Bell Canada

SABRE / DEAI — Data Engineering for BBM

CS Attack View Enhancement Project

Complete Engineering Portfolio

Every Query • Every Result • Every Decision

Sabbir Hossain

Data Engineer I | SABRE / DEAI Team

Bell Canada | January – March 2026

5	50+	29	10+	19
Workstreams	SQL Queries	ETL Nodes	Source Tables	Investigation Steps

Table of Contents

1. EXECUTIVE SUMMARY

The **CS Attack View Enhancement** is a multi-sprint data engineering initiative at **Bell Canada** to modernize the CS Attack reporting pipeline — a Teradata view (`v_fact_tti_cs_attack`) that tracks telecommunications orders stuck in WIP (Work in Progress) status. The view is consumed by operations teams to investigate why orders are delayed, identify roadblocks, and prioritize resolution.

Originally scoped as a single-field addition (PMPC comments), the project expanded into **five major workstreams** after each investigation revealed additional complexity. The project spans JIRA tickets DEBBM-18705, 18787, 18788, 18822, 18823, 18830, 18920-18923, and 19015 across Sprints 60-62 (January - March 2026).

Metric	Value	Details
Workstreams	5 completed + 2 investigations	PMPC Comments, Task Descriptions, Circuits, Won-to-Order Age Fix, NRC/MRC feasibility
SQL Queries Developed	50+	Feasibility, diagnostics, implementation, validation — all documented in this portfolio
Source Tables Integrated	10+	fact_tti_ord, fact_eom_net_ord_ckt, fact_ipact_net_ord, fact_ipact_ord, fact_pmpc_proj_order, fact_pmpc_task_details, fact_opty_to_ord, V_USF_ACCOUNT_OPTY, TSF_OPTY_LINE_ITEM, IDR tables
JIRA Tickets	8+ tickets	DEBBM-18705, 18787, 18788, 18822, 18823, 18830, 18920-23, 19015
ETL Pipeline Design	29 nodes, zero UWC	SAS DIS 4.905: Extract + SQL Join + Waterfall architecture
WIP Order Coverage	85.8% (comments), 72% (tasks)	85.8% = 236 of 279 WIP orders matched to PMPC source
Row Count Integrity	4,264 = 4,264	Zero row multiplication confirmed across ALL LEFT JOIN additions
RCA Resolution	10-step diagnostic	CHR(26) control character identified via ASCII() scan, resolved via SAS COMPRESS

2. PROJECT ARCHITECTURE & DATA FLOW

System Context

The CS Attack view (`GRP_JARVIS_ANALYSIS.v_fact_tti_cs_attack`) joins data from multiple source systems. Each system owns a different piece of the order lifecycle:

Layer	System	Key Tables in GRP_JARVIS_ANALYSIS	What It Provides
CRM	Salesforce (USFDC)	fact_opty_to_ord (via py_usfdc_opty_line_item_dl)	Opportunity: MRC, NRC, customer, product, opty_owner
Order Mgmt	EOM	fact_eom_net_ord_ckt, fact_eom_ord	Order lifecycle, circuits, milestones, status
Provisioning	IPACT	fact_ipact_net_ord, fact_ipact_ord	Network provisioning, circuit assignments
Delivery	TTI	fact_tti_ord	WIP tracking, build type, roadblocks, ages, jeopardy flags
Process Control	PMPC (Pathway)	fact_pmpc_proj_order, fact_pmpc_task_details	Operational comments, task descriptions, roadblocks
Milestones	SmartPath	v_fact_smtpth_cs_attack	Request IDs, assignment dates, control desk
Billing	IDR (Oracle)	INVC_INVNTRY (via SAS Oracle connection)	Billed MRC per circuit

View Architecture: UNION + Cascading Exclusion Joins

The view uses a **UNION of two queries**. Understanding this structure is critical — it drove every architectural decision in the project:

Top Query (orders with at least one source match): Starts with fact_opty_to_ord as the driver (filtered by opty_campaign_id LIKE '%CS Attack%', ord_num IS NOT NULL). LEFT JOINs to 5 source systems (TTI, ENOC, EO, INO, FIO) plus enrichments (SmartPath, PMPC). Each source system uses ROW_NUMBER() OVER (PARTITION BY ord_num ORDER BY CASE...) for priority-based deduplication. COALESCE picks the highest-priority available source for each output column.

Bottom Query (opportunity-only records): Same fact_opty_to_ord driver, but WHERE ord_num IS NULL. These are opportunities that exist in Salesforce but haven't yet created an order in EOM. All order-derived columns are NULL; result_source = 'opty_only'.

Source Priority Waterfall

Priority	Source	Why This Order	Columns Contributed
1 (highest)	TTI (fact_tti_ord)	TTI is the delivery tracking system — most authoritative for WIP orders	Build type, status, milestone, roadblock (catg+name), ages, regions, jeopardy flag, target days
2	ENOC (fact_eom_net_ord_ckt)	EOM circuit-level data — second most complete source	Same fields. Note: status column is ord_status_ctgry (different spelling from TTI)
3	EO (fact_eom_ord)	EOM order-level — less detail but broader coverage	MAX-only assembly (no priority ranking). Includes RID date.
4	INO (fact_ipact_net_ord)	IPACT network — good for milestones	Only milestone uses priority ranking; everything else is MAX
5 (lowest)	FIO (fact_ipact_ord)	IPACT order — fallback source	MAX-only + province-to-region CASE mapping

COALESCE(TTI_value, ENOC_value, EO_value, INO_value, FIO_value) picks the first non-NULL, implementing the priority waterfall. If an order exists in TTI, TTI columns are used. If not, ENOC is tried, then EO, etc.

