
regoUniversity 2021
VIRTUAL
Performance Optimization
| SQL Tuning
Your Guides:
Davey Zywiec & Dave Matzdorf

Introductions

- Take 5 Minutes
- Turn to a Person Near You
- Introduce Yourself
- Business Cards

Agenda

- Introduction
- IN vs EXISTS
- DISTINCT vs EXISTS
- OBS Filtering
- UNION Queries
- Inline Views
- Subquery Factoring
- Analytic Functions

Part I: Miscellaneous Examples

<Descriptor>

Let Rego be your guide.

IN vs. EXISTS

- IN is typically better when the inner query contains a small result set
- EXISTS is typically better when the inner query contains a large result set

```
SELECT SRMR.FULL_NAME
FROM SRM_RESOURCES SRMR
WHERE SRMR.ID IN (SELECT TM.PRRESOURCEID FROM PRTEAM TM)
```

- Vs

```
SELECT SRMR.FULL_NAME
FROM SRM_RESOURCES SRMR
WHERE EXISTS (SELECT 1 FROM PRTEAM TM WHERE TM.PRRESOURCEID = SRMR.ID)
```

DISTINCT vs. EXISTS

- EXISTS is preferable to DISTINCT
- DISTINCT produces the entire result set (including duplicates), sorts, and then filters out duplicates

```
SELECT DISTINCT SRMR.FULL_NAME
FROM SRM_RESOURCES SRMR
JOIN PRTEAM TM ON SRMR.ID = TM.PRRESOURCEID
```

- EXISTS proceeds with fetching rows immediately after the sub-query condition has been satisfied the first time

```
SELECT SRMR.FULL_NAME
FROM SRM_RESOURCES SRMR
WHERE EXISTS (SELECT 1 FROM PRTEAM TM WHERE TM.PRRESOURCEID = SRMR.ID)
```

OBS Filtering

- Seen many ways to filter based on OBS
- Many rely on complex logic, left joins to inline views, or multiple sub-queries
- Using EXISTS and the OBS_UNITS_FLAT_BY_MODE table provides an easy solution
- Filter by Unit Only, Unit and Descendants, or Units and Ancestors

```
SELECT SRMR.FULL_NAME
FROM SRM_RESOURCES SRMR
WHERE (:OBS_ID IS NULL OR
        EXISTS (SELECT 1
                FROM OBS_UNITS_FLAT_BY_MODE OBSM
                JOIN PRJ_OBS_ASSOCIATIONS OBSA ON OBSM.LINKED_UNIT_ID = OBSA.UNIT_ID AND
                    OBSA.TABLE_NAME = 'SRM_RESOURCES'
                WHERE OBSM.UNIT_ID = :OBS_ID
                AND OBSM.UNIT_MODE = NVL(:OBS_MODE, 'OBS_UNIT_AND_CHILDREN')
                AND OBSA.RECORD_ID = SRMR.ID))
```

UNION Queries

- UNION queries perform poorly as they scan through the same data multiple times
- Require any logic changes to be made in multiple locations

```
SELECT CODE, NAME, SUM(FORECAST_COST) FORECAST_COST, SUM(BUDGET_COST) BUDGET_COST
FROM (SELECT INVI.CODE, INVI.NAME, FP.TOTAL_COST FORECAST_COST, 0 BUDGET_COST
    FROM INV_INVESTMENTS INVI
    JOIN FIN_PLANS FP ON INVI.ID = FP.OBJECT_ID AND INVI.ODF_OBJECT_CODE = FP.OBJECT_CODE
    WHERE FP.IS_PLAN_OF_RECORD = 1 AND FP.PLAN_TYPE_CODE = 'FORECAST'
    UNION ALL
    SELECT INVI.CODE, INVI.NAME, 0 FORECAST_COST, FP.TOTAL_COST BUDGET_COST
    FROM INV_INVESTMENTS INVI
    JOIN FIN_PLANS FP ON INVI.ID = FP.OBJECT_ID AND INVI.ODF_OBJECT_CODE = FP.OBJECT_CODE
    WHERE FP.IS_PLAN_OF_RECORD = 1 AND FP.PLAN_TYPE_CODE = 'BUDGET')
WHERE 1=1
GROUP BY CODE, NAME
```

- Most UNION queries can easily be replaced with logic

```
SELECT INVI.CODE, INVI.NAME
, SUM(CASE WHEN FP.PLAN_TYPE_CODE = 'FORECAST' THEN FP.TOTAL_COST END) FORECAST_COST
, SUM(CASE WHEN FP.PLAN_TYPE_CODE = 'BUDGET' THEN FP.TOTAL_COST END) BUDGET_COST
FROM INV_INVESTMENTS INVI
JOIN FIN_PLANS FP ON INVI.ID = FP.OBJECT_ID AND INVI.ODF_OBJECT_CODE = FP.OBJECT_CODE
WHERE 1=1
GROUP BY INVI.CODE, INVI.NAME
```

