28

Overview of SQL Plan Management

SQL plan management is a preventative mechanism that enables the optimizer to automatically manage execution plans, ensuring that the database uses only known or verified plans.

About SQL Plan Baselines

SQL plan management uses a mechanism called a **SQL plan baseline**, which is a set of accepted plans that the optimizer is allowed to use for a SQL statement.

In this context, a plan includes all plan-related information (for example, SQL plan identifier, set of hints, bind values, and optimizer environment) that the optimizer needs to reproduce an execution plan. The baseline is implemented as a set of plan rows and the outlines required to reproduce the plan. An outline is a set of optimizer hints used to force a specific plan.

The main components of SQL plan management are as follows:

- Plan capture

  This component stores relevant information about plans for a set of SQL statements.
- Plan selection

  This component is the detection by the optimizer of plan changes based on stored plan history, and the use of SQL plan baselines to select appropriate plans to avoid potential performance regressions.
- Plan evolution

  This component is the process of adding new plans to existing SQL plan baselines, either manually or automatically. In the typical use case, the database accepts a plan into the plan baseline only after verifying that the plan performs well.

Purpose of SQL Plan Management

SQL plan management prevents performance regressions caused by plan changes.

A secondary goal is to gracefully adapt to changes such as new optimizer statistics or indexes by verifying and accepting only plan changes that improve performance.

> **Note:**
>
> SQL plan baselines cannot help when an event has caused irreversible execution plan changes, such as dropping an index.

Benefits of SQL Plan Management

SQL plan management can improve or preserve SQL performance in database upgrades and system and data changes.

Specifically, benefits include:

- A database upgrade that installs a new optimizer version usually results in plan changes for a small percentage of SQL statements.

  Most plan changes result in either improvement or no performance change. However, some plan changes may cause performance regressions. SQL plan baselines significantly minimize potential regressions resulting from an upgrade.

  When you upgrade, the database only uses plans from the plan baseline. The database puts new plans that are not in the current baseline into a holding area, and later evaluates them to determine whether they use fewer resources than the current plan in the baseline. If the plans perform better, then the database promotes them into the baseline; otherwise, the database does not promote them.
- Ongoing system and data changes can affect plans for some SQL statements, potentially causing performance regressions.

  SQL plan baselines help minimize performance regressions and stabilize SQL performance.
- Deployment of new application modules introduces new SQL statements into the database.

  The application software may use appropriate SQL execution plans developed in a standard test configuration for the new statements. If the system configuration is significantly different from the test configuration, then the database can evolve SQL plan baselines over time to produce better performance.

> **See Also:**
>
> *Oracle Database Upgrade Guide* to learn how to upgrade an Oracle database

Differences Between SQL Plan Baselines and SQL Profiles

Both SQL profiles and SQL plan baselines help improve the performance of SQL statements by ensuring that the optimizer uses only optimal plans.

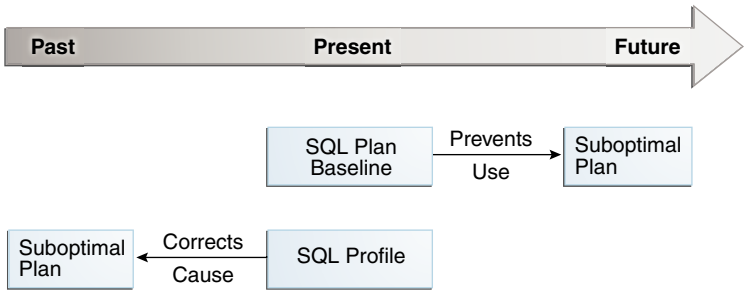
Both profiles and baselines are internally implemented using hints. However, these mechanisms have significant differences, including the following:

- In general, SQL plan baselines are proactive, whereas SQL profiles are reactive.

  Typically, you create SQL plan baselines *before* significant performance problems occur. SQL plan baselines prevent the optimizer from using suboptimal plans in the future.

  The database creates SQL profiles when you invoke SQL Tuning Advisor, which you do typically only *after* a SQL statement has shown high-load symptoms. SQL profiles are primarily useful by providing the ongoing resolution of optimizer mistakes that have led to suboptimal plans. Because the SQL profile mechanism is reactive, it cannot guarantee stable performance as drastic database changes occur.