Architectural Decision: View → SAS ETL Job

Why the conversion was needed: The circuit cross-join problem. Circuits are not uniquely tied to orders across source systems — the same order can appear in TTI with 3 circuits, in ENOC with 5 circuits, and in IPACT with 2 circuits. A view cannot produce both an order-level result (1 row per order) and a circuit-level result (1 row per order+circuit) without architectural gymnastics that become fragile. The solution: convert to a maintained SAS DIS 4.905 ETL job producing **two**

target tables: `fact_cs_attack` (order-level) and `fact_cs_attack_ckt` (circuit-level).

Non-negotiable rule (from Sr. Manager): Every source table must have an explicit join in the ETL job. No implicit dependencies through views-of-views.

3. WORKSTREAM 1: PMPC COMMENTS INVESTIGATION & DEPLOYMENT

JIRA: DEBBM-18705 (Feasibility, 0.5 SP) → DEBBM-18788 (Development, 1 SP)
Date Range: January 19-27, 2026 (Sprint 60)
Outcome: pmpc_comments field DEPLOYED to v_fact_tti_cs_attack_temp

The business stakeholder needed to see operational comments from Pathway (PMPC) directly in the CS Attack report, instead of manually opening each order in Pathway to investigate why it was stuck. The investigation needed to: (1) find which tables contain comments, (2) verify the data is useful, (3) assess impact on view performance, and (4) implement without breaking the existing view.

Phase 1: Table Discovery

Query 1a — Find all PMPC tables in the database

WHY THIS QUERY: *The JIRA ticket mentioned 'Pathway tables' but didn't specify exact table names. We needed to discover what PMPC tables exist in the analytics schema before we could identify which ones contain comment data.*

```
SELECT DISTINCT TableName
FROM dbc.ColumnsV
WHERE DatabaseName = 'GRP_JARVIS_ANALYSIS'
  AND LOWER(TableName) LIKE '%pmpc%';
```

RESULT: Found 38 PMPC tables including fact_pmpc_proj_order, fact_pmpc_task_details, and various views/staging tables.

Query 1b — Find tables with description-like columns

WHY THIS QUERY: *The BA mentioned a 'description' field in addition to comments. This query found it lives in a DIFFERENT table than comments — a critical finding that prevented us from looking in the wrong table.*

```
SELECT TableName, ColumnName
FROM dbc.ColumnsV
WHERE DatabaseName = 'GRP_JARVIS_ANALYSIS'
  AND TableName LIKE '%pmpc%'
  AND LOWER(ColumnName) LIKE '%desc%';
```

RESULT: Found task_description in fact_pmpc_task_details (NOT in fact_pmpc_proj_order as initially assumed).

Phase 2: Column Identification

Query 2a — Get all columns in fact_pmpc_proj_order

WHY THIS QUERY: *Before building anything, we need to confirm exactly what columns exist. The initial assumption was that 'description' was in this table — this query proved it wrong, preventing a failed implementation.*

```
SELECT ColumnName
FROM dbc.ColumnsV
WHERE DatabaseName = 'GRP_JARVIS_ANALYSIS'
  AND TableName = 'fact_pmpc_proj_order';
```

RESULT: Found 98 columns including 'comments' and 'cust_comments'. NO 'description' column exists in this table. Join key: ord_num.

Query 2b — Get all columns in fact_pmpc_task_details

WHY THIS QUERY: *This confirmed where task_description lives AND revealed the join key is 'order_id' not 'ord_num' — a naming difference that would cause a silent join failure if not caught.*

```
SELECT ColumnName
FROM dbc.ColumnsV
WHERE DatabaseName = 'GRP_JARVIS_ANALYSIS'
  AND TableName = 'fact_pmpc_task_details';
```

RESULT: Found 37 columns including 'task_description', 'order_id' (join key — note: NOT ord_num), task_status, task_created_dt.

Phase 3: Pre-Join Population Analysis

Why pre-join analysis matters: Population rate = % of records with non-null data. If a field is only 5% populated, adding it to the view provides almost no value while adding join overhead. We measure BEFORE joining to CS Attack (entire source table) and AFTER joining (only WIP orders).

Query 3a — Pre-join population: comments

WHY THIS QUERY: *99% population means virtually every record has data — excellent candidate. Median 79 chars is acceptable length (no performance concerns from VARCHAR overflow). We can proceed to join testing.*

```
SELECT
  COUNT(*) AS total_records,
  COUNT(comments) AS records_with_comments,
  CAST(COUNT(comments) AS DECIMAL(10,4)) / COUNT(*) * 100 AS pop_pct,
  PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY CHAR_LENGTH(comments)) AS median_len
FROM GRP_JARVIS_ANALYSIS.fact_pmpc_proj_order;
```

RESULT: 62,667 total records, 61,904 with comments = 99% population rate. Median length: 79 characters.

Query 3b — Pre-join population: cust_comments

WHY THIS QUERY: *Only 18% populated source-wide. Still might be useful if the 18% aligns with WIP orders. But 500-char median is concerning for performance. Need to check WIP-specific population next.*

```
SELECT
  COUNT(*) AS total_records,
  COUNT(cust_comments) AS records_with_cust_comments,
  CAST(COUNT(cust_comments) AS DECIMAL(10,4)) / COUNT(*) * 100 AS pop_pct,
  PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY CHAR_LENGTH(cust_comments)) AS median_len
FROM GRP_JARVIS_ANALYSIS.fact_pmpc_proj_order;
```

RESULT: 11,525 with cust_comments = 18% population rate. Median length: 500 characters.

