# SQL Tuning Without Trying

**Arup Nanda**

*Longtime Oracle DBA*

## Scenario

- Situation:
  - Query from hell pops up, brings the DB to its knees
  - DBA is blamed for the failure
- Aftermath
  - DBA: "Developer should have taken care of this."
  - Developer: "Why is the DBA not aware of this problem?"
  - Manager: "DBA will review all queries and approve them."
- Challenge
  - What is the most efficient way to manage this process?

# Why Good SQLs Go Bad

- Missing, Incomplete or Inaccurate Statistics
- Improper or Lack of Indexing
- Bad Syntax
  - `WHERE COL1+20 = COL2`
  - `WHERE UPPER(COL1) = 'XYZ'`
- High Demand for Data Buffers
- Bind peeking
- Upgrades, patches



# Solutions

- Adding or Correcting Indexing
  - Index Absent
  - Proper Index- B-tree? Bitmap? Unique?
- Rewriting the SQL
  - e.g. `col1+10=:v1` becomes `col1=:v1-10`
  - Nested Loop to Hash Join
- Reduce I/O
  - Materialized Views
  - Partitioning
- Collect Accurate Statistics
- Put Hints
- Create Outlines

# Challenges

- Tough to determine why plans go bad, at least quickly
- Requires development skills
  - Not typical DBA skills
- Volume of statements to tune
- Time
  - Almost always reactive
  - Do it *now*. Under pressure!
- Not in the loop for application deployment
- Code can't be changed, i.e. no hints
- Lack of Testing
  - Time
  - Resources

