LAS VEGAS

regoUniversity 2025

Sponsored by

ValueOps by Broadcom | Clarity by Broadcom | Rally by Broadcom | ConnectALL by Broadcom | Insights by Broadcom

Best Practices in SQL

Your Guides:

David Matzdorf & Rahul Agrawal

Gold Sponsor

aws

Introductions

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

Agenda

- Introduction
- IN vs EXISTS
- DISTINCT vs EXISTS
- OBS Filtering
- UNION Queries
- Inline Views
- Subquery Factoring
- Double Dipping
- Recursion
- Analytic Functions
- Working examples

Introduction

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

- Multiple 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
```

UNION Queries

- 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

```sql
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-25', '01-OCT-25')
)
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
```

Double Dipping

- Accessing the same data twice
- Forecast and Budget totals

```
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 FP.IS_PLAN_OF_RECORD = 1

GROUP BY INVI.CODE
, INVI.NAME
```

- Availability vs Allocation vs ETC vs Actuals

Recursion

- Using the WITH clause to recurse
- Get OBS Full Path

```
WITH OBS_PATH (ID, TYPE_ID, UNIQUE_NAME, NAME, LVL, OBS_PATH) AS (
  SELECT OBSU.ID, OBSU.TYPE_ID, OBSU.UNIQUE_NAME, OBSU.NAME, 1 LVL
  , '/' || OBSU.NAME OBS_PATH
  FROM PRJ_OBS_UNITS OBSU
  WHERE OBSU.PARENT_ID IS NULL
  UNION ALL
  SELECT OBSU.ID, OBSU.TYPE_ID, OBSU.UNIQUE_NAME, OBSU.NAME
  , OBS.LVL + 1 LVL, OBS.OBS_PATH || '/' || OBSU.NAME OBS_PATH
  FROM PRJ_OBS_UNITS OBSU
  JOIN OBS_PATH OBS ON OBSU.PARENT_ID = OBS.ID
  WHERE 1=1
)

SELECT OBST.UNIQUE_NAME OBS_TYPE_CODE
, OBST.NAME OBS_TYPE
, OBSP.UNIQUE_NAME
, OBSP.NAME
, OBSP.LVL
, OBSP.OBS_PATH
FROM PRJ_OBS_TYPES OBST
JOIN OBS_PATH OBSP ON OBST.ID = OBSP.TYPE_ID
WHERE OBST.UNIQUE_NAME = 'dlm_project_obs'
```

Best Practices

- **Avoid SELECT *** – Retrieve only the columns you actually need.
- **Use Appropriate Joins** – Choose INNER, LEFT, or RIGHT joins based on actual requirements.
- **Filter Early** – Apply WHERE and JOIN conditions to reduce data sets before grouping or ordering.
- **Write Readable Queries** – Use consistent formatting, meaningful aliases, and clear indentation.
- **Test with Realistic Data Volumes** – Ensure performance and behavior hold up in production-sized data.
- **Leverage Set-Based Operations** – Avoid row-by-row (cursor) processing when possible.
- **Use Proper Indexing** – Match indexes to query patterns to improve performance.

Analytic Functions

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
- Operate with OVER() and optional PARTITION BY / ORDER BY clauses to define the calculation window

Why Use Analytic Functions

- **Preserve Detail** – Return aggregated metrics while keeping all individual rows visible.
- **Enhanced Analysis** – Enable calculations like rankings, running totals, and moving averages directly in SQL.
- **Performance Gains** – Reduce the need for complex self-joins or multiple subqueries.
- **Flexible Windows** – Allow calculations over dynamic ranges using PARTITION BY and ORDER BY
- **Accurate Insights** – Handle complex reporting needs (e.g., "top N per group" or period-over-period comparisons) with minimal SQL complexity

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 – Assign a unique sequential value to each row
- LAG/LEAD – Finds rows a number of rows from the current row
- FIRST_VALUE/LAST_VALUE – Finds first or last value in an ordered group
- RANK/DENSE_RANK – Rank items in a group
- SUM – Compute running totals
- Most recent status report

```
SELECT INVI.CODE
, INVI.NAME
, SR.REPORT_DATE
, SR.RNUM

FROM INV_INVESTMENTS INVI
JOIN (SELECT SR.ID
      , SR.ODF_PARENT_ID
      , SR.COP_REPORT_DATE REPORT_DATE
      , ROW_NUMBER() OVER (PARTITION BY SR.ODF_PARENT_ID
                           ORDER BY SR.COP_REPORT_DATE DESC, SR.CREATED_DATE DESC) RNUM
      FROM ODF_CA_COP_PRJ_STATUSRPT SR
      WHERE 1=1) SR ON INVI.ID = SR.ODF_PARENT_ID AND SR.RNUM = 1

