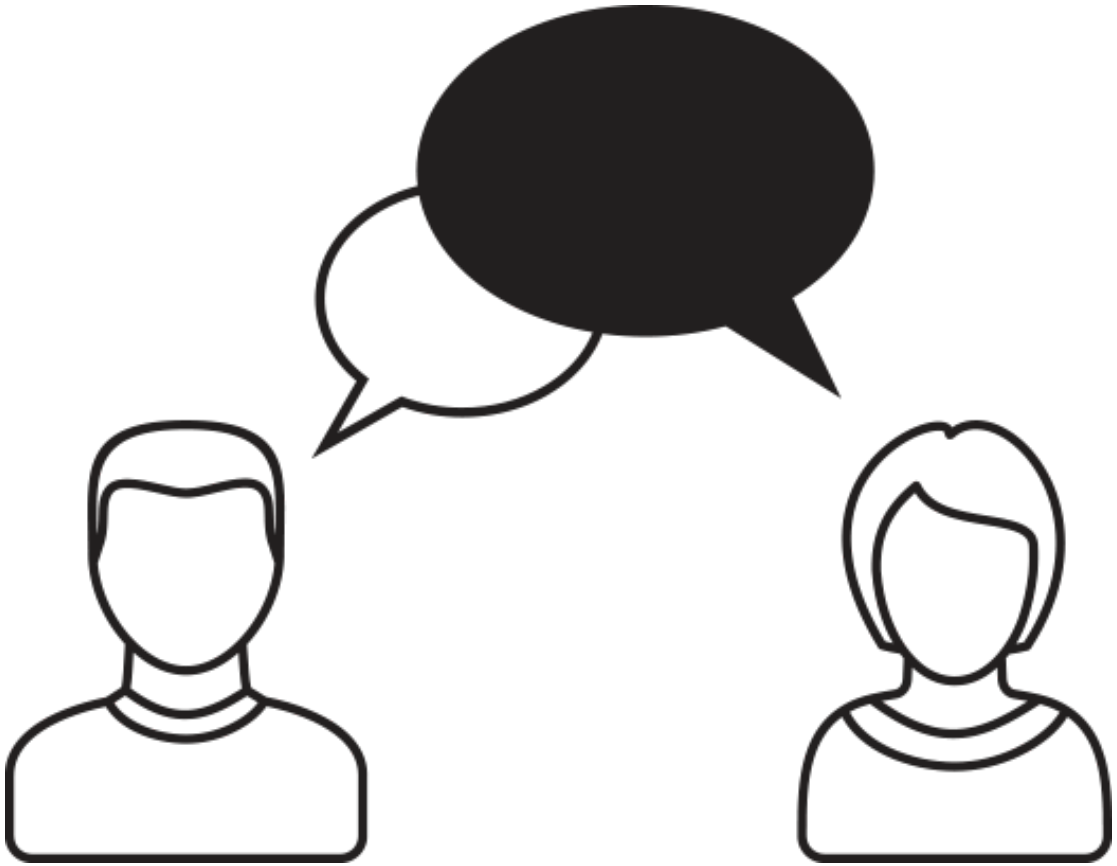

regoUniversity 2017
SQL Tuning |
Tips and Tricks
Your Guides: Vipin Chouhan, James Gille

Introductions

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



Agenda

- Common Issues
- General Tips
- Pre-Aggregating Data

Double Dipping

- Accessing the same data twice
- Requires multiple scans of the same data
- Restructure query to avoid

Double Dipping – Actuals vs ETC Bad Example

```
SELECT  i.id project_id,
        i.name project_name
        s.slice_date
        SUM(CASE s.slice_type WHEN 'ACTUALS' THEN s.slice ELSE 0 END) actuals
        SUM(CASE s.slice_type WHEN 'ETC' THEN s.slice ELSE 0 END) etc
FROM    inv_investments i
        JOIN (SELECT t.prprojectid, 'ACTUALS' slice_type, s.slice_date, SUM(s.slice) slice
                FROM  prtask t
                        JOIN prassignment a ON t.prid = a.prtaskid
                        JOIN prj_blb_slices s ON a.prid = s.prj_object_id AND s.slice_request_id = 4
                GROUP BY t.prprojectid, s.slice_date
                UNION ALL
                SELECT t.prprojectid, 'ETC' slice_type, s.slice_date, SUM(s.slice) slice
                FROM  prtask t
                        JOIN prassignment a ON t.prid = a.prtaskid
                        JOIN prj_blb_slices s ON a.prid = s.prj_object_id AND s.slice_request_id = 5
        GROUP BY t.prprojectid, s.slice_date) s ON i.id = s.prprojectid
GROUP BY i.id, invi.name, s.slice_date
```