Query 3c — Pre-join population: task_description

WHY THIS QUERY: *Excellent population. However, ~500K rows for ~60K orders means this is task-level data (1:many). Each order has multiple tasks. This granularity difference will require an aggregation strategy decision.*

```
SELECT
  COUNT(*) AS total_records,
  COUNT(task_description) AS with_task_desc,
  PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY CHAR_LENGTH(task_description)) AS median_len
FROM GRP_JARVIS_ANALYSIS.fact_pmpc_task_details;
```

RESULT: ~500K total rows, ~500K with task_description = ~100% population. Median: 70 characters.

Phase 4: WIP-Specific Join Testing

Query 4a — Join to CS Attack WIP: comments match rate

WHY THIS QUERY: *85.8% match rate is excellent. The 14.2% unmatched are orders too new for Pathway or not yet in the PMPC system. Every matched order has comments — this is the strongest candidate field.*

```
SELECT
  COUNT(DISTINCT A.ord_num) AS wip_orders,
  COUNT(DISTINCT CASE WHEN P.ord_num IS NOT NULL THEN A.ord_num END) AS matched,
  COUNT(DISTINCT CASE WHEN P.comments IS NOT NULL THEN A.ord_num END) AS with_comments
FROM GRP_JARVIS_ANALYSIS.v_fact_tti_cs_attack A
INNER JOIN GRP_JARVIS_ANALYSIS.fact_pmpc_proj_order P ON A.ord_num = P.ord_num
WHERE A.ord_status_catg = 'WIP';
```

RESULT: WIP orders: 279. Matched to PMPC: 236 (85.8%). With comments: 236 (100% of matched).

Query 4b — Join to CS Attack WIP: cust_comments check

WHY THIS QUERY: *This killed cust_comments as a candidate. Despite 18% source-wide population, it's 0% for WIP — the customer-facing comments are cleared/empty for in-progress orders. DO NOT ADD.*

```
SELECT
  COUNT(DISTINCT CASE WHEN P.cust_comments IS NOT NULL THEN A.ord_num END) AS with_cust_comments
FROM GRP_JARVIS_ANALYSIS.v_fact_tti_cs_attack A
INNER JOIN GRP_JARVIS_ANALYSIS.fact_pmpc_proj_order P ON A.ord_num = P.ord_num
WHERE A.ord_status_catg = 'WIP';
```

RESULT: 0 orders with cust_comments for WIP. Confirmed USELESS for WIP orders.

Query 4c — Join to CS Attack WIP: task_description match rate

WHY THIS QUERY: *Near-complete coverage but with 1:many complexity. 1,426 task rows for ~240 orders means we CANNOT just LEFT JOIN directly — it would multiply the view's row count. We need an aggregation strategy.*

```
SELECT
  COUNT(DISTINCT A.ord_num) AS wip_orders,
  COUNT(DISTINCT CASE WHEN T.task_description IS NOT NULL THEN A.ord_num END) AS with_task_desc,
  COUNT(*) AS total_task_rows
FROM GRP_JARVIS_ANALYSIS.v_fact_tti_cs_attack A
INNER JOIN GRP_JARVIS_ANALYSIS.fact_pmpc_task_details T ON A.ord_num = T.order_id
WHERE A.ord_status_catg = 'WIP';
```

RESULT: 97% of matched WIP orders have task_description. 1,426 total task rows (avg ~6 tasks per order). Confirms 1:many granularity.

Phase 5: XMLAGG Implementation (pmpc_comments)

Query 5 — XMLAGG implementation (DEPLOYED)

WHY THIS QUERY: *XMLAGG concatenates all comments per order, pipe-delimited, newest first (ORDER BY transform_dt DESC). LEFT JOIN preserves all existing rows. GROUP BY ensures 1 row per ord_num. VARCHAR(4000) is the safe maximum for Teradata.*

```
-- Added as LEFT JOIN subquery in v_fact_tti_cs_attack_temp:
LEFT JOIN (
  SELECT ord_num,
    TRIM(TRAILING '|' FROM
      XMLAGG(TRIM(comments) || '|'
        ORDER BY transform_dt DESC
      )(VARCHAR(4000))
    ) AS pmpc_comments
  FROM GRP_JARVIS_ANALYSIS.fact_pmpc_proj_order
  GROUP BY ord_num
) PMPC_COM ON OPTY.ord_num = PMPC_COM.ord_num
```

RESULT: SUCCESS. View compiles, runs, returns correct data. pmpc_comments populated for 85.8% of WIP orders. Row count: 4,264 (unchanged).

Field Population Summary

Field	Source Table	Pre-Join Pop%	WIP Pop%	WIP Median Len	Decision
comments	fact_pmpc_proj_order	99% (61,904/62,667)	100% of matched	94 chars	ADD — DEPLOYED
cust_comments	fact_pmpc_proj_order	18% (11,525/62,667)	0% for WIP	N/A	DO NOT ADD
task_description	fact_pmpc_task_details	~100% (~500K)	97% of matched	52 chars	ADD — DEPLOYED (see next section)

Row count integrity: 4,264 before = 4,264 after adding comments. Zero row multiplication from LEFT JOIN + GROUP BY subquery.

4. WORKSTREAM 2: TASK DESCRIPTIONS — RCA & RESOLUTION

JIRA: DEBBM-18830 (RCA, 2 SP) — split from DEBBM-18788 when XMLAGG failed
Date Range: February 2-6, 2026 (Sprint 62)
Outcome: ROW_NUMBER approach DEPLOYED; ETL-level encoding fix IMPLEMENTED

Applying the same proven XMLAGG pattern to task_description triggered **Teradata Error 6706**: "The string contains an untranslatable character." This launched the most complex diagnostic investigation in the project — a systematic 10-step journey from failure to root cause.

