

Oracle Data Redaction

Oracle Advance Features

Complete Reference Guide

*Planning, Policy Types, Multitenant Scope, Auditing, and Rollout Guidance on
Oracle Database 19c*

*A complete reference to Oracle Data Redaction what it actually does
and does not protect against, how to plan a redaction program before
writing a single policy, how to build every policy type, how scope
works across a multitenant database, how to audit who changes a
policy, and the practical limitations every DBA should understand
before relying on it.*

Author	Muhammad Ilyas Awan <i>Oracle ACE Associate ♠ Oracle DBA Oracle EBS & Oracle APEX Techno-Functional Consultant ERP Solution Architect</i>
Mobile	+92-336-963-8284
Email	ilyas8284@outlook.com
LinkedIn	www.linkedin.com/in/ilyas8284
Address	Multan, Punjab, Pakistan (60000)
Platform	Oracle Database 19c, Oracle Advanced Security, DBMS_REDACT
Audience	Oracle DBAs and security teams protecting sensitive columns
In this guide	What redaction is and how it works • Planning a redaction program • Building every policy type • Managing, testing, and auditing policies • What redaction does not protect against • A practical build order and checklist

Part I — Understanding Data Redaction

1.1 What Is Data Redaction?

Oracle Data Redaction masks sensitive column values in a query's result set at the moment the query runs, without changing a single byte of the data actually stored in the table. The same row looks completely different to two sessions running the identical query, depending on whether a redaction policy applies to each session the data on disk never moves.

1.2 Redaction vs. Other Data Protection

- Transparent Data Encryption protects data at rest and in transit; it says nothing about what an authenticated session is allowed to see once connected redaction is the layer that governs that
- Virtual Private Database controls which rows a session can see at all; redaction controls what a visible row's column values look like, a different and complementary problem
- Data masking and subsetting permanently rewrites data, typically for a non-production copy; redaction changes nothing at rest and applies live, every time, against the one real copy.

1.3 The Four Redaction Types

- Full: Replaces the entire value with a fixed, type-appropriate placeholder: zero for numbers, a space for text, a base date for dates
- Partial: Masks only part of the value, such as all but the last four digits of a credit card number, leaving the rest visible
- Random: Returns a different, randomly generated value of the same data type on every execution, so no two query results ever match
- Regular expression: Matches a pattern within the value and redacts only the matched portion, useful for semi-structured text like an email address or a formatted identifier

The Four Redaction Types

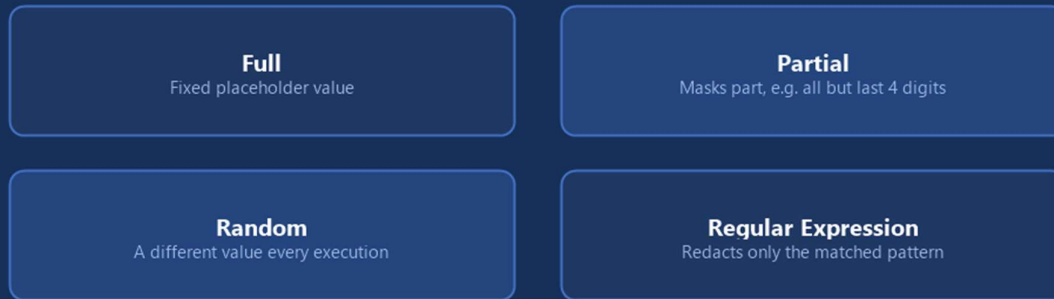


Figure 1 — Four ways to mask a value, from a fixed placeholder to a matched pattern

1.4 How Redaction Works at Query Time

Client Query → Redaction Policy Evaluated → Column Value Replaced → Result Set Returned



Figure 2 — The substitution happens last, right before the result leaves the database

The database executes the query normally against the real stored data, then substitutes the redacted value into the result set immediately before it leaves the database the substitution happens after the real value has already been read and evaluated internally, which is exactly why redaction is not a substitute for controlling who can query the table at all.