Figure 28-1 SQL Plan Baselines and SQL Profiles



- SQL plan baselines reproduce a specific plan, whereas SQL profiles correct optimizer cost estimates.

  A SQL plan baseline is a set of accepted plans. Each plan is implemented using a set of outline hints that fully specify a particular plan. SQL profiles are also implemented using hints, but these hints do not specify any specific plan. Rather, the hints correct miscalculations in the optimizer estimates that lead to suboptimal plans. For example, a hint may correct the cardinality estimate of a table.

  Because a profile does not constrain the optimizer to any one plan, a SQL profile is more flexible than a SQL plan baseline. For example, changes in initialization parameters and optimizer statistics enable the optimizer to choose a better plan.

Oracle recommends that you use SQL Tuning Advisor. In this way, you follow the recommendations made by the advisor for SQL profiles and plan baselines rather than trying to determine which mechanism is best for each SQL statement.

> **See Also:**
>
> - "About Optimizer Hints"
> - "Managing SQL Profiles"
> - "Analyzing SQL with SQL Tuning Advisor"

Plan Capture

SQL plan capture refers to techniques for capturing and storing relevant information about plans in the SQL Management Base for a set of SQL statements.

Capturing a plan means making SQL plan management aware of this plan. You can configure initial plan capture to occur automatically by setting an initialization parameter, or you can capture plans manually by using the `DBMS_SPM` package.

Automatic Initial Plan Capture

When enabled, the database checks whether executed SQL statements are eligible for automatic capture.

You can enable **automatic initial plan capture** by setting `OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES` to `true` (the default is `false`). Note that the initialization parameter `OPTIMIZER_USE_SQL_PLAN_BASELINES` is independent. For example, if

OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES is true, then the database creates initial plan baselines regardless of whether OPTIMIZER_USE_SQL_PLAN_BASELINES is true or false.

> **See Also:**
>
> - "Plan Selection"
> - *Oracle Database Reference* to learn about the OPTIMIZER_USE_SQL_PLAN_BASELINES initialization parameter

Eligibility for Automatic Initial Plan Capture

To be eligible for automatic plan capture, an executed statement must be repeatable, and it must not be excluded by any capture filters.

By default, the database considers all repeatable SQL statements as eligible for capture, with the following exceptions:

- CREATE TABLE when the AS SELECT clause is *not* specified
- DROP TABLE
- INSERT INTO ... VALUES

The first check for eligibility is repeated execution. If a statement is executed less than twice, then the database does not consider it eligible for a SQL plan baseline. If a statement is executed at least twice, then it is by definition repeatable, and so the database considers it eligible for further checking.

> **Note:**
>
> SQL plan management does not protect statements that have been explained using EXPLAIN PLAN but have not been executed.

For repeatable statements, the DBMS_SPM.CONFIGURE procedure enables you to create an automatic capture filter. Thus, you can capture only statements that you want, and exclude noncritical statements, thereby saving space in the SYSAUX tablespace. Noncritical queries often have the following characteristics:

- Not executed often enough to be significant
- Not resource-intensive
- Not sufficiently complex to benefit from SQL plan management

For a specified parameter, a filter either includes (allow=>TRUE) or excludes (allow=>FALSE) plans for statements with the specified values. To be eligible for capture, a repeatable statement must not be *excluded* by any filter. The DBMS_SPM.CONFIGURE procedure supports filters for SQL text, parsing schema name, module, and action.

A null value for any parameter removes the filter. By using parameter_value=>'%' in combination with allow=FALSE, you can filter out all values for a parameter, and then create a separate filter to include only specified values. The DBA_SQL_MANAGEMENT_CONFIG view shows the current filters.

> **See Also:**
>
> - "Configuring Filters for Automatic Plan Capture"
> - *Oracle Database PL/SQL Packages and Types Reference* to learn more about the `DBMS_SPM.CONFIGURE` procedure
> - *Oracle Database Reference* to learn more about the `DBA_SQL_MANAGEMENT_CONFIG` view

Plan Matching for Automatic Initial Plan Capture

If the database executes a repeatable SQL statement, and if this statement passes through the `DBMS_SPM.CONFIGURE` filters, then the database attempts to match a plan in the SQL plan baseline.