Step-by-Step Root Cause Analysis

Step 1 — XMLAGG attempt on task_description (FAILED)

WHY THIS QUERY: *Identical XMLAGG pattern, different input column, different result. This tells us the issue is in the DATA, not the SQL pattern. Something in task_description's content triggers a charset conversion failure.*

```
LEFT JOIN (
  SELECT order_id,
    XMLAGG(TRIM(task_description) || '|'
      ORDER BY task_created_dt DESC
    )(VARCHAR(4000)) AS task_desc_concat
  FROM GRP_JARVIS_ANALYSIS.fact_pmpc_task_details
  GROUP BY order_id
) PMPC_TD ON OPTY.ord_num = PMPC_TD.order_id
```

RESULT: FAILED: Teradata Error 6706 — 'The string contains an untranslatable character.' Same pattern that worked for comments fails on task_description.

Step 2 — Isolate: Can we SELECT task_description at all?

WHY THIS QUERY: *This is the key paradox: individual rows are fine, but XMLAGG aggregation fails. XMLAGG must be doing something internally that triggers the failure — likely a charset conversion.*

```
SELECT task_description
FROM GRP_JARVIS_ANALYSIS.fact_pmpc_task_details
SAMPLE 1000;
```

RESULT: Works perfectly. Individual rows display fine. No visible encoding issues.

Step 3 — Isolate: ROW_NUMBER without aggregation

WHY THIS QUERY: *ROW_NUMBER doesn't aggregate — it just selects. No implicit LATIN→UNICODE conversion is triggered. This could be our fallback if XMLAGG can't be fixed.*

```
SELECT order_id, task_description,
  ROW_NUMBER() OVER (PARTITION BY order_id ORDER BY task_created_dt DESC) AS rn
FROM GRP_JARVIS_ANALYSIS.fact_pmpc_task_details
QUALIFY rn = 1;
```

RESULT: Works. No encoding issue. ROW_NUMBER returns one row per order without any charset conversion.

Step 4 — Isolate: XMLAGG on comments (same pattern, different column)

WHY THIS QUERY: *This eliminates the SQL pattern as the cause. The issue is specific to the task_description column's content. Something in that column's data is untranslatable.*

```
SELECT ord_num,
  XMLAGG(TRIM(comments) || '|' ORDER BY transform_dt DESC)(VARCHAR(4000))
FROM GRP_JARVIS_ANALYSIS.fact_pmpc_proj_order
GROUP BY ord_num
SAMPLE 100;
```

RESULT: Works perfectly. Same XMLAGG pattern on the comments column has zero issues.

Step 5 — Character diagnostic: ASCII scan (ROOT CAUSE FOUND)

WHY THIS QUERY: *CHR(26) is the ASCII 'Substitute' character. It's invisible in query output but XMLAGG's internal LATIN→UNICODE conversion cannot translate it. This is why SELECT works but XMLAGG fails — SELECT doesn't do charset conversion, XMLAGG does.*

```
SELECT DISTINCT
  ASCII(SUBSTR(task_description, pos, 1)) AS char_code,
```

```
  COUNT(*) AS occurrences
FROM GRP_JARVIS_ANALYSIS.fact_pmpc_task_details
CROSS JOIN (
  SELECT ROW_NUMBER() OVER (ORDER BY 1) AS pos
  FROM dbc.ColumnsV SAMPLE 4000
) positions
WHERE pos <= CHAR_LENGTH(task_description)
  AND ASCII(SUBSTR(task_description, pos, 1)) < 32
GROUP BY 1;
```

RESULT: Found CHR(26) — ASCII Substitute character — in 172 rows across 152 orders. This invisible control character is the root cause.

Step 6 — Attempt: OREPLACE cleanup in view

WHY THIS QUERY: *This was surprising. OREPLACE should strip CHR(26) before XMLAGG sees it. But XMLAGG's execution plan processes the aggregation in a way that triggers charset conversion before the cleaned value is used. View-layer cleaning doesn't work.*

```
XMLAGG(TRIM(OREPLACE(task_description, CHR(26), '')) || '|'
  ORDER BY task_created_dt DESC)(VARCHAR(4000))
```

RESULT: FAILED. XMLAGG still fails even after OREPLACE. The internal charset conversion happens AFTER the OREPLACE evaluation.

Step 7 — Attempt: Explicit TRANSLATE

WHY THIS QUERY: *TRANSLATE forces the conversion but doesn't strip the byte — it tries to map CHR(26) to Unicode and fails. The character needs to be REMOVED, not converted.*

```
XMLAGG(TRIM(TRANSLATE(task_description USING LATIN_TO_UNICODE)) || '|'
  ORDER BY task_created_dt DESC)(VARCHAR(4000))
```

RESULT: FAILED. Explicit charset conversion still fails on CHR(26).

Step 8 — Attempt: REGEXP_REPLACE

WHY THIS QUERY: *REGEXP_REPLACE should strip all non-printable ASCII. But Teradata's REGEXP engine itself may trigger charset interpretation before the replacement. Another dead end for view-layer cleaning.*

```
REGEXP_REPLACE(task_description, '[^\x20-\x7E]', '')
```

RESULT: FAILED. Teradata's REGEXP implementation has charset limitations that prevent this approach.

Step 9 — Partial: Exclude affected rows

WHY THIS QUERY: *TRANSLATE_CHK returns non-zero for rows with untranslatable characters. Filtering them out works but loses business data. The stakeholder needs those 152 orders.*

```
WHERE TRANSLATE_CHK(task_description USING LATIN_TO_UNICODE) = 0
```

RESULT: Works but loses 152 orders of data. Not acceptable — we'd be silently dropping data.