Nothing on disk ever changes: A redaction policy is entirely a display-time rule. Anyone who can read the data files directly, take a backup, or export the table sees the real, unredacted values every time.

Part II — Planning a Redaction Program

2.1 Discovering Sensitive Columns

Guessing which columns are sensitive from column names alone misses as much as it catches a column called NOTES or COMMENTS can carry a pasted-in credit card number just as easily as CREDIT_CARD_NO does. Oracle Data Safe's Sensitive Data Discovery scans column names, comments, and sampled data against a library of sensitive-type patterns and produces a candidate list a DBA can review, which is a considerably more thorough starting point than a manual pass through the data dictionary.



Figure 3 — Discovery comes before a single policy is written, not after

2.2 Choosing the Right Type for Each Column

The four types from Part I map onto different shapes of data. Full suits a value nobody outside a narrow group should ever see any part of, like a compensation figure. Partial fits an identifier where leaving a fragment visible is actually useful, like the last four digits of a card number a support agent needs to confirm against a caller. Random suits values where even a consistent masked pattern could leak information by letting someone infer which rows share a real value. Regular expression is the right tool for semi-structured free text, an email address or a formatted government ID embedded in a larger string, where only the matched portion should be hidden.

2.3 Designing Policy Expressions Around Roles, Not Individuals

A policy expression that names a specific username breaks the moment that person changes teams or leaves, and nobody remembers to update it. Build expressions around a database role via `SYS_CONTEXT('USERENV', 'SESSION_ROLES')`, or against an application-set client identifier, so access follows the role a person currently holds rather than a name hardcoded into the policy years earlier.

Redaction Tiers by Role



Figure 4 — Visibility should widen by role, deliberately, not by accident

2.4 Redaction in a Multitenant Environment

A redaction policy is local to the pluggable database its object lives in there is no CDB-wide switch that applies one policy across every PDB automatically. A common user connecting to two different PDBs is governed by whatever policy each PDB happens to define independently, which means a policy created in PDB1 has to be deliberately scripted into PDB2 as well if the same protection is meant to apply everywhere the same table structure exists.

Redaction Scope Across a Multitenant Database



Figure 5 — A policy in one PDB says nothing about the identical table one PDB over

Planning happens before the first ADD_POLICY call: Discovery, type selection, role-based expressions, and multitenant scope all belong in a design document a business owner signs off on, not decisions made ad hoc while writing PL/SQL.

Part III — Building a Redaction Policy

3.1 Prerequisites

Data Redaction is part of Oracle Advanced Security and needs the `EXECUTE` privilege on `DBMS_REDACT` to create policies. Confirm the option is available and check licensing before building anything, since Advanced Security is a separately licensed option on some editions.

3.2 A Full Redaction Policy

```
BEGIN  DBMS_REDACT.ADD_POLICY(      object_schema => 'HR',      object_name => 'EMPLOYEES',      column_name => 'SALARY',      policy_name => 'SALARY_REDACT',      function_type => DBMS_REDACT.FULL,      expression => '1=1'      ); END; /
```

The `expression` parameter controls when the policy applies; `1=1` means always, for every session with no exemption.

3.3 A Partial Redaction Policy

```
BEGIN  DBMS_REDACT.ADD_POLICY(      object_schema => 'SALES',      object_name => 'CUSTOMERS',      column_name => 'CREDIT_CARD_NO',      policy_name => 'CC_PARTIAL_REDACT',      function_type => DBMS_REDACT.PARTIAL,      function_parameters => 'VVVVVVVVVVV1111,VVVVVVVVVVVVFFF,x,13,16',      expression => 'SYS_CONTEXT(''USERENV'', ''SESSION_USER'') != ''BILLING_ADMIN'''      ); END; /
```

Every session except `BILLING_ADMIN` sees only the last four digits of the card number; everyone else sees the rest of the digits replaced with the masking character.