Double Dipping – Actuals vs ETC Good Example

```
SELECT  i.id project_id
        i.name project_name
        s.slice_date
        SUM(CASE s.slice_request_id WHEN 4 THEN s.slice ELSE 0 END) actuals
        SUM(CASE s.slice_request_id WHEN 5 THEN s.slice ELSE 0 END) etc
FROM    inv_investments i
        JOIN prtask t ON i.id = t.prprojectid
        JOIN prassignment a ON t.prid = a.prtaskid
        JOIN prj_blb_slices s ON a.prid = s.prj_object_id AND s.slice_request_id IN(4, 5)
GROUP BY i.id, i.name, s.slice_date
```

Inline Views

- Great for including complex logic in a query
- Removes the need to create database views or functions
- LEFT OUTER joins to them can be a performance bottleneck
- Often seen availability vs allocation, allocations vs ETCs, or forecast vs actuals
- Replace with a UNION and standard join for better performance

Inline Views – Bad Example

```
SELECT  i.id, i.code,  i.name,  fp.transaction_class_id,  fp.period_id, fp.period, fp.slice forecast, wip.slice actuals
FROM    inv_investments i
        LEFT JOIN (SELECT  i.id project_id, fd.transaction_class_id, bp.id period_id, bp.start_date period,
                           SUM(c.slice * (finish_date - c.start_date)) slice
                   FROM    inv_investments i
                           JOIN fin_plans fp ON i.id = fp.object_id AND i.odf_object_code = fp.object_code
                           JOIN fin_cost_plan_details fd ON fp.id = fd.plan_id
                           JOIN odf_ssl_cst_dtl_cost c ON c.prj_object_id = fd.id
                           JOIN entity e ON i.entity_code = e.entity
                           JOIN biz_com_periods bp ON e.id = bp.entity_id AND bp.period_type = 'MONTHLY'
                                                      AND c.start_date BETWEEN bp.start_date AND bp.end_date
                   WHERE fp.plan_type_code = 'FORECAST‘ AND fp.is_plan_of_record = 1
                   GROUP BY i.id, fd.transaction_class_id, bp.id, bp.start_date) fp ON i.id = fp.project_id
        LEFT JOIN (SELECT  wip.investment_id project_id, tc.id transaction_class_id, bp.id period_id, bp.start_date period,
                           SUM(wv.totalcost) slice
                   FROM    ppa_wip wip
                           JOIN ppa_wip_values wv ON wip.transno = wv.transno AND wv.currency_type = 'HOME'
                           JOIN entity e ON wip.entity = e.entity
                           JOIN biz_com_periods bp ON e.id = bp.entity_id AND bp.period_type = 'MONTHLY'
                                                      AND wip.transdate BETWEEN bp.start_date AND bp.end_date
                           LEFT JOIN transclass tc ON wip.transclass = tc.transclass
                   WHERE wip.status = 0
                   GROUP BY wip.investment_id, tc.id, bp.id, bp.start_date) wip ON i.id = wip.project_id AND fp.period_id = wip.period_id
                                                                                   AND fp.transaction_class_id = wip.transaction_class_id
WHERE   i.code = 'PRJ0022'
```