Step 10 — SOLUTION: SAS COMPRESS at ETL layer

WHY THIS QUERY: *Sr. Manager directed: 'Sanitation belongs in ETL (Job → Table). The view remains a presentation layer only.' SAS COMPRESS with 'kw' strips non-printable characters while preserving French accented characters (critical for Bell Canada's bilingual data).*

```
/* In SAS DIS job: load_fact_pmpc_task_details */
/* Data cleaning step: */
task_description = COMPRESS(task_description, , 'kw');
/* 'kw' modifier: Keep printable + Whitespace */
/* Strips ALL control characters including CHR(26) */
/* Preserves French: e with accent, e grave, etc. */
```

RESULT: SUCCESS. CHR(26) removed at source before data reaches Teradata. XMLAGG would now work. But ROW_NUMBER was already deployed as the production solution.

Deployed Solution: ROW_NUMBER

Query 9 — ROW_NUMBER implementation (DEPLOYED to production)

WHY THIS QUERY: *ROW_NUMBER avoids aggregation entirely — no charset conversion, no encoding risk. The rn=1 filter guarantees exactly 1 row per order (most recent task). LEFT JOIN preserves all existing rows.*

```
LEFT JOIN (
  SELECT order_id,
    CAST(task_description AS VARCHAR(4000)) AS pmpc_task_description
  FROM (
    SELECT order_id, task_description,
      ROW_NUMBER() OVER (
        PARTITION BY order_id
        ORDER BY task_created_dt DESC
      ) AS rn
    FROM GRP_JARVIS_ANALYSIS.fact_pmpc_task_details
  ) sub
  WHERE rn = 1
) PMPC_TD ON OPTY.ord_num = PMPC_TD.order_id
```

RESULT: SUCCESS. Latest task description per order. 72% WIP coverage (239/332). Zero encoding errors. Row count: 4,264 = 4,264.

Alternative: Three-Column Approach (v2) — Available

Query 10 — Three-column pivot (available in v_fact_tti_cs_attack_temp_v2)

WHY THIS QUERY: *MAX(CASE WHEN rn=N...) with GROUP BY pivots task rows into columns. Gives stakeholders top 3 reasons an order is stuck without any XMLAGG. Stakeholders can choose between 1-column and 3-column approaches.*

```
LEFT JOIN (
  SELECT order_id,
    MAX(CASE WHEN rn=1 THEN CAST(task_description AS VARCHAR(4000)) END) AS pmpc_task_desc_1,
    MAX(CASE WHEN rn=2 THEN CAST(task_description AS VARCHAR(4000)) END) AS pmpc_task_desc_2,
    MAX(CASE WHEN rn=3 THEN CAST(task_description AS VARCHAR(4000)) END) AS pmpc_task_desc_3
  FROM (
    SELECT order_id, task_description,
      ROW_NUMBER() OVER (PARTITION BY order_id ORDER BY task_created_dt DESC) AS rn
    FROM GRP_JARVIS_ANALYSIS.fact_pmpc_task_details
  ) sub WHERE rn <= 3
  GROUP BY order_id
) PMPC_TD ON OPTY.ord_num = PMPC_TD.order_id
```

RESULT: Available. desc_1: 72% (239 orders), desc_2: 62% (206), desc_3: 52.7% (175). No XMLAGG = no encoding risk.

Validation Queries

Query 11 — Row count integrity check

WHY THIS QUERY: *This is the CRITICAL validation. If the count changed, a LEFT JOIN is creating duplicates (assembly has multiple rows per ord_num). 4,264 = 4,264 proves all subqueries produce exactly 1 row per ord_num.*

```
-- Before adding PMPC:
SELECT COUNT(*) FROM GRP_JARVIS_ANALYSIS.v_fact_tti_cs_attack;
-- After:
SELECT COUNT(*) FROM GRP_JARVIS_ANALYSIS.v_fact_tti_cs_attack_temp;
```

RESULT: Both return 4,264. Row count parity confirmed. No row multiplication from LEFT JOINs.

5. WORKSTREAM 3: NRC INVESTIGATION — SOURCE DISCOVERY

JIRA: DEBBM-18922 (Feasibility, 0.5 SP), DEBBM-18923 (Development, 2 SP)
Status: Feasibility Complete — Blocked on SELECT Access to GRP_BDM_TABLE/GRP_BDM_ETL
Problem: NRC was manually tracked in Excel. Stakeholder needs it automated from Salesforce.

Why NRC is hard: The Salesforce field Total_Non_Recurring_Charge_NRC__c was confirmed in the UI, but NRC does NOT exist in fact_opty_to_ord (the primary opportunity table in the view). MRC is there as opty_mrc, but whoever built the ETL for fact_opty_to_ord chose not to include NRC. We had to hunt for it across multiple schemas.

Q1A — Direct search: Does NRC exist in fact_opty_to_ord?

WHY THIS QUERY: *The obvious first check. fact_opty_to_ord has opty_mrc (FLOAT, 8 bytes) but no NRC counterpart. This means NRC was deliberately excluded from the ETL that builds this table — we need to find it elsewhere.*

```
SELECT ColumnName, ColumnType, ColumnLength, Nullable
FROM dbc.ColumnsV
WHERE DatabaseName = 'GRP_JARVIS_ANALYSIS'
  AND TableName = 'fact_opty_to_ord'
  AND UPPER(ColumnName) LIKE '%NRC%';
```

RESULT: Zero rows returned. No column containing 'NRC' exists in fact_opty_to_ord.