3.4 Random and Regular Expression Redaction

```
BEGIN  DBMS_REDACT.ADD_POLICY(      object_schema => 'HR',      object_name => 'EMPLOYEES',      column_name => 'PERFORMANCE_SCORE',      policy_name => 'SCORE_RANDOM_REDACT',      function_type => DBMS_REDACT.RANDOM,      expression => '1=1'      );  DBMS_REDACT.ADD_POLICY(      object_schema => 'HR',      object_name => 'EMPLOYEES',      column_name => 'EMAIL',      policy_name => 'EMAIL_REGEXP_REDACT',      function_type => DBMS_REDACT.REGEXP,      regexp_pattern => DBMS_REDACT.RE_PATTERN_EMAIL_ADDRESS,      regexp_replace_string => DBMS_REDACT.RE_REDACT_EMAIL_ADDRESS,      expression => '1=1'      ); END; /
```

3.5 Writing the Policy Expression

The expression is ordinary SQL, evaluated per session, and almost always built around `SYS_CONTEXT` the session user, an application context value set by your own logon trigger, or a client identifier passed up from the middle tier. Whatever the expression evaluates to for a given session decides whether that session gets the real value or the redacted one, so the expression is really the entire access-control decision in one line, which is exactly what section 2.3 is about designing deliberately.

Part IV — Managing, Testing & Auditing Policies

4.1 Altering a Policy

```
BEGIN  DBMS_REDACT.ALTER_POLICY(      object_schema      => 'HR',      object_name  
=> 'EMPLOYEES',      policy_name      => 'SALARY_REDACT',      action  
=> DBMS_REDACT.MODIFY_EXPRESSION,      expression      =>  
'SYS_CONTEXT(''USERENV'', ''SESSION_USER'') != ''HR_MANAGER''' ); END; /
```

4.2 Enabling, Disabling, and Dropping

```
EXEC DBMS_REDACT.DISABLE_POLICY('HR', 'EMPLOYEES', 'SALARY_REDACT'); EXEC  
DBMS_REDACT.ENABLE_POLICY('HR', 'EMPLOYEES', 'SALARY_REDACT'); EXEC  
DBMS_REDACT.DROP_POLICY('HR', 'EMPLOYEES', 'SALARY_REDACT');
```

4.3 Testing Redaction Behavior

Test as more than one session, deliberately: connect as an account the policy expression should redact for, and separately as one it should not, and compare the actual result sets rather than trusting the policy definition alone. A policy expression with a typo can silently redact for nobody, or everybody, and the difference only shows up by actually querying the data both ways.

4.4 The EXEMPT_REDACTION_POLICY Privilege

Any account granted **EXEMPT_REDACTION_POLICY** bypasses every redaction policy in the database entirely, regardless of what any individual policy's expression says. Grant it as narrowly as any other high-value privilege it is a single point that overrides every policy built in this guide at once.

4.5 Viewing Existing Policies

```
SELECT object_owner, object_name, policy_name, enable FROM redaction_policies;  
SELECT object_owner, object_name, column_name, policy_name, function_type FROM  
redaction_columns;
```

4.6 Auditing Policy Changes

A policy quietly disabled during troubleshooting and never re-enabled is one of the most common ways redaction protection actually lapses, and it is invisible unless someone is specifically looking for it. A unified audit policy on **DBMS_REDACT** closes that gap by recording every ADD_POLICY, ALTER_POLICY, DISABLE_POLICY, and DROP_POLICY call, who issued it, and when:

```
CREATE AUDIT POLICY redact_policy_changes ACTIONS EXECUTE ON DBMS_REDACT; AUDIT  
POLICY redact_policy_changes; SELECT event_timestamp, dbusername, action_name,  
unified_audit_policies FROM unified_audit_trail WHERE unified_audit_policies =  
'REDACT_POLICY_CHANGES' ORDER BY event_timestamp DESC;
```

Auditing Policy Changes

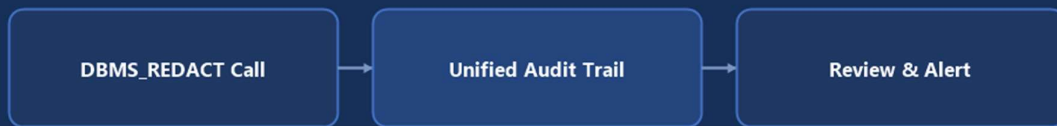


Figure 6 — A disabled policy should be a recorded event, not a silent one

Test both sides, every time: A redaction policy that has only ever been checked from the account that should see everything has only been half tested. Confirm the redacted view looks right too, from an account the policy is actually meant to restrict.