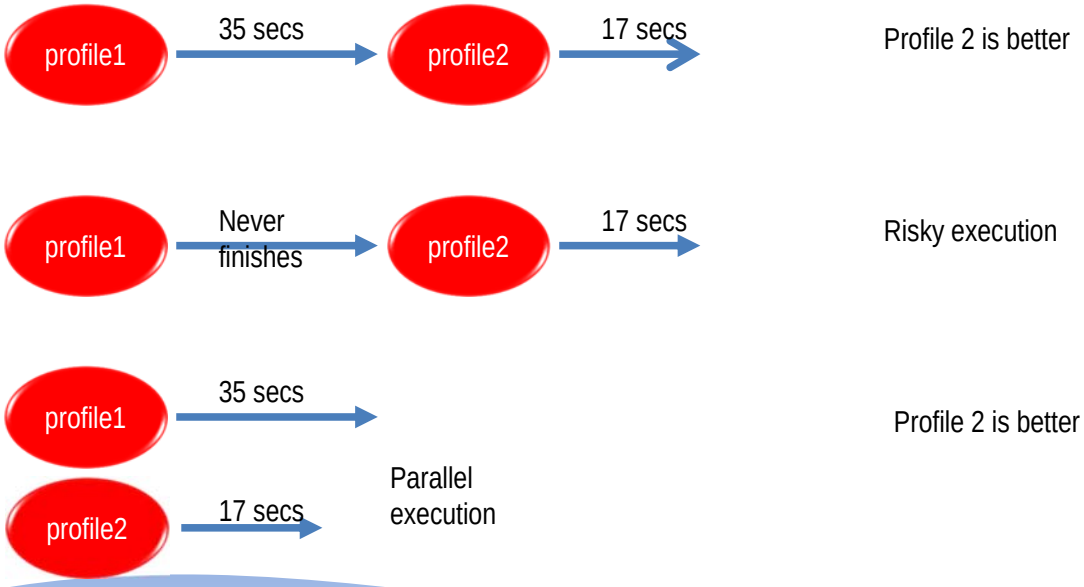


# SQL Profile

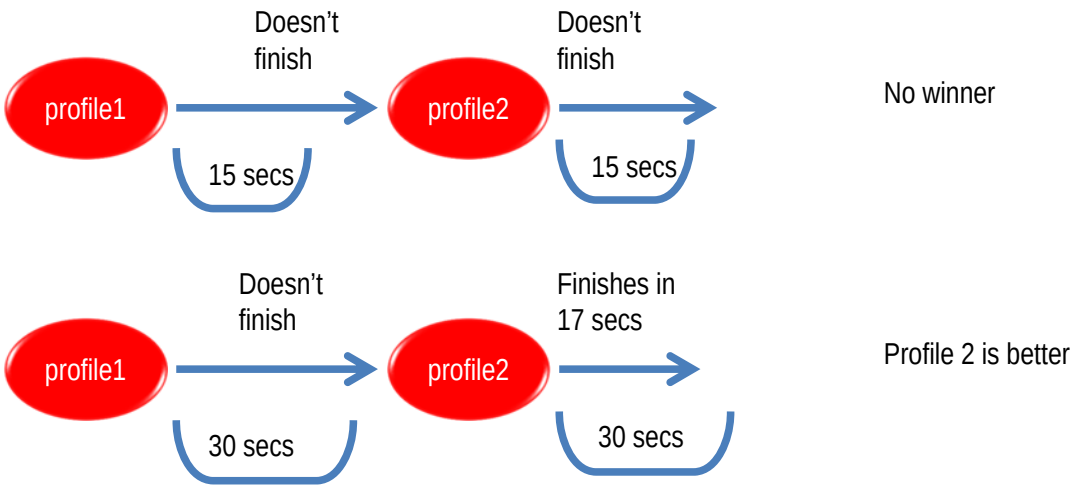
- Hints are *automatically* added to queries
- Gives more information about the accessed objects, data, etc.

```
<outline_data>
  <hint><![CDATA[BEGIN_OUTLINE_DATA]]></hint>
  <hint><![CDATA[IGNORE_OPTIM_EMBEDDED_HINTS]]></hint>
  <hint><![CDATA[OPTIMIZER_FEATURES_ENABLE('11.2.0.3')]]></hint>
  <hint><![CDATA[DB_VERSION('11.2.0.3')]]></hint>
  <hint><![CDATA[OPT_PARAM('optimizer_dynamic_sampling' 7)]]></hint>
  <hint><![CDATA[ALL_ROWS]]></hint>
  <hint><![CDATA[OUTLINE_LEAF(@"SEL$2")]]></hint>
  <hint><![CDATA[OUTLINE_LEAF(@"SEL$1")]]></hint>
  <hint><![CDATA[NO_ACCESS(@"SEL$1" "from$_subquery$_001"@"SEL$1")]]></hint>
  <hint><![CDATA[INDEX_RS_ASC(@"SEL$2" "CH"@"SEL$2" ("T1"."COL1" "T1"."COL2"
"T1"."COL3"))]]></hint>
  <hint><![CDATA[OPT_ESTIMATE(@"SEL$1", TABLE, "T"@"SEL$1", SCALE_ROWS=0.15")]]></hint>
  <hint><![CDATA[END_OUTLINE_DATA]]></hint>
</outline_data>
```

# How Oracle Selects a Profile





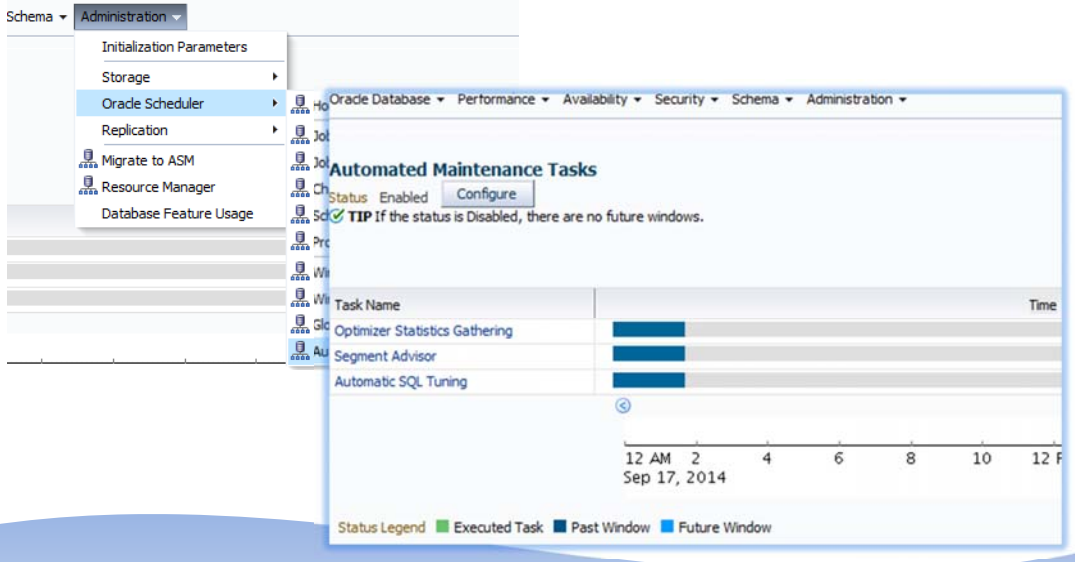
# Adding SQL Profiles?

