
3. Create the `public_pkg` package:

```sql
CREATE OR REPLACE PACKAGE public_pkg
IS
  PROCEDURE public_proc;
END;
/
CREATE OR REPLACE PACKAGE BODY public_pkg
IS
  PROCEDURE public_proc
  IS
    BEGIN
      DBMS_OUTPUT.PUT_LINE ('This is Public Procedure from Public Package!');
      protected_pkg.protected_proc;
    END;
  END;
/
```

4. Execute the `public_proc` procedure from `public_pkg`:

```sql
SQL> set serveroutput on
SQL> EXEC public_pkg.public_proc;
This is Public Procedure from Public Package!
This is a Protected Procedure that can only be accessed from Public Package
PL/SQL procedure successfully completed.
```

5. Try to directly execute `protected_proc` from `protected_pkg` and observe the error:

```sql
SQL> EXEC protected_pkg.protected_proc;
BEGIN protected_pkg.protected_proc; END;
```

```
ERROR at line 1:
ORA-06550: line 1, column 7:
PLS-00904: insufficient privilege to access object
PROTECTED_PKG
ORA-06550: line 1, column 7:
PL/SQL: Statement ignored
```
6. Try to create another package that accesses `protected_proc` from `protected_pkg`:

```sql
CREATE OR REPLACE PACKAGE other_pkg
IS
  PROCEDURE other_proc;
END;
/
CREATE OR REPLACE PACKAGE BODY other_pkg
IS
  PROCEDURE other_proc
  IS
    BEGIN
      DBMS_OUTPUT.PUT_LINE ('This is Other Procedure from Other Package!');
      protected_pkg.protected_proc;
    END;
  END;
/
Warning: Package Body created with compilation errors.
```

7. Find the compilation errors, as follows:

```
SQL> show errors
Errors for PACKAGE BODY OTHER_PKG:
LINE/COL ERROR
-------- --------------------------------------------------
7/7     PL/SQL: Statement ignored
7/7     PLS-00904: insufficient privilege to access object PROTECTED_PKG
```

**How it works…**

An `accessible by` clause enables us to specify which packages can access procedures and functions of another package. This process is called **white listing**. In step 2, we created the `protected_pkg` package and we specified that procedures and functions of this package can be accessed only by procedures and functions of `public_pkg` package. In step 4, we executed the `public_proc` procedure from the `public_pkg` package and, in output, we can observe that the `protected_proc` procedure has been successfully executed. However, if we try to execute `protected_proc` directly, we get an *insufficient privileges* error because the `accessible by` clause restricts execution of this procedure (step 5). Even if we try to create a new package with the procedure that calls the `protected_proc` procedure, we get an *insufficient privileges* error (steps 6 and 7).
Virtual Private Database

In this chapter, we will cover the following tasks:

- Creating different policy functions
- Creating Oracle Virtual Private Database row-level policies
- Creating column-level policies
- Creating a driving context
- Creating policy groups
- Setting context as a driving context
- Adding a policy to a group
- Exempting users from VPD policies

Introduction

Oracle Virtual Private Database (VPD) is a security feature, introduced in Oracle Database 8i. It is available only in Enterprise Edition of Oracle Database. Discretionary access control (DAC) grants/restricts access to data at an object level (for example, table level). This means that a user can access either the entire data in a table or no data. VPD enables you more granular control over security of your data. Using VPD, you can restrict access to data at row level or column level.

VPD doesn't replace DAC, but it is complimentary to DAC. VPD can further restrict access to users who have been given access to data by DAC.
Virtual Private Database

There are five types of policies based on how often a **policy function** is evaluated:

- DBMS_RLS.DYNAMIC
- DBMS_RLS.STATIC
- DBMS_RLS.SHARED_STATIC
- DBMS_RLS.CONTEXT_SENSITIVE
- DBMS_RLS.SHARED_CONTEXT_SENSITIVE

**DBMS_RLS.DYNAMIC** is default.

Although it is not necessary to use application contexts when implementing VPD policies, it is a common practice. Figure 1 shows usual steps that you will need to complete to implement the VPD policy on protected objects, such as table or view:

![Figure 1 – Steps to implement the VPD policy](image)

**A driving context** is an application context that has at least one attribute and its purpose is to determine which group of policies will be applied. The driving context is set by an application that is trying to access the data.
The default VPD behavior is that all policies defined on a table or a view are enforced for all SQL statements regardless of the application that executes them. If multiple applications share a table or a view, it is highly likely that you will either need to establish more complex logic to handle security requirements (to determine in which case, which predicate should be returned) or change the default VPD behavior by creating and using policy groups. If policies are already defined, you should identify which policies should be in effect when each application accesses the table or view. Each object has a predefined default policy group (SYS_DEFAULT), and the policies defined in this group are always applied for that particular object. A driving context determines which other policy group will also be applied at that time.

Suppose that there are two applications (A and B) that access data in table HR.EMP_VPD_TEST. Their specific policies are defined in two policy groups (HR_GRP_A and HR_GRP_B), and policies that should be enforced in any case are defined in the default group (SYS_DEFAULT). When application A accesses the data, policies that belong to HR_GRP_A and SYS_DEFAULT groups are applied, and when application B accesses the data, policies that belong to HR_GRP_B and SYS_DEFAULT groups are applied (it is assumed that the driving context is properly set).

Steps to implement policy groups are shown in Figure 2:

![Figure 2 – Steps to implement policy groups](image-url)
Creating different policy functions

The purpose of a policy function is to return a predicate that will be applied in \texttt{WHERE} clause of the statement (except for \texttt{INSERT} operation). In this recipe, you'll create several simple policy functions, based on different business and security requirements.

Getting ready

To complete this recipe, you'll need to create the table \texttt{hr.emp_vpd_test}, insert several values into that table, and create several users (in our case, \texttt{susan}, \texttt{joel}, \texttt{emma}, \texttt{maja}, and \texttt{zoran} already exist).

![SQL CREATE TABLE](image)

Table created.

Figure 3 – A test table
If you won't use the same data as shown in Figure 4, then keep in mind to accordingly make changes in the How to do it section and the rest of the recipes in this chapter.

![Oracle Database](image)

**Figure 4 – Test data in the table hr.emp_vpd_test**

**How to do it…**

1. Connect to the database as a user who has appropriate privileges (for example, user maja):

   ```
   $ sqlplus maja
   ```
2. Create a policy function that satisfies this condition: The user susan can't access data in a table (for example, hr.emp_vpd_test) and other users can access entire data in the table.

![Generate Policy Function](image)

Figure 5

3. Create an application context that has the emp_id attribute and the value is emp_id (from the hr.emp_vpd_test) of the connected user or if the connected user is not employee. See Chapter 12, Appendix – Application Contexts for detailed explanation (recipes: Creating an application context and Setting application context attributes) and make appropriate changes.

- Create an application context

```sql
SQL> connect maja
Enter password:
Connected.
SQL> create context hremp_ctx using hremp_ctx_pkg;
Context created.
```
• Create a PL/SQL package

```
CREATE OR REPLACE PACKAGE hremp_ctx_pkg
IS
  PROCEDURE set_emp_id;
END;
/
```

```
CREATE OR REPLACE PACKAGE BODY hremp_ctx_pkg
IS
  PROCEDURE set_emp_id
  IS
    v_emp_id NUMBER;
  BEGIN
    SELECT emp_id
    INTO v_emp_id
    FROM hr.emp_vpd_test
    WHERE UPPER(mail) = (SYS_CONTEXT('USERENV', 'SESSION_USER') || '@COMPANY.EXAMPLE.COM');
    WHEN no_data_found THEN
      DBMS_SESSION.SET_CONTEXT ('hremp_ctx','emp_id',v_emp_id);
  END;
END;
/
```

```
Package hremp_ctx_pkg compiled
```

```
Package body hremp_ctx_pkg compiled
```

• Create a logon trigger

```
SQL> CREATE OR REPLACE TRIGGER hremp_ctx_logon
2  AFTER LOGON ON DATABASE
3  BEGIN
4    hremp_ctx_pkg.set_emp_id();
5  END;
6 /
```

Trigger created.
4. Create a policy function (for example, `emp_access`) that satisfies this condition: a “regular” employee can access only his or her data in a table (for example, `hr.emp_vpd_test`) and manager users can access his or her data in the table and data for employees he or she directly manages.

![Figure 9 – The emp_access policy function](image)

5. Create a role (for example, `HREMP_TEST`).

```
SQL> create role HREMP_TEST;
Role created.
```
6. Create a policy function that satisfies this condition: Only users who have the HREMP_TEST role can view data in a table (for example, hr.emp_vpd_test).

![Function ROLE_ACCESS compiled](image)

**How it works…**

In step 4, you created a policy function that uses the application context you created, where as other policy functions you created use built-in application contexts.

A policy function can be part of a package or is standalone.
There's more…

To test whether the function defined in step 2 works properly, perform the following tasks:

1. Connect to the database as the user maja and execute the following statement:

   ```sql
   SQL> select no_access('a','b') from dual;
   NO_ACCESS('A','B')
   ------------------
   1=1
   ```

   Figure 12 – Maja can access data

2. Grant the user susan execute on no_access function and connect to the database as the user susan.

   ```sql
   SQL> grant execute on no_access to susan;
   Grant succeeded.
   SQL> connect susan
   Enter password: Connected.
   ```

   Figure 13 – Temporary grant the susan privilege

3. Execute the following statement:

   ```sql
   select maja.no_access('a','b') from dual;
   ```

   ```sql
   SQL> select maja.no_access('a','b') from dual;
   MAJA.NO_ACCESS('A','B')
   ------------------------
   1=2
   ```

   Figure 14 – Susan can't access data
4. Connect to the database as the user maja and revoke execute on no_access function from the user susan.

   SQL> connect maja
   Enter password:
   Connected.
   SQL> revoke execute on no_access from susan;
   Revoke succeeded.

Figure 15 – Clean up environment (revoke privilege)

See also

- All recipes of Chapter 12, Appendix – Application Contexts

Creating Oracle Virtual Private Database row-level policies

Oracle VPD row-level policies restrict users' access per row for a protected object. This means that two users who execute the same query against, for example, a table may, as a result, receive different number of rows.

Getting ready

See the Getting ready section of the recipe Creating different policy functions.
How to do it…

1. Connect to the database as a user who has appropriate privileges (for example, the user maja):

   ```
   $ sqlplus maja
   ```

2. Create a VPD policy (for example, test_pol1) that protects the hr.emp_vpd_test table in the following way: it restricts SELECT operation based on a policy function (for example, no_access).

   ```
   SQL> exec DBMS_RLS.ADD_POLICY('HR','EMP_VPD_TEST','TEST_POL1','MAJA','NO_ACCESS','SELECT')
   PL/SQL procedure successfully completed.
   ```

3. To test VPD policy created in the previous step, connect as the user susan to the database (keep in mind that she has the SELECT ANY TABLE privilege) and try to access data in the table hr.emp_vpd_test.

   ```
   SQL> connect susan
   Enter password:
   Connected.
   SQL> SELECT * FROM HR.EMP_VPD_TEST;
   no rows selected
   ```

   Figure 17 – Susan can’t access data
4. Connect to the database as a user who can create a VPD policy (for example, user maja). Create a VPD policy (for example, test_pol2) that additionally protects the hr.emp_vpd_test table in the following way: it restricts the SELECT and DELETE operations based on a policy function (for example, emp_access).

```sql
SQL> connect maja
```

```sql
SQL> exec DBMS_RLS.ADD_POLICY('HR','EMP_VPD_TEST','TEST_POL2','MAJA','EMP_ACCESS ','SELECT,DELETE')
PL/SQL procedure successfully completed.
```

Figure 18 – The VPD policy TEST_POL2

5. Connect to the database as the user joel and execute the following query:

```
SELECT * FROM HR.EMP_VPD_TEST;
```

The result will show 3 rows, because joel can view his data and data for his direct employees (policy function emp_access).

```sql
SQL> connect joel
Enter password:
Connected.
SQL> select * from hr.emp_vpd_test;
```

<table>
<thead>
<tr>
<th>EMAIL</th>
<th>SALARY</th>
<th>COMM_PCT</th>
<th>MGR_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:maja@company.example.com">maja@company.example.com</a></td>
<td>11000</td>
<td>.05</td>
<td>3</td>
</tr>
<tr>
<td><a href="mailto:joel@company.example.com">joel@company.example.com</a></td>
<td>15000</td>
<td>.04</td>
<td></td>
</tr>
<tr>
<td><a href="mailto:zoran@company.example.com">zoran@company.example.com</a></td>
<td>11500</td>
<td>.02</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 19 – Joel can view his data and data for his direct employees
6. Connect to the database as the user emma and execute the following query:

```
SELECT * FROM HR.EMP_VPD_TEST;
```

The result will show only 1 row, because emma is a “regular” employee, so she can view only her own data (policy function emp_access).

```
SQL> connect emma
Enter password:
Connected.
SQL> select * from hr.emp_vpd_test;
EMP_ID  FIRST_NAME  LAST_NAME
EMAIL           SALARY  COMM_PCT MGR_ID
----- -------- --------- ----
  4  Emma     Cole 8000   .1     5
emma@company.example.com
```

Figure 20 – Emma can only view her own data

There's more…

You defined two VPD policies on the same table, and they are both enabled. The first one only restricts the user susan from accessing the table, whereas the other one affects all users connected to the database (with some exceptions, see the recipe Exempting users from VPD policies). If susan connects to the database, both policies will determine whether she can access the data and if yes, which data. The way the policies are defined, she won’t be able to view data in the table.

See also

- The recipe Exempting users from VPD policies
Creating column-level policies

When you create a column-level VPD policy, you define sensitive columns, and if those columns are referenced in a query, statement will be rewritten. To create a column-level VPD policy, you also use the `DBMS_RLS.ADD_POLICY` procedure.

Getting ready

See the Getting ready section for the first recipe in this chapter. Results shown in this recipe assume that you completed previous recipes in this chapter.

How to do it…

1. Connect to the database as a user who has appropriate privileges (for example, the user maja):

   ```
   $ sqlplus maja
   ```

2. Create a VPD policy (for example, `test_col`) that protects the `hr.emp_vpd_test` table in the following way: it defines that `salary` and `comm_pct` are sensitive columns and a user can access them only if he or she has the `HREMP_TEST` role (the `role_access` policy function).
3. Grant the role HREMP_TEST to user zoran:

   SQL> grant HREMP_TEST to zoran;

   Grant succeeded.

   SQL> connect zoran
   Enter password:
   Connected.
   SQL> select * from hr.emp_vpd_test;

<table>
<thead>
<tr>
<th>EMP_ID</th>
<th>FIRST_NAME</th>
<th>LAST_NAME</th>
<th>EMAIL</th>
<th>SALARY</th>
<th>COMM_PCT</th>
<th>MGR_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Zoran</td>
<td>Pavlovic</td>
<td><a href="mailto:zoran@company.example.com">zoran@company.example.com</a></td>
<td>11500</td>
<td>.02</td>
<td>3</td>
</tr>
</tbody>
</table>

4. Connect to the database as the user zoran and view data in the table hr.emp_vpd_test.

5. Connect to the database as the user maja and disable the VPD policy TEST_POL2.

   SQL> connect maja
   Enter password:
   Connected.
   SQL> exec DBMS_RLS.ENABLE_POLICY('HR','EMP_VPD_TEST','TEST_POL2',FALSE);

   PL/SQL procedure successfully completed.

```
SQL> connect zoran
Enter password:
Connected.
SQL> select * from hr.emp_vpd_test;

<table>
<thead>
<tr>
<th>EMP_ID</th>
<th>FIRST_NAME</th>
<th>LAST_NAME</th>
<th>EMAIL</th>
<th>SALARY</th>
<th>COMM_PCT</th>
<th>MGR_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Maja</td>
<td>Veselica</td>
<td><a href="mailto:maja@company.example.com">maja@company.example.com</a></td>
<td>11000</td>
<td>.05</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Zoran</td>
<td>Pavlovic</td>
<td><a href="mailto:zoran@company.example.com">zoran@company.example.com</a></td>
<td>11500</td>
<td>.02</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Joel</td>
<td>Adams</td>
<td><a href="mailto:joel@company.example.com">joel@company.example.com</a></td>
<td>15000</td>
<td>.04</td>
<td></td>
</tr>
</tbody>
</table>

SQL>
```

7. Connect to the database as the user `joel` and execute the same statement as in the previous step.

```
SQL> connect joel
Enter password:
Connected.
SQL> select * from hr.emp_vpd_test;

<table>
<thead>
<tr>
<th>EMP_ID</th>
<th>FIRST_NAME</th>
<th>LAST_NAME</th>
<th>EMAIL</th>
<th>SALARY</th>
<th>COMM_PCT</th>
<th>MGR_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Maja</td>
<td>Veselica</td>
<td><a href="mailto:maja@company.example.com">maja@company.example.com</a></td>
<td>11000</td>
<td>.05</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Zoran</td>
<td>Pavlovic</td>
<td><a href="mailto:zoran@company.example.com">zoran@company.example.com</a></td>
<td>11500</td>
<td>.02</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Joel</td>
<td>Adams</td>
<td><a href="mailto:joel@company.example.com">joel@company.example.com</a></td>
<td>15000</td>
<td>.04</td>
<td></td>
</tr>
</tbody>
</table>

SQL>
```
How it works…

In step 2, the test_col VPD policy is created. In step 3, the user zoran is granted the role (HREMP_TEST) that will allow him to view entire data in step 6 (after test_pol2 is disabled). In step 4, displayed rows are restricted by TEST_POL2, so user zoran can view only his data. In step 5, you disabled the TEST_POL2 policy using the DBMS_RLS.ENABLE_POLICY procedure (to disable the policy, you set enable parameter to false). The syntax is:

```sql
DBMS_RLS.ENABLE_POLICY (
    object_schema IN VARCHAR2 NULL,
    object_name   IN VARCHAR2,
    policy_name   IN VARCHAR2,
    enable        IN BOOLEAN TRUE)
```

In step 7, the user joel can view all rows, but cannot view salary and comm_pct, because he doesn’t have the HREMP_TEST role.

Creating a driving context

In the previous recipe, you saw that having multiple VPD policies (most probably created because multiple application use that same table) is harder to manage, and it can lead to unexpected/unwanted results.

For example, you have two applications and want to create two policy groups. If the first application accesses the table, the test_pol1 and test_col policies should be enforced, and if second application accesses the table, the test_pol2 policies should be applied. There will be no default policies.

In this recipe, you’ll create an application context and set it.

Getting ready

To complete this recipe, you’ll need an existing user who can create an application context (for example, the user maja).
How to do it…

1. Connect to the database as a user who has appropriate privileges (for example, the user maja):

   $ sqlplus maja

2. Create a driving context (for example, driver_ctx):

   SQL> CREATE CONTEXT driver_ctx using driver_ctx_pkg;

3. Set the driving context:

   SQL> CREATE OR REPLACE PACKAGE driver_ctx_pkg IS
       PROCEDURE set_driver (p_group varchar2);
       END;
   /
   SQL> CREATE OR REPLACE PACKAGE BODY driver_ctx_pkg IS
       PROCEDURE set_driver (p_group varchar2)
       IS
       BEGIN
       DBMS_SESSION.SET_CONTEXT('driver_ctx','ACTIVE',p_group);
       END;
       END;
   /

Creating policy groups

In this recipe, you'll create two policy groups that will be applied to table hr.emp_vpd_test.

Getting ready

To complete this recipe, you'll need an existing user who has appropriate privileges (for example, the user maja).
How to do it…

1. Connect to the database as a user who has appropriate privileges (for example, the user maja):

   $ sqlplus maja

2. Create the first policy group (for example, pol_grp_A):

   SQL> BEGIN
      DBMS_RLS.CREATE_POLICY_GROUP(
         object_schema => 'HR',
         object_name => 'EMP_VPD_TEST',
         policy_group => 'pol grp_A');
   END;
   /

3. Create the second policy group (for example, pol_grp_B):

   SQL> BEGIN
      DBMS_RLS.CREATE_POLICY_GROUP(
         object_schema => 'HR',
         object_name => 'EMP_VPD_TEST',
         policy_group => 'pol grp_B');
   END;
   /

Setting context as a driving context

In this recipe, you'll make an existing application context a driving context (you'll associate it with the protected object).

Getting ready

To complete this recipe, you'll need an existing application context (for example, driver_ctx) and an existing user who has appropriate privileges (for example, maja).
How to do it…

1. Connect to the database as a user who has appropriate privileges (for example, the user maja):

   $ sqlplus maja

2. Make an existing application context a driving context.

   SQL> connect maja
   Enter password:
   Connected.
   SQL> BEGIN
   2   DBMS_RLS.ADD_POLICY_CONTEXT('HR', 'EMP_VPD_TEST', 'DRIVER_CTX', 'ACTIVE');
   3   END;
   4   /
   PL/SQL procedure successfully completed.

Adding policy to a group

In this recipe, create VPD policies as part of a policy group.

Getting ready

To complete this recipe, you’ll need an existing user who has appropriate privileges (for example, maja). If you completed previous recipes, drop all VPD policies using the DBMS_RLS.DROP_POLICY procedure.

   SQL> BEGIN
   2   DBMS_RLS.DROP_POLICY('HR', 'EMP_VPD_TEST', 'TEST_POL1');
   3   DBMS_RLS.DROP_POLICY('HR', 'EMP_VPD_TEST', 'TEST_POL2');
   4   DBMS_RLS.DROP_POLICY('HR', 'EMP_VPD_TEST', 'TEST_COL');
   5   END;
   6   /
   PL/SQL procedure successfully completed.

Figure 27 – Drop policies
How to do it...

1. Connect to the database as a user who has appropriate privileges (for example, the user maja):

   $ sql plus maja

2. Add TEST_POL1 to policy group pol_grp_A.
3. Add `TEST_COL` to policy group `pol_grp_A`.

4. Add `TEST_POL2` to policy group `pol_grp_B`.
5. Create a logon trigger.

```
CREATE OR REPLACE TRIGGER driver_ctx_logon
    AFTER LOGIN ON DATABASE
BEGIN
    IF SYS_CONTEXT('USERENV','CLIENT_PROGRAM_NAME') = 'SQL Developer'
    THEN DRIVER_CTX_PROG.SET_DRIVER('pol_grp_A');
    Else DRIVER_CTX_PROG.SET_DRIVER('pol_grp_B');
    END IF;
END;
/
```

6. Connect to the database as the user joel using SQL*Plus and execute the `SELECT` statement, as shown in Figure 32.

```
SQL> select sys_context('driver_ctx','ACTIVE') from dual;
SYS_CONTEXT('driver_ctx','ACTIVE')
--------------------------
pol_grp_B
```

Figure 32
7. View data in the table `hr.emp_vpd_test`.

```
SQL> connect joel
Enter password:
Connected.
SQL> select * from hr.emp_vpd_test;

EMP_ID FIRST_NAME LAST_NAME
EMAIL SALARY COMM_PCT MGR_ID
1 Maja Veselica 11800 .05 3
maja@company.example.com
2 Zoran Pavlovic 11500 .02 3
zoran@company.example.example.com
3 Joel Adams 15800 .04
joel@company.example.example.com
```

Figure 33

8. Connect to the database as the user `susan` using SQL*Plus and view data in the table `hr.emp_vpd_test`:

```
SQL> connect susan

SQL> select * from hr.emp_vpd_test;

EMP_ID FIRST_NAME LAST_NAME
EMAIL SALARY COMM_PCT MGR_ID
4 Emma Cole 8000 .1 5
emma@company.example.example.com
5 Susan Smith 16000 0
susan@company.example.example.com
```
9. Connect as the user emma using SQL Developer and view data in the table hr.emp_vpd_test.

![Query Result](query_result.png)

Exempting users from VPD policies

VPD policies are not enforced for users who connect as sysdba, during direct path export, and for users who have the EXEMPT ACCESS POLICY privilege.

Getting ready

To complete this recipe, you’ll connect to the database as SYS user and grant EXEMPT ACCESS POLICY to an existing user.
**Virtual Private Database**

## How to do it...

1. Connect to the database as **SYS** user:

   ```
   $ sqlplus / as sysdba
   ```

2. Grant the **EXEMPT ACCESS POLICY** privilege to an existing user (for example, **susan**):

   ```
   SQL> grant EXEMPT ACCESS POLICY to susan;
   ```

3. Connect to the database as the user **susan** and verify that now she can access data in the **hr.emp_vpd_test** table.

```sql
SQL> connect susan
Enter password:
Connected.
SQL> select * from hr.emp_vpd_test;

<table>
<thead>
<tr>
<th>EMP_ID</th>
<th>FIRST_NAME</th>
<th>LAST_NAME</th>
<th>EMAIL</th>
<th>SALARY</th>
<th>COMM_PCT</th>
<th>MGR_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Maja</td>
<td>Veselica</td>
<td><a href="mailto:maja@company.example.com">maja@company.example.com</a></td>
<td>11000</td>
<td>.05</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Zoran</td>
<td>Pavlovic</td>
<td><a href="mailto:zoran@company.example.com">zoran@company.example.com</a></td>
<td>11500</td>
<td>.02</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Joel</td>
<td>Adams</td>
<td><a href="mailto:joel@company.example.com">joel@company.example.com</a></td>
<td>15000</td>
<td>.04</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Emma</td>
<td>Cole</td>
<td><a href="mailto:emma@company.example.com">emma@company.example.com</a></td>
<td>8000</td>
<td>.1</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>Susan</td>
<td>Smith</td>
<td><a href="mailto:susan@company.example.com">susan@company.example.com</a></td>
<td>16000</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
```
5

Data Redaction

In this chapter, we will cover the following tasks:

- Creating a redaction policy when using full redaction
- Creating a redaction policy when using partial redaction
- Creating a redaction policy when using random redaction
- Creating a redaction policy when using regular expression redaction
- Using Oracle Enterprise Manager Cloud Control 12c to manage redaction policies
- Changing the function parameters for a specified column
- Adding a column to the redaction policy
- Enabling, disabling, and dropping a redaction policy
- Exempting users from data redaction policies

Introduction

Oracle Data Redaction is a new security feature, introduced in Oracle Database 12c. From a licensing viewpoint, it is part of the Advanced Security Option (only available as an option for Oracle Database Enterprise Edition). However, afterwards, Oracle decided to make it available in Oracle Database 11g as well (only in version 11.2.0.4). The main idea behind this feature is to mask (hide/redact) some (sensitive) data from end-users. Having this in mind, it is logical that you will primarily use this security solution in a production environment.
Oracle Data Redaction and Oracle Data Masking are both used to mask sensitive data, but these solutions are completely different—from the way they are designed (how they work) to their target implementation use cases. Oracle Data Masking enables organizations to use production data in development and test environments by changing production data with realistic data (transformation is done by using masking rules).

Oracle Data Redaction masks sensitive data just before the results of the SQL query are returned to the application that issued the query. Data stored in the database is NOT changed in any way.

When you implement Oracle Data Redaction, you have to decide the following:

- What data should be redacted
- Which redaction method is most suitable for the identified data
- In which situations the redaction should take place

You define all these decisions by creating a redaction policy (Figure 1), and they are enforced as long as the policy is enabled.
Different types of redaction are shown in Figure 2.

You can define only one redaction policy on a table (or view).

To view which data redaction policies are defined and whether they are enabled, you can query the `redaction_policies` view. Also, it is very useful to query the `redaction_columns` view, which shows which columns will be masked and what type of redaction will be used. Note that the names of those two views do not have any prefix (such as `DBA_`, `USER_`, or `ALL_`).

Although Oracle Data Redaction as a concept is fantastic, you should keep in mind that there are some implementation limitations (for example, unsupported data types) and unexpected behavior (most likely bugs) observed in Oracle Database 12.1.0.2.

For all recipes in this chapter, we assume that database is up and running, and each user has at least a `create session` privilege. Recipes are tested on Oracle Database 12.1.0.2.
Creating a redaction policy when using full redaction

In this recipe, you will create a redaction policy on the income_level column (on the income_level column on the CUSTOMERS table in the sample schema OE), find the default values (for full redaction) for different data types, and change the default value for the varchar2 data type.

You may consider data about customer address to be sensitive. Unfortunately, you can't create a redaction policy on the CUST_ADDRESS column (the table CUSTOMERS in the sample schema OE) because its data type is not supported (its data type is TYPE, which is not a literal value, so it can’t be redacted). If you try to create a redaction policy, you will receive the following error: ORA-28073. The column CUST_ADDRESS has an unsupported data type.