Part V — Practical Limitations

5.1 Redaction Is Not Encryption

This is the single most important thing to internalize about this feature. The real value is stored, unencrypted with respect to redaction, exactly as it always was. Redaction only ever changes what a SQL query's result set looks like when it leaves the database — it does nothing to protect the data at the storage layer, and nothing here reduces the need for Transparent Data Encryption if the underlying data files themselves need protecting.

5.2 What Redaction Does Not Protect Against

- A WHERE clause still filters on the real, unredacted value a session can search for a specific real credit card number and confirm a match exists, even while the displayed digits stay masked
- Indexes are still built on the real values, and query plans, statistics, and error messages can all leak information about data a policy is meant to hide
- Exports, backups, and any direct access to the data files bypass redaction completely, since it is purely a SQL-result-time behavior, not a property of the stored data itself

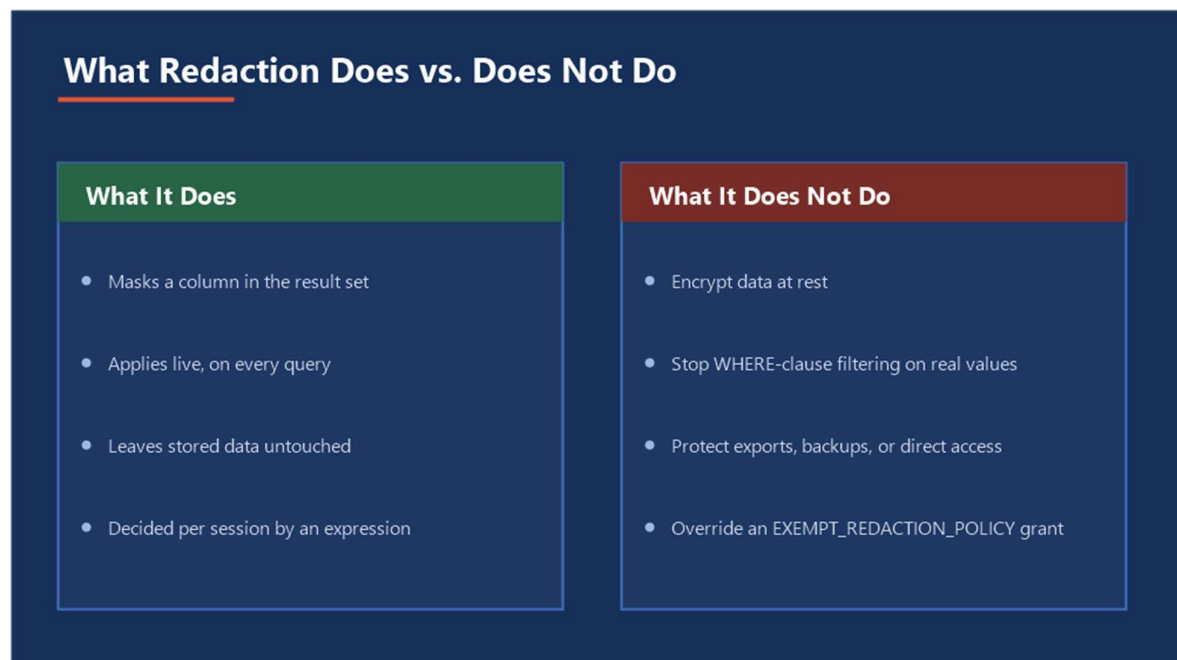


Figure 7 — The boundary every redaction decision has to respect

5.3 Performance Considerations

A redaction policy adds a per-row, per-column evaluation cost to every query that touches a redacted column, generally small but not zero. Regular expression redaction is the most expensive of the four types, since it evaluates a pattern match rather than a simple

substitution, and is worth benchmarking specifically on high-volume queries before rolling it out broadly.