For automatic initial plan capture, the plan matching algorithm is as follows:

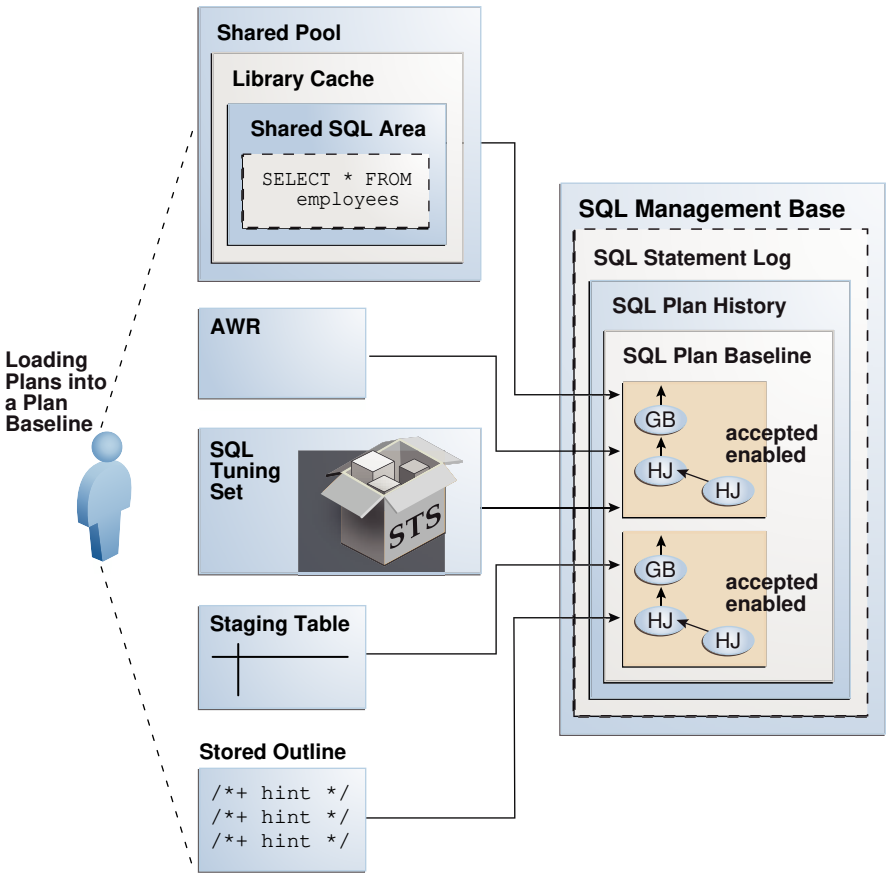
- If a SQL plan baseline does *not* exist, then the optimizer creates a plan history and SQL plan baseline for the statement, marking the initial plan for the statement as accepted and adding it to the SQL plan baseline.
- If a SQL plan baseline exists, then the optimizer behavior depends on the cost-based plan derived at parse time:
  - If this plan does *not* match a plan in the SQL plan baseline, then the optimizer marks the new plan as unaccepted and adds it to the SQL plan baseline.
  - If this plan *does* match a plan in the SQL plan baseline, then nothing is added to the SQL plan baseline.

Manual Plan Capture

In SQL plan management, **manual plan capture** refers to the user-initiated bulk load of existing plans into a SQL plan baseline.

Use Cloud Control or PL/SQL to load the execution plans for SQL statements from AWR, a SQL tuning set (STS), the shared SQL area, a staging table, or a stored outline.

Figure 28-2 Loading Plans into a SQL Plan Baseline



The loading behavior varies depending on whether a SQL plan baseline exists for each statement represented in the bulk load:

- If a baseline for the statement does not exist, then the database does the following:
  1. Creates a plan history and plan baseline for the statement
  2. Marks the initial plan for the statement as accepted
  3. Adds the plan to the new baseline
- If a baseline for the statement exists, then the database does the following:
  1. Marks the loaded plan as accepted
  2. Adds the plan to the plan baseline for the statement *without* verifying the plan's performance

Manually loaded plans are always marked accepted because the optimizer assumes that any plan loaded manually by the administrator has acceptable performance. You can load plans without enabling them by setting the `enabled` parameter to `NO` in the `DBMS_SPM.LOAD_PLANS_FROM_%` functions.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* to learn more about the `DBMS_SPM.LOAD_PLANS_FROM_%` functions