Getting ready

To complete this recipe, you’ll need the following:

- An existing user who can view data in OE.CUSTOMERS sample table but doesn’t have exempt redaction policy privilege (for example, oe)
- To connect as a SYS user to the database
- To restart the database (There’s more… section of the recipe)

How to do it…

1. Connect to the database as a user who has the SELECT privilege on the OE.CUSTOMERS table or the SELECT ANY TABLE privilege (for example, the oe user):

   ```
   $ sqlplus oe
   ```
2. Verify that the user (for example, the user oe) can view data by executing the following query:

```sql
select customer_id, cust_last_name, income_level from oe.customers order by customer_id fetch first 10 rows only;
```

![Figure 3 – Data in the clear text format (before redaction) in the OE.CUSTOMERS table](image)

3. Connect to the database as a user who can create the user secmgr (who will be responsible for managing redaction policies) and grant him appropriate privileges (for example, SYS):

```sql
SQL> create user secmgr identified by oracle;

SQL> grant create session to secmgr;

SQL> grant execute on dbms_redact to secmgr;
```
4. Connect to the database as the `secmgr` user:

   ```sql
   SQL> connect secmgr/oracle
   ```

5. Create the redaction policy `CUST_POL` in such a manner that data in the column `income_level` (the table `oe.customers`) is redacted using full redaction:

   ```sql
   SQL> begin
       2  dbms_redact.add_policy
       3  (object_schema => 'OE',
       4  object_name => 'CUSTOMERS',
       5  policy_name => 'CUST_POL',
       6  column_name => 'INCOME_LEVEL',
       7  function_type => DBMS_REDACT.FULL,
       8  expression => '1=1');
   9  end;
10  /
   ```

   **PL/SQL procedure successfully completed.**

6. Connect to the database as the same user as in step 1 (for example, `oe`) and execute the same query as in step 2.

![Figure 4 – After applying the redaction policy](image-url)
How it works…

In order to manage redaction policies, you need to connect to a database as a user who has an execute privilege on the dbms_redact package (in this recipe, that user is created in step 3 – the secmgr user). In step 5, you defined redaction policy CUST_POL. Let’s examine that step in more detail. Creating a new redaction policy is done by using the ADD_POLICY procedure in the DBMS_REDACT package (line 2). A policy consists of several distinct sections (see Figure 1). In step 5, lines 3-6 are part of the WHAT section in Figure 1. Lines 3, 4, and 6 define on which column redaction policy (whose name is defined on line 5) should be applied. Line 7 is part of the HOW section in Figure 1. It defines the redaction type (in this case, full redaction). Line 8 is a part of the WHEN section in Figure 1. It defines the conditions when protected data will be masked (in situations when the expression is evaluated to TRUE, the data is masked). In this case, the expression is always TRUE (1=1), meaning that data in column INCOME_LEVEL will always* be masked. Defining different, more complex expressions (using application contexts, and roles) will be done in the next few recipes in this chapter.

* Assume that the redaction policy CUST_POL is enabled and user doesn’t have strong privileges. For more information about for which users/operations the data redaction policy doesn’t have any effect, see the recipe Exempting users from data redaction policies.

In Figure 4, the result of applying the data redaction policy to the column INCOME_LEVEL (whose data type is varchar2) is shown, and as you can see, data in the column is masked. Every row is masked with the same value, in this case, whitespace (the default value when masking column whose data type is varchar2).

Keep in mind that whitespace is exactly one blank space and that it is different from NULL.
Figure 5 shows the example of creating a redaction policy (SAL_POLICY) using a full redaction method on a column (SALARY in table EMPLOYEES in schema GLDB) whose data type is a number. As you can see, the masked value is .

To find out default values (for full redaction) for other data types, query REDACTION_VALUES_FOR_TYPE_FULL. Finding the default value for DATE data type is depicted in Figure 6.
At the time when you define data redaction policy, you can specify redaction for *only one column*. If you want to redact more than one column in a table, you can later modify the policy. For more information, see the recipe *Changing redaction policy*.

**There's more...**

In the following section, you will learn to change default value for full redaction type. Step-by-step instructions are given on how to change default value from whitespace to T for *varchar2* data type.

It is important to remember that changing default value for full redaction will affect *ALL* the defined redaction policies in a database (that use full redaction type).
How to change the default value

You will use the `update_full_redaction_values` procedure to change the default value from whitespace to `T` for the `varchar2` data type. Note that you can create a redaction policy (for example, as you have already done in step 5 in the beginning of the recipe), but you can't change the default value (Figure 7) even though the user `secmgr` has been granted EXECUTE on the `DBMS_REDACT` package.

![Figure 7 – An unsuccessful change of default value](image)

Connect to the database as the `SYS` user and change the default value.

```
SQL> exec dbms_redact.update_full_redaction_values (varchar_val => 'T')
BEGIN dbms_redact.update_full_redaction_values (varchar_val => 'T'); END;

ERROR at line 1:
ORA-00942: table or view does not exist
ORA-06512: at "SYS.DBMS_REDACT", line 363
ORA-06512: at line 1
```

![Figure 8 – A successful change of the default value](image)
Optionally, verify that the default value is changed and that there is no effect before you restart the database. (Figure 9).

```
SQL> select varchar_value from redaction_values_for_type_full;

+------------------+
<table>
<thead>
<tr>
<th>VARCHAR_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
+------------------+

SQL> connect oe
Enter password:
Connected.
SQL> select customer_id, cust_last_name, income_level from oe.customers
  2  order by customer_id
  3  fetch first 10 rows only;

<table>
<thead>
<tr>
<th>CUSTOMER_ID</th>
<th>CUST_LAST_NAME</th>
<th>INCOME_LEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Welles</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>Pacino</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>Taylor</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>Sutherland</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>MacGraw</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>Hannah</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>Cruise</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>Mason</td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>Cage</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Sutherland</td>
<td></td>
</tr>
</tbody>
</table>

10 rows selected.
SQL> 
```

Figure 9 – The changed default value still has no effect on the displayed value in the column income_level
Restart the database and verify that the modified default value is displayed (Figure 10).

Figure 10 – A new default value is displayed

See also

- Changing redaction policy
- Exempting users from data redaction policies
Creating a redaction policy when using partial redaction

In this recipe, you will implement partial redaction on columns of two different types: Number and Varchar2. Partial redaction means that only part (hence the name partial) of the data in a specified column will be masked (redacted), whereas the other part of the data will be visible to the user – for instance, the first 12 digits of credit card number will be redacted, whereas other 4 digits will be visible.

How to do it...

1. Log in to database as a user who has a DBA role (for instance, zoran):
   $ sqlplus zoran/oracle

2. Create a test table and insert some data in it:
   SQL> create table tbl (a number);
   SQL> insert into tbl values (123456);
   SQL> insert into tbl values (234567);
   SQL> insert into tbl values (345678);
   SQL> commit;

3. Create role (that is going to be used in redaction policy) and user usr1 as the first test user:
   SQL> create role myrole;
   SQL> create user usr1 identified by oracle1;
   SQL> grant create session to usr1;

4. Grant the select privilege and role to usr1:
   SQL> grant select on zoran.tbl to usr1;
   SQL> grant myrole to usr1;
5. Create the second test user and grant him create session and select privilege, but don’t grant him the role myrole:

```
SQL> create user usr2 identified by oracle2;
SQL> grant create session to usr2;
SQL> grant select on zoran.tbl to usr2;
```

6. Create redaction policy to redact column a of the type Number using partial redaction (first four digits will be redacted and won’t be seen at all). This redaction policy will be applied only to users that don’t have role myrole and don’t have the EXEMPT REDACTION POLICY privilege:

```
SQL> BEGIN
  2  DBMS_REDACT.ADD_POLICY(
  3    object_schema => 'zoran',
  4    object_name => 'tbl',
  5    column_name => 'a',
  6    column_description => 'Sensitive column A',
  7    policy_name => 'a_tbl_partial',
  8    policy_description => 'Redact column A of tbl',
  9    function_type => DBMS_REDACT.PARTIAL,
 10    function_parameters => '0, 1, 4',
 11    expression => '
    SYS_CONTEXT(
      'SYS_SESSION_ROLES',
      'MYROLE') = 'FALSE');
 12  END;
 13  /
```

7. Connect to database as the user usr1 and select from the table tbl in the schema zoran:

```
SQL> connect usr1/oracle1
SQL> select a from zoran.tbl;
```

```
A
---
123456
234567
345678
```
8. Now, connect to database as the user **usr2** and again select from the table **tbl** in the schema **zoran**:

    SQL> connect usr2/oracle2
    usr2@ORA12CR1> select a from zoran.tbl;
    A
    --------
    56
    67
    78

9. Log in to database as a user who has a DBA role (for instance, **zoran**):

    $ sqlplus zoran/oracle

10. Create the test table to store credit cards data and insert some data in it:

    SQL> create table customers (name varchar2(20 CHAR),
        credit_card varchar2(20 CHAR));

    SQL> insert into customers values ('tom',
        '3455647456589132');

    SQL> insert into customers values ('mike',
        '3734982321225691');

    SQL> insert into customers values ('john',
        '3472586894975806');

    SQL> commit;

11. Grant select privilege on table customers in the schema **zoran** to **usr1**:

    SQL> grant select on zoran.customers to usr1;
12. Create a redaction policy to redact column `credit_card` of type `VARCHAR2` using partial redaction (first 12 values will be redacted with `#` sign). This redaction policy will be applied to all users, except those who have the `EXEMPT REDACTION POLICY` privilege (see the `Exempting users from data redaction policies` recipe):

```
SQL> BEGIN
  2  DBMS_REDACT.ADD_POLICY(
  3    object_schema          => 'zoran',
  4    object_name            => 'customers',
  5    column_name            => 'credit_card',
  6    column_description     => 'Credit Card numbers',
  7    policy_name            => 'CCN_POLICY',
  8    policy_description     => 'Redact column credit_card of table customers',
  9    function_type          => DBMS_REDACT.PARTIAL,
 10    function_parameters    => 'VVVVVVVVVVVVVVVV, VVVVVVVVVVVVVVV, #, 1, 12',
 11    expression             => '1=1' );
 12  END;
 13  /
```

13. Connect to database as the user `usr1` and select from the table `customers` in the schema `zoran`:

```
SQL> connect usr1/oracle1
SQL> select * from zoran.customers;
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>CREDIT_CARD</th>
</tr>
</thead>
<tbody>
<tr>
<td>tom</td>
<td>############9132</td>
</tr>
<tr>
<td>mike</td>
<td>############5691</td>
</tr>
<tr>
<td>john</td>
<td>############5806</td>
</tr>
</tbody>
</table>

**How it works…**

In order to manage redaction policies and also to create some test tables, you can connect to a database as a user who has a dba role (for example, `zoran`). If you just need to manage redaction policies, you can connect with user who has the execute privilege on the `dbms_redact` package.
The previous section is divided into two parts. The first part shows the creation of redaction policy for number type column, in such a way that redaction should only be applied to users that don't have a particular role. The second part shows the creation of a redaction policy for `Varchar2` type column.

In step 6, you created a redaction policy named `a_tbl_partial`. Creating a new redaction policy is done by using the `ADD_POLICY` procedure in the `DBMS_REDACT` package (line 2). A policy consists of several distinct sections (see Figure 1). Lines 3, 4, and 5 define on which column our redaction policy should be applied. Line 9 defines redaction type (in this case, partial redaction). Line 10 is used for function parameters (in our case, it is defined that first four digits will be redacted to 0). In line 11, you defined condition when protected data will be masked (in our case, it is when user doesn't have role `myrole`), and it is evaluated using the following expression: `SYS_CONTEXT('SYS_SESSION_ROLES','MYROLE') = 'FALSE'`. In this case, expression is true only if user doesn't have role `myrole`, and in this case, data in column a will be redacted (which is a case with the user `usr2`, whereas the user `usr1`, who has the role `myrole`, can see the unmasked data).

Step 12 shows the creation of redaction policy for the `Varchar2` column type. The difference is on line 10- function parameters (in our case, the first 12 values will be redacted or changed with symbol `#`, so only last four digits will be visible) and on line 11-condition is always `TRUE`.

![Figure 11 – Partial redaction](image-url)
There's more…

Even though users can't see unmasked data, they can use redacted columns in `where` clause:

```
SQL> select * from zoran.customers;
NAME                 CREDIT_CARD
-------------------- ------------------------------
tom                  ############9132
mike                 ############5691
john                 ############5806
```

```
SQL> select * from zoran.customers where credit_card like '34%';
NAME                 CREDIT_CARD
-------------------- ------------------------------
tom                  ############9132
john                 ############5806
```

Creating a redaction policy when using random redaction

Random redaction type is usually used for the number and date-time data types because for these data types, it is hard to make a distinction between the redacted (random) and real data. In this recipe, you will create redaction policy `EMP_POL` using random redaction type on `hr.employees` table, column `salary`, by using SQL*Plus. In the *Changing redaction policy* recipe, you will modify the `EMP_POL` redaction policy.

Getting ready

To complete this recipe, you'll need:

- An existing user who can view data in the `HR.EMPLOYEES` sample table but doesn’t have an exempt redaction policy privilege (for example, `hr`)
- The `secmgr` user created in the *Creating a redaction policy using full redaction* recipe or another user who can create redaction policies (has execute on the `dbms_redact` package)
How to do it...

1. Connect to the database as a user who has the `SELECT` privilege on the `HR.EMPLOYEES` table or the `SELECT ANY TABLE` privilege (for example, hr user):

   ```bash
   $ sqlplus hr
   ```

2. Verify that the user (for example, hr user) can view data by executing the following query:

   ```sql
   select employee_id, salary, commission_pct from hr.employees where commission_pct IS NOT NULL order by employee_id fetch first 10 rows only;
   ```

![Figure 12 – Data in the clear text format in the HR.EMPLOYEES table](image-url)
3. Connect to the database as the secmgr user:

   SQL> connect secmgr/oracle

4. Create the redaction policy EMP_POL in such a way that data in column salary (the table hr.employees) is redacted using random redaction only when user in step 1 (for example, hr) tries to view it. If you don't use the hr user, modify line 8 to reflect that change:

   SQL> begin
   2   dbms_redact.add_policy
   3   (object_schema => 'HR',
   4   object_name => 'EMPLOYEES',
   5   policy_name => 'EMP_POL',
   6   column_name => 'SALARY',
   7   function_type => DBMS_REDACT.RANDOM,
   8   expression => 'SYS_CONTEXT(''USERENV'', 'SESSION_USER'') = ''HR''');
   9   end;
10  /

   PL/SQL procedure successfully completed.

5. Connect to the database as the same user as in step 1 (for example, hr) and execute the same query, as in step 2, twice.
How it works…

In step 4, you created the redaction policy EMP_POL by using the procedure ADD_POLICY in the DBMS_REDACT package. Line 7 defines that random redaction type will be used to redact data. Line 8 (policy expression) in this case specifies the blacklist (which contains only user HR). This means only the hr user is not allowed to view data in the column salary. To define a whitelist (for example, list only users who are allowed to view data, range of only those IP addresses from which access is allowed, and so on) change operator = to operator <> and define left and right operand according to your needs.
When defining security policies, it is a good practice to create whitelists.

When the number data type is redacted using random redaction type, the redacted value will belong to the interval $[0, |n|]$, where $|n|$ is the absolute value of the original data. According to the official Oracle documentation (*Database Advanced Security Guide*, Chapter 9), the only exception to this rule is when original data is an integer between $-1$ and 9, and in that case, the redacted value will belong to the interval $[0, 9]$.

### Creating a redaction policy when using regular expression redaction

A regular expression redaction type enables you to create and implement flexible redaction rules. You define patterns that will be used in order to match and replace data, as well as some other parameters of the search. In this recipe, you will create the redaction policy `SHORT_POL`, which will be used to mask customers’ phone numbers.

### Getting ready

To complete this recipe, you’ll need:

- An existing user who can view data in the `SH.CUSTOMERS` sample table but doesn’t have an exempt redaction policy privilege (for example, `sh`)
- The `secmgr` user you created in the *Creating redaction policy using full redaction* recipe or another user who can create redaction policies (has execute on `dbms_redact` package)

### How to do it…

1. Connect to the database as a user who has the `SELECT` privilege on the `SH.CUSTOMERS` table or the `SELECT ANY TABLE` privilege (for example, the `sh` user):

   ```bash
   $ sqlplus sh
   ```
2. Verify that the user (for example, the user sh) can view data by executing the following query:

```sql
SELECT cust_id, cust_main_phone_number FROM sh.customers
ORDER BY cust_id FETCH FIRST 10 ROWS ONLY;
```

![Query execution result](image)

Figure 14 – Data in the clear text format (before redaction) in the SH.CUSTOMERS table

3. Connect to the database as the secmgr user:

```sql
SQL> connect secmgr/oracle
```

4. Create the redaction policy `SHORT_POL` in such a manner that data in the column `cust_main_phone_number` (the table `sh.customers`) is redacted using regular expression redaction:

```sql
SQL> begin
2  dbms_redact.add_policy
3  (object_schema => 'SH',
4  object_name => 'CUSTOMERS',
5  policy_name => 'SHORT_POL',
6  column_name => 'CUST_MAIN_PHONE_NUMBER',
7  function_type => DBMS_REDACT.REGEXP,
8  expression => '1=1',
9  regexp_pattern => DBMS_REDACT.RE_PATTERN_US_PHONE,
10 regexp_replace_string => DBMS_REDACT.RE_REDACT_US_PHONE_L7,
11 regexp_position => DBMS_REDACT.RE_BEGINNING,
12 regexp_occurrence => DBMS_REDACT.RE_FIRST);
13 end;
14 /
PL/SQL procedure successfully completed.
```

5. Connect to the database as the same user as in step 1 (for example, `sh`) and execute the same query as in step 2.
How it works…

When creating redaction policies that use regular expression redaction type, you can choose between redaction shortcuts (they exist for commonly redacted data, such as e-mail address, social security number, and postal code) and the creation of custom regular expressions. In this recipe, in step 4, lines 9-12, you used redaction shortcuts.

In Figure 15, the value T is displayed in the places where regular expression didn’t find a match and because of that full redaction was applied to them. With this kind of implementation, Oracle prevented accidental exposure of sensitive data.

Using Oracle Enterprise Manager Cloud Control 12c to manage redaction policies

In this recipe, you will perform several tasks with Data Redaction policies using Oracle Enterprise Manager Cloud Control 12c, including creation, modification, and deletion. Many tasks from other recipes, described in this chapter, can be done very easily using Enterprise Manager.

Getting ready

To complete this recipe, you need Enterprise Manager Cloud Control 12c and HR sample schema in the database.

How to do it…

2. Go to a Database home page (if it is a container database, you should go to a home page of PDB that contains sample schemas).
3. On menu, select **Security | Data Redaction** (see Figure 16).
4. On the **Data Redaction** screen, select **Create** (Figure 17).

![Image of Data Redaction screen]

Figure 17 – Creating a redaction policy
5. Set **Schema** as `HR` and the table as `EMPLOYEES`. Enter `SAL_POLICY` as a policy name. Click on the **Add** button, to add column that is going to be redacted. (Figure 18).

![Create Data Reduction Policy: SAI_POLICY](image)

*Figure 18 – The addition of a column*
6. Select the **SALARY** column and specify **RANDOM** as a **Redaction Function**. Click on **OK**. (Figure 19). On the next screen, click on **OK** at the top-right corner.

![Figure 19 – Choose random redaction type](image)

7. To edit **SAL_POLICY**, select it and click on **Edit** (you can search for policies by specifying schema, table, or policy name) (Figure 20).

![Figure 20 – Alter policy](image)
8. Select the **SALARY** column and click on **Modify** (Figure 21).

![Figure 21 – Modifying a column](image)

9. Change **Redaction Function** from **RANDOM** to **FULL**. Click on **OK** (Figure 22).

![Figure 22 – Changing redaction type for salary column](image)
10. Click on **Add** in order to add one more column to the redaction policy. (Figure 23).

![Figure 23 – Adding a column to the redaction policy](image-url)
11. Select the EMAIL column, and as **Redaction Template**, select **Email Address**. You can see that this pattern uses **Regular expression type** of **Data Redaction**. You can also change any of the parameters. Click on **OK**. (Figure 24).

![Defining redaction type for email column](image)

*Figure 24 – Defining redaction type for email column*
12. On next page you can change **Policy Expression**. Click **OK** on the top right corner.

![Figure 25 – You can change the policy expression](image)

13. To disable **SAL_POLICY**, select it and click on **Disable**. (Figure 26).

![Figure 26 – Disabling the sal_policy redaction policy](image)
14. To enable SAL_POLICY, select it and click on **Enable**. (Figure 27).

![Figure 27 – Enabling the sal_policy redaction policy](image)

15. To delete SAL_POLICY, select it and click on **Delete**. (Figure 28).

![Figure 28 – Deleting the sal_policy redaction policy](image)
16. You should see the **Confirmation** message (Figure 29).

![Oracle Enterprise Manager Cloud Control 12c](image)

**Figure 29 – The SAL_POLICY policy has been successfully deleted**

### Changing the function parameters for a specified column

There are several ways in which you can change an existing redaction policy. In this recipe and the next one, you will:

- Change the function parameters for a specified column (the `a_tbl_partial` policy, which you created in the recipe *Creating a redaction policy when using partial redaction*)
- Add a column (`commission_pct` in the `hr.employees` table) to the redaction policy `EMP_POL` (you defined it in the *Creating a redaction policy when using random redaction* recipe)
Also, it is possible to remove column from the redaction policy, alter the policy expression, and modify the type of redaction for a specified column.

You concluded that the `a_tbl_partial` redaction policy doesn’t satisfy the requirements for your application anymore because it redacts first four digits with 0 and leading zeros are not displayed in the application. You decide to alter the `a_tbl_partial` policy. You want all digits to be displayed and to have them redacted with some value (for example, 9).

**Getting ready**

Before doing this recipe, you should have completed the *Creating a redaction policy when using partial redaction* recipe. You will use the `secmgr` user you created in the *Creating a redaction policy when using full redaction* recipe.

**How to do it…**

1. Connect to the database as the `secmgr` user and alter the policy `EMP_POL`:

```sql
$ sqlplus secmgr
SQL> BEGIN
  2  DBMS_REDACT.ALTER_POLICY(
  3      object_schema   => 'zoran',
  4      object_name     => 'tbl',
  5      policy_name     => 'a_tbl_partial',
  6      action          => DBMS_REDACT.MODIFY_COLUMN,
  7      column_name     => 'a',
  8      function_type   => DBMS_REDACT.PARTIAL,
  9      function_parameters    => '9,1,4');
 10  END;
 11  /
```
2. Connect as the user `usr2` to the database and view data in column `A` in the `zoran.tbl` table:

```
SQL> connect usr2/oracle2
Connected.

SQL> select a from zoran.tbl;
```

```
A
---------
999956
999967
999978
```

**Add a column to the redaction policy**

You have to modify the existing redaction policy in order to redact more than one column in the table. In the table `HR.EMPLOYEES`, besides the column `SALARY`, you want to redact the column `COMMISSION_PCT`. You will modify the redaction policy `EMP_POL`. You decide that you want to use full redaction type for the column `COMMISSION_PCT`.

Note that in the same redaction policy (in this case, `EMP_POL`) the different “protected” columns can use different redaction types (in this case, random and full redaction).

**Getting ready**

Before doing this recipe, you should have completed the *Creating redaction policy when using random redaction* recipe. You will use the `secmgr` user you created in the *Creating redaction policy when using full redaction* recipe.
How to do it...

1. Connect to the database as the secmgr user and alter the EMP_POL policy:

   $ sqlplus secmgr
   SQL> BEGIN
   2  DBMS_REDACT.ALTER_POLICY(
   3     object_schema   => 'HR',
   4     object_name     => 'EMPLOYEES',
   5     policy_name     => 'EMP_POL',
   6     action          => DBMS_REDACT.ADD_COLUMN,
   7     column_name     => 'COMMISSION_PCT',
   8     function_type   => DBMS_REDACT.FULL);
   9  END;
 10  /
   PL/SQL procedure successfully completed.

2. Connect the user hr to the database and execute the following query:

   select employee_id, salary, commission_pct from hr.employees
   where commission_pct IS NOT NULL order by employee_id fetch first 10 rows only;

   Figure 30 – Two columns are redacted
How it works…

You used the procedure ALTER_POLICY in the PL/SQL package DMBS_REDACT to change redaction policies. On line 6 (in both examples), you specified value for the ACTION parameter, which defines what kind of change will happen.

See also

- Using Oracle Enterprise Manager Cloud Control 12c to manage redaction policies

Enabling, disabling, and dropping redaction policy

In this recipe, you will perform the three basic tasks: enabling, disabling, and dropping the same redaction policy (CUST_POL), which you defined in the Creating a redaction policy when using full redaction recipe using SQL*Plus. Also, you will check which redaction policies exist in the database and whether they are enforced (enabled).

To minimize dependence on the previous recipes in this chapter, a result shown after querying data dictionary view is equivalent to the one you would get if you completed only the Creating a redaction policy when using full redaction recipe before starting to do this recipe. The only difference you may see in the result is the number of existing redaction policies in the database.

Getting ready

Before doing this recipe, you should have completed the Creating a redaction policy when using full redaction recipe.
How to do it...

To complete the tasks, you will use procedures in the dbms_redact package (disable_policy, enable_policy, and drop_policy).

1. Connect to the database as a user who has an execute privilege on dbms_redact package and select_catalog_role role (for example, secmgr user):

   $ sqlplus secmgr

2. Find out which redaction policies exist in the database by querying the redaction_policies view:

   ```sql
   SQL> col policy_name format A20
   SQL> select policy_name, enable from redaction_policies;
   ```

   ![Figure 31 – Finding defined redaction policies](image)

3. Connect to the database as the oe user and grant the SELECT privilege on OE.CUSTOMERS to the secmgr user. Connect to the database as the secmgr user. Verify that the secmgr user can't see original data in the column INCOME_LEVEL:

   ```sql
   SQL> connect oe
   SQL> grant select on oe.customers to secmgr;
   SQL> connect secmgr
   SQL> select customer_id, cust_last_name, income_level
       from oe.customers
       2  order by customer_id
       3  fetch first 10 rows only;
   ```
Figure 32 – Redacted data is displayed even to the user who created the policy

4. Disable the redaction policy **CUST_POL** (as the **secmgr** user):

```sql
SQL> begin
  2  dbms_redact.disable_policy
  3  (object_schema => 'OE',
  4  object_name  => 'CUSTOMERS',
  5  policy_name => 'CUST_POL');
  6  end;
  7 /

PL/SQL procedure successfully completed.
```
5. Verify that now the secmgr user can view original data in the column INCOME_LEVEL and query the redaction_policies view by executing the following statements:

- \texttt{select customer_id, cust_last_name, income_level from oe.customers order by customer_id fetch first 10 rows only;}

- \texttt{col policy_name format A20}

- \texttt{select policy_name, enable from redaction_policies;}

![Image](image-url)

Figure 33 – secmgr can view unmasked data in the column income_level, because the cust_pol policy is disabled
• Enable the redaction policy `CUST_POL`:

```
SQL> begin
2  dbms_redact.enable_policy
3  (object_schema => 'OE',
4  object_name => 'CUSTOMERS',
5  policy_name => 'CUST_POL');
6  end;
7 /
```

PL/SQL procedure successfully completed.

6. Verify that redaction is working properly by executing the following statements:

```
• select customer_id, cust_last_name, income_level from oe.customers order by customer_id fetch first 10 rows only;

• col policy_name format A20

• select policy_name, enable from redaction_policies;
```
• Drop the redaction policy CUST_POL:

```sql
SQL> begin
    2    dbms_redact.drop_policy
    3      (object_schema => 'OE',
    4      object_name => 'CUSTOMERS',
    5      policy_name => 'CUST_POL');
    6  end;
    7 /