5.4 Common Bypass Risks

The realistic ways redaction gets bypassed are rarely exotic: an over-broad EXEMPT_REDACTION_POLICY grant, a reporting tool connecting through a shared service account that happens to be exempt, or a data pump export scheduled by someone who assumed redaction would apply to it. None of these are flaws in the feature they are exactly the boundaries described in this Part, encountered by someone who had not read this far yet.

5.5 Redaction and Application Design

A connection pool that hands out the same physical session to many logical users defeats a per-session expression unless the application resets the client identifier between users. Without that reset, the redaction decision follows whichever logical user happened to set the context last, not the one actually running the current request. Result caching, whether at the client, the middle tier, or the database's own result cache, carries the same risk: a cached result computed for one session's redaction context can be served to a different session with a different context if the cache key does not account for it.

A shared connection is a shared redaction context: If the application layer doesn't reset the session's identifying context between logical users, redaction quietly stops following the person and starts following whichever session happened to be handed to them.

Part VI — Practical Guidance

6.1 A Realistic Build Order

Discover & Classify → Choose a Redaction Type → Write the Policy Expression → Test as Two Different Sessions → Review EXEMPT Grants



Figure 8 — The last step is not optional — it is the one that closes the biggest hole

6.2 Integrating with Oracle Data Safe

Beyond the discovery step in section 2.1, Data Safe's Security Assessment reviews the database's overall configuration and flags exactly the kind of drift this guide warns about an unusually broad EXEMPT_REDACTION_POLICY grant, or a sensitive column with no policy at all despite being flagged by discovery. Feeding Data Safe's findings directly into the checklist below turns a one-time build exercise into something that gets re-verified on a schedule.

6.3 Common Pitfalls

- Treating redaction as encryption, and skipping TDE on the assumption the data is already protected
- Granting EXEMPT_REDACTION_POLICY to a shared or application service account far more broadly than any single reporting job actually needs
- Never testing the policy from the restricted side, so a broken expression goes unnoticed until an audit finds it
- Forgetting that a policy has to be created in every PDB that holds the table, not just the one it was first built in
- Forgetting that exports and backups carry the real data, and treating a redacted production database as if that makes its backups safe to handle casually
- Rolling out regular expression redaction broadly without benchmarking its cost on the actual production query volume first

6.4 A Redaction Policy Checklist

1. Sensitive columns were identified through structured discovery, not guessed from column names
2. Every genuinely sensitive column has an owner who signed off on which redaction type applies
3. Policy expressions are built around roles or application context, not hardcoded individual usernames
4. Every policy has been deployed to every PDB that holds the relevant table, not just the first one
5. Every policy has been tested from both an exempt and a non-exempt session
6. EXEMPT_REDACTION_POLICY is granted to specific, documented accounts only, never broadly
7. A unified audit policy is watching every ADD_POLICY, ALTER_POLICY, DISABLE_POLICY, and DROP_POLICY call
8. TDE, backup handling, and export procedures are governed independently, not assumed to be covered by redaction
9. redaction_policies and redaction_columns have been reviewed and match what is actually documented as in place

6.5 Where This Fits in the Series

Redaction policies and EXEMPT_REDACTION_POLICY grants belong directly in the security review section of the Database Health Check guide already in this series an account that quietly picked up the exempt privilege, or a policy someone disabled during troubleshooting and never re-enabled, are exactly the kind of drift a periodic check, and now the unified audit policy from section 4.6, exist to catch.

The Most Important Point

Data Redaction changes what a query shows, not what the database stores. Use it to control what appears on a screen or in a report, and keep relying on encryption, access control, careful handling of backups and exports, and a planning process that starts with discovery for everything redaction was never designed to touch.