Q1B — Broader search with multiple naming patterns

WHY THIS QUERY: *Exhaustive search. NRC could be labeled 'non_recurring', 'one_time_charge', or 'installation_fee'. None exist. Confirms we need to look outside this table.*

```
SELECT ColumnName, ColumnType, ColumnLength
FROM dbc.ColumnsV
WHERE DatabaseName = 'GRP_JARVIS_ANALYSIS'
  AND TableName = 'fact_opty_to_ord'
  AND (UPPER(ColumnName) LIKE '%NRC%'
    OR UPPER(ColumnName) LIKE '%NON_RECUR%'
    OR UPPER(ColumnName) LIKE '%ONE_TIME%'
    OR UPPER(ColumnName) LIKE '%INSTALL%');
```

RESULT: Zero rows. No NRC under any naming convention.

Q2A — Search ALL tables in GRP_JARVIS_ANALYSIS for NRC

WHY THIS QUERY: *These NRC columns represent per-order or per-circuit charges from the provisioning systems. They're NOT the Salesforce opportunity-level NRC the stakeholder needs. We need to find the Salesforce NRC landing spot.*

```
SELECT DatabaseName, TableName, ColumnName, ColumnType, ColumnLength
FROM dbc.ColumnsV
WHERE DatabaseName = 'GRP_JARVIS_ANALYSIS'
  AND UPPER(ColumnName) LIKE '%NRC%';
```

RESULT: 29 rows returned. NRC exists in fact_ipact_ord (ord_dscntd_nrc), fact_eom_ord_item (item_nrc), fact_data_ord (ord_nrc), and many others — but ALL are at order/circuit level, not opportunity level.

Q2D — Cross-database search: opportunity tables with NRC columns

WHY THIS QUERY: *This is the breakthrough query. Searching across ALL databases for tables that have both 'opty' in the name AND 'NRC' in a column. Found 4 candidate sources we didn't know existed.*

```
SELECT DatabaseName, TableName, ColumnName
FROM dbc.ColumnsV
WHERE UPPER(TableName) LIKE '%OPTY%'
  AND (
    UPPER(ColumnName) LIKE '%NRC%'
    OR UPPER(ColumnName) LIKE '%NON_RECUR%'
  );
```

RESULT: 5 hits across 3 schemas: GRP_BDM_TABLE.V_USF_ACCOUNT_OPTY (NRC), GRP_BDM_ETL.TSF_OPTY_LINE_ITEM (Non_Recurring_Charge__c), GRP_JARVIS_USFDC.py_usfdc_oppty_line_item_dl (sf_non_recurring_chrg_c), SMBBI_VIEW.V_Fact_OptyItems (Nrc).

Q3A — Explore V_USF_ACCOUNT_OPTY (PRIMARY candidate)

WHY THIS QUERY: *This is our best candidate. EOM_OPTY_NO probably maps to our opty_num. Has both NRC and MRC. Product/LOB granularity means multiple rows per opportunity — we'd need SUM(NRC) GROUP BY EOM_OPTY_NO.*

```
SELECT ColumnName, ColumnType, ColumnLength
FROM dbc.ColumnsV
WHERE DatabaseName = 'GRP_BDM_TABLE'
  AND TableName = 'V_USF_ACCOUNT_OPTY';
```

RESULT: 35 columns. Key: EOM_OPTY_NO (likely join key to opty_num), NRC, MRC, Product, LOB, Practice, Account_Name. Product/LOB columns indicate product-level granularity.

Q4A — Explore TSF_OPTY_LINE_ITEM (BACKUP candidate)

WHY THIS QUERY: *Problem: OpportunityId is Salesforce's 18-character ID (like '006XXXXXXXXXXXXXXX'), NOT our integer opty_num. We'd need a bridge table to map between them. Also: __c suffix means raw Salesforce custom field — less curated.*

```
SELECT ColumnName, ColumnType, ColumnLength
FROM dbc.ColumnsV
WHERE DatabaseName = 'GRP_BDM_ETL'
  AND TableName = 'TSF_OPTY_LINE_ITEM';
```

RESULT: 71 columns. Key: Non_Recurring_Charge__c (FLOAT, 8), OpportunityId (VARCHAR 25 — SF 18-char ID), Product_MRC__c, Product_TCV__c, IsDeleted flag.

NRC Source Comparison Matrix

Attribute	V_USF_ACCOUNT_OPTY (PRIMARY)	TSF_OPTY_LINE_ITEM (BACKUP)	py_usfdc_oppty_line_item_dl (FALLBACK)	V_Fact_OptyItems (DEPRIORITIZED)
Database	GRP_BDM_TABLE	GRP_BDM_ETL	GRP_JARVIS_USFDC	SMBBI_VIEW
NRC Column	NRC	Non_Recurring_Charge__c	sf_non_recurring_chrg_c	Nrc
Join Key	EOM_OPTY_NO (likely = opty_num)	OpportunityId (SF 18-char ID)	Unknown	Opty_Key (surrogate)
Granularity	Product/LOB (SUM needed)	Line-item (SUM needed)	Line-item (SUM needed)	Item level
Access	NO — blocked	NO — blocked	YES	Unknown
Complexity	LOW	MEDIUM	MEDIUM	HIGH

Status: Blocked on SELECT access to GRP_BDM_TABLE and GRP_BDM_ETL. Access request submitted. Once granted, validation queries Q7A-Q10 will confirm the join key mapping and NRC population rate.

6. WORKSTREAM 4: MRC INVESTIGATION

JIRA: DEBBM-18822 (Feasibility, 0.5 SP), DEBBM-18823 (Development — blocked)
Problem: Stakeholder's team searched s_asset, s_order, s_order_item in EOM — could not find MRC.