PL/SQL procedure successfully completed.
```
7. Verify that the redaction policy CUST_POL doesn't exist in the database by executing the following statements:

- `select customer_id, cust_last_name, income_level from oe.customers order by customer_id fetch first 10 rows only;`

- `col policy_name format A20`

- `select policy_name, enable from redaction_policies;`

![Database Query Results](image)

Figure 35 – The redaction policy cust_pol doesn't exist anymore

See also

- Creating a redaction policy when using full redaction
- Using Oracle Enterprise Manager Cloud Control 12c to manage redaction policies
Exempting users from data redaction policies

In this recipe, you will create a user and then exempt that user from Data Redaction. This user will be exempted from all redaction policies in the database.

Getting ready

Before doing this recipe, you should have completed the Creating a redaction policy when using the partial redaction recipe.

How to do it…

1. Connect to the database as a user who has a DBA role (for example, user zoran):

   $ sqlplus zoran/oracle

2. Create a new user (for example, vipuser) and grant him the create session privilege and select privilege on table customers in schema zoran:

   SQL> create user vipuser identified by oracle;
   SQL> grant create session to vipuser;
   SQL> grant select on zoran.customers to vipuser;

3. Connect as a newly created user and try to select from the zoran.customers table:

   SQL> connect vipuser/oracle
   SQL> select * from zoran.customers;

<table>
<thead>
<tr>
<th>NAME</th>
<th>CREDIT_CARD</th>
</tr>
</thead>
<tbody>
<tr>
<td>tom</td>
<td>#9132</td>
</tr>
<tr>
<td>mike</td>
<td>#5691</td>
</tr>
<tr>
<td>john</td>
<td>#5806</td>
</tr>
</tbody>
</table>
4. Connect again as the user zoran, and grant the EXEMPT REDACTION POLICY privilege to the vipuser user:

   SQL> connect zoran/oracle
   SQL> grant exempt redaction policy to vipuser;

5. As the user vipuser, now try to select from the table zoran.customers:

   SQL> connect vipuser/oracle
   SQL> select * from zoran.customers;

   NAME     CREDIT_CARD
   -------- ------------------------------
   tom      345647456589132
   mike     3734982321225691
   john     3472586894975806

How it works…

There is a new system privilege that is used to control which users will be exempted from data redaction in Oracle Database. This privilege is EXEMPT REDACTION POLICY. Users who are granted this privilege will be able to see clear (unmasked) data in the whole database if they have (select) privilege to access that data. This means that all redaction policies in the database will not be applied to these users. The DBA and EXP_FULL_DATABASE roles both contain this privilege, so any user that has either of these roles is exempt from data redaction.

Backup/restore as well as import and export operations are not subject to data redaction. However, data redaction policies are included in export and import operations.
In this chapter, we will cover the following tasks:

- Creating a sensitive type
- Determining sensitive columns
- Creating transparent sensitive data protection policy
- Associating transparent sensitive data protection policy with sensitive type
- Enabling, disabling, and dropping policy
- Altering transparent sensitive data protection policy

**Introduction**

Oracle **Transparent Sensitive Data Protection (TSDP)** is a new security feature, introduced in Oracle Database 12c (available only in Enterprise Edition). TSDP provides a way to create classes of sensitive data and enables more centralized control of how sensitive data is protected. In database versions 12.1.0.1 and 12.1.0.2, it leverages two Oracle security mechanisms:

- Oracle **Virtual Private Database (VPD)**, described in Chapter 4, *Virtual Private Database*
- Oracle Data Redaction, explained in Chapter 5, *Data Redaction*
To implement TSDP, you should complete steps shown in Figure 1:

![Figure 1 – Steps to implement TSDP]

For all recipes in this chapter, we assume that the database is up and running and each user has at least a create session privilege. In this chapter, it is assumed that user `c##zoran` has a DBA role and it executes privileges on the following packages:

- `DBMS_TSDP_MANAGE`
- `DBMS_TSDP_PROTECT`
- `DBMS_RLS`
- `DBMS_REDACT`

Recipes are tested on Oracle Database 12.1.0.2 in multitenant and non-CDB environment. If you use non-CDB, connect to that database instead of `pdb1` (as is done in recipes).

## Creating a sensitive type

To create a sensitive type, you can use Oracle Enterprise Manager or a command-line interface. In this recipe, you'll use the command-line interface to execute a PL/SQL procedure. You decided that you want to protect e-mail addresses stored in your database, so first you are going to create sensitive type `email_type`. 
Getting ready

To complete this recipe, you’ll need an existing user who can create a sensitive type (for example, c##zoran).

How to do it…

1. Connect to the database (for example, pdb1) as a user who has appropriate privileges (for example, c##zoran):

   $ sqlplus c##zoran@pdb1

2. Create a sensitive type (for example, email_type):

   ```sql
   SQL> BEGIN
   2   DBMS_TSDP_MANAGE.ADD_SENSITIVE_TYPE (sensitive_type => '<your_type>',
   3       user_comment=> '<description>');
   4   END;
   5   /
   6
   PL/SQL procedure successfully completed.
   ```

   Figure 2 – Creating a sensitive type

How it works…

In step 2, you created a sensitive type (for example, email_type), which you can use to consistently mask (protect), in our case, e-mail information throughout the database. By creating a sensitive type, you only define that in the database, there exists a class of sensitive data and you name it. In later recipes in this chapter, you’ll define where that sensitive data resides (in which columns) and the way that data will be protected.
The name of a sensitive type (for example, `email_type`) is case-sensitive.

There's more…
To view existing sensitive types, execute the following query:

```sql
select name from DBA_SENSITIVE_COLUMN_TYPES;
```

![SQL query result](image)

Figure 3 – Finding information about defined sensitive types

Determining sensitive columns
After you decide which data is sensitive, you'll need to find all places where that data is stored. Once you do that, you'll classify the data (associate sensitive columns with sensitive types). In this recipe, you'll associate two sensitive columns (from two tables) with sensitive type you created in the previous recipe.

Getting ready
To complete this recipe, create a user `challengezoran`, create table `T1`, and insert several values into the table (see Figure 1) or use your own table. Also, you'll need an existing user who has an execute privilege on `dbms_tsdp_manage` package (for example, `c##zoran`).
How to do it...

1. Connect to the database (for example, pdb1) as a user who has appropriate privileges (for example, c##zoran user):

   $ sqlplus c##zoran@pdb1

2. Associate a sensitive column (for example, schema CHALLENGEZORAN, table T1, column EMAIL_ADDRESS) with sensitive type you created in the previous recipe (for example, email_type)
3. Associate another sensitive column (for example, schema HR, table EMPLOYEES, column EMAIL) with the same sensitive data type (for example, email_type).

![SQL code]

Figure 6 – Adding sensitive column email to sensitive type email_type

**How it works…**

In step 2 and 3, you defined where sensitive data resides and associated it with previously created sensitive data type.

You can associate a column with only one sensitive type. If you try to associate it with another type, you’ll receive ORA-45607.

**Creating transparent sensitive data protection policy**

This step defines the way you want to protect sensitive data. You can use Data Redaction or VPD settings for your TSDP policy. In this recipe, you’ll use regular expression redaction to protect previously defined sensitive data.
Getting ready

To complete this recipe, you’ll need an existing user who has the execute privilege on the dbms_tsdp_protect package (for example, c##zoran).

How to do it…

1. Connect to the database (for example, pdb1) as a user who has appropriate privileges (for example, c##zoran user):

   ```bash
   $ sqlplus c##zoran@pdb1
   ```

2. Create TSDP policy using Data Redaction.

   ```sql
   DECLARE
   2 redact_feature_options DBMS_TSDP_PROTECT.FEATURE_OPTIONS;
   3 policy_conditions DBMS_TSDP_PROTECT.POLICY_CONDITIONS;
   4 BEGIN
   5 redact_feature_options('expression') := 'i=1';
   6 redact_feature_options('function_type') := 'DBMS_REDACT.REGEXP';
   7 redact_feature_options('regexp_pattern') := '([A-Za-z0-9-.]+\.[A-Za-z0-9-.]+)0([A-Za-z0-9-.]+\.[A-Za-z0-9-]{2,4})';
   8 policy_conditions(DBMS_TSDP_PROTECT.DATATYPE) := 'VARCHAR2';
   9 DBMS_TSDP_PROTECT.ADD_POLICY ('redact_regexp_email1', DBMS_TSDP_PROTECT.REDACT, redact_feature_options, policy_conditions);
   10 END;
   11 /
   SQL>
   PL/SQL procedure successfully completed.
   SQL>
   ```

   Figure 7 – TSDP policy using Oracle Data Redaction

How it works…

In step 2, lines 2 and 3 define variables `redact_features_options` and `policy_conditions`. Data redaction settings, for TSPD policy, are defined by using `redact_features_options` variable that holds parameter-value pairs that correspond with the parameters in `DBMS_REDACT.ADD_POLICY` procedure (lines 4-8). Line 9 specifies that data type of protected columns should be `VARCHAR2` in order for redaction settings to be applied on the column.
See also

- You can see Chapter 5, Data Redaction.

Associating transparent sensitive data protection policy with sensitive type

In this recipe, you’ll associate TSDP policy and sensitive type you created in the previous recipes.

Getting ready

To complete this recipe, you’ll need an existing user who has the execute privilege on the dbms_tsdp_protect package (for example, c##zoran).

How to do it…

1. Connect to the database as a user (for example, pdb1) who has appropriate privileges (for example, c##zoran user):

   $ sqlplus c##zoran@pdb1

2. Associate TSDP policy with sensitive type:

   SQL> BEGIN
   2  DBMS_TSDP_PROTECT.ASSOCIATE_POLICY (
   3      policy_name => 'redact_regexp_email',
   4      sensitive_type => 'email_type',
   5      associate => true);
   6  END;
   7 /

   PL/SQL procedure successfully completed.
   SQL>
There's more…
To verify that you successfully associated the TSDP policy and the sensitive type, execute the following query:

```sql
SQL> SELECT POLICY_NAME, SENSITIVE_TYPE FROM DBA_TSDP_POLICY_TYPE;
```

See also
- Creating a sensitive type
- Determining sensitive columns
- Creating transparent sensitive data protection policy

Enabling, disabling, and dropping policy
In this recipe, you’ll learn to enable, disable, and drop transparent sensitive data protection policies.

Getting ready
To complete this recipe, you’ll need two existing users—one to manage TSDP policies and the other to view sensitive data.

How to do it…
1. Connect to the database (for example, pdb1) as a user who has the SELECT privilege on the HR.EMPLOYEES table and the CHALLENGEZHORAN.T1 table or the SELECT ANY TABLE privilege (for example, maja).

   ```bash
   $ sqlplus maja@pdb1
   ```
2. View sensitive data by executing the following two queries:

```sql
SELECT EMAIL FROM HR.EMPLOYEES FETCH FIRST 10 ROWS ONLY;
```

![Figure 9 – Before enabling the policy](image)

```sql
SELECT EMAIL_ADDRESS FROM CHALLENGEZORAN.T1;
```

![Figure 10 – Before enabling the policy](image)
3. Connect to the database (for example, \texttt{pdb1}) as a user who can manage TSDP policies (for example, \texttt{c##zoran}). Enable the TSDP policy:

```sql
SQL> connect c##zoran/oracle@pdb1
Connected.
SQL> BEGIN
  2  DBMS_TSDP_PROTECT.ENABLE_PROTECTION_TYPE (
  3      sensitive_type => 'email_type');
  4  END;
  5 /
PL/SQL procedure successfully completed.
SQL> 
```

4. Repeat step 2 as user \texttt{maja}.

```sql
SQL> select email from hr.employees fetch first 10 rows only;
EMAIL
---
10 rows selected.
```

Figure 12 – Sensitive data is protected

5. Result of the second query is shown in Figure 13:

```sql
SQL> connect maja/oracle@pdb1
Connected.
SQL> select email_address from challengezoran.t1;
EMAIL_ADDRESS
---
ZORAN.PAVLOVIC@xxxx.com
MAJA.VESELICA@xxxx.com
SQL> 
```

Figure 13 – After enabling the policy
6. Connect to the database (for example, pdb1) as a user who can manage TSDP policies (for example, c##zoran). Disable the TSDP policy.

```
SQL> BEGIN
2  DBMS_TSDP_PROTECT.DISABLE_PROTECTION_TYPE (3    sensitive_type => 'email_type');
4  END;
5  /
PL/SQL procedure successfully completed.
SQL> 
```

7. Repeat step 2 as user maja.

```
SQL> SELECT EMAIL FROM HR.EMPLOYEES FETCH FIRST 10 ROWS ONLY;
EMAIL
-------
ABANDA
ABULL
ACABIRO
AERRAZUR
AFRIFF
AKUNOL
AKHNITO
AKHOO
AMCEWEN
AMALSH
10 rows selected.
SQL> 
```

Figure 15 – After the policy was disabled

8. In Figure 16, the result of the second query is shown:

```
SQL> connect maja/oracle@pdb1
Connected.
SQL> select email_address from challengezoran.t1;
EMAIL_ADDRESS
---------------------
ZORAN.PAVLOVCI@CHALLENGEZORAN.COM
MAJA.VESELICA@CHALLENGEZORAN.COM
SQL> 
```

Figure 16 – After the policy was disabled
9. Connect to the database (for example, `pdb1`) as a user who can manage TSDP policies (for example, `c##zoran`). Drop both sensitive columns.

```
SQL> BEGIN
2 DBMS_TSDP_MANAGE.DROPSENSITIVE_COLUMN
3   schema_name => 'CHALLENGEZORAN',
4   table_name => 'T1',
5   column_name => 'EMAIL_ADDRESS');
6 END;
7 /
PL/SQL procedure successfully completed.
```

```
SQL> BEGIN
2 DBMS_TSDP_MANAGE.DROPSENSITIVE_COLUMN
3   schema_name => 'HR',
4   table_name => 'EMPLOYEES',
5   column_name => 'EMAIL');
6 END;
7 /
PL/SQL procedure successfully completed.
```

10. Drop the sensitive type.

```
SQL> BEGIN
2 DBMS_TSDP_MANAGE.DROPSENSITIVE_TYPE
3   sensitive_type => 'email_type');
4 END;
5 /
PL/SQL procedure successfully completed.
```

11. Drop the TSDP policy.

```
SQL> BEGIN
2 DBMS_TSDP_PROTECT.DROP_POLICY
3   policy_name => 'redact_regexp_email');
4 END;
5 /
PL/SQL procedure successfully completed.
```
How it works…

In step 4, you got correct result-column `email_address` in schema `challengezoran` was masked like specified in the policy and full redaction was applied on all values in column `email` in schema `HR` where data wasn't matched to the specified pattern. For more information about redaction policies, see Chapter 5, Data Redaction (the recipe Creating redaction policy when using regular expression redaction).

Before you drop the policy, you don't have to disable it.

There's more…

Another way to enable/disable protection is to use procedures `enable_protection_column` (`.disable_protection_column`):

```sql
SQL> begin
  DBMS_TSDP_PROTECT.ENABLE_PROTECTION_COLUMN(
    schema_name => 'CHALLENGEZORAN',
    table_name => 'T1',
    column_name => 'EMAIL_ADDRESS',
    policy => 'redact_regexp_email');
end;
/
SQL> begin
  DBMS_TSDP_PROTECT.ENABLE_PROTECTION_COLUMN(
    schema_name => 'HR',
    table_name => 'EMPLOYEES',
    column_name => 'EMAIL',
    policy => 'redact_regexp_email');
end;
/
SQL> begin
  DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN(
    schema_name => 'CHALLENGEZORAN',
    table_name => 'T1',
    column_name => 'EMAIL_ADDRESS',
    policy => 'redact_regexp_email');
end;
/
SQL> begin
  DBMS_TSDP_PROTECT.DISABLE_PROTECTION_COLUMN(
    schema_name => 'HR',
    table_name => 'EMPLOYEES',
    column_name => 'EMAIL',
    policy => 'redact_regexp_email');
```
Altering transparent sensitive data protection policy

In this recipe, you'll alter policy you created in recipe Creating transparent sensitive data protection policy and enable it.

Getting ready

To complete this recipe, you’ll need two existing users (for example, c##zoran and maja). Also, update the table hr.employees, as shown in Figure 20:

```
SQL> UPDATE hr.employees SET EMAIL = EMAIL || '@example.com' WHERE 1=1;
107 rows updated.
SQL> commit;
Commit complete.
```

Figure 20 – Set new e-mail addresses in the hr.employees table

How to do it…

1. Connect to the database (for example, pdb1) as a user who can manage TSDP policies (for example, c##zoran):

   ```bash
   $ sqlplus c##zoran@pdb1
   ```

2. If the policy is enabled, disable it for all columns (for instructions how to disable the TSDP policy, see recipe Enabling, disabling, and dropping policy).
3. Connect to the database (for example, pdb1) as a user who can view sensitive data (for example, maja). Execute the following queries:

```
SELECT EMAIL FROM HR.EMPLOYEES FETCH FIRST 10 ROWS ONLY;
```

Figure 21 – Before altering and enabling the policy

```
SQL> SELECT EMAIL FROM HR.EMPLOYEES FETCH FIRST 10 ROWS ONLY;
EMAIL
------------------------
ABANDA@example.com
ABULL@example.com
ACABRIO@example.com
AERRAZU8@example.com
AFRIPP@example.com
AHUNOL8@example.com
AHUTTON@example.com
AKHO00@example.com
AMCE00N@example.com
AWALSH@example.com
10 rows selected.
SQL> 
```

4. Connect to the database (for example, pdb1) as a user who can manage TSDP policies (for example, c##zoran). Alter the TSDP policy and enable it.

```
SELECT EMAIL_ADDRESS FROM CHALLENGEZORAN.T1;
```

Figure 22 – Before altering and enabling the policy

```
SQL> SELECT EMAIL_ADDRESS FROM CHALLENGEZORAN.T1;
EMAIL_ADDRESS
ZORAN.PAVLOVIC@CHALLENGEZORAN.COM
MAJA.VESELICA@CHALLENGEZORAN.COM
SQL> 
```
5. View sensitive data as the user maja (repeat step 3).
How it works…
After you alter the policy, you have to manually enable it (it isn’t automatically enabled).

See also

- Creating transparent sensitive data protection policy, from this chapter
In this chapter, we will cover the following tasks:

- Creating a database analysis policy
- Creating a role analysis policy
- Creating a context analysis policy
- Creating a combined analysis policy
- Starting and stopping privilege analysis
- Reporting on used system privileges
- Reporting on used object privileges
- Reporting on unused system privileges
- Reporting on unused object privileges
- How to revoke unused privileges
- Dropping the analysis

Introduction

Privilege analysis is a new security feature, introduced in Oracle Database 12c. It is only available in Oracle Database Enterprise Edition, and from licensing viewpoint, it is part of Oracle Database Vault option.

Privilege analysis is very useful to implement and maintain the least privilege principle by identifying both privileges that users are actually using (used privileges) and those that are only granted to them (unused privileges).
General steps to analyze privileges using this feature are shown in Figure 1.

![Flowchart](image)

**Figure 1 – The steps to analyze the used and unused privileges**

In this chapter, it is assumed that all users have a `create session` privilege, and in the following table, other privileges and roles granted to the users and roles are listed:

<table>
<thead>
<tr>
<th>USER/ROLE</th>
<th>HR.EMPLOYEES</th>
<th>OE.ORDERS</th>
<th>ROLES/SYS.PRIVS.</th>
</tr>
</thead>
<tbody>
<tr>
<td>BARBARA</td>
<td></td>
<td></td>
<td>P1_ROLE</td>
</tr>
<tr>
<td>NICK</td>
<td></td>
<td></td>
<td>DBA</td>
</tr>
<tr>
<td>ALAN</td>
<td>SELECT, INSERT, UPDATE, DELETE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STEVE</td>
<td></td>
<td></td>
<td>P2_ROLE</td>
</tr>
<tr>
<td>P1_ROLE</td>
<td>SELECT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2_ROLE</td>
<td></td>
<td>SELECT, INSERT, UPDATE, DELETE</td>
<td>SELECT ANY TABLE, CREATE TABLE</td>
</tr>
</tbody>
</table>

Depending on your needs, you can create and use four different types of privilege analysis policies that differ in the scope of the analysis. This scope can be:

- An entire database
- Role-based
- Context-based
- Role- and context-based
Creating database analysis policy

In this recipe, you'll learn to create database privilege analysis policy. It analyzes privileges in the whole database (except privileges used by SYS user). You can use SQL*Plus and Enterprise Manager Cloud Control 12.1.0.3+ (in our case, EM12cR4) to create privilege analysis policies.

Getting ready

You'll need an existing user who can create a privilege analysis policy (has CAPTURE_ADMIN role and SELECT ANY DICTIONARY privilege), for example, SYSTEM user.

How to do it…

1. Connect to the database as system or a user who has appropriate privilege:

   $ sqlplus system

2. Create a privilege analysis policy that captures all the used privileges in the database:

   SQL> BEGIN
   2   SYS.DBMS_PRIVILEGE_CAPTURE.CREATE_CAPTURE(
   3       name => '<policy_name>',
   4       description => '<your_desc>',
   5       type => DBMS_PRIVILEGE_CAPTURE.G_DATABASE);
   6   END;
   7 /

   PL/SQL procedure successfully completed.

   Figure 2 – Database (unconditional) analysis policy
How it works…
In step 2, you created database-wide policy that will capture privileges, which are used (and which are granted, but are unused) by users (except the SYS user). However, to start gathering data about privilege usage, you have to enable the policy (see recipe Starting and stopping privilege analysis).

There's more…
Another way to create the same policy is to use Enterprise Manager Cloud Control 12c (EM).

1. Login to EM as a user who has appropriate privileges and select Privilege Analysis from Security drop-down menu (see Figure 3):

![Figure 3 – The choose privilege analysis]
2. Log in to the database as **SYSTEM** user or a user who has appropriate privileges (CAPTURE_ADMIN role and SELECT ANY DICTIONARY privilege).

3. Click on the **Create** button in the **Policy** section (see Figure 5):

![Figure 4 – The login screen](image)

![Figure 5 – Start the process of creating a privilege analysis policy](image)
4. To create a database policy, choose that scope is **Database**, name the policy, and optionally write a description (see Figure 6). Click on the **OK** button:

![Figure 6 – The create policy](image)

5. You should receive a confirmation message and see your newly created policy listed in the table (see Figure 7):

![Figure 7 – A successful message](image)

**See also**

- You can see the *Starting and stopping privilege analysis* recipe.
Creating role analysis policy

In this recipe, you'll create a role analysis policy using SQL*Plus and Enterprise Manager Cloud Control 12c (EM). The usage of directly and indirectly granted privileges to the roles listed in the policy, will be captured if the roles are active for the session.

Getting ready

You'll need an existing user who can create a privilege analysis policy (has a CAPTURE_ADMIN role and a SELECT ANY DICTIONARY privilege), for example, SYSTEM user.

How to do it…

1. Connect to the database as system or a user who has appropriate privileges:

   $ sqlplus system

2. Create a privilege analysis policy that captures all the used privileges granted through roles DBA and P1_ROLE:

   SQL> BEGIN
   2   SYS.DBMS_PRIVILEGE_CAPTURE.CREATE_CAPTURE(
   3       name => '<policy_name>',
   4       description => '<your_desc>',
   5       type => DBMS_PRIVILEGE_CAPTURE.G_ROLE,
   6       roles => role_name_list (<'role1',..., 'role10'>));
   7   END;
   8 /

   PL/SQL procedure successfully completed.

Figure 8 – The role analysis policy
There's more…

Another way to create a role privilege analysis policy is to use EM12c. Repeat steps 1, 2, and 3 from the There’s more… section in the previous recipe. Name the policy, select roles, optionally write a description, and click on OK button (see Figure 9):

![Figure 9 – Creating a role policy](image)

You should receive a confirmation message and see your newly created policy listed in the table (see Figure 10):
Privilege Analysis

Figure 10 – The successful creation of the policy

See also

- You can refer to the Starting and stopping privilege analysis recipe.

Creating context analysis policy

In this recipe, you’ll create a context analysis policy. After the policy is enabled, it will capture privileges when the condition specified in the policy evaluates to true.

Getting ready

You’ll need an existing user who can create a privilege analysis policy (has the CAPTURE_ADMIN role and the SELECT ANY DICTIONARY privilege), for example, the SYSTEM user.
Privilege Analysis

How to do it…

1. Connect to the database as system or a user who has appropriate privileges:

   $ sqlplus system

2. Create a privilege analysis policy that captures all the used (and unused) privileges by Steve:

   SQL> BEGIN
       2   SYS.DBMS_PRIVILEGE_CAPTURE.CREATE_CAPTURE(
       3       name => '<policy_name>',
       4       description => '<your_desc>',
       5       type => DBMS_PRIVILEGE_CAPTURE.G_CONTEXT,
       6       condition => '<your_condition>);
       7   END;
       8 /

   Figure 11 – The context analysis policy

There's more…

Another way to create a context privilege analysis policy is to use EM12c. Repeat steps 1, 2, and 3 from the There’s more… section in the Creating database analysis policy recipe. Name the policy and optionally write a description (see Figure 12):
Figure 12 – The create context policy

Click on the **Build Context Expression** button (a pencil icon; see Figure 13). You can enter expression manually (select **Edit** checkbox) or use the built-in help (select the checkbox **Policy is in effect when** select appropriate options from drop-down menus, click on the **Add** button). Click on the **OK** button.

Figure 13 – The Expression Builder
Make sure that you chose options you wanted (see Figure 14) and then click on the **OK** button:

![Privilege Analysis: Create Policy](image)

**Figure 14 – Checking the filled-out context policy**

You should receive a confirmation message and see your newly created policy listed in the table (see Figure 15):
See also

- You can refer to the *Starting and stopping privilege analysis* recipe. For more information about application contexts, see Chapter 12, *Appendix – Application Contexts*.

Creating combined analysis policy

In this recipe, you’ll create a combined analysis policy. This type of policy defines that the usage of directly and indirectly granted privilege to specified roles will be gathered if roles are enabled in the session and the context condition is satisfied. The context condition can consist of one or more conditions (you can use the **AND** or **OR** Boolean operators).

Getting ready

You’ll need an existing user who can create a privilege analysis policy (has the `CAPTURE_ADMIN` role and the `SELECT ANY DICTIONARY` privilege), for example, the `SYSTEM` user.
Privilege Analysis

How to do it...

1. Connect to the database as system or a user who has appropriate privileges:

   `$ sqlplus system`

2. Create a privilege analysis policy that captures the usage of privileges, when using SQL Developer, which are granted through the role `P2_ROLE`:

   ```sql
   SQL> BEGIN
   2   SYSDBMS_PRIVILEGE_CAPTURE.CREATE_CAPTURE(
   3       name => '<policy_name>',
   4       description => '<your_desc>',
   5       type => DBMS_PRIVILEGE_CAPTURE.G_ROLE_AND_CONTEXT,
   6       roles => role_name_list ('role1','role2','role3','role4','role5','role6','role7','role8','role9','role10'),
   7       condition => '<your_condition>');
   8   END;
   9 END;
   /
   
   SQL> BEGIN
   2   SYSDBMS_PRIVILEGE_CAPTURE.CREATE_CAPTURE(
   3       name => 'COM_PRIV_POL',
   4       description => 'Usage of privileges when using SQL Developer that are granted through role P2_ROLE',
   5       type => DBMS_PRIVILEGE_CAPTURE.G_ROLE_AND_CONTEXT,
   6       roles => role_name_list ('P2_ROLE'),
   7       condition => 'SYS_CONTEXT(''USERENV'',''CLIENT_PROGRAM_NAME'')=''SQL Developer''');
   8   END;
   9 END;
   
   PL/SQL procedure successfully completed.
   ```

   Figure 16 – The combined analysis policy

There's more...

Another way to create a context privilege analysis policy is to use EM12c. Repeat steps 1, 2, and 3 from the There’s more… section in the recipe Creating database analysis policy. Name the policy, select roles, and optionally write a description. Click on Build Context Expression (see Figure 17).
Manually write the policy expression. Click on the **Validate** button and then on the **OK** button (see Figure 18):
You should receive a confirmation message and see your newly created policy listed in the table (see Figure 19).

![Figure 19 – The successful message](image)

**See also**

- You can refer to the *Starting and stopping privilege analysis* recipe. For more information about application contexts, see Chapter 12, *Appendix – Application Contexts.*

**Starting and stopping privilege analysis**

To start capturing privileges, you’ll enable privilege analysis policies you created in the previous recipes.

**Getting ready**

You’ll need an existing user who can manage privilege analysis policies (has the CAPTURE_ADMIN role and the SELECT ANY DICTIONARY privilege), for example, the SYSTEM user.
How to do it...

1. Connect to the database as system or a user who has appropriate privileges:

   ```
   $ sqlplus system
   ```

2. List all existing privilege analysis policies by querying `DBA_PRIV_CAPTURES`.

   ```
   SQL> column name format A20
   SQL> select name, type, enabled
         2  from DBA_PRIV_CAPTURES;
   NAME           TYPE   E
   ROLE_PRIV_POL  ROLE   N
   ALL_PRIV_POL   DATABASE N
   CONT_PRIV_POL  CONTEXT N
   COM_PRIV_POL   ROLE AND CONTEXT N
   ORA$DEPENDENCY DATABASE N
   ```

   Figure 20 – Finding all defined policies

3. Enable a privilege analysis (for example, `ALL_PRIV_POL`, which you created in the first recipe in this chapter):

   ```
   SQL> BEGIN
       2  SYS.DBMS_PRIVILEGE_CAPTURE.ENABLE_CAPTURE(
       3     name          => '<policy_name>' );
       4  END;
   ```

   ```
   SQL> BEGIN
       2  SYS.DBMS_PRIVILEGE_CAPTURE.ENABLE_CAPTURE(
       3     name          => 'ALL_PRIV_POL');
       4  END;
   ```

   PL/SQL procedure successfully completed.

   Figure 21 – Start capturing all privileges
4. Connect to the database as the user **alan** and view the first names of employees who have salary less than **1000**:

```sql
SQL> connect alan
Enter password:
Connected.
SQL> select first_name from HR.EMPLOYEES
  2  WHERE SALARY < 1000;
no rows selected
```

Figure 22 – the first test of select privilege

5. Find first names of employees who earn less than **3 000**.