Inline Views – Good Example

```
SELECT   i.id, i.code, i.name, fin.transaction_class_id, fin.period_id, fin.period, fin.slice,
         SUM(CASE WHEN fin.slice_type = 'PLAN' THEN fin.slice ELSE 0 END) plan_slice,
         SUM(CASE WHEN fin.slice_type = 'ACTUALS' THEN fin.slice ELSE 0 END) act_slice
FROM     inv_investments i
         JOIN (SELECT  i.id project_id, fd.transaction_class_id, bp.id period_id, bp.start_date period, 'PLAN' slice_type,
                       SUM(c.slice * (finish_date - c.start_date)) slice
               FROM    inv_investments i
                       JOIN fin_plans fp ON i.id = fp.object_id AND i.odf_object_code = fp.object_code
                       JOIN fin_cost_plan_details fd ON fp.id = fd.plan_id
                       JOIN odf_ssl_cst_dtl_cost c ON c.prj_object_id = fd.id
                       JOIN entity e ON i.entity_code = e.entity
                       JOIN biz_com_periods bp ON e.id = bp.entity_id AND bp.period_type = 'MONTHLY'
                                                  AND c.start_date BETWEEN bp.start_date AND bp.end_date
               WHERE fp.plan_type_code = 'FORECAST' AND fp.is_plan_of_record = 1
               GROUP BY i.id, fd.transaction_class_id, bp.id, bp.start_date
               UNION ALL
               SELECT  wip.investment_id project_id, tc.id transaction_class_id, bp.id period_id, bp.start_date period, 'ACTUALS' slice_type,
                       SUM(wv.totalcost) slice
               FROM    ppa_wip wip
                       JOIN ppa_wip_values wv ON wip.transno = wv.transno AND wv.currency_type = 'home'
                       JOIN entity e ON wip.entity = e.entity
                       JOIN biz_com_periods bp ON e.id = bp.entity_id AND bp.period_type = 'MONTHLY'
                                                  AND wip.transdate BETWEEN bp.start_date AND bp.end_date
                       LEFT JOIN transclass tc ON wip.transclass = tc.transclass
               WHERE wip.status = 0
               GROUP BY wip.investment_id, tc.id, bp.id, bp.start_date) fin ON i.id = fin.project_id
WHERE  i.code = 'PRJ0022'
GROUP BY i.id, i.code, i.name, fin.transaction_class_id, fin.period_id, fin.period, fin.slice
```

Analytic Functions

- Replace complex logic with built in functions
- 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
- ROW_NUMBER – Assign a unique sequential value to each row
- SUM – Compute running totals

Analytic Functions – Example Data

Project: Test Project - *Properties - Main - Status Reports*

Filter: System Default

Current	Name	Report Date	Scope
✓	Last Status Report	2/17/17	
	Middle Status Report	2/10/17	
	First Status Report	2/3/17	

Green = 10

Yellow = 20

Red = 30

Analytic Functions – LAG/LEAD

- LAG (value_expression [,offset] [,default]) OVER ([query_partition_clause] order_by_clause)
- LEAD (value_expression [,offset] [,default]) OVER ([query_partition_clause] order_by_clause)

```
SELECT    sr.name status_report,
          sr.cop_report_date,
          LAG(sr.cop_report_date) OVER (PARTITION BY i.id ORDER BY sr.cop_report_date) last_sr,
          LEAD(sr.cop_report_date) OVER (PARTITION BY i.id ORDER BY sr.cop_report_date) next_sr
FROM      inv_investments i
          JOIN odf_ca_cop_prj_statusrpt sr ON i.id = sr.odf_parent_id
WHERE     i.id = 5145296
ORDER BY sr.cop_report_date
```

STATUS_REPORT	COP_REPORT_DATE	LAST_SR	NEXT SR
First Status Report	2/3/2017		2/10/2017
Middle Status Report	2/10/2017	2/3/2017	2/17/2017
Last Status Report Date	2/17/2017	2/10/2017	

Analytic Functions – FIRST_VALUE/LAST_VALUE

- FIRST_VALUE (expression) [RESPECT|IGNORE NULLS] OVER ([analytic_clause])
- LAST_VALUE (expression) [RESPECT|IGNORE NULLS] OVER ([analytic_clause])

```
SELECT    sr.name status_report, sr.cop_report_date,
          sr.cop_scope_status,
          FIRST_VALUE(sr.cop_scope_status) OVER (PARTITION BY i.id ORDER BY sr.cop_report_date)
          first_status,
          LAST_VALUE(sr.cop_scope_status) OVER (PARTITION BY i.id ORDER BY sr.cop_report_date
          RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) last_status
FROM      inv_investments i
          JOIN odf_ca_cop_prj_statusrpt sr ON i.id = sr.odf_parent_id
WHERE     i.id = 5145296
ORDER BY sr.cop_report_date
```

STATUS_REPORT	COP_REPORT_DATE	COP_SCOPE_STATUS	FIRST_STATUS	LAST_STATUS
First Status Report	2/3/2017	30	30	10
Middle Status Report	2/10/2017	20	30	10
Last Status Report Date	2/17/2017	10	30	10