- You add it by a tool “SQL Tuning Advisor”
- What it is:
  - A built-in tool for SQL Tuning
  - Can suggest alternatives, some pretty good
- Suggests:
  - Indexes
  - Rewriting
  - Materialized Views
  - Partitioning
  - Statistics
  - SQL Profiles
  - Baselines



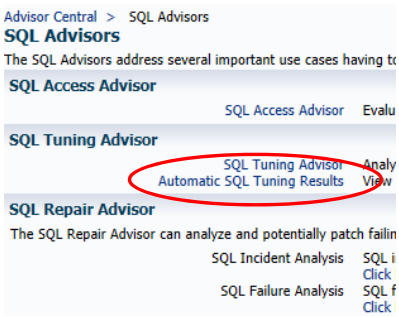
# SQL Tuning Advisor

- From Top Menu -> Administration -> Oracle Scheduler -> Automated Maintenance Tasks



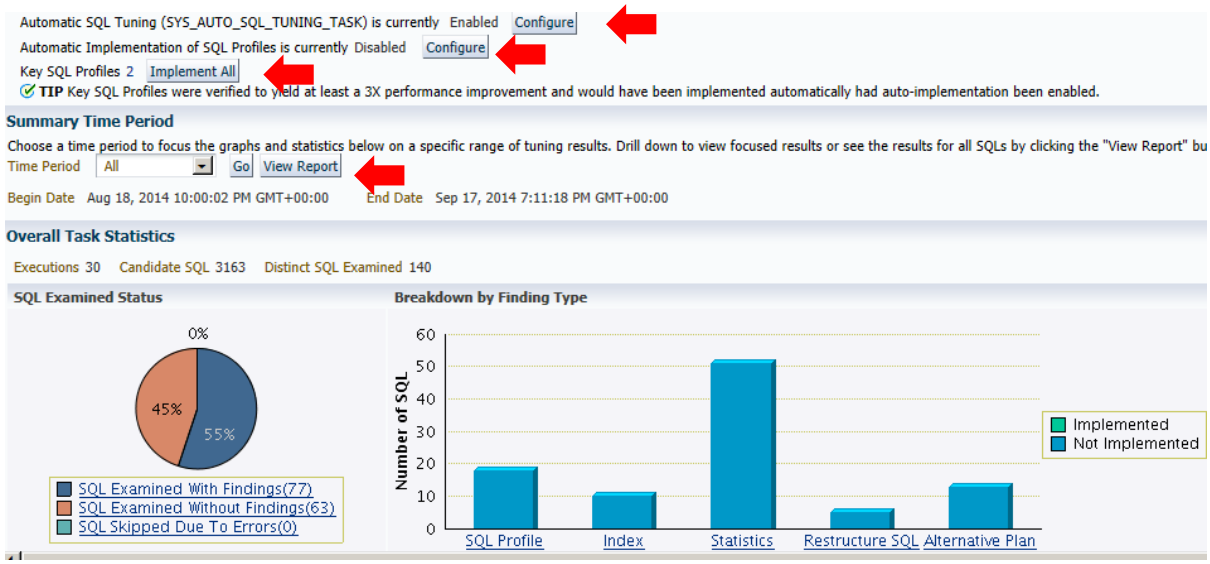
# Automatic

- Automatic since Oracle 11g
- Or, from Top Menu -> Performance -> Advisor Home -> SQL Advisors






# Automatic SQL Tuning



**Recommendations**

Only profiles that significantly improve SQL performance were implemented.