```sql
SQL> select first_name from HR.EMPLOYEES
  2  WHERE SALARY < 3000;
```

Figure 23 – The second test of select privilege

6. Try to delete all employees whose first name is **Karen**.

```sql
SQL> DELETE FROM HR.EMPLOYEES
  2  WHERE FIRST_NAME = 'Karen';
```

Figure 24 – The test of delete privilege: integrity constraint violation

7. Connect to the database as system or a user who has appropriate privileges. Stop collecting data about privileges:

```sql
SQL> connect system
SQL> BEGIN
  2  SYS.DBMS_PRIVILEGE_CAPTURE.DISABLE_CAPTURE(
  3    name => '<policy_name>');</n  4  END;
```

[ 198 ]
Privilege Analysis

8. Generate the result:

```
SQL> BEGIN
    2    SYS.DBMS_PRIVILEGE_CAPTURE.DISABLE_CAPTURE(
    3    name => 'ALL_PRIV_POL');
    4    END;
    5  /

PL/SQL procedure successfully completed.
```

Figure 25 – Stop capturing

```
SQL> BEGIN
    2    SYS.DBMS_PRIVILEGE_CAPTURE.GENERATE_RESULT(
    3    name => '<policy_name>');
    4    END;
    5  /

PL/SQL procedure successfully completed.
```

Figure 26 – Generating the report

How it works…

In step 3, you started capturing privileges according to the policy ALL_PRIV_POL. Then, you executed several statements as the user ALAN. The point of those statements is to generate records, you'll see in the next recipes.

Delete operation wasn't able to delete row(s) because of integrity constraint violation, but you will see in the next recipes it generated record that DELETE privilege was used.

In step 7, you stopped capturing the privilege usage. In step 8, you populated DBAUSED_XXX and DBAUNUSED_XXX data dictionary views. You can see how to use the results of capture later in this chapter.
There's more…

You can also use Enterprise Manager Cloud Control 12c to manage privilege analysis policies. Repeat steps 1 and 2 from the *There's more* section in the recipe *Creating database analysis policy*.

Select the database policy and click on the Start Capture button (see Figure 27):

![Figure 27 – Enabling a policy](image)

You can either start capture immediately or schedule it. Leave the defaults and click on the OK button (see Figure 28):

![Figure 28 – Start capture immediately](image)
Privilege Analysis

You should receive a confirmation message and see that your policy is active (see Figure 29):

![Figure 29 – An active capture]

Select the role policy and click on **Start Capture** (see Figure 30):

![Figure 30 – Enabling role policy]
You should see under the **Policies** section that both policies are active (see Figure 31):

![Figure 31 – Active policies](image)

Keep in mind that only one policy whose type is **Database** and one policy whose type is not **Database** could be active at any given time.

Verify that you can't enable another non-database policy while role policy is active. Select **CONT_PRIV_POL** and click on the **Start Capture** button. You'll receive warning message, and you'll only be able to schedule job to run at later point in time (see Figure 32).

![Figure 32 – Warning message](image)
To disable capture, select an active policy (for example, ALL_PRIV_POL) and click on the button **Stop Capture** (see Figure 33):

![Figure 33 – Stop capture](image)

Choose to immediately stop capture and tick generate report checkbox. Click on the **OK** button (see Figure 34).

![Figure 34 – Stop capture and generate report](image)
Privilege Analysis

You should receive confirmation message that capture has been stopped and that job has been submitted (see Figure 35):

![Confirmation](image1)

Figure 35 – Confirmation

Refresh page (it may take up to several minutes to complete). You should receive result similar to the one shown in Figure 36.

![Generated Report](image2)

Figure 36 – The generated report

Test all policies you have created in the previous recipes.

**Reporting on used system privileges**

In this recipe, you'll view collected data about the usage of system privileges during a capture interval.
Privilege Analysis

Getting ready
You'll need an existing user who can create a privilege analysis policy (has the CAPTURE_ADMIN role and the SELECT ANY DICTIONARY privilege), for example, the SYSTEM user.

How to do it...

1. Connect to the database as system or a user who has appropriate privileges:
   
   ```
   $ sqlplus system
   ```

2. View system privileges that the user ALAN used:
   
   ```
   SQL> select username, sys_priv
   2     from DBA_USED_SYS_PRIVS
   3     where username='ALAN';
   
   USERNAME   SYS_PRIV
   -------     --------
   ALAN        CREATE SESSION
   ```
   
   Figure 37 – The used system privileges

3. View grant path for the used system privileges generated by ALL_PRIV_POL for the user ALAN:
   
   ```
   SQL> column path format A20
   SQL> select sys_priv, path
   2     from DBA_USED_SYS_PRIVS_PATH
   3     where capture='ALL_PRIV_POL' and username='ALAN';
   
   SYS_PRIV       PATH
   -------        --------
   CREATE SESSION GRANT_PATH('ALAN')
   ```
   
   Figure 38 – The Grant path
There's more...

In EM 12c, after you have generated the report, select the policy and from Actions drop-down menu, select Reports. The Usage Summary report will open (see Figure 39).

Click on the tab Used and choose All for Match radio button, Policy: ALL_PRIV_POL, User Name: ALAN, and click on the Search button. Results are shown in Figure 40:
If you haven’t generated report for the role policy, do it now and return to this tab (the **Used** tab). Find all records generated by `ROLE_PRIV_POL` for user **Nick** (who has a DBA role). Results are presented in Figure 41:

![Privilege Analysis Reports](image)

Figure 41 – The used privileges recorded for the user Nick based on role policy

### Reporting on used object privileges

In this recipe, you’ll view collected data about the usage of object privileges during the capture interval.

### Getting ready

You’ll need an existing user who can create a privilege analysis policy (has the `CAPTURE_ADMIN` role and the `SELECT ANY DICTIONARY` privilege), for example, the `SYSTEM` user.

### How to do it…

1. Connect to the database as system or a user who has appropriate privileges:

```
$ sqlplus system
```
2. View which object privileges the user Alan has used while database policy \texttt{ALL_PRIV_POL} has been active.

![Figure 42 – The used object privileges](image)

3. View grant path by querying \texttt{DBA_USED_OBJPRIVS_PATH}:

![Figure 43 – Object privileges grant path](image)

**There's more...**

In EM 12c, after you have generated the report, select the policy, and from \textbf{Actions} dropdown menu, select \textbf{Reports}. The \textbf{Usage Summary} report will open. Click on the \textbf{Used} tab and verify that the user Alan has used the SELECT and DELETE privileges while \texttt{ALL_PRIV_POL} has been active (see Figure 44):
Reporting on unused system privileges

In this recipe, you'll view the collected data about the unused system privileges during the capture interval.

Getting ready

You'll need an existing user who can create a privilege analysis policy (has the CAPTURE_ADMIN role and the SELECT ANY DICTIONARY privilege), for example, the SYSTEM user.

How to do it…

1. Connect to the database as system or a user who has appropriate privileges:

   
   $ sqlplus system
2. View that the user Alan has used all system privileges that have been granted to him (there are no unused system privileges):

```
SQL> select username, sys_priv
2  from DBA_UNUSED_SYSPRIVS
3  where username='ALAN';

no rows selected
```

Figure 45 – The unused system privileges for the user Alan during the database policy ALL_PRIV_POL capture interval

**There's more...**

To view report about the unused system privileges in EM12c, see instructions to view the used system privileges and under Privilege Analysis: Reports, choose the Unused tab instead of the Used tab.

**Reporting on unused object privileges**

In this recipe, you'll view collected data about the unused object privileges during the capture interval.

**Getting ready**

You'll need an existing user who can create a privilege analysis policy (has the CAPTURE_ADMIN role and the SELECT ANY DICTIONARY privilege), for example, the SYSTEM user.

**How to do it...**

1. Connect to the database as system or a user who has appropriate privileges:

   `$ sqlplus system`

2. View which object privileges the user Alan has used during the database policy capture interval:
Privilege Analysis

There's more...

In EM 12c, after you have generated the report, select the policy, and from **Actions** drop-down menu, select **Reports**. The **Usage Summary** report will open. Click on the **Unused** tab and verify that the user Alan hasn't used the **INSERT** and **UPDATE** privileges while **ALL_PRIV_POL** has been active.

```
SQL> select username, object_owner, object_name, obj_priv
    2     from DBA_UNUSED_OBJPRIVS
    3     where username='ALAN';
```

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>OBJECT_OWNER</th>
<th>OBJECT_NAME</th>
<th>OBJ_PRIV</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALAN</td>
<td>HR</td>
<td>EMPLOYEES</td>
<td>UPDATE</td>
</tr>
<tr>
<td>ALAN</td>
<td>HR</td>
<td>EMPLOYEES</td>
<td>INSERT</td>
</tr>
</tbody>
</table>

Figure 46 – The unused object privileges

Figure 47 – The Unused object privileges report
How to revoke unused privileges

You can manually revoke unused privileges one by one from users, write your own scripts to complete that task, or use Enterprise Manager Cloud Control 12c. In this recipe, you’ll use EM12c to efficiently revoke unused privileges based on reports you generated in the previous recipes.

How to do it…

1. Select policy, and from Actions drop-down menu, choose Revoke Scripts (see Figure 48):

   ![Figure 48 – Create revoke scripts](image)

2. You’ll see a message about required privileges (see Figure 49). Click on the OK button.

   ![Figure 49 – The info message](image)
3. Select policy (Policy Name) and click on the Generate button (see Figure 50):

![Figure 50 – Generating a script](image)

4. Generate script to revoke all the unused object privileges from the user Alan. Fill out form as shown in Figure 51 and click on the Next button:

![Figure 51 – Revoking the script configuration](image)
5. Click on the Select None link and tick revoke checkbox for the user Alan (see Figure 52):

![Figure 52 – Choose to revoke privilege only from the user Alan](image)

Click on the Next button. Review your choices and click on the Save button (see Figure 53):

![Figure 53 – Review](image)
You should receive confirmation similar to the one shown in Figure 54:

![Figure 54- The confirmation message](image)

6. Click on the green arrow in the **Revoke Script** column (Figure 54) to download the generated revoke script. Note that **Regrant Script** has also been generated.

7. View the generated revoke script - `ALAN_OBJ_PRIV_REV_revokeScript.sql` (see Figure 55):

```
-- REVOKE SCRIPT GENERATED BY SYSTEM AT Aug 05, 2015 07:35 AM.
-- SCRIPT FOR REVOKING DIRECTLY GRANTED UNUSED OBJECT PRIVILEGES :
REVOKE INSERT ON HR.EMPLOYEES FROM ALAN;
REVOKE UPDATE ON HR.EMPLOYEES FROM ALAN;
```

![Figure 55 – The generated revoke script](image)

**There's more…**

In EM 12c, there is another excellent option to create a new role based on privilege analysis results. This way, you won’t change an existing role (and affect other users and roles who have that role), but create a new one and afterwards revoke the old role and grant a newly created one.
You can select it from the **Actions** menu (**Create Role**). In Figure 56, the configuration part of a process for creating a new role is shown:

---

**Figure 56 – Create a new role based on policy**

---

**Dropping the analysis**

In this recipe, you'll drop an existing privilege analysis policy. It has to be disabled before dropping; otherwise, you'll receive an error.

**Getting ready**

You’ll need an existing user who can manage privilege analysis policies (has the **CAPTURE_ADMIN** role and the **SELECT ANY DICTIONARY** privilege), for example, the **SYSTEM** user and an existing privilege analysis policy.

**How to do it…**

1. Connect to the database as system or a user who has appropriate privileges:

   ```bash
   $ sqlplus system
   ```
2. Drop a privilege analysis policy (for example, ALL_PRIV_POL, which you created in the first recipe in this chapter):

```sql
SQL> BEGIN
    2   SYS.DBMS_PRIVILEGE_CAPTURE.DROP_CAPTURE(
    3       name => '<policy_name>');
END;
/

PL/SQL procedure successfully completed.
```

3. Verify that all the records about the used and unused privileges, which have been gathered according to the policy, are also dropped:

```sql
SQL> SELECT username, sys_priv, obj_priv, object_owner, object_name
    2   FROM DBA_USED_PRIVS
    3   WHERE capture='<policy_name>';
```

```sql
SQL> select username, sys_priv, obj_priv, object_owner, object_name
    2   from DBA_USED_PRIVS
    3   where capture='ALL_PRIV_POL';

no rows selected
```

There's more...

In EM 12c under the Policies section, select policy you want to drop and click on the Delete button.
8

Transparent Data Encryption

In this chapter, we will cover the following tasks:

- Configuring a keystore location in sqlnet.ora
- Creating and opening the keystore
- Setting a master encryption key in a software keystore
- Column encryption – adding a new encrypted column to a table
- Column encryption – creating a new table that has encrypted column(s)
- Using salt and MAC
- Column encryption – encrypting the existing column
- Autologin keystore
- Encrypting tablespace
- Rekeying
- Backup and recovery

Introduction

Encryption is a very important security mechanism used to enforce confidentiality of data. There are two types of encryption that can be used in the Oracle Database. The first type is application-based encryption, which is implemented using the DBMS_CRYPTO PL/SQL package (this type is not covered in this book), and the second type is Transparent Data Encryption (TDE). TDE is a part of Advanced Security option of Oracle Database Enterprise Edition. It can be used to encrypt data in rest (table columns and tablespaces inside the database) and in transit (network, Recovery Manager (RMAN) backups, and Data Pump Exports).
The word *transparent* in Transparent Data Encryption means that application is not aware that data is encrypted in any way. In other words, application will never see the encrypted data—if user is not authorized to see the data, error (for example, insufficient privileges, table, or view does not exist) will be shown. The only way that a user will see encrypted data is if he or she tries to avoid Oracle Database Access Controls, by reading data files directly.

TDE should never be used as a mechanism of access control. For this purpose, there is a large portfolio of access control mechanisms in Oracle Database (standard Discretionary Access Control, Mandatory Access Control-Oracle Label Security, Virtual Private Database, Database Vault, and so on).

There are two types of TDE: column and tablespace.

In column encryption, only user-selected columns (in user-selected tables) are encrypted. This encryption type is more suitable for systems with small number of columns that need to be encrypted. Encrypting large number of columns can lead to significant performance degradation. This type even encrypts data in memory, which prevents cold boot attacks. There are several encryption algorithms that can be chosen from: AES128, AES192, AES256, and 3DES168. The default one is AES192. Because these are block ciphers, each row that is going to be encrypted need to be padded to a multiple of 16 bytes (for example, if the size of value in row is 11 bytes, additional 5 bytes of storage is needed to encrypt this row). By default, salt and MAC are used (salt and MAC are covered in the Using salt and MAC recipe). There are several restrictions of column encryption:

- Foreign key constraints are not supported because each table has a different table key
- B-Tree indexes are not supported when using salt
- Bitmap indexes are not supported
- Transportable tablespaces are not supported
- Synchronous Change Data Capture (CDC) is not supported
- External Large Objects (LOBs) are not supported
- SYS schema objects cannot be encrypted
Tablespace encryption is the second type of TDE, which has better performance and has fewer restrictions. This type of TDE is usually more suitable for systems that need to encrypt large portion of data in the database. Using this type, all data that resides inside encrypted tablespace is encrypted (no restrictions on data types). Encryption/decryption is performed on the I/O level, so performance overhead can be expected to be seen on that level. Tablespace encryption doesn't require additional storage. Unlike column encryption, tablespace encryption supports the following:

- Foreign keys
- Bitmap indexes
- Transportable tablespaces (as long as platforms are of the same endian and the same keystore exists on both locations)
- All data types

However, there are still some limitations. Following things are not supported in tablespace encryption:

- BFILE cannot be encrypted
- External tables cannot be encrypted
- UNDO tablespace cannot be encrypted
- TEMP tablespace cannot be encrypted
- SYSTEM tablespace cannot be encrypted
- Key for tablespace cannot be rekeyed (workaround is to create another encrypted tablespace and move all data to this newly created tablespace)
TDE uses two-tier key architecture. For column encryption, columns are encrypted using column (also known as table) keys. There is only one key per table regardless of number of columns that are encrypted in that particular table. For tablespace encryption, tablespaces are encrypted using tablespace keys. Both table and tablespace keys are stored in data dictionary inside Oracle Database. These keys are encrypted using a master key. There is only one master key per database (in Oracle multitenant environment, there is one master key per pluggable database). This master key is stored in a keystore outside the Oracle Database. This keystore can be a software keystore (in previous versions of Oracle Database, it's been named Oracle Wallet) or a hardware keystore (for example, Hardware Security Module). There is only one keystore per database (in Oracle Multitenant environment, there is only one keystore per whole container database). This means that in Oracle Multitenant, there will be one keystore (software or hardware) per container database, which contains multiple master keys (one for each pluggable database that is plugged in that particular container database). A keystore is secured by a password, which is used during maintenance operations (keystore opening and closing, rekeying master key, and so on).

Keystore's password is not the same as the master key.

Configuring keystore location in sqlnet.ora

In this recipe, you're going to configure the location of a software keystore in a regular file system. If you want to use Hardware Security Module (HSM), see the official Oracle documentation (Chapter 3 in Oracle Advanced Security Guide, part named Configuring Hardware Keystore).
How to do it…

1. Create a directory, to hold a keystore, that is accessible to the owner of Oracle software (for example, $ORACLE_BASE/admin/ora12cR1/wallet). See Figure 1:

![Figure 1 – Create a directory and edit sqlnet.ora](image)

2. Edit sqlnet.ora and add entry to specify the location of the keystore (see Figure 1 and 2). This step is optional if you are using default location for the wallet, which is $ORACLE_HOME/admin/<db_name>/wallet.

![Figure 2 – Define ENCRYPTION_WALLET_LOCATION parameter](image)

Creating and opening the keystore

In this recipe, you’re going to create a password-based keystore. Open it and learn to check its status.
Getting ready

It is assumed that the keystore location is already configured (instructions are given in the recipe Configuring keystore location in sqlnet.ora). In this recipe, you’ll grant, as the SYS user, administer key management privilege, or SYSKM administrative privilege to an existing user (for example, maja).

How to do it…

1. Connect to the database as a user who can grant administer key management privilege (for example, SYS) and grant the privilege to an existing user (for example, maja).
2. To create a password-based software keystore, connect to the database as the user in the previous step (for example, maja) and execute the following statement (after you change parameters so that they are appropriate for your environment) (an example is shown in Figure 3):

   ```sql
   SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE 'keystore_location' IDENTIFIED BY keystore_password;
   ```

   ```sql
   SQL> grant administer key management to maja;
   Grant succeeded.

   SQL> connect maja
   Enter password: Connected.
   SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/u01/app/oracle/admin/oral2cR1/wallet' identified by welcome1;
   keystore altered.
   ```

   Figure 3 – Creating a password-based software keystore
3. Open the keystore you created in the previous step by executing the following statement (see Figure 4):

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY keystore_password;
```

![Figure 4 – Opening the password-based keystore](image)

**How it works…**

In step 2, you create a new wallet, which is a file with `.p12` extension, in a wallet directory. In step 3, you opened the keystore. It will remain open until you manually close it.

**There's more…**

Verify that the keystore has been successfully created in step 2 by checking that the file `ewallet.p12` exists in the directory you specified as a keystore location (`ENCRYPTION_WALLET_LOCATION` parameter in `sqlnet.ora`). You should get the similar result to the one shown in Figure 5.

![Figure 5](image)

To view the status of the keystore execute the following statements:

```bash
$ sqlplus / as syskm
SQL> SELECT STATUS, WALLET_TYPE FROM V$ENCRYPTION_WALLET;
```
You should receive the same result as shown in Figure 6. The OPEN_NO_MASTER_KEY status means that the keystore is opened, but a master key hasn’t been generated yet.

<table>
<thead>
<tr>
<th>STATUS</th>
<th>WALLET_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPEN_NO_MASTER_KEY</td>
<td>PASSWORD</td>
</tr>
</tbody>
</table>

Figure 6

### Setting master encryption key in software keystore

In this recipe, you’re going to create the first master key for the password-based software keystore you created and opened in the previous recipe.

### Getting ready

It is assumed that software keystore is already opened. To complete this recipe, you’ll need an existing user who has the SYSKM administrative or administer key management privilege (for example, maja).

### How to do it…

1. Connect to the database as a user who has the SYSKM administrative or administer key management privilege (for example, maja):

   ```bash
   $ sqlplus maja
   ```
2. Create a master key for the password-based keystore (Figure 7 shows the creation of master key for the keystore you created in the recipe *Creating and opening the keystore*):

```
SQL> ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY
keystore_password
WITH BACKUP
USING 'desc_purpose';
```

```
SQL> ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY welcome1 WITH BACKUP USING
'transparent';
keystore altered.
```

Figure 7

**There's more…**

The `WITH BACKUP` clause in step 2 instructs Oracle Database to create a backup of a keystore before the creation of a master key. This backup is created in the same directory where keystore resides and is created in the form `ewallet_timestamp.p12` (where timestamp represents timestamp of backup creation).

Verify the status of the keystore (Figure 8):

```
SQL> select status, wallet_type from v$encryption_wallet;

<table>
<thead>
<tr>
<th>STATUS</th>
<th>WALLET_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPEN</td>
<td>PASSWORD</td>
</tr>
</tbody>
</table>
```

Figure 8 – The status of the keystore after master key was created

**See also**

- If you want to learn to change a master key, see the *Rekeying* recipe.
Column encryption – adding new encrypted column to table

In this recipe, you'll add a new column, which will be encrypted using a nondefault encryption algorithm, to an existing table.

Getting ready

It is assumed that a keystore is opened and a master key is created.

How to do it...

1. Connect to the database as a user who has administer key privilege or SYSKM privilege (for example, maja) and verify that the keystore is in the OPEN status. You should get the result similar to the one depicted in Figure 9:

   $ sqlplus maja

   SQL> SELECT WRL_PARAMETER, STATUS, WALLET_TYPE FROM V$ENCRYPTION_WALLET;

   WRL_PARAMETER | STATUS | WALLET_TYPE
   ------------- |------- |------------
   /u01/app/oracle/admin/oral2cR1/wallet/ | OPEN   | PASSWORD

   Figure 9

2. Add a column (for example, bonus) to a table (for example, hr.employees), encrypted using the AES 256 algorithm.

   SQL> ALTER TABLE HR.EMPLOYEES ADD (BONUS NUMBER(10) ENCRYPT USING 'AES256');

   Table altered.

   Figure 10 – Adding the new encrypted column to the table
Column encryption – creating new table that has encrypted column(s)

In this recipe, you’re going to learn to use TDE column encryption to encrypt columns in a newly created table.

Getting ready

It is assumed that a keystore is opened and a master key is created.

How to do it…

1. Connect to the database as a user who has administer key privilege or SYSKM privilege (for example, maja):

   ```
   $ sqlplus maja
   ```

2. Create a new table (for example, table enc_cols in schema hr) that has, for example, the following structure:

   ![Table Structure]

   ```
   SQL> CREATE TABLE HR.ENC_COLS (
   2       NAME VARCHAR2(50),
   3       CREDIT_LIMIT NUMBER(10) ENCRYPT,
   4       SALARY NUMBER(10) ENCRYPT);
   Table created.
   ```

   Figure 11 – A syntax to create the table hr.enc_cols
3. Connect to the database as a user who can insert and view data in the table (for example, hr user):

   SQL> connect hr

4. Insert several arbitrary values into the table HR.ENC_COLS.

   ```sql
   SQL> INSERT INTO HR.ENC_COLS VALUES ('Debra',50000,20000);
   1 row created.
   SQL> INSERT INTO HR.ENC_COLS VALUES ('Sarah',48000,18000);
   1 row created.
   SQL> INSERT INTO HR.ENC_COLS VALUES ('Tim',45000,14000);
   1 row created.
   SQL> INSERT INTO HR.ENC_COLS VALUES ('Alex',49000,23000);
   1 row created.
   ```

   Figure 12 – Test values

5. Verify that the user can view unencrypted values in all columns.

   ```sql
   SQL> SET LINESIZE 300
   SQL> COLUMN NAME FORMAT A10
   SQL> select * from hr.ENC_COLS;
   
   NAME | CREDIT_LIMIT | SALARY
   --------------------------
   Debra    | 50000        | 20000
   Sarah    | 48000        | 18000
   Tim      | 45000        | 14000
   Alex     | 49000        | 23000
   ```

   Figure 13- Encryption is transparent
6. Connect to the database as a user who can't view data in the table (for example, **james**) and try to view data in all columns:

```sql
SQL> connect james
SQL> select * from hr.enc_cols;
```

| SQL> connect james
Enter password:  
Connected. 
SQL> select * from hr.enc_cols;  
select * from hr.enc_cols  
*  
ERROR at line 1:  
ORA-00942: table or view does not exist |

---

**Using salt and MAC**

In this recipe, you'll understand when you should use salt and MAC.

**Getting ready**

It is assumed that a keystore is opened and a master key is created.

**How to do it...**

1. Connect to the database as a user who has administer key privilege or **SYSKM** privilege (for example, **maja**):

```
$ connect maja
```
2. Encrypt two columns in an existing table (for example, `sh.customers`)

```sql
SQL> ALTER TABLE SH.CUSTOMERS MODIFY (  2  CUST_LAST_NAME ENCRYPT USING 'AES256',  3  CUST_STREET_ADDRESS ENCRYPT USING 'AES256' NO SALT);
Table altered.
```

Figure 15 – Using salt and MAC

**How it works…**

In step 2:

- You encrypted the `last_name` column using the AES256 algorithm with salt and used MAC
- You encrypted the `cust_street_address` column using the AES256 algorithm with no salt and used MAC

In general, you have to use same encryption algorithm for all encrypted columns at the same time. You can choose a `SALT` option on the encrypted column level in a table, but you have to choose either the `MAC` or `NOMAC` option on a table level (meaning that all encryption columns in a table must use the same option).

**There's more…**

To understand why salt is important, let’s consider a basic scenario that doesn’t use salt. For example, if we have 100 rows and they contain only values `A, B, C,` and `D`, this will mean that there are only 4 different values in 100 rows. If we know that value `A` exists in 3 rows, value `B` exists in 20, value `C` exists in 30, and value `D` exists in 47 rows, we can then check cyphertexts (because there will be only 4 different values in cyphertext as well). By evaluating it, we can find that one cyphertext that exists in 3 rows will be value `A`, one that exists in 20 rows will be value `B`, and so on. In order to avoid this problem, we can introduce salt. Salt is used to ensure that each encrypted row has different cyphertext regardless of number of same values in plaintext rows. In our previous example, if we used salt, even though there were only 4 different plaintext values in 100 rows, there will be 100 different cyphertext values in 100 rows, which will be almost impossible for attacker to presume which value corresponds to which row. Consequently, there is no need for salt if plaintext values are unique. There is additional storage cost of 16 bytes per row for salt.
Salt cannot be used on indexed columns.