WHERE 1=1
```

Summing

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

Window clause

- BETWEEN … AND
- UNBOUND PRECEDING
- UNBOUND FOLLOWING
- CURRENT ROW
- X PRECEDING OR X FOLLOWING

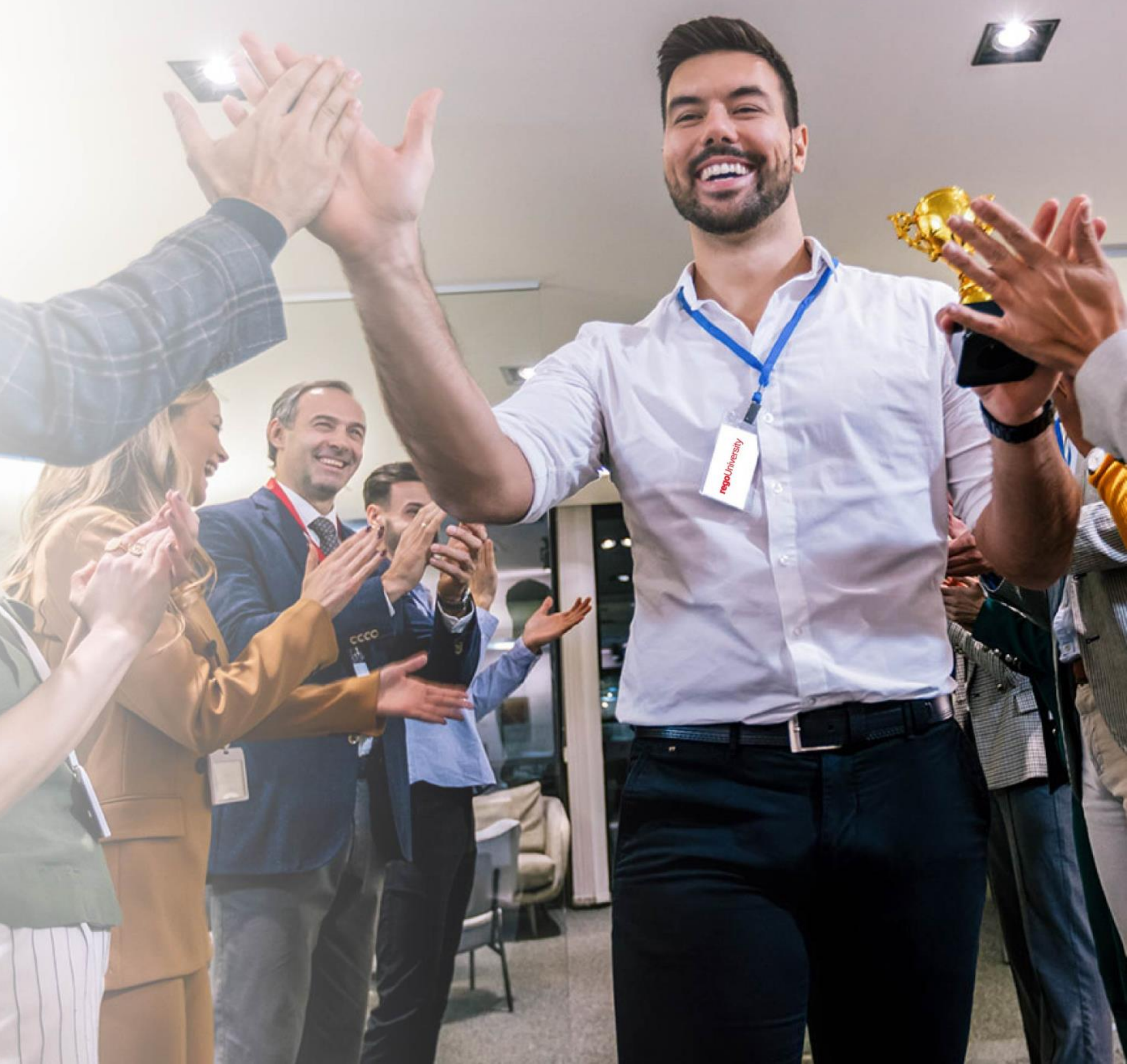
SQL Tuning 2025 Queries.txt

regoUniversity

Master Clarity with Rego University

Earn Certifications in Administration, Leadership, and Technical Proficiency

Let Rego be your guide.



Elevate Your Professional Expertise with Rego University Certifications

Rego is excited to continue our **certification programs**, designed to enhance your expertise in Clarity administration, leadership, and technical skills. These certifications provide hands-on experience and knowledge to excel in your career.

Certification Requirements:

- **Completion**: 12 units per certification track
- **Eligibility**: Open to all Rego University attendees

Important Reminder:

To have your certification **credits tracked**, ensure you **complete the class surveys in the app** after each session. This step is critical for certification progress.

Questions?

LAS VEGAS regoUniversity 2025

Surveys

Please take a few moments to fill out the class survey.
Your feedback is extremely important for future events.

Thank You For Attending Rego University

Instructions for PMI credits

- Access your account at pmi.org
- Click on **Certifications**
- Click on **Maintain My Certification**
- Click on **Visit CCR's** button under the **Report PDU's**
- Click on **Report PDU's**
- Click on **Course or Training**
- Class Provider = **Rego Consulting**
- Class Name = **regoUniversity**
- Course **Description**
- Date Started = **Today's Date**
- Date Completed = **Today's Date**
- Hours Completed = **1 PDU per hour of class time**
- Training classes = **Technical**
- Click on **I agree** and **Submit**

Let us know how we can improve!
Don't forget to fill out the class survey.

Phone
888.813.0444

Email
info@regoconsulting.com

Website
www.regouniversity.com

Continue to Get Resources and Stay Connected

1. Use RegoXchange.com for instructions and how-tos.
2. Talk with your account managers and your Rego consultants.
3. Connect with each other and Clarity experts at RegoGroups.com.
4. Sign up for webinars and join in-person Rego groups near you through at RegoConsulting.com
5. Join us for the next Rego University!