View Recommendations | Implement All SQL Profiles

| Select | SQL Text | Parsing Schema | SQL ID | Weekly DB Time Benefit(sec) | Per-Execution % Benefit | Statistics | SQL Profile | Index |
|---|---|---|---|---|---|---|---|---|
| ◉ | SELECT PROP_ID PROPERTY_ID,AVAIL_INV_DAT... | REX | bwsgtxfb505fq | 211438.92 | 47 | ✓ | (47%) ✓ | |
| ○ | INSERT INTO RMS_PT_CAL (AVAIL_INV_DATE, ... | REX | g1fy66a9kjda9 | 11355.43 | 76 | ✓ | | (76%) ✓ |
| ○ | INSERT INTO RMS_PT_CAL (AVAIL_INV_DATE, ... | REX | drb8p000mp8kg | 8392.38 | 74 | ✓ | | (74%) ✓ |
| ○ | UPDATE RMS_PT_CAL A SET A.LOS_SWITCH_MAS... | REX | 9cjybwtg5fbvs | 8034.56 | 85 | ✓ | | (85%) ✓ |
| ○ | UPDATE PRODUCT_CAL A SET A.LOS_SWITCH_MA... | REX | cvhr162tckssw | 6053.95 | 68 | ✓ | | (68%) ✓ |
| ○ | UPDATE RATE_CAT_CAL A SET (LOS_SWITCH_MA... | REX | 5cag5nx81cb1n | 931.15 | <10 | ✓ | (<10%) ✓ | |
| ○ | SELECT ROWID "ROWID", ORA_ROWSCN "ORA_RO... | REX | 8w64p983441bv | 110.95 | 98 | ✓ | (98%) ✓ | |
| ○ | SELECT PROP_ID PROPERTY_ID, AVAIL_INV_DA... | REX | c2v0m8j8s4zpb | 107.08 | 86 | ✓ | (86%) ✓ | |
| ○ | INSERT INTO RATE_CAT_CAL (AVAIL_INV_DATE,... | REX | cufv4r8wsxtpd | 93.39 | 31 | ✓ | (31%) ✓ | |
| ○ | DELETE FROM RMS_PT_CAL WHERE PROP_ID = :... | REX | 1gaxyb9td18tq | 54.33 | 95 | ✓ | | (95%) ✓ |
| ○ | DELETE FROM RMS_PT_CAL WHERE PROP_ID | REX | db254d2n7kt57 | 39.94 | 55 | ✓ | | (55%) ✓ |