MAC (short for **Message Authentication Code**) is a hash value computed on encrypted data, which is used for data integrity verification. There is the additional storage cost of 20 bytes per row.

By default, both salt and MAC are used.

```
SQL> alter table hr.test modify (ID ENCRYPT);
alter table hr.test modify (ID ENCRYPT)
* 
ERROR at line 1:
ORA-28338: Column(s) cannot be both indexed and encrypted with salt
```

Figure 16 – TDE column restriction

```
SQL> alter table hr.test modify (ID ENCRYPT NO SALT);
Table altered.
```

Figure 17 – Encrypted primary key with no salt

It is not possible to have salt on indexed column. In Figure 16, it is shown that column ID (which is primary key) cannot be encrypted with salt. In Figure 17 is shown that after changing attribute to **NOSALT**, the primary key column is successfully encrypted.
Column encryption – encrypting existing column

It is common case that organizations first create database and later decide that they want to implement encryption. In this recipe, you're going to encrypt an existing column using TDE column encryption.

Getting ready

It is assumed that a keystore is opened and a master key is created.

How to do it…

1. Connect to the database as a user who can read data from the OE.CUSTOMERS table (for example, the oe user):

   $ sqlplus oe

2. Select data from column you want to encrypt (for example, cust_email), just to verify that the user can view it.

   SQL> SELECT CUST_EMAIL FROM OE.CUSTOMERS WHERE CUST_EMAIL LIKE 'Am%';

   CUST_EMAIL
   1
   Amanda.Brown@THRASHER.EXAMPLE.COM
   Amanda.Finney@STILT.EXAMPLE.COM
   Amanda.Tanner@TEAL.EXAMPLE.COM
   Amrish.Palin@EIDER.EXAMPLE.COM

   Figure 18 – A test query

3. Connect to the database as a user who has administer key privilege or SYSKM privilege (for example, maja):

   SQL> connect maja
4. Encrypt the `cust_email` column in the `oe.customers` table using the default encryption algorithm (AES192) and no salt.

```
SQL> ALTER TABLE OE.CUSTOMERS MODIFY (CUST_EMAIL ENCRYPT NO SALT);
Table altered.
```

Figure 19 – Encrypting an existing column, which has an index

5. Execute steps 1 and 2 again to verify that there is no change in the way user/application views data after TDE column encryption is applied.

There's more...

This example demonstrates that you can't use TDE column encryption to encrypt column, which is a foreign key. If you need to encrypt that kind of column, use TDE tablespace encryption.

1. Connect to the database as a user who can select data from a table, for example, `OE.ORDERS` (for example, the `oe` user):

```
$ sqlplus oe
```

2. Select data from the foreign key column you want to encrypt (for example, `customer_id`), just to verify the user can view it.

```
SQL> select distinct(customer_id) from oe.orders
   2  order by order_total desc
   3  fetch first 8 rows only;

CUSTOMER_ID
-----------
   147
   150
   149
   148
   188
   122
   117
   104

8 rows selected.
```

Figure 20 – A simple test query
3. Connect to the database as a user who has administer key privilege or SYSKM privilege (for example, maja):

   SQL> connect maja

4. Encrypt the customer_id column in the oe.orders table using the default encryption algorithm (AES192).

   SQL> ALTER TABLE oe.orders MODIFY (CUSTOMER_ID ENCRYPT);
   ALTER TABLE oe.orders MODIFY (CUSTOMER_ID ENCRYPT)
   ERROR at line 1:
  ORA-28335: referenced or referencing FK constraint column cannot be encrypted

Figure 21 – A TDE column encryption restriction

Auto-login keystore

Auto-login keystore is a type of keystore that doesn't need to be manually opened. The local autologin keystore can be opened only from computer where it has been created. Autologin keystores have system-generated passwords. They are less secure than password-based keystores. They are created from password-based software keystores.

Getting ready

It is assumed that password-based software keystore is created.

How to do it...

1. Connect to the database as a user who has administer key privilege or SYSKM privilege (for example, maja):

   $ sql plus maja
2. Create (local) an autologin keystore. In our case, keystore_location is /u01/app/oracle/admin/ora12cR1/wallet and keystore_password is welcome1:

   \[\text{SQL} \geq \text{ADMINISTER KEY MANAGEMENT CREATE LOCAL AUTO_LOGIN KEYSTORE FROM KEYSTORE 'keystore\_location' IDENTIFIED BY keystore\_password;}\]
   \[\text{OR}\]
   \[\text{SQL} \geq \text{ADMINISTER KEY MANAGEMENT CREATE AUTO_LOGIN KEYSTORE FROM KEYSTORE 'keystore\_location' IDENTIFIED BY keystore\_password;}\]

**How it works…**

After you executed statement in step 2, in directory that holds password-based keystore, the cwallet.sso file was created. That file represents autologin keystore.

**Encrypting tablespace**

It is not possible to encrypt an existing tablespace using TDE tablespace encryption. In this recipe, you’ll create a new encrypted tablespace.

**Getting ready**

It is assumed that a keystore is opened and a master key is created.

**How to do it…**

1. Connect to the database as a user who has a create tablespace privilege (for example, zoran):

   \[$\text{sql\_plus\ zoran}\$\]
2. Create encrypted tablespace (for example, TEST_ENC) using AES192 encryption algorithm:

```
SQL> CREATE TABLESPACE TEST_ENC
DATAFILE '/u01/app/oracle/oradata/ORA12CR1/datafile/testenc01.dbf'
SIZE 20M
ENCRYPTION USING 'AES192'
DEFAULT STORAGE (ENCRYPT);
```

Figure 22 – Encrypting tablespace

How it works…

In step 2, you create an encrypted tablespace TEST_ENC. To find information about encrypted tablespaces, you can query the V$ENCRYPTED_TABLESPACES view.

```
SQL> connect / as sysdba
Connected.
SQL> desc v$encrypted_tablespaces
Name Null? Type
--------- -------- -----------------------------------------------
TS# NUMBER
ENCRYPTIONALG VARCHAR2(7)
ENCRYPTEDTS VARCHAR2(3)
ENCRYPTEDKEY RAW(32)
MASTERKEYID RAW(16)
BLOCKS_ENCRYPTED NUMBER
BLOCKS_DECRYPTED NUMBER
CON_ID NUMBER

SQL> select encryptedts, encryptionalg from v$encrypted_tablespaces;
ENC ENCRYPT
-------
YES AES192
```

Figure 23 – Finding information about encrypted tablespace
There's more…
You can import existing tables into encrypted tablespace using Oracle Data Pump. Another option is to use SQL statements, for example, CTAS (short for CREATE TABLE AS).

Rekeying
You can change (rekey) a master key and table keys. You cannot rekey tablespace keys.

Getting ready
It is assumed that a keystore is opened and a master key is created.

How to do it…

1. Connect to the database as a user who has administer key privilege or SYSKM privilege (for example, maja):

   $ sqlplus maja

2. To rekey a table (for example, the oe.customer) using a different encryption algorithm (for example, AES128), execute the following statement:

   SQL> ALTER TABLE oe.customers REKEY USING 'AES128';
   Table altered.

Figure 24 – Rekeying a table key
3. Change a master key by executing the following statement (in our example, 
   \textit{keystore\_password} is \textit{welcome1}): 

   \begin{verbatim}
   SQL> ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY 
   keystore\_password 
   WITH BACKUP;
   \end{verbatim}

   \begin{verbatim}
   SQL> ADMINISTER KEY MANAGEMENT SET KEY IDENTIFIED BY welcome1 
   2 WITH BACKUP;
   \end{verbatim}

   keystore altered.

   Figure 25 – Rekeying a master key

**How it works...**

When you changed a table key, in step 2, all encrypted data in the oe.customers table were decrypted and then encrypted using the new table key and the new encryption algorithm. If you just want to change key and use the same algorithm as before, syntax for rekeying is:

   \begin{verbatim}
   ALTER TABLE table\_name REKEY;
   \end{verbatim}

In step 3, you created a backup of the keystore and created a new master key in the keystore. Old master keys are held in the keystore.

   It is extremely important to have backup of thekeystore.
Backup and Recovery

RMAN supports three encryption modes:

- Transparent mode
- Password mode
- Dual mode

In this recipe, you’re going to learn to create encrypted backups using RMAN.

How to do it…

1. Connect to the RMAN as user who has the `sysbackup` privilege:

   ```
   $ rman target '"zoran@orcl as sysbackup'"
   ```

2. Configure encryption on a database level:

   ```
   RMAN> CONFIGURE ENCRYPTION FOR DATABASE ON;
   ```

3. Backup a tablespace example in transparent mode:

   ```
   RMAN> BACKUP TABLESPACE EXAMPLE tag 'tran_mode';
   ```

4. Enable dual mode encryption and backup tablespace example in dual mode:

   ```
   RMAN> SET ENCRYPTION ON IDENTIFIED BY "password_1";
   RMAN> BACKUP TABLESPACE EXAMPLE tag 'dual_mode';
   ```

5. Enable password mode and backup tablespace example in password mode:

   ```
   RMAN> SET ENCRYPTION ON IDENTIFIED BY "password_2" ONLY;
   RMAN> BACKUP TABLESPACE EXAMPLE tag 'pass_mode';
   ```
There's more…

If a backup is created in transparent mode, it can be restored only by using a key that is used to create the backup (stored in the external keystore).

If the backup is created in password mode, it can be restored only by using a password that is provided during the backup.

If the backup is created in dual mode, it can be restored by either key that is stored in the external keystore or the password that is provided during the backup.
In this chapter, we will cover the following tasks:

- Registering Database Vault
- Preventing users from exercising system privileges on schema objects
- Securing roles
- Preventing users from executing a specific command on a specific object
- Creating a rule set
- Creating a secure application role
- Using Database Vault to implement that administrators cannot view data
- Running Oracle Database Vault reports
- Disabling Database Vault
- Re-enabling Database Vault

**Introduction**

Introduction of Oracle Database Vault in 2005 brought a major change in the way security is enforced. Today, 10 years after it was introduced, it remains the most significant tool to control data access and enforce separation of duties in Oracle Database.

From licensing viewpoint, it is only available as an option for Oracle Database Enterprise Edition.
You need to understand how, when, why, and which component of Database Vault you should implement in order to successfully protect your database. In this chapter, you are going to learn to create and appropriately use realms, rules, rule sets, command rules, factors, and secure application roles. Basic concepts are covered in this chapter, whereas doing everyday administration tasks in Database Vault environment, more advanced topics, and security in more complex environments are explained in Chapter 11, Additional Topics.

For all recipes in this chapter, we assume that database is up and running, and each user has at least a create session privilege. Also, you will use Oracle Enterprise Manager Cloud Control 12c.

A **SYS** user, because he is the most powerful user, will be used to test that security is correctly enforced (even for him).

Recipes are tested on Oracle Database 12.1.0.2 in multitenant environment.

## Registering Database Vault

In Oracle Database 12c process of configuring and enabling Database Vault is different than in Oracle Database 11g. In this recipe, you will learn to register Oracle Database Vault in multitenant environment in two situations:

- When Oracle Database 12c is already installed
- During the installation of Oracle Database 12c

## Getting ready

To complete this recipe, you’ll need an existing common user who has a privilege to create users and grant create session and set container privileges (for example, c##maja).
How to do it…

To register Database Vault with Oracle Database 12c when the database is already installed, perform the following steps:

1. Connect to the root container as a user who has privileges to create users and grant create session and set container privileges (for example, c##maja):

   $ sqlplus c##maja

2. Create two users (for example, c##dbv_owner and c##dbv_acctmgr) and grant them create session and set container privileges:

   SQL> create user c##dbv_owner identified by oraDVO123 CONTAINER = ALL;
   SQL> grant create session, set container to c##dbv_owner CONTAINER = ALL;
   SQL> create user c##dbv_acctmgr identified by oraDVA123 CONTAINER = ALL;
   SQL> grant create session, set container to c##dbv_acctmgr CONTAINER = ALL;

3. Connect to the root as a SYS user:

   SQL> connect sys as sysdba

4. Configure the Database Vault users:

   SQL> begin
      DVSYS.CONFIGURE_DV ( dvowner_uname => 'c##dbv_owner',
                         dvacctmgr_uname => 'c##dbv_acctmgr');
   end;
   /

5. Execute the utlrp.sql script:

   SQL> @/rdbms/admin/utlrp.sql
6. Connect to the root as the Database Vault Owner user that you just configured (for example, the `c##dbv_owner`):

```
SQL> connect c##dbv_owner/oraDVO123
```

7. Enable Oracle Database Vault:

```
SQL> exec DBMS_MACADM.ENABLE_DV
```

8. Connect as a `SYS` user:

```
SQL> connect / as sysdba
```

9. Restart the database.

For each PDB, perform step 3 through step 8 and then close and reopen the pluggable database (for example, `PDB1`).

```
SQL> alter pluggable database pdb1 close immediate;

SQL> alter pluggable database pdb1 open;
```

**How it works…**

After you register Oracle Database Vault with Oracle Database 12c, there are number of changes in the Oracle Database. Some database parameters change values, separation of duties is enabled by revoking privileges from some roles and by creating new users.

**There's more…**

You use **Database Configuration Assistant (DBCA)** when you configure Database Vault during the database installation. When you get to step 9 (**Database Options**), click on tab **Database Vault & Label Security**. Select both available checkboxes and fill out text fields to create users: **Database Vault Owner** and **Account Manager** (see Figure 1). You should complete the rest of the installation in the same way you usually do.
See also

- Disabling Database Vault
- Re-enabling Database Vault

Preventing users from exercising system privileges on schema objects

In this recipe, to prevent users to exercise system privileges (such as `select any table`), you are going to first create a realm and then you are going to change it to a mandatory realm. The mandatory realm further restricts access to protected objects. Schema owners and users with object privileges cannot access mandatory realm-secured objects if they are not authorized in realm.
Getting ready

To complete this recipe, you’ll need an existing common user who has a DBA role in the pluggable database PDB1 (for example, c##zoran).

How to do it…

1. Connect to a pluggable database (for example, pdb1) as a Database Vault account manager (for example, c##dbv_acctmgr):
   
   ```sql
   SQL> connect c##dbv_acctmgr@pdb1
   ```

2. Create a new local user in the pluggable database (for example, usr1):
   
   ```sql
   SQL> create user usr1 identified by oracle;
   ```

3. Connect to the pluggable database as a common user who has a DBA role in pdb1 (for example, c##zoran):
   
   ```sql
   SQL> connect c##zoran@pdb1
   ```

4. Grant the select privilege on the table HR.EMPLOYEES and the create session privilege to the user usr1:
   
   ```sql
   SQL> grant select on hr.employees to usr1;
   SQL> grant create session to usr1;
   ```

5. Connect to the Enterprise Manager Cloud Control 12c (EM) as a privileged user (SYSMAN or some other privileged user, for example, zoran). From Security drop-down menu, choose Database Vault (see Figure 2).
6. Log in to the pluggable database PDB1 as a user who is the Database Vault Owner (see Figure 3).
7. On the next page, click on the **Administration** tab (see Figure 4).

8. Create **HR_Realm**, as shown in the following figures (Figure 5 – 8). First, click on the **Create** button.
9. Name the realm (for example, HR_Realm) and leave default values for other parts of the form.

10. Securing all tables in HR schema.
11. Add realm participant (for example, C##ZORAN).

![Figure 8 – Adding authorized user(s)](image)

12. After you make sure that you chose the options you wanted, click on the **Finish** button.

![Figure 9 – Reviewing and clicking on the Finish button](image)
13. Verify that the **usr1** and **hr** users can view data in the **HR.EMPLOYEES** table:

```sql
SQL> connect usr1@pdb1
SQL> select count(*) from hr.employees;

    COUNT(*)
-------------
        107

SQL> connect hr@pdb1
SQL> select count(*) from hr.employees;

    COUNT(*)
-------------
        107
```

14. To provide better security, edit the realm **HR_Realm** and select the checkbox **Mandatory Realm** (see Figure 10 – Figure 12).

![Figure 10 – Editing HR_Realm](image-url)
15. Clicking on the **Finish** button.
Database Vault

There's more...

The difference between participant and owner of the realm is that a realm participant can only exercise system privileges on realm-secured objects, whereas an owner besides that can grant object privileges on realm-secured objects to other users and roles.

Verify that the SYS user can't create a user, after Database Vault is registered:

```sql
SQL> connect sys@pdb1 as sysdba
Enter password:
Connected.

SQL> create user usr1 identified by oracle;
create user usr1 identified by oracle
* 
ERROR at line 1:  
ORA-01031: insufficient privileges
```

Verify that after you created realm HR_Realm, the SYS user can't access data in the table HR.EMPLOYEES.

```sql
SQL> connect sys@pdb1 as sysdba
Enter password:
Connected.

SQL> select count(*) from hr.employees;
select count(*) from hr.employees
* 
ERROR at line 1:  
ORA-01031: insufficient privileges
```

This is the expected behavior because realm protects secured objects from users who try to use their system privileges. In our example, SYS tried to use `SELECT ANY TABLE`, and because he doesn't have direct object privilege (SELECT on HR.EMPLOYEES), he is restricted from selecting data in the table HR.EMPLOYEES.

```sql
SQL> conn c##zoran@pdb1
Enter password:
Connected.
```
After mandatory realm is created, the user `usr1` can't access data in the table `HR.EMPLOYEES` because he/she is not authorized in the realm.

```
SQL> connect usr1@pdb1
Enter password:
Connected.

usr1@CDB1> select count(*) from hr.employees;

ERROR at line 1:
ORA-01031: insufficient privileges
```

The same principle applies even to the schema owner (`HR`).

```
SQL> connect hr@pdb1
Enter password:
Connected.

SQL> select count(*) from hr.employees;

ERROR at line 1:
ORA-01031: insufficient privileges
```
See also

- Securing roles

Securing roles

In the recipe *Preventing users from exercising system privileges on schema objects*, you secured the table `HR.EMPLOYEES` by creating the `HRRealm` realm, and afterwards, you edit it and made it mandatory. In this recipe, you'll learn to protect roles using a realm and a mandatory realm.

Getting ready

To complete this recipe, you’ll need to use a `SYS` user.

How to do it…

1. Connect to the pluggable database `PDB1` as a `SYS` user:

   ```sql
   SQL> connect sys@pdb1 as sysdba
   ```

2. Create the role `role1`:

   ```sql
   SQL> create role role1;
   ```

3. Grant the `create session` and `select any table` privileges to the role:

   ```sql
   SQL> grant create session, select any table to role1;
   ```
4. Create realm **ROLE1_Realm** in Enterprise Manager Cloud Control 12c (see Figure 13).

![Figure 13 – Creating ROLE1_Realm](image)

5. Add realm-secured objects (see Figure 14).

![Figure 14 – Adding secured objects](image)
6. Add realm authorizations and click on the **Next** button (see Figure 15).

![Figure 15 – Realm authorizations](image)

7. Review and click on the **Finish** button (see Figure 16).

![Figure 16](image)
8. Connect to the pluggable database PDB1 as a SYS user:

   SQL> connect sys@pdb1 as sysdba

9. Verify that SYS still can revoke/grant privileges from/to role role1, even though role1 is protected by the realm:

   SQL> revoke select any table from role1;

   SQL> grant drop any synonym to role1;

10. Edit the realm ROLE1_Realm and make it mandatory (select the Mandatory Realm checkbox).

   ![Figure 17 – Editing realm](image-url)
11. Review and confirm the change of ROLE1_Realm.

![Database Vault](image)

There's more…

After we created a realm, the SYS user (or any user that is not authorized in realm) cannot grant the realm-protected role:

```sql
SQL> connect sys@pdb1 as sysdba
Enter password: Connected.

SQL> grant role1 to usr1;
grant role1 to usr1 *
ERROR at line 1:
ORA-47410: Realm violation for GRANT on ROLE1
```
However, user c##zoran is authorized in realm as owner, so he can grant this role:

```sql
SQL> connect c##zoran@pdb1
Enter password: 
Connected.

SQL> grant role1 to usr1;
Grant succeeded.
```

In step 9, we've seen that the SYS user can grant or revoke privileges from role even though the role is protected by realm. After we make the realm mandatory (steps 10 and 11), this is no longer possible:

```sql
SQL> connect sys@pdb1 as sysdba
Enter password: 
Connected.

SQL> revoke drop any synonym from role1;
revoke drop any synonym from role1
* 
ERROR at line 1: 
ORA-47410: Realm violation for REVOKE on ROLE1

SQL> grant update any table to role1;
grant update any table to role1
* 
ERROR at line 1: 
ORA-47410: Realm violation for GRANT on ROLE1
```

**See also**

- *Preventing users from exercising system privileges on schema objects*
Preventing users from executing specific command on specific object

In this recipe, you’ll learn to create command rules. A command rule defines a protected database operation on a specific database object (for example, UPDATE on all tables in HR schema). The evaluation of associated rule set determines if statement will be allowed (executed) or blocked.

How to do it…
Create a command rule by following these steps depicted in Figures 19 and 20.

Figure 19 – Creating a command rule
In the **Command** field, write **UPDATE**; in the **Applicable Object Owner** field, write **OE**; in the **Applicable Object Name** field, write **ORDERS**; and select **Disabled** for **Rule Set** (see Figure 20).

![Create Command Rule](image)

**Figure 20** – A Command rule to secure the UPDATE operation on OE.ORDERS

### How it works…

Command rules can be understood this way: In order to execute command **X** on object **Y** in schema **Z**, rule set with name **A** needs to evaluate **TRUE**.

In our case, it can be understood this way: In order to execute **UPDATE** on the table **ORDERS** in schema **OE**, rule set **Disabled** needs to evaluate **TRUE**. However, because rule set **DISABLED** will evaluate **FALSE** always, consequently, this command is disabled for all users in the database:

```
SQL> connect sys@pdb1 as sysdba
Enter password:
Connected.

SQL> UPDATE OE.ORDERS SET ORDER_MODE = 'TEST' WHERE ORDER_ID < 3000;
ERROR at line 1:
ORA-01031: insufficient privileges
```
Creating a rule set

A rule set is a group of rules, which will be evaluated as a whole, using only **AND** or only **OR** operator. The Boolean result of logical evaluation is used in other Oracle Database Vault components to grant or deny certain actions (for example, deleting data from a table). In this recipe, you’ll learn to create rules and rule sets.

Getting ready

In this recipe, you are going to use Enterprise Manager Cloud Control 12c.

How to do it…

1. Go to Rule Sets component and then click on **Create** (Figure 21).
2. As a name, enter **Working Hours** and click on **Next** (Figure 22). For **Evaluation Options**, choose **All True**.

![Figure 22 – Our rule set “Working Hours”](image)

3. Add two rules (**Is Working Day** and **Is Working Hour**) by clicking on **Create Rule** before adding each of them. Enter the details in **Rule Name** and **Rule Expression** as shown in Figure 23. After you added both rules, click on **Next**.

![Figure 23 – Create two rules](image)
4. Leave all options on defaults and click on Next.

![Figure 24 – Error handling and audit options](image)

5. Click on Finish.

![Figure 25 – Finish](image)
There's more…

To use rule set you have created in this recipe, create command rule for `UPDATE` operation on schema `SCOTT`, table `EMP`, and choose that condition for evaluation whether update operation will be executed is defined by rule set `Working Hours`.

![Create Command Rule](image)

Figure 26 – Create command rule using your rule set

Check time and day (your result will be different). In this case, it's NOT a work day, so rule set will evaluate to `false`.

```
SQL> !date
Sun Jun 14 02:12:02 CEST 2015

Try to increase salary by 300 for the employee whose `empno` is 7902.

```
SQL> UPDATE SCOTT.EMP SET SAL = SAL + 300 WHERE EMPNO = 7902;
```

```
ERROR at line 1:
ORA-01031: insufficient privileges
```
Check time and day (your result will be different). In this case, it is a work day and it is during working hours, so rule set will evaluate to \text{true}.

\[
\text{SQL} > \! \text{date}
\]

\text{Mon Jun 15 14:27:24 CEST 2015}

\[
\text{SQL} > \text{UPDATE SCOTT.EMP SET SAL = SAL + 300 WHERE EMPNO = 7902;}
\]

1 row updated.

Creating a secure application role

A secure application role is a database role whose enablement depends on the evaluation of a specified condition. In this recipe, you’ll learn to create secure application role using Oracle Database Vault. The condition that determines whether the role will be enabled is specified by rule set (you can use built-in rule set or create your own).

How to do it…

1. Create rule set with name \text{Can Access Customer Data} and with rule \text{DVF.F$MACHINE = <your host name>} \text{ (for example, name it: Is Local Machine). In our case, hostname is host01.challengezoran.com (see Figure 27). Refer to the recipe Creating rule sets for full explanation.}
2. In the Database Vault Components panel, click on the **Secure Application Roles** link and then click on the **Create** button (see Figure 28).

![Create a secure application role](image)

Figure 28 – Create a secure application role

3. Define secure application role settings. In our case, we secure the role `cust_role` and condition for enablement is defined by the **Can Access Customer Data** rule set (see Figure 29).

![Define secure application role](image)

Figure 29 – Define secure application role
There's more...

Now, you are going to test behavior of the secure application role.

Connect to pluggable database pdb1 as a user who has the Oracle Database Vault Account Manager role and create the user usr2.

```
SQL> connect c##dbv_acctmgr/oraDVA123@pdb1
Connected.
SQL> create user usr2 identified by oracle1;
User created.
```

Connect to the pluggable database pdb1 as a SYS user and grant a create session privilege to usr2 and select and update privileges to the role cust_role.

```
SQL> connect sys/oracle@pdb1 as sysdba
Connected.
SQL> grant create session to usr2.
Grant succeeded.
SQL> grant select on oe.customers to cust_role;
Grant succeeded.
SQL> grant update on oe.customers to cust_role;
Grant succeeded.
```

Connect to pluggable database pdb1 as usr2 and view information about machine you are accessing the database from. In this example, we are using a built-in factor to get that information. If you want to learn more about factors in Oracle Database Vault, see Chapter 11, Additional Topics.

```
SQL> connect usr2/oracle1@pdb1
SQL> select dvf.f$machine from dual;

+-----------------------------+
<table>
<thead>
<tr>
<th>F$MACHINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>host01.challengezoran.com</td>
</tr>
</tbody>
</table>
+-----------------------------+
```
Set `cust_role` by using the PL/SQL package `DBMS_MACSEC_ROLES`:

```sql
SQL> EXEC DBMS_MACSEC_ROLES.SET_ROLE('CUST_ROLE');
PL/SQL procedure successfully completed.
```

View number of rows in the table `OE.CUSTOMERS`:

```sql
SQL> select count(*) from oe.customers;

COUNT(*)
----------
   319
```

When the same user tries to connect from another machine, he or she won't be able to set the role, which in turn means that he or she won't be able to view data in the table `OE.CUSTOMERS`:

```sql
SQL> connect usr2/oracle1@pdb1
Connected.