Analytic Functions – RANK/DENSE_RANK

- RANK () OVER ([query_partition_clause] order_by_clause)
- DENSE_RANK () OVER ([query_partition_clause] order_by_clause)

```
SELECT    sr.name status_report,
          sr.cop_report_date,
          sr.cop_scope_status,
          RANK() OVER (PARTITION BY i.id ORDER BY sr.cop_scope_status) status_rank,
          DENSE_RANK() OVER (PARTITION BY i.id ORDER BY sr.cop_scope_status) status_dense_rank
FROM      inv_investments i
          JOIN odf_ca_cop_prj_statusrpt sr ON i.id = sr.odf_parent_id
WHERE     i.id = 5145296
ORDER BY sr.cop_report_date
```

STATUS_REPORT	COP_REPORT_DATE	COP_SCOPE_STATUS	STATUS_RANK	STATUS_DENSE_RANK
First Status Report	2/3/2017	30	3	2
Middle Status Report	2/10/2017	10	1	1
Last Status Report Date	2/17/2017	10	1	1

Analytic Functions – ROW_NUMBER

- ROW_NUMBER () OVER ([query_partition_clause] order_by_clause)

```
SELECT    sr.name status_report,
          sr.cop_report_date,
          CASE WHEN ROW_NUMBER() OVER (PARTITION BY i.id ORDER BY sr.cop_report_date DESC)
                    = 1 THEN 1 END is_current,
          ROW_NUMBER() OVER (PARTITION BY i.id ORDER BY sr.cop_report_date DESC) rnum
FROM      inv_investments i
          JOIN odf_ca_cop_prj_statusrpt sr ON i.id = sr.odf_parent_id
WHERE     i.id = 5145296
ORDER BY sr.cop_report_date
```

STATUS_REPORT	COP_REPORT_DATE	IS_CURRENT	RNUM
First Status Report	2/3/2017		3
Middle Status Report	2/10/2017		2
Last Status Report Date	2/17/2017	1	1

Analytic Functions – SUM

- SUM (expression) OVER (analytic_clause)

```
SELECT    sr.name status_report,
          sr.cop_report_date,
          sr.cop_scope_status,
          SUM(sr.cop_scope_status) OVER (PARTITION BY i.id) total_sum,
          SUM(sr.cop_scope_status) OVER (PARTITION BY i.id ORDER BY sr.cop_report_date) run_sum
FROM      inv_investments i
          JOIN odf_ca_cop_prj_statusrpt sr ON i.id = sr.odf_parent_id
WHERE     i.id = 5145296
ORDER BY sr.cop_report_date
```

STATUS_REPORT	COP_REPORT_DATE	COP_SCOPE_STATUS	TOTAL_SUM	RUN_SUM
First Status Report	2/3/2017	30	60	30
Middle Status Report	2/10/2017	20	60	50
Last Status Report Date	2/17/2017	10	60	60

Analytic Functions – Get Latest Status Report

```
SELECT  i.code,
        sr.*
FROM    inv_investments i
        JOIN (SELECT sr.id,
                     sr.odf_parent_id,
                     sr.cop_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) sr ON i.ID = sr.odf_parent_id AND sr.rnum = 1
```

General Tips

- Only pull data you actually need
  - Eliminate unnecessary JOINs
  - Don’t blindly reuse code
- Try rewriting subqueries to use JOIN
- Limit use of Outer JOINs
- Use UNION ALL instead of UNION if duplicate rows aren’t an issue
- Use bind variables if possible (GEL Scripts)

Pre-Aggregate Data

WIP / Financial Reporting

Project costs by month by cost type

regoUniversity 2017

Let Rego be your guide.