Plan Selection

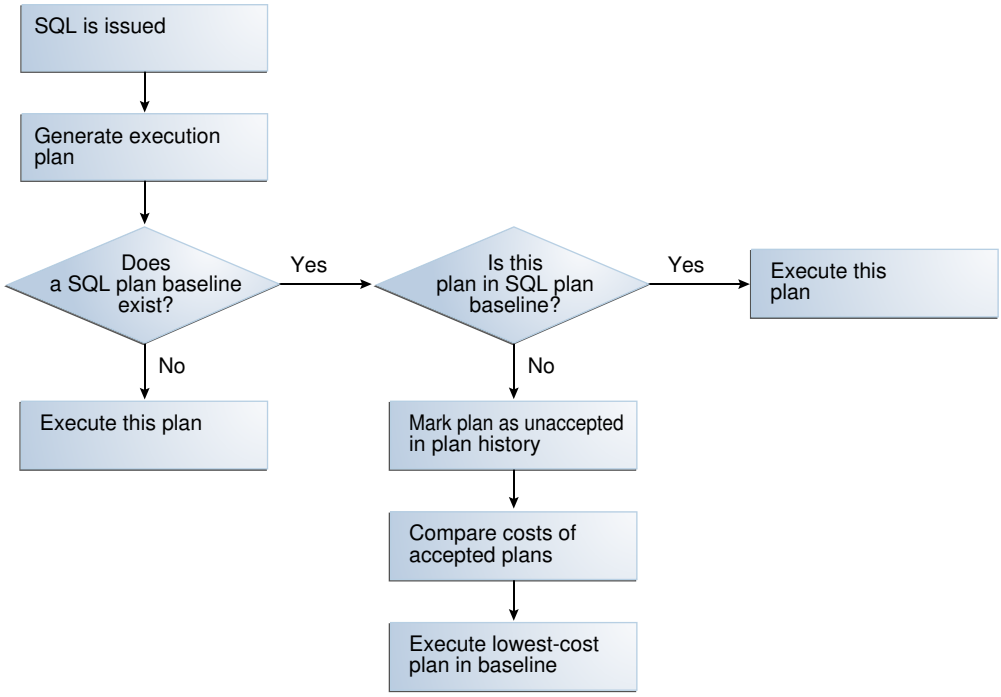
SQL plan selection is the optimizer ability to detect plan changes based on stored plan history, and the use of SQL plan baselines to select plans to avoid potential performance regressions.

When the database performs a hard parse of a SQL statement, the optimizer generates a best-cost plan. By default, the optimizer then attempts to find a matching plan in the SQL plan baseline for the statement. If no plan baseline exists, then the database runs the statement with the best-cost plan.

If a plan baseline exists, then the optimizer behavior depends on whether the newly generated plan is in the plan baseline:

- If the new plan is in the baseline, then the database executes the statement using the found plan.
- If the new plan is *not* in the baseline, then the optimizer marks the newly generated plan as unaccepted and adds it to the plan history. Optimizer behavior depends on the contents of the plan baseline:
  - If fixed plans exist in the plan baseline, then the optimizer uses the fixed plan with the lowest cost.
  - If no fixed plans exist in the plan baseline, then the optimizer uses the baseline plan with the lowest cost.
  - If no reproducible plans exist in the plan baseline, which could happen if every plan in the baseline referred to a dropped index, then the optimizer uses the newly generated cost-based plan.

Figure 28-3 Decision Tree for SQL Plan Selection



> **See Also:**
>
> "Fixed Plans"

Plan Evolution

In general, SQL **plan evolution** is the process by which the optimizer verifies new plans and adds them to an existing SQL plan baseline.

Purpose of Plan Evolution

Typically, a SQL plan baseline for a statement starts with one accepted plan.

However, some SQL statements perform well when executed with different plans under different conditions. For example, a SQL statement with bind variables whose values result in different selectivities may have several optimal plans. Creating a materialized view or an index or repartitioning a table may make current plans more expensive than other plans.

If new plans were never added to SQL plan baselines, then the performance of some SQL statements might degrade. Thus, it is sometimes necessary to evolve newly accepted plans into SQL plan baselines. Plan evolution prevents performance regressions by verifying the performance of a new plan before including it in a SQL plan baseline.