Q1 — Check existing CS Attack view for MRC

WHY THIS QUERY: *Before building anything new, check what already exists. opty_mrc is already in the view from Salesforce. The stakeholder may not have realized this. But they might want IDR billed MRC (actual invoiced rate), which is different.*

```
SELECT ord_num, ckt_num, cust_name, ord_status_catg, opty_mrc
FROM GRP_JARVIS_ANALYSIS.v_fact_tti_cs_attack_ckt
WHERE opty_mrc IS NOT NULL
SAMPLE 20;
```

RESULT: 20 rows returned. opty_mrc populated with values: 535, 1000, 3117, 998, 9115, 5355, 6225, 20173. Salesforce MRC already exists in the view!

Q2 — JIRA test case validation

WHY THIS QUERY: *Validation against the JIRA test case. The stakeholder cited this order as needing $535/month MRC. It's already in the view via opty_mrc. Confirmed the existing data is correct.*

```
SELECT ord_num, ckt_num, opty_mrc, cust_name
FROM GRP_JARVIS_ANALYSIS.v_fact_tti_cs_attack_ckt
WHERE ord_num = '3-41461381514';
```

RESULT: 2 rows returned. Both circuits show opty_mrc = 535. Matches expected $535/month from the JIRA ticket.

Q6 — Attempt to query IDR table directly

WHY THIS QUERY: *IDR data exists in Teradata but we can't access it. However, SAS CAN access the Oracle source directly via LIBNAME. This means IDR MRC can only be added via the SAS ETL path, not via a view-layer SQL join.*

```
SELECT TOP 10 SUBSCRIBER_NUM, MONTHLY_RATE, BILLING_NUMBER, BILLED_DATE
FROM GRP_SHR.IRD_INVC_INVNTRY_ACTIVITY;
```

RESULT: FAILED: Error 5315 — user does not have SELECT access to GRP_SHR.IRD_INVC_INVNTRY_ACTIVITY.BILLED_DATE.

Q8-Q9 — CS Attack vs BB Staging overlap analysis

WHY THIS QUERY: *This is the most important finding in the MRC investigation. The existing BB staging table (where IDR MRC is loaded) is an entirely different population from CS Attack. Only 0.2% overlap, and even those 9 have null IDR MRC. We cannot reuse the existing pipeline — CS Attack needs its own IDR MRC extraction.*

```
SELECT
  CASE WHEN B.ord_num IS NOT NULL THEN 'Match' ELSE 'No Match' END AS match_status,
  COUNT(*) AS order_count
FROM GRP_JARVIS_ANALYSIS.v_fact_tti_cs_attack A
LEFT JOIN GRP_JARVIS_STAGING.STG_CS_SP_PM_BB_ORD_CKT B ON A.ord_num = B.ord_num
GROUP BY 1;
```

RESULT: Match: 9 orders. No Match: 4,672 orders. Only 0.2% overlap. All 9 matched orders have Salesforce MRC but ALL have null IDR MRC.

SAS ETL Pattern for IDR MRC (Reverse-Engineered)

Reviewed the ext_cs_sp_pm_bb_ord_ckt SAS job (12,000+ lines of generated code) and documented the 3-step MRC pattern for replication in the CS Attack pipeline:

Step	SAS Code Pattern	Key Logic
EXT_MRC	PROC SQL; CREATE TABLE work.EXT_MRC AS SELECT DISTINCT MONTHLY_RATE, SUBS_NO, MAX(LAST_UPDT_DT) AS LAST_UPDT_DT FROM skyidr.INVC_INVNTRY...	Oracle connection: LIBNAME skyidr ORACLE PATH=SKY_IDR SCHEMA=IDR AUTHDOMAIN='SKY_IDR_AUTH'. Filters: INVC_SYSTEM_CD='NM1', SP_CD='BLCO', non-zero MRC, non-null circuit, recent bills.
SUM_MRC	PROC SQL; CREATE TABLE work.SUM_MRC AS SELECT SUM(MONTHLY_RATE) AS MONTHLY_RATE, SUBS_NO FROM work.EXT_MRC WHERE LAST_UPDT_DT = (SELECT MAX(...)) GROUP BY SUBS_NO	Filters to most recent billing record per circuit (MAX LAST_UPDT_DT), then SUMs rate components. A circuit can have multiple billing line items.

Step	SAS Code Pattern	Key Logic
JOIN_MRC	LEFT JOIN work.SUM_MRC ON ord_ckt_num IS NOT NULL AND ord_ckt_num = SUBS_NO	Output: ord_discounted_mrc (aliased from MONTHLY_RATE). LEFT JOIN preserves orders without billing data.

7. WORKSTREAM 5: CIRCUITS & WON-TO-ORDER FIX

Circuit-Level View: Why UNION Beats Waterfall

The circuit view (v_fact_tti_cs_attack_ckt) needs to show every circuit associated with each order. The problem: the same order appears in multiple source systems with different circuit sets. Two approaches were tested:

Priority Waterfall (abandoned): If an order has circuits in TTI, use only TTI circuits. Otherwise try ENOC, then INO, then FIO. Problem: TTI might have 2 circuits while ENOC has 5 for the same order. The waterfall throws away 3 circuits because TTI is higher priority. Coverage dropped.

UNION + Deduplication (adopted): Append ALL circuits from all 4 sources, then deduplicate on (ord_num, ckt_num). Result: ~95.5% circuit coverage. Maximum coverage preserved.