Pre-Aggregate Data

- Pros
  - Huge performance gains
  - Data is pre-aggregated and concise
  - Simplify data access
- Cons
  - Data can be stale
  - Scope of table data is limited
  - More complex to develop

WIP Reporting Strategies

- Staging Table
  - Pre-aggregated data populated by a job
  - Custom table or Clarity object
  - Purge and repopulate data
- Materialized View
  - DB Object that contains query results
  - Refresh Types - Complete vs Fast / Incremental refresh
  - Not an option for on-demand environments

Table Structure

- Clarity Object
  - Viewable through User Interface
  - SaaS compliant
  - Contains extra data elements
  - Dependent tables
  - Cannot be indexed or constrained
  - Primary key derived through sequence <table_name>_S1 (Oracle)
- Custom Table
  - Cannot be viewed through User Interface
  - Not SaaS compliant
  - No extra fields or dependent tables
  - Can be indexed and constrained

Refreshing Data

- Purge
  - Truncate
    - Fast, generates little redo log, deallocates space of deleted rows, resets the high water mark of table, not transaction safe
  - Delete
    - Slow, generates large amount of redo log, does not deallocate space of deleted rows, results in a segmented table, transaction safe
- Populate
  - Row by row
    - Slow, inefficient, generates large amount of redo log, large amount of extra processing
  - Insert into from select
    - Fast, efficient, generates little redo log
- SQL Transaction
  - Perform the purge and populate within a SQL transaction
  - Data is always available

Original Query

```
SELECT    i.code project_code,
          i.name project_name,
          bp.id period_id,
          bp.period_name,
          SUM(CASE wip.cost_type WHEN 'CAPITAL' THEN wv.totalcost ELSE 0 END) capital,
          SUM(CASE wip.cost_type WHEN 'OPERATING' THEN wv.totalcost ELSE 0 END) operating,
          SUM(wv.totalcost) total
FROM      inv_investments i
          JOIN ppa_wip wip ON i.id = wip.investment_id
          JOIN ppa_wip_values wv ON wip.transno = wv.transno AND wv.currency_type = 'HOME'
          JOIN entity e ON wip.entity = e.entity
          JOIN biz_com_periods bp ON e.id = bp.entity_id AND bp.period_type = 'MONTHLY'
                                     AND wip.transdate BETWEEN bp.start_date AND bp.end_date-
1
WHERE     wip.status = 0
GROUP BY i.code,
          i.name,
          bp.id,
          bp.period_name
```

Staging Table

```
CREATE TABLE REGO_WIP_SUMMARY (
   INVESTMENT_ID     NUMBER,
   PERIOD_ID         NUMBER,
   COST_TYPE         VARCHAR2(30),
   QUANTITY          NUMBER,
   TOTAL_COST        NUMBER
)

CREATE INDEX REGO_WIP_SUMMARY_INV_IDX ON REGO_WIP_SUMMARY (INVESTMENT_ID)
CREATE INDEX REGO_WIP_SUMMARY_PER_IDX ON REGO_WIP_SUMMARY (PERIOD_ID)
```

Populate Staging Table

```
INSERT INTO rego_wip_summary (investment_id, period_id, cost_type, quantity, total_cost)
SELECT    wip.investment_id
          bp.id period_id
          wip.cost_type
          SUM(wip.quantity) quantity
          SUM(wv.totalcost) total_cost
FROM      ppa_wip wip
          JOIN ppa_wip_values wv ON wip.transno = wv.transno AND wv.currency_type = 'HOME'
          JOIN entity e ON wip.entity = e.entity
          JOIN biz_com_periods bp ON e.id = bp.entity_id AND bp.period_type = 'MONTHLY'
                                     AND wip.transdate BETWEEN bp.start_date AND bp.end_date-
1
WHERE     wip.status = 0
GROUP BY wip.investment_id,
          bp.id,
          wip.cost_type
```

New Query

```
SELECT   i.id, ws.*
FROM     inv_investments i
         JOIN rego_wip_summary ws ON i.id = ws.investment_id
```

Questions?

regoUniversity 2017

Let Rego be your guide.

Thank You For Attending regoUniversity

Instructions for PMI credits

- Access your account at pmi.org
- Click on **Certification**
- Click on **Maintain My Certification**
- Scroll down to **Report PDU's**
- Click on Course Training (or other appropriate category)
- Enter **Rego Consulting**
- Enter Activity- **Enter Name of Course**
- Enter **Description**
- Enter **Date Started**
- Enter **Date Completed**
- Provide Contact Person **Name of Person to Contact**
- Provide Contact E-Mail **E-Mail of Person to Contact**
- Enter Number of **PDU's Claimed** (1 PDU per course hour)
- Click on the **I agree this claim is accurate box**
- Click **Submit** button

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

Phone
888.813.0444

Email
info@regouniversity.com

Website
www.regouniversity.com