How Plan Evolution Works

Plan evolution involves both verifying and adding plans.

Specifically, plan evolution consists of the following distinct steps:

1. Verifying

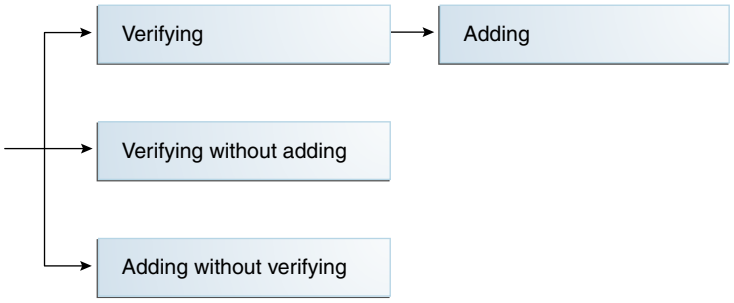
   The optimizer ensures that unaccepted plans perform at least as well as accepted plans in a SQL plan baseline (known as plan verification).

2. Adding

   After the database has proved that unaccepted plans perform as well as accepted plans, the database adds the plans to the baseline.

In the standard case of plan evolution, the optimizer performs the preceding steps sequentially, so that a new plan is not usable by SQL plan management until the optimizer verifies plan performance relative to the SQL plan baseline. However, you can configure SQL plan management to perform one step without performing the other. The following graphic shows the possible paths for plan evolution.

Figure 28-4 Plan Evolution



PL/SQL Subprograms for Plan Evolution

The `DBMS_SPM` package provides procedures and functions for plan evolution.

These subprograms use the task infrastructure. For example, `CREATE_EVOLVE_TASK` creates an evolution task, whereas `EXECUTE_EVOLVE_TASK` executes it. All task evolution subprograms have the string `EVOLVE_TASK` in the name.

Use the evolve procedures on demand, or configure the subprograms to run automatically. The automatic maintenance task `SYS_AUTO_SPM_EVOLVE_TASK` executes daily in the scheduled maintenance window. The task perform the following actions automatically:

1. Selects and ranks unaccepted plans for verification
2. Accepts each plan if it satisfies the performance threshold

> **See Also:**
>
> - "Managing the SPM Evolve Advisor Task"
> - "Evolving SQL Plan Baselines Manually"
> - *Oracle Database PL/SQL Packages and Types Reference* to learn about the `DBMS_SPM` package

About Automatic SQL Plan Management

Automatic SQL plan management provides an automated end-to-end solution to detect and correct SQL statement performance regressions.

Automatic SQL plan management detects SQL execution plan changes and, if a previous plan exists, it uses execution statistics to evaluate whether a new plan performs poorly compared to an earlier plan. If that is the case, a SQL plan baseline is created to enforce the earlier plan and prevent subsequent performance regressions.

Use Cases for Automatic SQL Plan Management

The following is a typical scenario where automatic SQL plan management can provide benefit.

SQL statements have been performing well, but there is a sudden change of execution plan for a statement. This causes a severe performance regression.

- Without automatic SQL plan management:
  The new plan may be suboptimal and in some cases poor performance may seriously impact the application service level.
- With automatic SQL plan management
  The database is able to rapidly recovery from the suboptimal plan and application service level is restored. This occurs transparently, without manual intervention.

SQL Plan Management (With Background Verification)

SQL plan management (with background-verification) uses a high-frequency background task to identify SQL statements consuming significant system resources (by inspecting the automatic workload repository and the automatic SQL tuning set. Historic SQL performance data is used to establish whether there has been a likely performance regression. Alternate SQL execution plans are located automatically and test executed (using the SPM evolve advisor). Where necessary, the best plans are enforced using SQL plan baselines without manual intervention.

About Real-Time SQL Plan Management

Real-time SQL plan management works in the foreground to immediately detect and transparently repair SQL performance regressions.

Automatic SQL plan management proactively prevents SQL performance regressions in real-time. Execution plan changes are immediately detected and the performance of statements affected by these changes are evaluated against earlier execution plans. SQL plan baselines are used to select the best-performance known plans if necessary. This process works transparently and without manual intervention to improve application service levels and reduce service outages caused by SQL performance regressions

Real-time SQL plan management is similar to SQL plan management (with background-verification), except that performance is evaluated in the foreground by the database session executing the SQL statement. This reduces the amount of time to detect and repair the performance regression. At the end of SQL execution, the performance of the new plan is compared with the performance of a previous lowest-cost plan. The new plan is accepted or

rejected based on its performance compared to previously known plans. SQL plan baselines are created to control which plan is accepted.