SQL> select dvf.f$machine from dual;

F$MACHINE
---------------------------------------------------
host02.challengezoran.com

SQL> EXEC DBMS_MACSEC_ROLES.SET_ROLE('CUST_ROLE');
BEGIN DBMS_MACSEC_ROLES.SET_ROLE('CUST_ROLE'); END;

*  
ERROR at line 1:
ORA-47305: Rule Set violation on SET ROLE (CUST_ROLE)
ORA-06512: at "DVSYS.DBMS_MACUTL", line 49
ORA-06512: at "DVSYS.DBMS_MACUTL", line 398
ORA-06512: at "DVSYS.DBMS_MACSEC", line 306
ORA-06512: at "DVSYS.ROLE_IS_ENABLED", line 4
ORA-06512: at "DVSYS.DBMS_MACSEC_ROLES", line 55
ORA-06512: at line 1

SQL> select count(*) from oe.customers;

select count(*) from oe.customers

*  
ERROR at line 1:
ORA-00942: table or view does not exist
```
Using Database Vault to implement that administrators cannot view data

In this recipe, you will use multiple components (realms, command rules, and rule sets) to secure data in database from administrators.

How to do it…

1. Connect to the pluggable database PDB1 as the user c##dbv_acctmgr:

   SQL> connect c##dbv_acctmgr@pdb1
   SQL> create user orders_dba identified by oracle1;
   SQL> create user orders_user identified by oracle2;

2. Connect to the pluggable database PDB1 as a SYS user and execute the following statements:

   SQL> connect sys@pdb1 as sysdba
   SQL> grant dba to orders_dba;
   SQL> grant create session to orders_user;
   SQL> grant select on oe.orders to orders_user;
   SQL> grant update on oe.orders to orders_user;
   SQL> create role ord_usr_role;
   SQL> grant ord_usr_role to orders_user;
3. Create a realm that protects all objects in OE schema and authorize user orders_dba as owner (for detailed explanation on creating realms, see recipe Preventing users from exercising system privileges on schema objects) – Figure 30.

![Create realm OE_Realm](image)

Figure 30 – Create realm OE_Realm

4. Create realm that protects the ORD_USR_ROLE role and authorize the user c##zoran as owner (for detailed explanation on creating realms, see recipe Preventing users from exercising system privileges on schema objects) – Figure 31.

![Create realm Orders_Role_Realm](image)

Figure 31 – Create realm Orders_Role_Realm
5. Create rule set (exp. role check) that has one rule with name `Has ORD_USR_ROLE` and expression `DBMS_MACUTL.USER_HAS_ROLE_VARCHAR('ORD_USER_ROLE') = 'Y'` (for detailed explanation on how to create rule sets see the recipe *Creating a rule set*) – Figure 32.

![Figure 32 – Create Rule Set Role check](image)

6. Create a command rule for Select on all objects in OE schema, and as rule set, select one that you created in previous step (exp role check) (for detailed explanation on how to create command rules, see recipe *Preventing users from executing a specific command on a specific object*) – Figure 33.

![Figure 33 – Create a command rule](image)
There's more…

We can show that the user `orders_dba` in fact can manage objects in OE schema (for instance, he can create and drop a table test) – this is because he is authorized in realm that protects oe schema:

```
SQL> connect orders_dba@pdb1
Enter password:
Connected.

SQL> create table oe.test(a int);
Table created.

SQL> drop table oe.test;
Table dropped.
```

However, the user `orders_dba` cannot view data that resides inside objects in OE schema – select on objects in this schema is restricted to users that have the role `ORD_USR_ROLE` using command rule:

```
SQL> select count(*) from oe.orders;

ERROR at line 1:
ORA-01031: insufficient privileges
```

The user `orders_user` has the role `ORD_USER_ROLE` and he or she can select data from table in OE schema:

```
SQL> connect orders_user@pdb1
Enter password:
Connected.

SQL> select count(*) from oe.orders;

   COUNT(*)
----------
       105
```
An example of adding a new user to the system and authorizing him to access the data:

Because separation of duties is implemented, there are several users that need to grant certain privileges.

Only account manager can create users in database:

```sql
SQL> connect c##dbv_acctmgr@pdb1
Enter password:
Connected.

SQL> create user orders_user2 identified by oracle3;
User created.
```

The `SYS` user is one of the few users who are authorized to grant a `create session` privilege (after Database Vault is implemented, users with the DBA role cannot grant the `create session` privilege, unless they are authorized in Database Vault)

```sql
SQL> connect sys@pdb1 as sysdba
Enter password:
Connected.

SQL> grant create session to orders_user2;
Grant succeeded.
```

Because `c##zoran` is the only authorized user in realm that protects the role `ORD_USR_ROLE`; he is the only user that can grant that role:

```sql
SQL> connect c##zoran@pdb1
Enter password:
Connected.

SQL> grant ord_usr_role to orders_user2;
Grant succeeded.
```
Orders_dba is the only user that is authorized in realm that protects OE schema, so he is the only user that can grant object privileges on objects in OE schema.

    SQL> connect orders_dba@pdb1
    Enter password:
    Connected.

    SQL> grant select on oe.orders to orders_user2;
    Grant succeeded.

    SQL> grant update on oe.orders to orders_user2;
    Grant succeeded.

After a user is granted all necessary privileges, he or she is able to connect to the database and select data from table in secured schema.

    SQL> connect orders_user2@pdb1
    Enter password:
    Connected.

    SQL> select count(*) from oe.orders;

    COUNT(*)
    --------
    105

Running Oracle Database Vault reports
In this recipe, you will intentionally violate some security controls in order to have data for reports.
How to do it…

Let’s connect as user system and violate some restrictions. First, we are going to select from hr schema, which is going to violate HR realm, and second, we are going to update sal in the scott.emp table, which is going to violate the command rule (we are updating it outside of working hours).

1. SQL> connect system@pdb1

2. SQL> select count(*) from hr.employees;

3. SQL> update scott.emp set sal = sal * 1.20 where empno = 7839;

Let’s see reports for these violations:

1. Go to Database Vault home page (See Figure 2).

2. Click on Enforcement Audit Reports (See Figure 34).
3. Click on **Realm Audit Report** (see Figure 35). Observe the line marked in red (violation from step 2 is audited).

![Figure 35]

4. Next, click on **Command Rule Audit Report** (see Figure 36). Observe the line marked in red (violation from step 3 is audited).

![Figure 36]
Disabling Database Vault

In this recipe, you will disable Database Vault in two ways: Using Enterprise Manager 12c Cloud Control and command line.

How to do it…

1. Go to Database Vault home page of your database or pluggable database and click on **Disable** (see Figure 37):

![Figure 37](image)

2. Click on continue in a small pop-up window (see Figure 38)

![Figure 38](image)
Or

Connect to the database as Database Vault owner and disable it through command line:

```
SQL> EXEC DBMS_MACADM.DISABLE_DV;
```

3. Connect to your database or pluggable database and restart it:

```
SQL> connect / as sysdba
SQL> alter pluggable database pdb1 close immediate;
SQL> alter pluggable database pdb1 open;
```

4. Confirm that Database Vault is disabled:

```
SQL> connect c##dbv_owner@pdb1
SQL> SELECT PARAMETER, VALUE FROM V$OPTION WHERE PARAMETER = 'Oracle Database Vault';
PARAMETER                                VALUE
---------------------------------------- --------------
Oracle Database Vault                     FALSE
```

Re-enabling Database Vault

In this recipe, you will enable previously disabled Database Vault in two ways: Using Enterprise Manager 12c Cloud Control and command line.
How to do it…

1. Go to Database Vault home page of your database or pluggable database and click on **Enable**, then click on continue in a small pop-up window (see Figures 39, 40).
Connect to the database as Database Vault owner and enable it through command line:

```sql
SQL> EXEC DBMS_MACADM.ENABLE_DV;
```

2. Connect to your database or pluggable database and restart it:

```sql
SQL> connect / as sysdba
SQL> alter pluggable database pdb1 close immediate;
SQL> alter pluggable database pdb1 open;
```

3. Confirm that Database Vault is enabled:

```sql
SQL> connect c##dbv_owner@pdb1
SQL> SELECT PARAMETER, VALUE FROM V$OPTION WHERE PARAMETER = 'Oracle Database Vault';
```

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Database Vault</td>
<td>TRUE</td>
</tr>
</tbody>
</table>
In this chapter, we will cover the following tasks:

- Enabling the Unified Auditing mode
- Configuring whether loss of audit data is acceptable
- Which roles do you need to have to be able to create audit policies and to view audit data?
- Auditing RMAN operations
- Auditing Data Pump operations
- Auditing Database Vault operations
- Creating audit policies to audit privileges, actions, and roles under specified conditions
- Enabling an audit policy
- Finding information about audit policies and audited data
- Auditing application contexts
- Purging audit trail
- Disabling and dropping audit policies

Introduction

Unified Auditing is a new feature in Oracle Database 12c, and it introduces new auditing architecture. Some of the characteristics of unified auditing are:

- A single audit trail
- Being based on a read-only table
Unified Auditing

- Extensible Audit Framework for additional columns
- The separation of audit administration with new roles
- Auditing performance is better, especially when used in the queued-write mode

Figure 1 depicts that in preunified auditing architecture, there were many audit trails. Now, there is one consolidated unified audit trail, which simplifies management, and auditors can more easily find audited data they are looking for.

![Figure 1](image1.png)

In Figure 2, a new architecture is shown:

![Figure 2](image2.png)
Enabling Unified Auditing mode

In Oracle Database 12c, unified auditing is not enabled by default. The process of enabling it is simple and equivalent to enabling of other database options.

Getting ready

To complete this recipe, you’ll need to shut down the database.

How to do it…

The process of enabling unified auditing is depicted in Figure 3.
Unified Auditing

1. In our case, there is only one database instance. Connect to the instance as sysoper and shut it down. Also, stop the listener:

$ sqlplus / as sysoper

SQL> shutdown immediate
SQL> exit
$ lsnrctl stop

2. Relink Oracle binaries with the uniaud_on option:

$ cd $ORACLE_HOME/rdbms/lib
$ make -f ins_rdbms.mk uniaud_on ioracle

3. Start the listener and the database instance:

$ lsnrctl start

$ sqlplus / as sysoper
SQL> startup

To verify that unified auditing is enabled, issue the following SQL statement:

SQL> SELECT PARAMETER, VALUE
FROM v$option
WHERE PARAMETER = 'Unified Auditing';

You should see that value for Unified Auditing parameter is true:

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unified Auditing</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

How it works…

When database is upgraded to 12c, by default, it uses the traditional way of auditing (everything like it was in previous versions). However, when you directly install a new database 12c, default auditing is set to **mixed auditing mode**. In both cases, the procedure to enable the **unified auditing mode** is the same.
Unified Auditing

After you enable the unified auditing mode, traditional auditing doesn’t work anymore. Old audit instance parameters (AUDIT_TRAIL, AUDIT_FILE_DEST, AUDIT_SYSLOG_LEVEL, and AUDIT_SYS_OPERATIONS) are disregarded. Also, using syslog and writing audit records to OS are not supported. Predefined unified audit policies that are enabled by default are:

- ORA_SECURECONFIG (database versions: 12.1.0.1, 12.1.0.2)
- ORA_LOGON_FAILURES (Oracle Database 12.1.0.2)

### Predefined unified audit policies

A predefined unified audit policy is a named set of commonly used and recommended audit settings, which already exists in Oracle Database 12c. In Oracle Database 12.1.0.1, there are five predefined unified audit policies, whereas there are eight predefined audit policies in Oracle Database 12.1.0.2. Table 1 lists predefined audit policies.

<table>
<thead>
<tr>
<th>Predefined audit policy</th>
<th>Oracle Database 12.1.0.1</th>
<th>Oracle Database 12.1.0.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORA_RAS_POLICY_MGMT</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ORA_DATABASE_PARAMETER</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ORA_RAS_SESSION_MGMT</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ORA_ACCOUNT_MGMT</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ORA_SECURECONFIG</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ORA_LOGON_FAILURES</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>ORA_CIS_RECOMMENDATIONS</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>ORA_DV_AUDPOL</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 1 – The list of predefined unified audit policies

Even though predefined audit policies have the same name in different versions of Oracle Database, it doesn’t necessarily mean that they are always identical.
If you execute the following statement in both 12.1.0.1 and 12.1.0.2 database versions, as a user who has the audit_admin or dba role:

```sql
SQL> select audit_option from audit_unified_policies
    where policy_name='ORA_SECURECONFIG'
    order by 1;
```

You will note that the ORA_SECURECONFIG predefined unified audit policy is slightly different (for example, audit_options: LOGON, LOGOFF that exist in 12.1.0.1 are removed from the policy in 12.1.0.2 and LOGON is part of ORA_LOGON_FAILURES policy; also some audit options are added in ORA_SECURECONFIG in 12.1.0.2 such as ALTER PLUGGABLE DATABASE).

**There's more…**

In Oracle Database 12cR1 Standard Edition (SE), when you enable unified auditing mode and query the v$option view to verify that it’s enabled you may see the following:

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unified Auditing</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

This bug has been reported in My Oracle Support (17466854) and patch has been released.

**See also**

- Finding information about audit policies and audited data
- Create audit policies to audit privileges, actions, and roles under specified conditions
- Enabling audit policy

**Configuring whether loss of audit data is acceptable**

In this recipe, you’ll learn to set whether audit data is queued in memory or is immediately written to audit trail.
Getting ready

To complete this recipe, you'll need an existing user who has the `audit_admin` role (for example, `jack`).

How to do it…

1. Connect to the database as user who has the `audit_admin` role (for example, `jack`):

   ```sql
   SQL> connect jack
   ```

2. If you want audit records to be immediately written to the unified audit trail set immediate-write mode:

   ```sql
   SQL> EXEC DBMS_AUDIT_MGMT.SET_AUDIT_TRAIL_PROPERTY
       (DBMS_AUDIT_T_MGMT.AUDIT_T_TRAIL_L_UNIFIED,
        DBMS_AUDIT_T_MGMT.AUDIT_T_TRAIL_L_IMMEDIATE_WRITE);
   ```

3. Check that the mode is set to immediate-write:

   ```sql
   SQL> select * from dba_audit_mgmt_config_params
       where parameter_name='AUDIT WRITE MODE';
   ```

   You should see that the value for the `AUDIT WRITE MODE` parameter is `IMMEDIATE WRITE MODE`:

<table>
<thead>
<tr>
<th>PARAMETER_NAME</th>
<th>PARAMETER_VALUE</th>
<th>AUDIT T_TRAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUDIT WRITE MODE</td>
<td>IMMEDIATE WRITE MODE</td>
<td>UNIFIED AUDIT TRAIL</td>
</tr>
</tbody>
</table>

   If you want audit records to be queued in memory and at later time persisted, then set the queued-write mode. Instead of step 2, execute:

   ```sql
   SQL> EXEC DBMS_AUDIT_T_MGMT.SET_AUDIT_T_TRAIL_L_PROPERTY
       (DBMS_AUDIT_T_MGMT.AUDIT_T_TRAIL_L_UNIFIED,
        DBMS_AUDIT_T_MGMT.AUDIT_T_TRAIL_L_QUEUED_WRITE);
   ```
How it works…

The default value for a write mode is the queued-write mode. In this mode, audit data is stored in SGA queues and later automatically persisted in the read-only table in the AUDSYS schema in the SYSAUX tablespace. You can also manually flush content of memory queues to the disk:

```sql
SQL> EXEC SYS.DBMS_AUDIT_MGMT.FLUSH_UNIFIED_AUDIT_TRAIL;
```

You’ll achieve better performance by using the queued-write mode, but in an event of instance crash, you may lose some audit records.

💡 TIP

It is recommended that you use the queued-write mode in case that possibility of some audit data loss is acceptable.

Which roles do you need to have to be able to create audit policies and to view audit data?

In this recipe, you’re going to create two users (for example, jack and jill). Jack’s job is to implement auditing requirements and to make sure that auditing is functioning properly. Jill is an auditor and her job is to analyze audit data.

Getting ready

To complete this recipe, you’ll need an existing user who has the DBA role (for example, maja).

How to do it…

1. Connect to the database as a user who has the dba role (for example, maja):

   ```bash
   $ sqlplus maja
   ```
2. Create the user jack and grant him the create session privilege and the audit_admin role.

    SQL> create user jack identified by pQB$s7a4w2;
    SQL> grant create session, audit_admin to jack;

3. Create the user jill and grant her the create session privilege and the audit_viewer role.

    SQL> create user jill identified by t1n5_R2f3;
    SQL> grant create session, audit_viewer to jill;

How it works...

In Oracle Database 12c, there are two new roles: AUDIT_ADMIN and AUDIT_VIEWER (Figure 4).

They enable the separation of duties in the auditing process. To configure auditing, you no longer need to have the dba role or connect as sysdba. From the security perspective, this is a significant improvement.
Unified Auditing

In step 2, you granted the `AUDIT_ADMIN` role to the newly created user `jack` because that role enables him to create, alter, enable, disable, and drop audit policies, view audit data, and manage the unified audit trail. In step 3, you granted the `AUDIT_VIEWER` role to the user `jill` because that role enables her to view audit data. You may wonder why the `AUDIT_ADMIN` role is designed in such a way that it enables a user to view audit data. One of the reasons could be that when you configure auditing (for example, create and enable audit policies), you have to be able to verify that audit records are generated in a way you have expected they would.

There's more...

To test what can and can't be done as a user who has the `audit_viewer` role, connect to the database as `jill` and try to create the unified audit policy `jill_policy`:

```sql
SQL> connect jill

SQL> create audit policy jill_policy
actions delete on oe.orders;
actions delete on oe.orders *
ERROR at line 2:
ORA-00942: table or view does not exist
```

Even if you grant object privileges on the `oe.orders` table to `jill`, she won't be able to create unified audit policy because she doesn't have the `audit_admin` role or the `AUDIT_SYSTEM` system privilege:

```sql
SQL> conn maja

SQL> grant select,delete on oe.orders to jill;

SQL> connect jill

SQL> create audit policy jill_policy
actions delete on oe.orders;
actions delete on oe.orders *
ERROR at line 2:
ORA-01031: insufficient privileges
```
Revoke select and delete on the `oe.orders` table from Jill:

```
SQL> connect maja
SQL> revoke select, delete on oe.orders from jill;
Revoke succeeded.
```

Grant the `AUDIT SYSTEM` privilege to jill and again try to create the audit policy `jill_policy`:

```
SQL> grant audit system to jill;
SQL> connect jill
SQL> create audit policy jill_policy
    actions delete on oe.orders;
Audit policy created.
```

Drop the unified audit policy `jill_policy` and revoke the `AUDIT SYSTEM` privilege from jill:

```
SQL> drop audit policy jill_policy;
Audit Policy dropped.
SQL> connect maja
SQL> revoke audit system from jill;
```

View audit data:

```
SQL> connect jill
SQL> select dbusername, action_name from unified_audit_trail
    where unified_audit_policies='ORA_SECURECONFIG';
```

Also, a user who has the `audit_viewer` role can access information about defined and enabled unified audit policies.
Throughout this chapter, you’ll use a user who has the audit_admin role (for example, jack), so only test you’ll do right now is to enable the predefined audit policy ORA_ACCOUNT_MGMT and then to disable it:

```sql
SQL> connect jack

SQL> audit policy ora_account_mgmt;
Audit succeeded.

SQL> noaudit policy ora_account_mgmt;
Nbaudit succeeded.
```

**Auditing RMAN operations**

In this recipe, you’ll see that RMAN operations are audited by default.

**Getting ready**

In this recipe, we assume that database is in the ARCHIVELOG mode. To complete this recipe, you’ll need an existing user who has the SYSBACKUP privilege (for example, tom) and an existing user who has the dba role (for example, maja).

**How to do it…**

1. Connect to the target database as a user who has the SYSBACKUP privilege (for example, tom).

   ```sh
   $ rman target "tom@ora12cR1 AS SYSBACKUP"
   ```

2. Backup the EXAMPLE tablespace and view information about backups:

   ```
   RMAN> backup tablespace EXAMPLE;
   RMAN> list backup;
   RMAN> exit
   ```
3. Connect to the database as a user who has the DBA role (for example, maja):

   $ sqlplus maja

4. Find the location of datafile for EXAMPLE tablespace:

   SQL> select file_name from dba_data_files where
tablespace_name='EXAMPLE';
   FILE_NAME
   ----------------------------------------------------------
   /u01/app/oracle/oradata/ORA12CR1/datafile/
o1_mf_example_9z79vpcj_.dbf

5. Remove the EXAMPLE tablespace datafile:

   SQL> rm /u01/app/oracle/oradata/ORA12CR1/datafile/
o1_mf_example_9z79vpcj_.dbf

6. Put the EXAMPLE tablespace offline:

   SQL> alter tablespace example offline immediate;
   SQL> exit

7. Restore the EXAMPLE tablespace datafile:

   $ rman target 'tom@ora12cr1 AS SYSBACKUP'
   RMAN> restore tablespace EXAMPLE;

8. Recover the EXAMPLE tablespace datafile:

   RMAN> recover tablespace EXAMPLE;
   RMAN> exit

9. Put tablespace back online:

   $ sqlplus maja
   SQL> alter tablespace EXAMPLE online;
10. To verify that RMAN operations were successfully audited, execute the following statements:

```sql
SQL> connect jack

SQL> EXEC SYS.DBMS_AUDIT_MGMT.FLUSH_UNIFIED_AUDIT_TRAIL;

SQL> select dbusername, rman_operation
    2    from unified_audit_trail
    3    where rman_operation is not null;
```

**How it works…**

When the mixed or unified auditing mode is enabled, RMAN operations are automatically audited. This means that you don’t create audit policies, but you view and manage audit data in the same way as for other components.

In step 2, you performed the backup of the tablespace `EXAMPLE`. Then, in step 5, you intentionally caused a problem by removing the datafile. Afterwards, you performed restore and recover RMAN operations. The whole point of the example is to execute several RMAN operations. In the `unified_audit_trail` data dictionary view, there are several columns that contain data pertaining to the RMAN events. Their names start with RMAN, so it’s easy to find them.

In step 10, you should get similar result to this one:

<table>
<thead>
<tr>
<th>DBUSERNAME</th>
<th>RMAN_OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOM</td>
<td>Backup</td>
</tr>
<tr>
<td>TOM</td>
<td>List</td>
</tr>
<tr>
<td>TOM</td>
<td>Restore</td>
</tr>
<tr>
<td>TOM</td>
<td>Recover</td>
</tr>
</tbody>
</table>

**See also**

- *The sysbackup privilege – How, when, and why you should use it?* (Chapter 1, Basic Database Security)
- *Finding information about audit policies and audited data*
Auditing Data Pump operations

You can audit Data Pump export, import, or both export and import operations by creating audit policies.

Getting ready

To complete this recipe, you’ll need an existing user who has the audit_admin role (for example, jack). Also, it is assumed that directory for export operations (for example, my_dir) is created and a user (for example, maja) who is going to perform the Data Pump export has read and write privileges on the directory.

```sql
SQL> CREATE DIRECTORY my_dir AS '/u01/app/oracle/oradata/export';

SQL> grant read, write ON DIRECTORY my_dir to maja;
```

How to do it…

1. Connect to the database as a user who has the audit_admin role (for example, jack):

   ```sh
   $ sqlplus jack
   ```

2. Create an audit policy to audit Data Pump export operations:

   ```sql
   SQL> CREATE AUDIT POLICY DP_POLICY ACTIONS
   COMPONENT=datapump export;
   ```

3. Enable the audit policy:

   ```sql
   SQL> AUDIT POLICY DP_POLICY;
   ```

4. Export the table hr.departments:

   ```sh
   $ expdp maja@ora12cR1 dumpfile=test tables=hr.departments DIRECTORY=my_dir
   ```
5. Verify that the export operation was successfully audited:

```sql
SQL> connect jack
SQL> select DP_TEXT_PARAMETERS1, DP_BOOLEAN_PARAMETERS1
    from unified_audit_trail
    where audit_type='Datapump' and dbusername='MAJA';
```

See also

- Enabling audit policy
- Finding information about audit policies and audited data

Auditing Database Vault operations

In this recipe, you'll learn to audit Oracle Database Vault events.

Getting ready

To complete this recipe, you'll need to use Oracle Database 12c, which has Oracle Database Vault enabled and at least some of the components configured (for example, the realm HR realm and rule set Working Hours). Also, you'll need an existing user who has the audit_admin role (for example, jack).

How to do it…

1. Connect to the database as a user who has the audit_admin role (for example, jack):

   ```
   $ connect jack
   ```

2. Create the audit policy dbv_policy:

   ```sql
   SQL> CREATE AUDIT POLICY dbv_policy
          ACTIONS COMPONENT = DV Rule Set Failure on "Working Hours", realm violation on "HR Realm";
   ```
3. Enable the audit policy `dbv_policy`:

   ```sql
   SQL> audit policy dbv_policy;
   ```

4. Execute several statements that will cause generation of audit records:

   ```sql
   SQL> select * from oe.orders;
   SQL> update hr.employees set salary=30000 where salary=24000;
   ```

**How it works...**

To create an audit policy that captures Oracle Database Vault events, specify `ACTIONS COMPONENT = DV <action> ON <object>`. In step 2, you defined the audit policy `dbv_policy` that encapsulates the rules: audit records should be generated when somebody tries to access protected objects during nonworking hours or when unauthorized person tries to access objects secured by HR Realm.

In the unified audit trail, Oracle Database Vault-specific audit data is stored in the columns whose name starts with `DV_`.

**There's more...**

When you are using Oracle Database Vault, you can also additionally secure your auditing infrastructure by creating a realm around the `AUDIT_ADMIN` and `AUDIT_VIEWER` roles. This allows you to control who can grant those roles.

**See also**

- *Re-enabling Database Vault* (Chapter 9, *Database Vault*)
- *Creating a rule set* (Chapter 9, *Database Vault*)
Creating audit policies to audit privileges, actions and roles under specified conditions

In this recipe, you will create several unified audit policies.

Getting ready

To complete this recipe, you’ll need two existing users:

- A user who has the audit_admin role (for example, jack)
- A user who has the create session privilege (for example, john)

Also, you should create the roles hr_role and oe_role as stated here and grant hr_role to the user john.