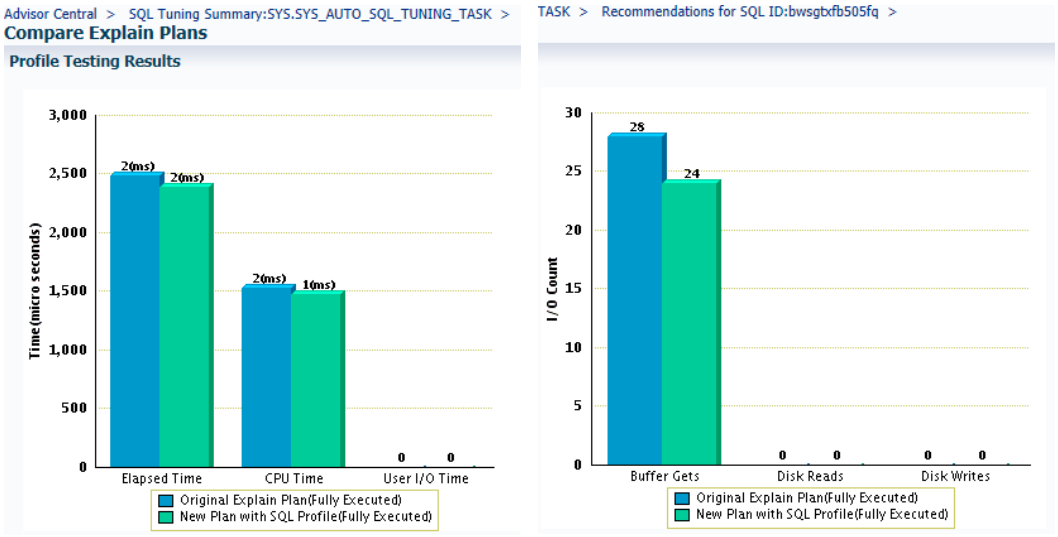


| | |
|---|---|
| SQL Profile | A potentially better execution plan was found for this statement. |
| Alternative Plans | Some alternative execution plans for this statement were found by searching the system's real-time and historical performance data. |

*Shows the comparison of plans before and after SQL Profile*

| | | | | |
|---|---|---|---|---|
| The SQL profile was not automatically created because its benefit could not be verified. | 47.84 | | oo | oo |
| Creating a plan baseline for the plan with the best elapsed time will prevent the Oracle optimizer from selecting a plan with worse performance. | | oo | | |

# Enhancement Comparison





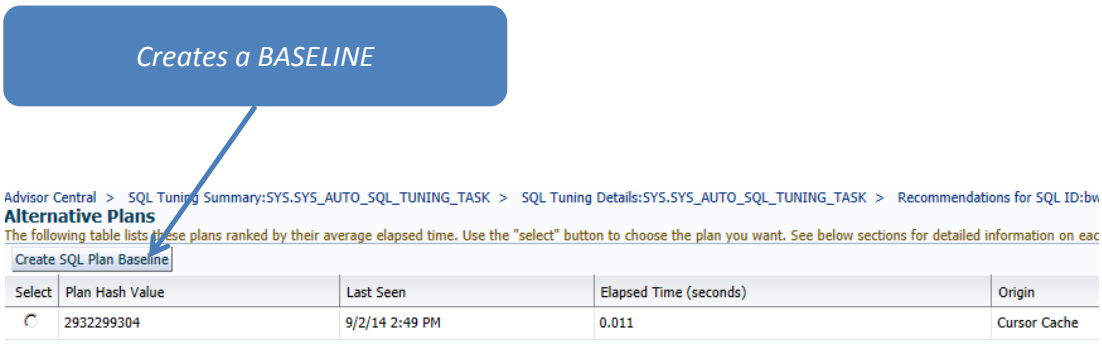

# Compare Plans

| Operation | Line ID | Object |
|---|---|---|
| ▽ SELECT STATEMENT | 0 | |
| ▽ SORT GROUP BY | | |
| ▽ FILTER | | |
| ▽ FILTER | | |
| ▽ NESTED LOOPS | | |
| ▽ HASH JOIN | | |
| ▽ PARTITION RANGE ITERATOR | | |

| Order | Rows | Bytes | Cost | Time | CPU Cost | I/O Cost |
|---|---|---|---|---|---|---|
| 27 | | 0.162 | 23 | 1 | 39,814,568 | 21 |
| 26 | | 0.162 | 23 | 1 | 39,814,568 | 21 |
| 25 | | | | | | |
| 23 | | | | | | |
| 22 | | 0.162 | 22 | 1 | 20,091,796 | 21 |
| 20 | | 0.307 | 22 | 1 | 20,087,996 | 21 |
| 3 | | 0.024 | 6 | 1 | 201,168 | 6 |
| 2 | | 0.024 | 6 | 1 | 201.168 | 6 |