Storage Architecture for SQL Plan Management

The SQL plan management infrastructure records the signatures of parsed statements, and both accepted and unaccepted plans.

SQL Management Base

The **SQL management base (SMB)** is a logical repository in the data dictionary.

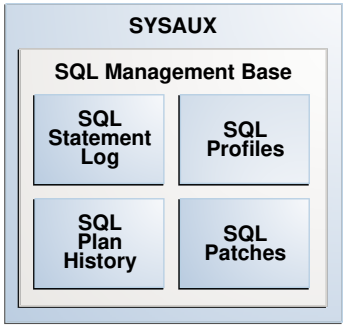
The SMB contains the following:

- SQL statement log, which contains only SQL IDs
- SQL plan history, which includes the SQL plan baselines
- SQL profiles
- SQL patches

The SMB stores information that the optimizer can use to maintain or improve SQL performance.

The SMB resides in the `SYSAUX` tablespace and uses automatic segment-space management. Because the SMB is located entirely within the `SYSAUX` tablespace, the database does not use SQL plan management and SQL tuning features when this tablespace is unavailable.

Figure 28-5 SMB Architecture



SMB data related to a PDB is stored in the PDB, and is included if the PDB is unplugged. A common user whose current container is the CDB root can view SMB data for PDBs. A user whose current container is a PDB can view the SMB data for the PDB only.

> **See Also:**
>
> *Oracle Database Administrator's Guide* to learn about the `SYSAUX` tablespace

SQL Statement Log

When automatic SQL plan capture is enabled, the **SQL statement log** contains the signature of statements that the optimizer has evaluated over time.

A SQL signature is a numeric hash value computed using a SQL statement text that has been normalized for case insensitivity and white space. When the optimizer parses a statement, it creates signature.

During automatic capture, the database matches this signature against the SQL statement log (`SQLLOG$`) to determine whether the signature has been observed before. If it has not, then the database adds the signature to the log. If the signature is already in the log, then the database has confirmation that the statement is a repeatable SQL statement.

> **Note:**
>
> If a filter excludes a statement, then its signature is also excluded from the log.

Example 28-1 Logging SQL Statements

This example illustrates how the database tracks statements in the statement log and creates baselines automatically for repeatable statements. An initial query of the statement log shows no tracked SQL statements. After a query of `hr.jobs` for `AD_PRES`, the log shows one tracked statement.

```
SQL> ALTER SYSTEM SET OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES=true;

System altered.

SQL> SELECT * FROM SQLLOG$;

no rows selected

SQL> SELECT job_title FROM hr.jobs WHERE job_id = 'AD_PRES';

JOB_TITLE
---------------------------------
President

SQL> SELECT * FROM SQLLOG$;

 SIGNATURE     BATCH#
---------- ----------
1.8096E+19          1
```

Now the session executes a different `jobs` query. The log shows two tracked statements:

```
SQL> SELECT job_title FROM hr.jobs WHERE job_id='PR_REP';

JOB_TITLE
---------------------------------
Public Relations Representative
```

```
SQL> SELECT * FROM SQLLOG$;

 SIGNATURE     BATCH#
---------- ----------
1.7971E+19          1
1.8096E+19          1
```

A query of `DBA_SQL_PLAN_BASELINES` shows that no baseline for either statement exists because neither statement is repeatable:

```
SQL> SELECT SQL_HANDLE, SQL_TEXT
  2  FROM DBA_SQL_PLAN_BASELINES
  3  WHERE SQL_TEXT LIKE 'SELECT job_title%';

no rows selected
```