```
SQL> create role hr_role;
SQL> grant select any table, create table to hr_role;
SQL> grant insert on hr.departments to hr_role;
SQL> create role oe_role;
SQL> grant drop any table to oe_role;
SQL> grant select, update on oe.orders to oe_role;
SQL> grant oe_role to hr_role;
SQL> grant hr_role to john;
```

How to do it…

1. Connect to the database as a user who has the audit_admin role (for example, jack):
   
   ```
   $ sqlplus jack
   ```

2. Create audit policy my_policy1:

   ```
   SQL> CREATE AUDIT POLICY MY_POLICY1
   PRIVILEGES SELECT ANY TABLE
   ACTIONS CREATE TABLE, DROP TABLE;
   ```
3. Create the audit policy `role_con_policy`:

```sql
SQL> CREATE AUDIT POLICY ROLE_CON_POLICY
    ROLES HR_ROLE
    WHEN 'SYS_CONTEXT(''USERENV'',''HOST'')='dbhost.orapassion.com''
    EVALUATE PER SESSION;
```

4. Create the audit policy `hr_policy`:

```sql
SQL> CREATE AUDIT POLICY HR_POLICY
    ACTIONS SELECT, INSERT, UPDATE, DELETE ON HR.DEPARTMENTS;
```

5. Create the audit policy `oe_policy`:

```sql
SQL> CREATE AUDIT POLICY OE_POLICY
    ACTIONS ALL ON OE.ORDERS;
```

**How it works…**

When you create a unified audit policy, it is stored in the first-class object owned by `SYS` schema (According to the official Oracle documentation, Oracle Database Security Guide 12c, E48135-09, p.22-4).

Audit records generation, as defined in a unified audit policy, starts after you enable the policy.

In step 2, you created the audit policy `my_policy1`.

In step 3, you created the audit policy `role_con_policy`, which defines that audit records will be generated when a user is connected to the database from the specified host (dbhost.orapassion.com) and system privileges that are directly granted to `HR_ROLE` are used. The Role `HR_ROLE` has to exist at the time audit policy `role_con_policy` is created because if it doesn’t exist you will get an error message:

```
ERROR at line 2:
ORA-01919: role 'HR_ROLE' does not exist
```
In step 4, you created the audit policy hr_policy, the way it is written, audit records will be generated for select, insert, and update operations on all objects and for delete operations on hr.departments.

A common pitfall: People often define object-wise audit policies, the way you did in step 4 (<object_action_1>, <object_action_2>, ..., <object_action_n> ON <object>). However, in most cases, the behavior they really want to get should be defined by writing <object_action_1> ON <object>, <object_action_2> ON <object>, ..., <object_action_n> ON <object>.

In step 5, you created the audit policy oe_policy, which will be used in order to audit all actions on table orders in oe schema. In Oracle Database 12.1.0.1 due to the bug, audit records for this policy are not generated (16714031- Audit policy using actions all does not record audit trails (MOS)). Workaround is to specify one by one actions instead of using keyword ALL. The bug is fixed in Oracle Database 12.1.0.2.

See also

- Enabling audit policy

Enabling audit policy

In this recipe, you will learn to use different options to enable unified audit policies.

Getting ready

To complete this recipe, you’ll need an existing user who has the audit_admin role (for example, jack) and several other existing users (for example, john, maja, and zoran).
How to do it…

1. Connect to the database as a user who has audit_admin role (for example, jack)
   
   SQL> connect jack

2. Enable audit policy oe_policy in such way that it applies only to user JOHN
   
   SQL> audit policy OE_POLICY BY JOHN;

3. Enable audit policy hr_policy to capture only successful events.
   
   SQL> AUDIT POLICY HR_POLICY WHENEVER SUCCESSFUL;

4. Enable policy my_policy1 to audit unsuccessful events for all users except maja and zoran.
   
   SQL> audit policy my_policy1 EXCEPT MAJA, ZORAN WHENEVER NOT SUCCESSFUL;

5. Enable audit policy role_con_policy using default options.
   
   SQL> audit policy role_con_policy;

How it works…

In step 2, you defined BY list, which means that only user(s) listed on that list will be affected by the policy.

In step 3, you defined that audit policy hr_policy is applied to all users, but only successful operations will generate audit records.

In step 4, you defined EXCEPT list, which means that listed users will not be affected by audit policy. Also, audit records will be generated only for the failed operations.
Unified Auditing

In step 5, you enabled audit policy using default options, which means that role_con_policy will affect all users for both successful and unsuccessful events.

You can't use both BY and EXCEPT lists for the same policy statement.

Finding information about audit policies and audited data

In this recipe, you will view audited data and find information about unified audit policies.

Getting ready

To complete this recipe, you'll need three existing users:

- A user who has audit_admin role (for example, jack)
- A user who has hr_role and oe_role (for example, john), created in recipe Creating audit policies to audit privileges, actions and roles under specified conditions
- A user who has admin_viewer role (for example, jill)

Also, you'll need to connect to the database as SYS user.

How to do it…

1. Connect to the database as a user who has the audit_admin role (for example, jack):

   $ connect jack

2. Find which unified audit policies are defined (exist in the database):

   SQL> select distinct policy_name
          from audit_unified_policies;
   SQL> desc audit_unified_policies
3. View which unified audit policies are enabled:

```
SQL> select * from audit_unified_enabled_policies;
```

4. Connect to the database as the user **john**:

```
SQL> connect john
```

5. Execute several statements on the tables **HR.EMPLOYEES**, **HR.DEPARTMENTS**, and **OE.ORDERS**:

```
SQL> create table t(a number(10));
SQL> select count(*) from oe.orders;
SQL> select first_name from hr.employees;
SQL> drop table t;
SQL> connect sys / as sysdba
SQL> create table hr.my_table(b varchar2(10));
SQL> connect john
SQL> drop table hr.my_table;
```

6. Connect to the database as a user who has the **audit_viewer** role (for example, **jill**):

```
SQL> connect jill
```

7. View audit records:

```
SQL> set linesize 250
SQL> col event_timestamp format a30
SQL> col action_name format a20
SQL> col unified_audit_policies format a20
SQL> col sql_text format a80
SQL> select event_timestamp, action_name, unified_audit_policies, sql_text from unified_audit_trail where DBUSERNAME = 'SYS' and ACTION_NAME NOT IN ('LOGON','LOGOFF') ORDER BY EVENT_TIMESTAMP DESC;
SQL> select event_timestamp, action_name, unified_audit_policies, sql_text from unified_audit_trail where DBUSERNAME = 'JONH' and ACTION_NAME NOT IN ('LOGON','LOGOFF') ORDER BY EVENT_TIMESTAMP DESC;
```
Auditing application contexts

In this recipe, you will configure auditing of information contained in an application context.

Getting ready

To complete this recipe, you’ll need an existing (or predefined) application context and a user who has the audit_admin role (for example, jack).

How to do it…

1. Connect to the database as a user who has the audit_admin role (for example, jack):

   ```bash
   $ sqlplus jack
   ```

2. Configure application context auditing:

   ```sql
   SQL> AUDIT CONTEXT NAMESPACE USERENV
   ATTRIBUTES SESSION_USER, SERVICE_NAME;
   Audit succeeded.
   SQL> AUDIT CONTEXT NAMESPACE USERENV
   ATTRIBUTES HOST BY jill;
   Audit succeeded.
   ```

3. View for which application contexts audit data is going to be captured:

   ```sql
   SQL> set linesize 180
   SQL> column namespace format A30
   SQL> column attribute format A30
   SQL> column user_name format A30
   SQL> select * from audit_unified_contexts;
   ```

4. Connect user jill as follows:

   ```sql
   SQL> connect jill
   ```
5. View audit records:

```sql
SQL> SELECT APPLICATION_CONTEXTS FROM UNIFIED_AUDIT_TRAIL
WHERE APPLICATION_CONTEXTS IS NOT NULL;
```

**How it works…**

The result of the statements in step 3:

<table>
<thead>
<tr>
<th>NAMESPACE</th>
<th>ATTRIBUTE</th>
<th>USER_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERENV</td>
<td>HOST</td>
<td>JILL</td>
</tr>
<tr>
<td>USERENV</td>
<td>SERVICE_NAME</td>
<td>ALL USERS</td>
</tr>
<tr>
<td>USERENV</td>
<td>SESSION_USER</td>
<td>ALL USERS</td>
</tr>
</tbody>
</table>

You can audit custom application contexts (for example, the ones you created) in the same way.

If needed, execute the following statement as the user jack:

```sql
SQL> EXEC SYS.DBMS_AUDIT_MGMT.FLUSH_UNIFIED_AUDIT_TRAIL;
```

Result after step 5:

<table>
<thead>
<tr>
<th>APPLICATION_CONTEXTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>(USERENV,SERVICE_NAME=SYS$USERS); (USERENV,SESSION_USER=JACK)</td>
</tr>
<tr>
<td>(USERENV,SERVICE_NAME=SYS$USERS); (USERENV,SESSION_USER=JILL)</td>
</tr>
<tr>
<td>(USERENV,HOST= dbhost.orapassion.com)</td>
</tr>
</tbody>
</table>

**There's more…**

To disable auditing of application contexts, you should use the `NOAUDIT` command:

```sql
SQL> connect jack
SQL> NOAUDIT CONTEXT NAMESPACE USERENV ATTRIBUTES HOST BY jill;
```
Purging audit trail
You can clean up audit data manually or by scheduling clean up job.

Getting ready
To complete this recipe, you’ll need a user who has the audit_admin role (for example, jack).

How to do it…

1. Connect to the database as a user who has the audit_admin role (for example, jack):

   $ sqlplus jack

2. View number of audit records in the unified audit trail before the cleanup:

   SQL> select count(*) from unified_audit_trail;

   • To perform the manual cleanup, execute:

     SQL> exec DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL(
             AUDIT_TRAIL_TYPE => DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED)

   • To create a purge job:

     SQL> exec DBMS_AUDIT_MGMT.CREATE_PURGE_JOB(
             AUDIT_TRAIL_TYPE => DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED,
             AUDIT_TRAIL_PURGE_INTERVAL => 24,
             AUDIT_TRAIL_PURGE_NAME => 'My_Job',
             USE_LAST_ARCH_TIMESTAMP => TRUE)
3. View number of audit records in the unified audit trail after the cleanup:

   SQL> select count (*) from unified_audit_trail;

How it works…

By default, USE_LAST_ARCH_TIMESTAMP is set to TRUE. It means that only records created before that time will be deleted. If you set that parameter to FALSE, all records will be deleted. It is recommended to use the default value.

There's more…

In multitenant environment, use CONTAINER clause as well (CONTAINER => DBMS_AUDIT_MGMT.CONTAINER_CURRENT or DBMS_AUDIT_MGMT.CONTAINER_ALL).

Disabling and dropping audit policies

In this recipe, you will learn to disable and drop audit policies.

Getting ready

To complete this recipe, you'll need an enabled unified audit policy (for example, oe_policy) and a user who has the audit_admin role (for example, jack).

How to do it…

1. Connect to the database as a user who has the audit_admin role (for example, jack):

   $ sqlplus jack
2. Verify that the policy is enabled:

```
SQL> SELECT POLICY_NAME, ENABLED_OPT, USER_NAME, SUCCESS, FAILURE
      FROM AUDIT_UNIFIED_ENABLED_POLICIES;
```

3. Disable the policy oe_policy:

```
SQL> NOAUDIT policy oe_policy BY JOHN;
```

4. Verify that oe_policy is disabled:

```
SQL> select * from AUDIT_UNIFIED_ENABLED_POLICIES;
```

5. Drop the policy oe_policy:

```
SQL> drop audit policy oe_policy;
```

**How it works...**

In step 2, you checked that the audit policy oe_policy is enabled. In step 3, you disabled it.

When you disable audit policy, make sure that in the NOAUDIT statement, a list of users (BY or EXCEPT) is the same as it was in the AUDIT statement. If in step 3, you omit BY JOHN, audit records will continue to be generated.

To be able to drop audit policy, you have to disable it first. In step 5, you dropped the audit policy oe_policy.

**See also**

- *Enabling audit policy*
In this chapter, we will cover the following tasks:

- Exporting data using Oracle Data Pump in the Oracle Database Vault environment
- Creating factors in Oracle Database Vault
- Using TDE in a multitenant environment

**Introduction**

An Oracle Database Vault component **factor** is a named variable, which can have one or more values, assigned in several ways. The actual value of factor is named **identity**. Each factor has a **factor type**. A factor type is used only for classification purposes. Factors are building blocks for configuring security policies. They can be used in rules/rule sets. You can configure factors by using Oracle Enterprise Manager or the Database Vault API.

**Exporting data using Oracle Data Pump in Oracle Database Vault environment**

In Oracle Database 12c, it is possible to perform Oracle Data Pump regular and transportable export and import operations in the Oracle Database Vault environment.

The process of exporting and importing data in Oracle Database 12c is a bit different than in Oracle Database 11g. The default rule set **Allow Oracle Data Pump Operation** is deprecated.
In this recipe, you'll export data that resides in a schema that is protected by a realm.

**Getting ready**

It is assumed that:

- You are using Oracle Database 12.1.0.2 (the traditional architecture) on Linux
- Sample schemas are installed (you'll use HR schema in this recipe)
- Database Vault is enabled and configured (a Database Vault owner is user `dbv_owner`, account manager is user `dbv_acctmgr`, and realm that protects HR schema is created). This is one way how you can create HR realm:

  SQL> connect dbv_owner
  SQL> BEGIN
  DBMS_MACADM.CREATE_REALM(  
    realm_name    => 'HR Realm',  
    description   => 'Protects HR schema',  
    enabled       => DBMS_MACUTL.G_YES,  
    audit_options => DBMS_MACUTL.G_REALM_AUDIT_OFF,  
    realm_type    => 0);
  END;
  /

  PL/SQL procedure successfully completed.

  The parameter `realm_type` specifies whether realm is a mandatory realm or not. Allowed values, for the parameter, are (realm) and 1 (mandatory realm).

  SQL> BEGIN
  DBMS_MACADM.ADD_OBJECT_TO_REALM(  
    realm_name   => 'HR Realm',  
    object_owner => 'HR',  
    object_name  => '%',  
    object_type  => '%');
  END;
  /

  PL/SQL procedure successfully completed.
A directory for export operations (for example, `dp_dir`) is created and a user (for example, `piter`) who is going to perform Data Pump export has read and write privileges on the directory. Also, the `DATAPUMP_EXP_FULL_DATABASE` role and the `CREATE TABLE` and `UNLIMITED TABLESPACE` privileges have been granted to the user:

```
SQL> connect system
SQL> create directory dp_dir as '/u01/app/oracle/oradata/dp_exp';
SQL> connect dbv_acctmgr
SQL> create user piter identified by T2abc_4z1;
SQL> grant create session to piter;
SQL> connect / as sysdba
SQL> grant create table, unlimited tablespace to piter;
SQL> grant read, write on directory dp_dir to piter;
SQL> grant DATAPUMP_EXP_FULL_DATABASE to piter;
```

How to do it…

1. Connect to the database as a user who has the `DV_OWNER` or `DV_ADMIN` role (for example, `dbv_owner`):

   
   $ sqlplus dbv_owner

2. Verify that the user `piter` has the `DATAPUMP_EXP_FULL_DATABASE` role:

   ```
   SQL> select granted_role from dba_role_privs
   where granted_role like '%FULL%' and grantee='PITER';
   ```

   ![Figure 1 – Prerequisite role](image)

3. Authorize the user `piter` to perform Data Pump operations on `HR` schema (execute the `DBMS_MACADM.AUTHORIZE_DATAPUMP_USER` procedure):

   ```
   SQL> exec dbms_macadm authorize_datapump_user ('PITER', 'HR');
   ```

   PL/SQL procedure successfully completed.
4. Query the `DVSYS.DBA_DV_DATAPUMP_AUTH` view to confirm that the user `piter` is authorized to perform export and import operations only on `HR` schema:

```
SQL> column grantee format A10
SQL> column schema format A15
SQL> column object format A15
SQL> SELECT * FROM DVSYS.DBA_DV_DATAPUMP_AUTH WHERE grantee = 'PITER';
```

<table>
<thead>
<tr>
<th>GRANTEE</th>
<th>SCHEMA</th>
<th>OBJECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>PITER</td>
<td>HR</td>
<td>%</td>
</tr>
</tbody>
</table>

Figure 2 – Authorized for all database object in schema HR

5. Export the `HR.EMPLOYEES` and `HR.DEPARTMENTS` tables:

```
$ expdp piter DIRECTORY=dp_dir DUMPFILE=exptables.dmp TABLES=hr.employees, hr.departments
```

```
Connected to: Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Product
With the Partitioning, Oracle Label Security, OLAP, Advanced Analytics,
Oracle Database Vault, Real Application Testing and Unified Auditing options
Starting "PITER"."SYS_EXPORT_TABLE_01": piter/******* DIRECTORY=dp_dir DUMPFILE=exptables.dmp TABLES=hr.employees hr.departments
ORA-39327: Oracle Database Vault data is being stored unencrypted in dump file set.
Estimate in progress using BLOCKS method...
Processing object type TABLE_EXPORT/TABLE/TABLE_DATA
Total estimation using BLOCKS method: 128 KB
Processing object type TABLE_EXPORT/TABLE/TABLE
Processing object type TABLE_EXPORT/TABLE/GRANT/OWNER_GRANT/OBJECT_GRANT
Processing object type TABLE_EXPORT/TABLE/COMMENT
Processing object type TABLE_EXPORT/TABLE/INDEX/INDEX
Processing object type TABLE_EXPORT/TABLE/CONSTRAINT/CONSTRAINT
Processing object type TABLE_EXPORT/TABLE/INDEX/STATISTICS/INDEX_STATISTICS
Processing object type TABLE_EXPORT/TABLE/CONSTRAINT/REF_CONSTRAINT
Processing object type TABLE_EXPORT/TABLE/TRIGGER
Processing object type TABLE_EXPORT/TABLE/STATISTICS/TABLE_STATISTICS
Processing object type TABLE_EXPORT/TABLE/STATISTICS/MARKER
. . . exported "HR"."DEPARTMENTS" 7.125 KB 27 rows
. . . exported "HR"."EMPLOYEES" 17.88 KB 107 rows
Master table "PITER"."SYS_EXPORT_TABLE_01" successfully loaded/unloaded
```

Figure 3 – The warning message
Additional Topics

6. Export \texttt{HR} schema in an unencrypted format:

\begin{verbatim}
$ expdp piter DIRECTORY=dp_dir DUMPFILE=expsh.dmp SCHEMAS=hr
   ENCRYPTION=NONE
\end{verbatim}

You'll receive the same message as in the previous step (ORA-39327), even though you explicitly stated that you don’t want to encrypt export. At the end of the job, you’ll see that it completed with one error (Figure 4) meaning that one warning:

\begin{verbatim}
Dump file set for PITER.SYS_EXPORT_SCHEMA_01 is:
   /u01/app/oracle/oradata/dp_exp/expsh.dmp
Job "PITER"."SYS_EXPORT_SCHEMA_01" completed with 1 error(s)
\end{verbatim}

Figure 4 – Successful export with warning

\section*{How it works…}

To be able to export data that is protected by Database Vault mechanisms, user has to be authorized (besides having appropriate privileges to perform Data Pump operations, for example, the \texttt{ROLE}). You can authorize a user to perform export and import operations:

- On specific database object in a schema, such as table. For example, it authorizes the user \texttt{amy} to export the table \texttt{HR.EMPLOYEES}:

\begin{verbatim}
SQL> EXEC DBMS_MACADM.AUTHORIZE_DATAPUMP_USER ('AMY', 'HR', 'EMPLOYEES');
\end{verbatim}

- On specific schema (you authorized the user \texttt{piter} to perform export and import operations on \texttt{HR} schema in step 3).

- For entire database. For example, it authorizes the user \texttt{kim} to export and import database object for the entire database:

\begin{verbatim}
SQL> EXEC DBMS_MACADM.AUTHORIZE_DATAPUMP_USER ('KIM');
SQL> grant DV_OWNER to k\texttt{im};
\end{verbatim}
There's more…

According to *Oracle Database Licensing Information, 12c Release 1 (12.1)*, if you want to encrypt Oracle Data Pump export, using its encryption features, Oracle Advanced Security option has to be enabled.

To encrypt export, specify appropriate value for ENCRYPTION parameter (instead of NONE which was shown in step 6 in the *How to do it* section). These are allowed values for the parameter:

```
ENCRYPTION=[ALL|DATA_ONLY|ENCRYPTED_COLUMNS_ONLY|METADATA_ONLY|NONE]
```

Also, before starting an export operation, make sure that the keystore is open.

See also

- A good reference to learn about Oracle Data Pump is official Oracle documentation – *Oracle Database Utilities, 12c Release 1 (12.1.0.2), Part I*.

Creating factors in Oracle Database Vault

In this recipe, you'll create three factors (*Day*, *Holiday*, and *NonWorkingDay*). The factor *Day* will return name of the day based on `sysdate`. The factor *Holiday* will return `TRUE` if it is a company nonworking holiday (for example, 1-JAN, 4-JUL, and 15-NOV) and `FALSE` otherwise. The factor *NonWorkingDay* will return whether it's a nonworking day (NO, WEEKEND, and COMPANY_HOLIDAY). We'll assume that a day is a nonworking day if it is a weekend or a company nonworking holiday (in case it is both weekend and holiday, it should resolve it to COMPANY_HOLIDAY).
Getting ready

It is assumed that:

- You are using Oracle Database 12.1.0.2 (the traditional architecture) on Linux and Oracle Enterprise Manager Cloud Control 12c
- Database Vault is enabled and configured (the Database Vault owner is the user `dbv_owner` and account manager is the user `dbv_acctmgr`).
- The user `dbv_owner` has been granted the `SELECT ANY DICTIONARY` privilege
- The user `piter` exists, the function `piter.get_function` has been created, and `DVSYS` has been granted the `EXECUTE` privilege on the function:

```sql
SQL> connect system
SQL> create or replace function piter.get_holiday
   2  return varchar2
   3  IS
   4  holiday varchar2(10);
   5  begin
   6  IF (RTRIM(TO_CHAR(SYSDATE, 'DD-MON')) IN ('1-JAN', '4-JUL',
   7   '15-NOV')) THEN
   8  holiday := 'TRUE';
   9  ELSE
  10  holiday := 'FALSE';
  11  END IF;
  12  RETURN holiday;
  13  end;
```

Function created.

```sql
SQL> grant execute on piter.get_holiday to dvsys;
Grant succeeded.
```

- Results in this recipe are shown for the situation that, at the same time, it is `SUNDAY` and it is a holiday.
How to do it…

1. Log in to EM12c as a SYSMAN or some other privileged user. Select your database. Then, from Security drop-down menu, choose Database Vault (Figure 5).

![Database Vault](image)

Figure 5

2. Log in as the dbv_owner user (Figure 6).

![Database Login](image)

Figure 6
3. Choose the **Administration** tab and click on the **Factors** link (Figure 7).

![Figure 7]

4. Click on the **Create** button to create your first custom factor (see Figure 8).

![Figure 8]
5. The name of the factor will be **Day**, the description will be **The name of day**, and **Factor Type** will be **Time** (shown in Figure 9). After you enter that information, click on the button **Next**.

![Figure 9](image)

6. Enter these configuration details for the factor and click on the button **Next**. It will appear as shown in this figure:

![Figure 10](image)
7. For Audit Options, choose Never. Leave other default values and click the button Next (see Figure 11).

![Create Factor: Options](image)

Figure 11

8. You won't create new identities at this moment, so just click on the button Next. After you finish reviewing the configuration, click on the Finish button. You should receive a confirmation message and see the newly created factor Day (result is shown in Figure 12).

![Confirmation](image)

Figure 12
9. Click on **Link Day** (in the column **Factor Name**, as shown in Figure 12). You will see that factor **Day** will get value **SUNDAY** (see figure 13). Click on the **OK** button.

![View Factor](image)

**Figure 13**

Now create the new factor **NonWorkingDay** (**Factor Type**: Time) which will, for the beginning, be based only on the factor **Day** and test it. After you create the factor **Holiday**, you'll edit the factor **NonWorkingDay** in such a way that it is based on both factors (**Day** and **Holiday**).

10. Repeat steps 4 and 5.

11. Enter these configuration details for the factor and click on the button **Next**:

<table>
<thead>
<tr>
<th>Factor Identification:</th>
<th>By Factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluation:</td>
<td>By Access</td>
</tr>
<tr>
<td>Factor Labeling:</td>
<td>By Self</td>
</tr>
</tbody>
</table>

12. Leave the default values and click on the **Next** button.
13. Click on the green plus button – **Add New Identity** (see Figure 14).

![Figure 14](image)

14. On the tab Identity, enter **Value** as **TRUE** and select **Untrusted** for **Trust Level** (see Figure 15).

![Figure 15](image)
15. Click on the **Map Identity** tab. Click on the green plus button – **Add Mapping** (see Figure 16).

![Add New Identity](image)

**Figure 16**

16. Select the following values and click on the **OK** button:

<table>
<thead>
<tr>
<th>Child Factor Name</th>
<th>Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operator</td>
<td>Like</td>
</tr>
<tr>
<td>Min Value</td>
<td>S%</td>
</tr>
</tbody>
</table>
17. You should see that identity is added (Figure 17).

18. Add the new identity \texttt{FALSE}. Repeat steps from 13 to 16 with appropriate values (for example, the value \texttt{FALSE}, \textbf{Trust Level} as \textbf{Somewhat trusted}; instead of the \textbf{Like} operator, choose \textbf{Not Like}).

19. Click on the \textbf{Next} button. Review the configuration and click on the \textbf{Finish} button. You should see confirmation message.
20. Click on the link NonWorkingDay (in the column Factor Name). You will see that the factor NonWorkingDay will get value TRUE (see Figure 18). Click on the OK button.

![View Factor]

Figure 18

21. On the Factors page (see Figure 12), in the table select row in which Day factor is displayed and click on the Edit button (pencil icon). Click on the Next button.

22. Change Retrieval Method to RTRIM(TO_CHAR(sysdate, 'DAY')) and click on the Done button.

23. Create the new factor Holiday (Factor Type: Time).
24. Enter these configuration details for the factor and click on the **Done** button:

<table>
<thead>
<tr>
<th>Factor Identification:</th>
<th>By Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluation:</td>
<td>By Access</td>
</tr>
<tr>
<td>Factor Labeling:</td>
<td>By Self</td>
</tr>
<tr>
<td>Retrieval Method:</td>
<td><code>PITER.GET_HOLIDAY</code></td>
</tr>
</tbody>
</table>

It will appear as shown in this figure:

![Create Factor: Configurations](image)

**Figure 19**

25. **Edit the factor NonWorkingDay so that it has three identities** (NO, WEEKEND, and COMPANY_HOLIDAY) **and click on OK.**
26. a. Edit the FALSE identity (change value to NO, add mapping in the Map Identity – Child Factor Name: Holiday, Operator: Like, Min Value: F%). Click on the OK button.

26. b. Edit the TRUE identity and click on the OK button (change value to WEEKEND, change mapping to have two rows:

<table>
<thead>
<tr>
<th>Child Factor Name</th>
<th>Operator</th>
<th>Min Value</th>
<th>Max Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day</td>
<td>Equal</td>
<td>SATURDAY</td>
<td></td>
</tr>
<tr>
<td>Day</td>
<td>Equal</td>
<td>SUNDAY</td>
<td></td>
</tr>
</tbody>
</table>

26. c. Add the new COMPANY_HOLIDAY identity (Trust Level: Untrusted). On the Map Identity tab, click on Add Mapping. Set the following values and click on OK:

<table>
<thead>
<tr>
<th>Child Factor Name</th>
<th>Operator</th>
<th>Min Value</th>
<th>Max Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Holiday</td>
<td>Equal</td>
<td>TRUE</td>
<td></td>
</tr>
</tbody>
</table>
This will appear as shown in this figure:

![Figure 21](image)

27. View evaluated value for factor Day (repeat step 9). The result is shown in Figure 22.

![Figure 22](image)
28. View evaluated value for factor *Holiday*. The result is shown in figure.

![Figure 23](image1)

29. View an evaluated value for factor *NonWorkingDay*. The result is shown in Figure 24.

![Figure 24](image2)
How it works…

The identity of a factor can be assigned by:

- Method
- Constant
- Factors

The process of assigning identity to a factor is named factor identification.

In this recipe, you created two factors (Day and Holiday) whose identities were assigned by methods and one factor (NonWorkingDay) whose identity was assigned by factors.

Factors can be evaluated when database session is created (By Session), each time factor is accessed (By Access), and when a database session starts (On Startup). Because you created factors that can change during a session, you chose evaluation by access.

Factor labeling is relevant for integration with Oracle Label Security.

In step 28, you verified that the factor NonWorkingDay got the value COMPANY_HOLIDAY (in case when it is SUNDAY and a holiday at the same time). That happened because factors that are based on other factors get actual value by evaluating identities in the order of their sorted ASCII identity values and first one that matches is assigned (evaluation stops). In our case, COMPANY_HOLIDAY was matched, so WEEKEND wasn't evaluated. In general, it is better to try to avoid overlapping conditions (if possible) because maintenance is easier and the risk of making a mistake is smaller.
There's more…

From Factors page in EM12c verify that you can't delete the factor Holiday because you use it to resolve identities for the factor NonWorkingDay (error ORA-47030: Factor Holiday is referred by one or more factor links).

1. Select the factor Holiday and click on the Delete button (see Figure 25).

   ![Figure 25](image)

2. Click on the button Yes (see Figure 26).

   ![Figure 26](image)
3. You will receive an error message (see Figure 27).

![Error]

When you create the factor PL/SQL, a function is created in schema `DVF` with the name `F$<factor_name>`.

**See also**


**Using TDE in a multitenant environment**

In this recipe, you will perform different operations using Transparent Data Encryption in a multitenant environment.
Getting ready

It is assumed that:

- You have two container databases (the multitenant architecture), version 12.1.0.2 in the same host.
- You have at least one pluggable database in each container database.
- You have sample schemes installed.

How to do it...

1. Enter the following text into your sqlnet.ora file located in a network/admin directory of your oracle home (for example, /u01/app/oracle/product/12.1.0/dbhome_1)

   ```
   ENCRYPTION_WALLET_LOCATION=
     (SOURCE=
       (METHOD=FILE)
       (METHOD_DATA=
         (DIRECTORY=/u01/app/oracle/admin/$ORACLE_SID/wallet)))
   ```

2. Change your environment to the first container database (for example, cdb1):

   ```
   [oracle@host01 ~]$ . oraenv
   ORACLE_SID = [oracle] ? cdb1
   ```

3. Connect as a user with the DBA role (for example, system), create a new user (for example, c##tdedba) to manage key management administration, and grant him appropriate privileges:

   ```
   $ sqlplus system
   SQL> create user c##tdedba identified by oracle123 container=all;
   SQL> grant administer key management to c##tdedba container=all;
   SQL> grant create session to c##tdedba container=all;
   SQL> grant select any dictionary to c##tdedba container=all;
   SQL> grant set container to c##tdedba container=all;
   ```
4. Connect as a user `c##tdedba` and create a keystore:

```sql
SQL> connect c##tdedba/oracle123
SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/u01/app/oracle/admin/cdb1/wallet' identified by oracle1;
```

5. See information about the previously created keystore and open it:

```sql
SQL> select wallet_type, wrl_type, status from v$encryption_wallet;

<table>
<thead>
<tr>
<th>WALLET_TYPE</th>
<th>WRL_TYPE</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNKNOWN</td>
<td>FILE</td>
<td>CLOSED</td>
</tr>
</tbody>
</table>

SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY oracle1;

SQL> select wallet_type, wrl_type, status from v$encryption_wallet;