*Shows that the plan steps are different as a result of SQL Profile*

# Alternative Plans






# Why only Profiles in Auto?

- Setup is quick
  - e.g. building an index takes time
- SQL does not need to change
- Testing localized to SQL only—effective
- Don't like it? Easily undone.
- Can be private, using SQL Tune Category

# More on Auto Profiles Tests

- Default Behavior
  - Uses MAINTENANCE_WINDOW_GROUP
  - SQL profiles are generated but not implemented
- You can configure
  - If, when, how long
  - Resources allowed to use
  - If profiles are automatically accepted
  - How many profiles it implements



# SQL Profiles or Baselines

| SQL Profiles | Baselines |
|---|---|
| Reactive | Proactive |
| Bad plan. Fix applied | Good Plan. Plan Fixed |
| Works by storing additional information about cardinality | Works by storing the plan. Cardinality is not the primary factor |
| Provides additional data to Optimizer | Helps Optimizer to choose from choices |
| No specific plan | Only the set of plans |
| When data changes are dramatic, this is a better approach | When data changes are dramatic, difficult |
| One execution is enough to generate profile | More than one execution is required for capture the baseline |
| Can still be valid if the access structures change | May not be valid when access structures change |

# Realtime SQL Monitoring

- From SQL Menu, Plan
  - Automatically monitors long running SQL
  - Shows the statistics and resources consumed at each step of the plan.
  - Shows actual cardinality at each step, helps resolve problems with poor cardinality estimates
- Exposes monitoring statistics
  - Plan operation level
  - Parallel Execution level
  - I/O, CPU, memory, network
  - Exadata Smart Scans

**Very Useful Tool: Active Reports**



# Active Reports without EM

- Built-In Functions Returning Report as CLOB
  - SQL Details `dbms_perf.report_session`
  - SQL Monitor `dbms_sqltune.report_sql_monitor_list`
  - SQL Perf Analyzer `dbms_sqlpa.report_analysis_task`
  - Performance Hub `dbms_perf.report_perfhub`
- Example

```
set pages 0 linesize 32767 trimspool on
set long 1000000 longchunksize 10000000
spool rep.html
select dbms_perf.report_perfhub (is_realtime=>1,
type=>'active') from dual;
```

# Don't Like GUI?

- Package DBMS_SQLTUNE Functions

| Function | Description |
|---|---|
| CREATE_TUNING_TASK | Creates a tuning task<br>• For a single SQL, a group of SQLs<br>• For SQL text, or SQL_ID<br>• From an SQL Tuning Set |
| EXECUTE_TUNING_TASK | Executes the task<br>• The parameters are defined here |
| REPORT_TUNING_TASK | Reports the findings |
| SCRIPT_TUNING_TASK | Implement the results. Creates a script to be implemented by SQL*Plus |



# Non-GUI Auto

- Package DBMS_AUTO_SQLTUNE

| Function | Description |
|---|---|
| SET_AUTO_TUNING_TASK_PARAMETER | Change the default parameters |
| EXECUTE_AUTO_TUNING_TASK | Executes the task<br>• The parameters are defined here |
| REPORT_AUTO_TUNING_TASK | Reports the findings |

# Sources for Tuning Set

All functions are in DBMS_SQLTUNE package

| Source | How to Get from it |
|---|---|
| Shared pool | `SELECT_CURSOR_CACHE ()` |
| From AWR Repository | `SELECT_WORKLOAD_REPOSITORY ()` |
| Oracle Trace Files | `SELECT_SQL_TRACE ()` |
| SQL Performance Analyzer task comparison results | `SELECT_SQLPA_TASK ()` |
| Another SQL Tuning Set | `SELECT_SQLSET ()` |



# Takeaways

- Enable SQL Tuning Advisor to run automatically
- Disable automatic application of SQL Profiles
- Check recommendations and apply them from one screen
  - In small databases, may want to enable automatic application of profiles
- Use Realtime Monitoring to find out issues at specific steps
- Generate Active Reports to explain database issues

# *Thank You!*

Blog: arup.blogspot.com
Tweeter: @arupnanda
Facebook.com/ArupKNanda