- Only use UNION when joining data from multiple tables

Inline Views

- Inline views can be very beneficial but can severely affect performance
- LEFT JOINs to large inline views is typically not a good idea

```
SELECT SRMR.FULL_NAME, SUM(AV.SLICE) AVAIL, AL.ALLOC
FROM SRM_RESOURCES SRMR
JOIN PRJ_BLB_SLICES AV ON SRMR.ID = AV.PRJ_OBJECT_ID AND AV.SLICE_REQUEST_ID = 7
LEFT JOIN (SELECT TM.PRRESOURCEID, SUM(AL.SLICE) ALLOC
        FROM PRTEAM TM
        JOIN PRJ_BLB_SLICES AL ON TM.PRID = AL.PRJ_OBJECT_ID
        WHERE AL.SLICE_REQUEST_ID = 6
        AND AL.SLICE_DATE BETWEEN '01-JAN-14' AND '30-JUN-14'
        GROUP BY TM.PRRESOURCEID) AL ON SRMR.ID = AL.PRRESOURCEID
WHERE AV.SLICE_DATE BETWEEN '01-JAN-14' AND '30-JUN-14'
GROUP BY SRMR.FULL_NAME, AL.ALLOC
ORDER BY SRMR.FULL_NAME
```

- Will talk through some examples to demonstrate alternatives

Subquery Factoring – WITH clause

- Simplify complex queries
- Reduce repeated table access by generating temporary datasets during query execution
- Can be used as an inline view or a table

```
WITH ALLOCS AS (
  SELECT INVI.ID, INVI.CODE, INVI.NAME, AL.SLICE_DATE, AL.SLICE
  FROM SRM_RESOURCES SRMR
  JOIN PRTEAM TM ON SRMR.ID = TM.PRRESOURCEID
  JOIN INV_INVESTMENTS INVI ON TM.PRPROJECTID = INVI.ID
  JOIN PRJ_BLB_SLICES AL ON TM.PRID = AL.PRJ_OBJECT_ID AND AL.SLICE_REQUEST_ID = 6
  WHERE SRMR.UNIQUE_NAME = 'dmatzdorf' AND AL.SLICE > 0
  AND AL.SLICE_DATE IN ('01-SEP-21', '01-OCT-21')
)
SELECT A.ID, A.CODE, A.NAME, A.SLICE_DATE, A.SLICE, 1 SORT_ORDER
FROM ALLOCS A
UNION ALL
SELECT NULL ID, NULL CODE, TO_CHAR(A.SLICE_DATE, 'Mon YY') || ' Total' NAME, A.SLICE_DATE, SUM(A.SLICE) SLICE, 2 SORT_ORDER
FROM ALLOCS A
GROUP BY A.SLICE_DATE
UNION ALL
SELECT NULL ID, NULL CODE, 'Total' NAME, NULL SLICE_DATE, SUM(A.SLICE) SLICE, 3 SORT_ORDER
FROM ALLOCS A
ORDER BY SLICE_DATE, SORT_ORDER, NAME
```

Part I:
Analytic Functions

<Descriptor>

Let Rego be your guide.

What Are Analytic Functions

- Used to compute aggregate values based on a group of rows
- Similar to aggregate functions but return multiple rows
- Can only appear in the SELECT or ORDER BY clause
- Used to compute cumulative, moving aggregates

Why Use Analytic Functions

- Can be done with native SQL
- Odd syntax
- Analytic functions are faster and more accurate
- Get the latest status report
  - Get the max updated date for each project and join to it
  - Not accurate if there are multiple reports updated at the same time
  - Not efficient

Available Functions

- AVG
- CORR
- COUNT
- COVAR_POP
- COVAR_SAMP
- CUME_DIST
- DENSE_RANK
- FIRST
- FIRST_VALUE
- LAG
- LAST
- LAST_VALUE
- LEAD
- LISTAGG
- MAX
- MEDIAN
- MIN
- NTH_VALUE
- NTILE
- PERCENT_RANK
- PERCENTILE_CONT
- PERCENTILE_DISC
- RANK
- RATIO_TO_REPORT
- REGR_
- ROW_NUMBER
- STDDEV
- STDDEV_POP
- STDDEV_SAMP
- SUM
- VAR_POP
- VAR_SAMP
- VARIANCE

Selecting Specific Records

- ROW_NUMBER: Get the latest status report
- LEAD/LAG: Get the previous and next status report
- FIRST_VALUE/LAST_VALUE: Get the first and last status report
- NTH_VALUE: Get the nth status report

Summing

- SUM – Calculate total allocations
- RATIO_TO_REPORT – Calculate percentage of total allocations
- SUM – Calculate total allocation hours
- SUM ORDER BY – Running allocation hours

Questions?

regoUniversity 2021
VIRTUAL

Let Rego be your guide.