<table>
<thead>
<tr>
<th>WALLET_TYPE</th>
<th>WRL_TYPE</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASSWORD</td>
<td>FILE</td>
<td>OPEN_NO_MASTER_KEY</td>
</tr>
</tbody>
</table>

SQL> select con_id, tag, key_id from v$encryption_keys;

no rows selected
```

6. Create a new master key for root container:

```sql
SQL> ADMINISTER KEY MANAGEMENT SET KEY USING TAG 'description: root key' IDENTIFIED BY oracle1 WITH BACKUP;

SQL> select con_id, tag, key_id from v$encryption_keys;

<table>
<thead>
<tr>
<th>CON_ID</th>
<th>TAG</th>
<th>KEY_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description: root key

SQL> select wallet_type, wrl_type, status from v$encryption_wallet;

<table>
<thead>
<tr>
<th>WALLET_TYPE</th>
<th>WRL_TYPE</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASSWORD</td>
<td>FILE</td>
<td>OPEN</td>
</tr>
</tbody>
</table>
```

```sql
```

Figure 28
7. Connect to a pluggable database (for example, `pdb11`) inside the first container database and check availability of a keystore:

```
SQL> alter session set container=pdb11;

SQL> select wallet_type, wrl_type, status from v$encryption_wallet;

<table>
<thead>
<tr>
<th>WALLET_TYPE</th>
<th>WRL_TYPE</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

8. Open a keystore, check availability of a master key, and create one:

```
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY oracle1;
SQL> select wallet_type, wrl_type, status from v$encryption_wallet;

<table>
<thead>
<tr>
<th>WALLET_TYPE</th>
<th>WRL_TYPE</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASSWORD</td>
<td>FILE</td>
<td>OPEN_NO_MASTER_KEY</td>
</tr>
</tbody>
</table>

SQL> select con_id, tag, key_id from v$encryption_keys;
no rows selected

SQL> ADMINISTER KEY MANAGEMENT SET KEY USING TAG 'description: pdb11 key' IDENTIFIED BY oracle1 WITH BACKUP;

SQL> select con_id, tag, key_id from v$encryption_keys;

<table>
<thead>
<tr>
<th>CON_ID</th>
<th>TAG</th>
<th>KEY_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>description: pdb11 key AeC4mqH5WUHmvxjEMBNk7lcAAAAAAAAAAAAAAAAAAAAAAAAAAAA</td>
<td></td>
</tr>
</tbody>
</table>

```

![Figure 29](image)

```
SQL> select wallet_type, wrl_type, status from v$encryption_wallet;

<table>
<thead>
<tr>
<th>WALLET_TYPE</th>
<th>WRL_TYPE</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASSWORD</td>
<td>FILE</td>
<td>OPEN</td>
</tr>
</tbody>
</table>
```
9. Change environment for the second container database (for example, \texttt{cdb2}):

```bash
[oracle@host01 ~]$ . oraenv
ORACLE_SID = [cdb1] ? cdb2
```

10. Connect as a user with the \texttt{sysdba} privileges, create a new user (for example, \texttt{c##tdedba}), and grant him appropriate privileges:

```bash
$ sqlplus / as sysdba
SQL> create user c##tdedba identified by oracle321 container=all;
SQL> grant syskm to c##tdedba container=all;
```

11. Connect as a user \texttt{c##tdedba} (as \texttt{syskm}), create a keystore, and open it for all pluggable databases:

```sql
SQL> ADMINISTER KEY MANAGEMENT CREATE KEYSTORE '/u01/app/oracle/admin/cdb2/wallet' identified by oracle2;
SQL> select wallet_type, wrl_type, status from v$encryption_wallet;
```

<table>
<thead>
<tr>
<th>WALLET_TYPE</th>
<th>WRL_TYPE</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNKNOWN</td>
<td>FILE</td>
<td>CLOSED</td>
</tr>
</tbody>
</table>

```sql
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY oracle2 container=all;
SQL> select wallet_type, wrl_type, status from v$encryption_wallet;
```

<table>
<thead>
<tr>
<th>WALLET_TYPE</th>
<th>WRL_TYPE</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASSWORD</td>
<td>FILE</td>
<td>OPEN_NO_MASTER_KEY</td>
</tr>
</tbody>
</table>

12. Create new master keys for all pdbs:

```sql
SQL> ADMINISTER KEY MANAGEMENT SET KEY USING TAG 'description: all pdbs' IDENTIFIED BY oracle2 WITH BACKUP container=all;
SQL> select con_id, tag, key_id from v$encryption_keys;
```

```
<table>
<thead>
<tr>
<th>CON_ID</th>
<th>TAG</th>
<th>KEY_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>description: all pdbs</td>
<td>AZ071jUQOU8cv/wRuWgzoBEAAAAAAAAAAAAAAAAAAAAAAAAAAA</td>
</tr>
<tr>
<td>0</td>
<td>description: all pdbs</td>
<td>AbwF8tkj0+ov9/HG430YiUAAAAAAAAAAAAAAAAAAAAAAAAAAA</td>
</tr>
</tbody>
</table>
```

Figure 30
13. Connect to a pluggable database as a SYS user and check keystore and masterkey:

```sql
SQL> connect / as sysdba
SQL> alter session set container=pdb21;
SQL> select wallet_type, wrl_type, status from v$encryption_wallet;

<table>
<thead>
<tr>
<th>WALLET_TYPE</th>
<th>WR_ TYPE</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASSWORD</td>
<td>FILE</td>
<td>OPEN</td>
</tr>
</tbody>
</table>

SQL> select con_id, tag, key_id from v$encryption_keys;

<table>
<thead>
<tr>
<th>CON_ID</th>
<th>TAG</th>
<th>KEY_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>description: all pdb</td>
<td>A2071JUQQU&amp;cv/wRulbgoBEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA</td>
</tr>
</tbody>
</table>
```

Figure 31

14. Change your environment to the first container database (for example, cdb1):

```bash
[oracle@host01 ~]$ . oraenv
ORACLE_SID = [cdb2] ? cdb1
```

15. Connect to the pluggable database as a user who has the DBA role (for example, c##zoran), create a test table with one encrypted column, and insert some data:

```sql
$ sqlplus c##zoran@pdb11
SQL> create table hr.enc_tbl(a int, b varchar2(20) encrypt);
SQL> insert into hr.enc_tbl values (1, 'value1');
SQL> insert into hr.enc_tbl values (2, 'value2');
SQL> commit;
SQL> select * from hr.enc_tbl;

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>value1</td>
</tr>
<tr>
<td>2</td>
<td>value2</td>
</tr>
</tbody>
</table>
```
16. Export a master key:

   SQL> ADMINISTER KEY MANAGEMENT EXPORT KEYS WITH SECRET "secret1"
   to '/home/oracle/keys.exp' IDENTIFIED BY oracle1;

17. Close the pluggable database pdb11 and unplug it:

   SQL> alter pluggable database pdb11 close immediate;
   SQL> alter pluggable database pdb11 unplug into
   '/home/oracle/pdb11.xml';
   SQL> drop pluggable database pdb11 keep datafiles;

18. Change your environment to the second container database (for example, cdb2):

   [oracle@host01 ~]$ . oraenv
   ORACLE_SID = [cdb1] ? cdb2

19. Connect to the second container database (for example, cdb2) as a sys user and
    plug the previously unplugged database (pdb11):

   $ sqlplus / as sysdba
   SQL> create pluggable database pdb11 using '/home/oracle/pdb11.xml';

20. Open the pluggable database:

   SQL> alter pluggable database pdb11 open;

   Warning: PDB altered with errors.

   SQL> show pdbs
21. Connect to pdb11, as a SYS user, open the keystore, and try to select from table with encrypted column:

```
SQL> alter session set container=pdb11;
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY oracle2;
SQL> select * from hr.enc_tbl;
```

```
ERROR at line 1:
ORA-28362: master key not found
```

22. Import the master key for this pluggable database and restart it:

```
SQL> ADMINISTER KEY MANAGEMENT IMPORT ENCRYPTION KEYS WITH SECRET "secret1" FROM '/home/oracle/keys.exp' IDENTIFIED BY oracle2 WITH BACKUP;
SQL> alter pluggable database pdb11 close immediate;
SQL> alter pluggable database pdb11 open;
```

```
SQL> show pdbs
```

```
CON_ID CON_NAME OPEN MODE RESTRICTED
---------- -------------- ----------- -----------
 2 PDB$SEED       READ ONLY    NO
 3 PDB21          READ WRITE   NO
 4 PDB11          READ WRITE   NO
```

23. Connect to the pluggable database (pdb11), open the keystore, and select from table with encrypted column:

```
SQL> alter session set container=pdb11;
SQL> ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY oracle2;
SQL> select * from hr.enc_tbl;
```

```
A          B
---------- --------------------
1           value1
2           value2
```
How it works…

In steps 1-6, the creation of keystore and master key in root container is shown. In step 7-8, the opening and creation of master key in the pluggable database is shown. There is only one keystore per entire container database, but that keystore contains multiple master keys (root container has its own master key, as well as every pluggable database in which transparent data encryption is used). In steps 9-13, another way of creation of the keystore and master key is shown (in the second container database). The user with the SYSKM system privilege is used, and opening of keystore as well as the creation of master keys are done by using container=all clause. This way, we are opening a keystore and creating master keys in all pluggable databases.

Because there is only one keystore per container database but multiple master keys, if database needs to be unplugged and plugged into another container database, a master key needs to be exported and imported into the target database also. In steps 16 and 17, we are exporting a master key and unplugging the database. In steps 18-20, we are plugging this database into another container database (cdb2). When we try to open the pluggable database in step 20, we get an error (because the master key is missing). The pluggable database is opened but in restricted mode. We can ignore this error for now and connect to that pluggable database as SYS user, but if we try to select from table that has encrypted columns, we get an error because the master key is missing. In step 22, we are importing a master key (that we exported in step 16). After importing a master key, we are restarting that pluggable database (now we can see that it can be opened without errors). And when we try to select from table that has encrypted columns, everything works perfectly.

See also

- Chapter 8, Transparent Data Encryption (in this book) and official Oracle documentation Oracle Advanced Security Guide.
Appendix – Application Contexts

In this chapter, we will cover the following tasks:

- Exploring and using built-in contexts
- Creating an application context
- Setting application context attributes
- Using an application context

Introduction

An application context is a memory container that holds a set of key-value pairs. You can think of an application context as an array of attributes where every attribute has a name (key) and value. Also, an application context is a namespace because in different application contexts, attributes that have the same name can exist (and there is no correlation between those attributes; they can store the same or different value).
To implement a local application context, you should complete steps shown in Figure 1 (the order of steps 1 and 2 is not important).

![Figure 1 – The steps to implement a local application context](image)

**Exploring and using built-in contexts**

The **USERENV** application context is a built-in context that contains information about the current session. In this recipe, you’ll learn to retrieve values from built-in contexts.

**Getting ready**

To complete this recipe, you’ll need an existing user who can get values from built-in namespaces by using the `SYS_CONTEXT` function (for example, user maja).
How to do it…

1. Connect to the database as a user who has appropriate privileges (for example, user maja):

   `$ sqlplus maja`

2. Find the name of host machine from which the client has connected to the database.

   ```
   SQL> select sys_context('USERENV','HOST') from dual;
   SYS_CONTEXT('USERENV','HOST')
   -------------------------------
   dbhost.orapassion.com
   ```

   Figure 2 – The name of the client host machine

3. Find the name of the user who logged on to the database.

   ```
   SQL> select sys_context('USERENV','SESSION_USER') from dual;
   SYS_CONTEXT('USERENV','SESSION_USER')
   ------------------------------------------
   MAJA
   ```

   Figure 3 – The name of the session user

4. Find the name of the program used for the database session.

   ```
   SQL> SELECT sys_context('USERENV', 'CLIENT_PROGRAM_NAME') FROM dual;
   SYS_CONTEXT('USERENV','CLIENT_PROGRAM_NAME')
   -----------------------------------------------
   sqlplus@dbhost.orapassion.com (TNS V1-V3)
   ```

   Figure 4 – The name of the client program
5. Find unified audit session ID.

```
SQL> select sys_context ('USERENV','UNIFIED_AUDIT_SESSIONID') from dual;
SYS_CONTEXT('USERENV','UNIFIED_AUDIT_SESSIONID')
-----------------------------------------------
2303011715
```

Figure 5 – A unified audit session ID

How it works…

In steps 2-5, you used the `SYS_CONTEXT` function to get values of several parameters from the `USERENV` context. You can use that function in both SQL and PL/SQL statements. It is expected that your results will differ from those shown in Figures 2-5, because they are system-specific.

The `UNIFIED_AUDIT_SESSIONID` attribute (parameter) is introduced in Oracle Database 12.1.0.2. The value of that parameter is `unified audit session ID` if the database uses unified auditing mode or mixed auditing mode, and `NULL` if the database uses traditional auditing (see Figure 6).

Note that in mixed auditing mode, the `UNIFIED_AUDIT_SESSIONID` value in the `USERENV` context is different from the `SESSIONID` value.

Figure 6 – The value of the `UNIFIED_AUDIT_SESSIONID`
There's more…

Another built-in context is SYS_SESSION_ROLES. You can use it to check whether a specified role is currently enabled for the session. For example, you'll create the test_role role, grant select privilege on hr.employees table, and grant the role to an existing user (for example, zoran). Afterwards, you'll verify that zoran has the test_role role by using the SYS_CONTEXT function. The example is shown in Figure 7.

```
SQL> create role test_role;
Role created.
SQL> grant select on hr.employees to test_role;
Grant succeeded.
SQL> grant test_role to zoran;
Grant succeeded.
SQL> select sys_context('SYS_SESSION_ROLES','TEST_ROLE') from dual;
SYS CONTEXT('SYS_SESSION_ROLES','TEST_ROLE')
-----------------------------------------------
FALSE
```

```
SQL> connect zoran
Enter password: Connected.
SQL> select sys_context('SYS_SESSION_ROLES','TEST_ROLE') from dual;
SYS CONTEXT('SYS_SESSION_ROLES','TEST_ROLE')
-----------------------------------------------
TRUE
```

When working in the multitenant environment, some useful attributes are CON_ID, CON_NAME, and CDB_NAME.

See also

- The full list of attributes that exist in the USERENV namespace is available in the official Oracle documentation—Oracle Database SQL Language Reference, Chapter 7, The SYS_CONTEXT function.
Creating an application context

In this recipe, you'll create a local application context (for example, sh_client). In the next recipes, you will use it to store clients' identifiers.

Getting ready

To complete this recipe, you'll need an existing user who can create an application context (it needs the CREATE ANY CONTEXT privilege or a DBA role), for example, the user maja.

How to do it…

1. Connect to the database as a user who has appropriate privileges (for example, user maja).

   $ sqlplus maja

2. Create a local application context (for example, sh_client).

   The PL/SQL package that will be used to set application context attributes doesn't have to exist at this time, but you have to specify its name.

   SQL> CREATE CONTEXT <context_name> USING <PL/SQL_package_name>;

   SQL> CREATE CONTEXT sh_client USING sh_ctx_pkg;

   Context created.

Figure 8 – Creating an application context
How it works…

In step 2, you created application context sh_client and defined that the PL/SQL package sh_ctx_pkg will be used to create and set application context attributes. At this moment, attributes aren’t set in the application context.

Context names must be unique within the database.

Setting application context attributes

In this recipe, you’ll create the PL/SQL package (for example, sh_ctx_pkg) that will set application context attributes for the application context you created in the previous recipe (for example, sh_client). Also, you’ll create a logon trigger.

Getting ready

To complete this recipe, you’ll need an existing user who can create sh_ctx_pkg. Make sure that the user has direct privileges on the sh.customers table (even if he/she has a DBA role) so that you don’t receive this message in SQL*Plus: Warning: Package Body created with compilation errors. or error Table or view doesn't exist in SQL Developer (for more information, see Chapter 3, PL/SQL Security).

How to do it…

1. Connect to the database as a user who has appropriate privileges (for example, user maja):

   $ sql pl us maja
2. Create the PL/SQL package that will set the `cust_id` attribute with the value, which is equal to the value of the `cust_id` column when the following statement is evaluated: `UPPER(cust_email) = (SYS_CONTEXT('USERENV', 'SESSION_USER') || '@COMPANY.EXAMPLE.COM')`. In case session user is not a customer, set the value for `cust_id` attribute in the application context to 0.

```sql
SQL> CREATE OR REPLACE PACKAGE sh_ctx_pkg IS
  2   PROCEDURE set_cust_id;
  3 END;
  4 /
Package created.

SQL> CREATE OR REPLACE PACKAGE BODY sh_ctx_pkg IS
  2   PROCEDURE set_cust_id
  3 IS
  4     v_cust_id NUMBER;
  5     BEGIN
  6       SELECT cust_id INTO v_cust_id FROM sh.customers
  7       WHERE UPPER(cust_email) = (SYS_CONTEXT('USERENV', 'SESSION_USER') || '@COMPANY.EXAMPLE.COM');
  8       DBMS_SESSION.SET_CONTEXT('sh_client','cust_id',v_cust_id);
  9     EXCEPTION
 10       WHEN no data found THEN
 11       DBMS_SESSION.SET_CONTEXT('sh_client','cust_id',0);
 12     END;
 13 END;
 14 /
Package body created.
```

Figure 9 – Creating a PL/SQL package

3. Create a logon trigger that calls the `sh_ctx_pkg.set_cust_id` procedure.

```sql
SQL> CREATE OR REPLACE TRIGGER sh_ctx_logon
  2   AFTER LOGON ON DATABASE
  3   BEGIN
  4     sh_ctx_pkg.set_cust_id();
  5   END;
  6 /
Trigger created.
```

Figure 10 – A logon trigger
How it works…
In step 3, you created a logon trigger so that every user who connects to the database is going to have an application context set. This step is optional because your application can set the application context by calling the same procedure.

There's more…
It is very important to note that if you try to set or change key-value pairs outside the package you specified when you created application context, you will receive the error insufficient privileges (see Figure 11).

```sql
SQL> exec DBMS_SESSION.SET_CONTEXT('sh_client','cust_id',101);
BEGIN DBMS_SESSION.SET_CONTEXT('sh_client','cust_id',101); END;

* ERROR at line 1:
ORA-01031: insufficient privileges
ORA-06512: at "SYS.DBMS_SESSION", line 122
ORA-06512: at line 1
```

Figure 11 – An error message

See also
- You can see Chapter 3, PL/SQL Security.

Using an application context
In this recipe, you’ll see one possible usage (in SQL) of the application contexts. Some other usages are shown in other parts of the book, and their references are given in the See also section of this recipe.
Getting ready

Create a new user (for example, sofia). Make sure that his or her e-mail in the format user@company.example.com is unique. Grant him or her privileges: create session and select on sh.customers table.

```
SQL> create user sofia identified by Q14be7WP;
User created.
SQL> grant create session to sofia;
Grant succeeded.
SQL> grant select on sh.customers to sofia;
Grant succeeded.
```

Figure 12 – New user

Insert data about him or her into the sh.customers table.

```
SQL> insert into sh.customers values (88000, 'Sofia', 'Smith', 'F', 1979, 'Married', 'Albert Embankment 10', 'SE1 7MD', 'London', 11111, 'England', '11111', 52790, '11111111', '12', 38, 'Sofia@company.example.com', 10000, 1, sysdate, sysdate, 'T');
1 row created.
SQL> commit;
```

Figure 13 – The new data in sh.customers

How to do it…

1. Connect to the database as a newly created user (for example, user sofia):

   
   ```bash
   $ sqlplus sofia
   ```
2. Verify that the user (for example, sofia) can access all data in the \texttt{sh.customers} table.

```sql
SQL> SELECT COUNT(*) FROM SH.CUSTOMERS;
COUNT(*)
----------
  55501
```

Figure 14 – The entire data in \texttt{sh.customers}

3. Verify that when executing the following statement, he or she (for example, sofia) can view only his or her data.

```sql
SQL> SELECT COUNT(*) FROM SH.CUSTOMERS
  2  WHERE cust_id = sys_context('sh_client','cust_id');
COUNT(*)
----------
    1
```

Figure 15 – Only data about newly created user

**How it works...**

In step 3, a simple way how an application can leverage application contexts in SQL statements was shown.

**See also**

- You can refer to Chapter 4, *Virtual Private Database*, Chapter 5, *Data Redaction*, and Chapter 10, *Unified Auditing*. 
Index

A
accessible by
  used, for restricting access to Program Units 86, 87, 88
application context attributes
  setting 349, 350, 351
application context
  about 343
  auditing 307, 308
  creating 348, 349
  implementing 344
  using 351, 352, 353
audit policies
  audited data, viewing 305, 306
  creating 301, 302, 303
  disabling 310, 311
  dropping 310, 311
  enabling 303, 304, 305
  information, finding 305, 306
audit trail
  purging 309, 310
authentication 7
auto-login keystore 235, 236

B
built-in context
  using 344, 345, 346, 347

C
code base access control
  about 72
  using 82, 83, 84, 85
  working 84
column encryption
  existing column, encrypting 235
  existing column, encrypting 233, 234
  new column, adding 227
  new table, creating 228, 229, 230
column-level VPD policy
  about 103
  creating 103, 104, 105
  working 106
combined analysis policy
  creating 193, 194
common privilege
  about 60
  creating 62
  granting 60, 61, 65
common role
  creating 54, 55
  creating, OEM 12c 57, 58
  granting 60, 61, 62, 65
common user
  creating 47, 48
  creating, OEM 12c used 49, 50, 51
  rules 49
container database (CDB) 45
context analysis policy
  creating 189, 190, 191, 192
context
  setting, as driving context 108, 109
cyphertexts 231

D
data dictionary seperation 46
data guard tasks 41
Data Pump operations
  auditing 298, 299
Data Redaction 168
data redaction policies
  users, exempting from 161, 162
data
  exporting in Oracle Database Vault environment,
Oracle Data Pump used 312, 313, 314, 315, 316
database administrators (dba) role 8
database analysis policy
  creating 183, 185
Database Configuration Assistant (DBCA) 245
database roles
  creating 30
  using 30
  working 31
Database Vault operations
  auditing 299, 300
Database Vault reports
  running 277, 278, 279
Database Vault
  about 242, 243
  disabling 280, 281
  for implementing, administrators cannot view
    data 272, 273, 274, 275, 276, 277
  re-enabling 281, 282, 283
  registering 243, 244, 245
DBMS_CRYPTO PL/SQL package 218
definer 71
definer's rights procedure
  about 71
  creating 72, 73
  using 73
  working 74
Discretionary access control (DAC) 89
driving context
  about 90
  creating 106, 107
E
EM Express
  for creating user 12, 13, 14, 15, 16
encrypted backups
  creating, RMAN used 240, 241
encryption 218
F
factor type 312
factors
  creating, in Oracle Database Vault 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334
function parameters
  changing, for specified column 150, 151, 152
H
Hardware Security Module (HSM) 221
I
immediate-write mode 290
invoker 71
invoker's right procedure
  about 71
  creating 74, 75, 76
  using 77, 78, 79, 80, 81, 82
  working 76
K
keystore location
  configuring, in sqlnet.ora 221, 222
keystore
  creating 222, 223, 224
  opening 222, 223, 224
L
local privilege
  about 65
  granting 66, 67
local role
  creating 58, 59
  creating, OEM 12c used 60
  granting 66, 67
local user
  creating 52
  creating, OEM 12c used 53, 54
  rules 53
loss of audit data
  checking 289, 290, 291
M
MAC
  about 232
  using 230, 231
mandatory realm 246
master encryption key
O
object privileges, usage
reporting on 207, 208

OEM 12c
used, for creating common user 49, 50, 51
used, for creating local role 60
used, for creating local user 53, 54
Oracle Data Masking 117
Oracle Data Redaction 116, 117
Oracle Database 12c 7
Oracle Database 12cR1 Standard Edition (SE) 289
Oracle Database Vault option 181
Oracle Database
multitenant architecture 47
traditional architecture 46
Oracle Enterprise Manager Cloud Control 12c
used, for managing redaction policies 140, 143, 144
Oracle Enterprise Manager Database Express 12c (EM Express) 10
Oracle multitenant environment 45
Oracle VPD row-level policies
about 99
creating 99, 100, 101, 102
Oracle Wallet 221
OS-authenticated users
creating 25, 26
using 25, 26

P
partial redaction 128
password profile
creating 8, 9, 10
password-authenticated users
creating 10, 11, 12
pluggable database (PDB) 45
policy function
about 90
creating 92, 93, 94, 96, 97
testing 98, 99
working 97

policy groups
creating 107, 108
policy
adding, to group 109, 110, 111, 112, 113, 114
predefined unified audit policy 288
privilege analysis
about 181
dropping 216
starting 196, 198, 199, 202, 204
stopping 196, 198, 199, 202, 204
privileges
effects, of plugging/unplugging operations 67, 68, 69
Program Units
access restricting, with accessible by 86, 87, 88
proxy authentication 28
proxy users
creating 27, 28
using 27, 28

Q
queued-write mode 290

R
random redaction type 133
realm 246
Recovery Manager (RMAN) backups 218
redaction policy
column, adding 152, 153, 154
creating, when using full redaction 119, 120, 121, 122, 123, 124
creating, when using partial redaction 128, 129, 130, 131, 132, 133
creating, when using random redaction 133, 134, 135, 136, 137
creating, when using regular expression
  redaction 137, 138, 139, 140
default value, changing 125, 126
disabling 154, 155, 156, 157, 158, 159, 160
dropping 154, 155, 156, 157, 158, 159, 160
enabling 154, 155, 156, 157, 158, 159, 160
managing, with Oracle Enterprise Manager Cloud Control 12c 140, 142, 144, 146, 147, 148, 149, 150
redaction
  types 118
regular expression redaction 137
rekeying 238, 239
RMAN operations
  auditing 295, 296, 297
  working 297
RMAN
  encryption modes 240
  used, for backup and recovery 240, 241
role analysis policy
  creating 187, 188
roles
  effects, of plugging/unplugging operations 67, 68, 69
  securing 256, 257, 258, 259, 260, 261
root container 45
rule set
  about 264
  creating 264, 265, 266
  using 267, 268
S
salt
  about 232
  using 230, 231
secure application role
  about 268
  behavior, testing 270, 271
  creating 268, 269
seed 45
sensitive columns
  determining 166, 167, 168
sensitive type
  creating 164, 165, 166
sysbackup privilege
  about 33
  instructions, for database authentication 34
  instructions, for OS authentication 35
  using 33
  working 35, 36, 37
sysdg privilege
  about 41
  instructions, for database authentication 42
  instructions, for OS authentication 42
  using 41
  working 43
syskm privilege
  about 38
  instructions, for database authentication 39
  instructions, for OS authentication 40
  using 39
  working 40
system privileges, usage
  reporting on 204, 205, 206
T
tablespace encryption 220
tablespace
  encrypting 236, 237
transparent data encryption (TDE) keystore 38
Transparent Data Encryption (TDE)
  about 218
  architecture 221
  column 219
tablespace 220
types 219
  using, in multitenant environment 334, 335, 336, 337, 338, 339, 340, 341, 342
Transparent Sensitive Data Protection (TSDP)
  about 163
  implementing 164
transparent sensitive data protection policy
  altering 177, 178, 179
  associating, with sensitive type 170, 171
  creating 168, 169
  disabling 171, 172, 173, 174, 175, 176
  dropping 171, 172, 173, 174, 175, 176
  enabling 171, 172, 173, 174, 175, 176
U
unified auditing
  about 284
  characteristics 284, 285
  enabling 286, 287, 288
  predefined unified audit policy 288, 289
unused object privileges
  reporting on 210, 211
unused privileges
  revoking 212, 213, 215, 216
unused system privileges
  reporting on 209
user account
  locking 21, 22, 23
user's password
  changing 16, 17, 18
  expiring 23, 24
users
  creating, EM Express used 12, 13, 14, 15, 16
  creating, with same credentials on another database 19, 20, 21
effects, of plugging/unplugging operations 67, 68, 69
exempting, from data redaction policies 161, 162
for creating audit policies 291, 292, 293, 294, 295
for viewing audit data 291, 292, 293, 294, 295
preventing, from executing specific command on specific object 262, 263
preventing, from exercising system privileges on schema objects 246, 248, 249, 250, 251, 252, 253, 254

V
Virtual Private Database (VPD) 89
VPD policies
  users, exempting from 114, 115