The session executes the query for `job_id='PR_REP'` a second time. Because this statement is now repeatable, and because automatic SQL plan capture is enabled, the database creates a plan baseline for this statement. The query for `job_id='AD_PRES'` has only been executed once, so no plan baseline exists for it.

```
SQL> SELECT job_title FROM hr.jobs WHERE job_id='PR_REP';

JOB_TITLE
----------------------------------
Public Relations Representative

SQL> SELECT SQL_HANDLE, SQL_TEXT
  2  FROM DBA_SQL_PLAN_BASELINES
  3  WHERE SQL_TEXT LIKE 'SELECT job_title%';

SQL_HANDLE           SQL_TEXT
-------------------- --------------------
SQL_f9676a330f972dd5 SELECT job_title FRO
                     M hr.jobs WHERE job_
                     id='PR_REP'
```

See Also:

- "Automatic Initial Plan Capture"
- *Oracle Database Reference* to learn about `DBA_SQL_PLAN_BASELINES`

SQL Plan History

The **SQL plan history** is the set of captured SQL execution plans. The history contains both SQL plan baselines and unaccepted plans.

In SQL plan management, the database detects new SQL execution plans for existing SQL plan baselines and records the new plan in the history so that they can be evolved (verified). Evolution is initiated automatically by the database or manually by the DBA.

Starting in Oracle Database 12c, the SMB stores the execution plans for all SQL statements in the SQL plan history. The `DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE` function fetches and displays the plan from the SMB. For plans created before Oracle Database 12c, the function must compile the SQL statement and generate the plan because the SMB does not store it.

See Also:

- "Displaying Plans in a SQL Plan Baseline"
- *Oracle Database PL/SQL Packages and Types Reference* to learn about the `DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE` function

Enabled Plans

An **enabled plan** is a plan that is eligible for use by the optimizer.

When plans are loaded with the `enabled` parameter set to `YES` (default), the database automatically marks the resulting SQL plan baselines as enabled, even if they are unaccepted. You can manually change an enabled plan to a disabled plan, which means the optimizer can no longer use the plan even if it is accepted.

Accepted Plans

An **accepted plan** is a plan that is in a SQL plan baseline for a SQL statement and thus available for use by the optimizer. An accepted plan contains a set of hints, a plan hash value, and other plan-related information.

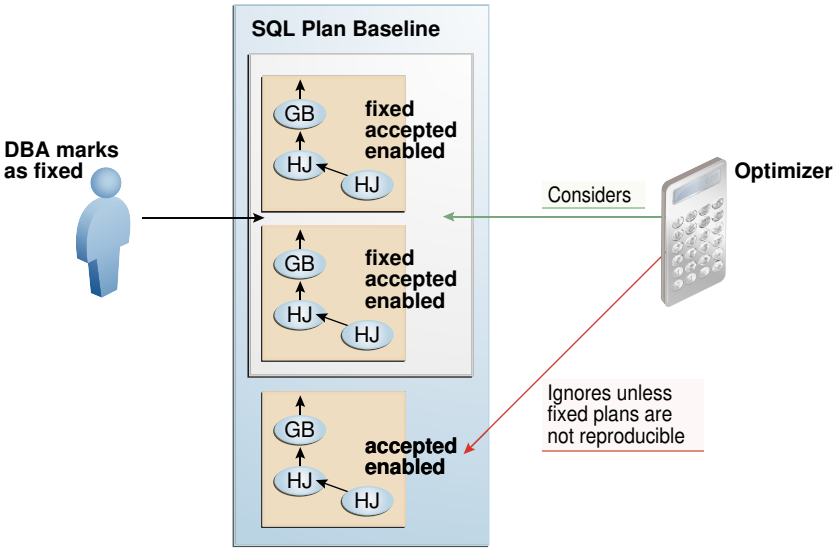
The SQL plan history for a statement contains all plans, both accepted and unaccepted. After the optimizer generates the first accepted plan in a plan baseline, every subsequent unaccepted plan is added to the plan history, awaiting verification, but is not in the SQL plan baseline.

Fixed Plans

A **fixed plan** is an accepted plan that is marked as preferred, so that the optimizer considers only the fixed plans in the baseline. Fixed plans influence the plan selection process of the optimizer.

Assume that three plans exist in the SQL plan baseline for a statement. You want the optimizer to give preferential treatment to only two of the plans. As shown in the following figure, you mark these two plans as fixed so that the optimizer uses only the best plan from these two, ignoring the other plans.

Figure 28-6 Fixed Plans



If new plans are added to a baseline that contains at least one enabled fixed plan, then the optimizer cannot use the new plans until you manually declare them as fixed.