Source	Circuit Column	Coverage Role
fact_tti_ord	ckt_num	Primary — TTI delivery tracking circuits
fact_eom_net_ord_ckt	ckt_num	EOM — order management circuits
fact_ipact_net_ord	ckt_num	IPACT — network provisioning circuits
fact_ipact_ord	ckt_num	IPACT — order-level circuits (fallback)

Won-to-Order Age Fix (DEBBM-19015)

The won-to-order age calculation (how long between opportunity 'won' and order creation) returned NULL for pending EOM lines where opty_closed_date exists but ord_create_dt IS NULL. Fix:

```
-- Original: only worked when both dates exist
CASE WHEN opty_closed_date IS NOT NULL AND ord_create_dt IS NOT NULL
  THEN ord_create_dt - opty_closed_date END

-- Fixed: added WHEN branches for pending lines
CASE WHEN opty_closed_date IS NOT NULL AND ord_create_dt IS NOT NULL
  THEN ord_create_dt - opty_closed_date
  WHEN opty_closed_date IS NOT NULL AND ord_create_dt IS NULL
  THEN CURRENT_DATE - opty_closed_date
END AS won_to_ord_age
```

Using TODAY() as the end date provides a running age metric. Stakeholders can now see how long an opportunity has been 'won' but not yet created as an order.

8. SAS DIS 4.905 ETL BUILD PLAN — 29 NODES, ZERO UWC

The view-to-job conversion design. Replaces v_fact_tti_cs_attack with a maintained SAS DIS 4.905 ETL job (`load_fact_cs_attack`) producing two target tables. Uses **zero User Written Code (UWC) nodes** — all logic through native transforms.

Three-Layer Architecture

Layer	Purpose	DIS Transform	Output
1. EXTRACT	Pull from Teradata. Computed priority columns in Mapping tab.	Extract	Row-level (multi-row per order)
2. ASSEMBLY	GROUP BY ord_num. MIN-concat priorities, CATX strings, MAX scalars.	SQL Join (1 input)	1 row per ord_num
3. WATERFALL	Left anchor on OPTY. LEFT JOIN all assemblies. COALESCE selects best source.	SQL Join (13+ inputs)	1 row per ord_num

Pattern A: MIN-Concat Trick (Replaces ROW_NUMBER)

The view uses ROW_NUMBER() OVER (PARTITION BY ord_num ORDER BY CASE...) = 1. SAS DIS has no ROW_NUMBER. The MIN-concat trick achieves identical results:

```
-- Step 1 (Extract Mapping): Compute priority number
CASE WHEN ord_build_type = 'Complex' THEN 1
     WHEN ord_build_type = 'Standard' THEN 2
     ... ELSE 9 END AS bt_priority

-- Step 2 (Assembly SQL Join, GROUP BY ord_num):
SUBSTR(MIN(PUT(bt_priority, Z2.) || COALESCE(ord_build_type, '')), 3) AS ord_build_type

-- How it works: Concatenates '01Complex', '02Standard', '09Unknown'
-- MIN picks '01Complex' (alphabetically smallest)
-- SUBSTR(..., 3) strips the '01' prefix → returns 'Complex'
-- Z2. format is CRITICAL: zero-pads so '09' < '10' (without it, '9' > '10')
```

Pattern B: CATX Pivot (Replaces XMLAGG)

```
-- Step 1: Assign row numbers (SQL Join, no GROUP BY):
(SELECT COUNT(DISTINCT b.cust_region) FROM input b
 WHERE b.ord_num = a.ord_num AND b.cust_region <= a.cust_region) AS region_rn

-- Step 2: CATX pivot (Assembly, GROUP BY ord_num):
CATX(',',
  MAX(CASE WHEN region_rn=1 THEN STRIP(cust_region) END),
  MAX(CASE WHEN region_rn=2 THEN STRIP(cust_region) END),
  MAX(CASE WHEN region_rn=3 THEN STRIP(cust_region) END)
) AS cust_region_concat
-- CATX skips NULLs automatically. 'ON,QC' for 2-region order.
```

Roadblock: Dual-Column MIN-Concat

The roadblock priority selects BOTH active_rdblk_catg AND active_rdblk_name from the same winner row. Solved by encoding both into one string with a ||| delimiter:

```
-- Encode both values:
MIN(PUT(rr_priority, Z2.) || STRIP(COALESCE(active_rdblk_catg,''))
   || '|||' || STRIP(COALESCE(active_rdblk_name,'')))

-- Split for catg: SUBSTR before '|||' delimiter
-- Split for name: SUBSTR after '|||' delimiter
-- 29-level CASE: Construction Charges(1) → Site Requirements(3-10) →
-- Site Access(11-19) → Customer Not Ready(20-26) → COVID(27-28) → ELSE(29)
```

Complete Node Inventory (29 Nodes)

#	Node	Type	Purpose
1	EXTRACT_OPTY	Extract	fact_opty_to_ord: campaign filter, ord_num IS NOT NULL (driver)
2	EXTRACT_OPTY_ONLY	Extract	fact_opty_to_ord: campaign filter, ord_num IS NULL (UNION bottom)
3	EXTRACT_TTI	Extract	fact_tti_ord + 4 computed priority columns (bt/sc/ml/rr)
4	EXTRACT_ENOC	Extract	fact_eom_net_ord_ckt + 4 priorities (note: ord_status_ctgry not ord_status_catg)
5	EXTRACT_EO	Extract	fact_eom_ord (no priorities — MAX-only assembly)
6	EXTRACT_INO	Extract	fact_ipact_net_ord + ml_priority only
7	EXTRACT_FIO	Extract	fact_ipact_ord (no priorities — MAX + region CASE)
8	EXTRACT_SMTPTH	Extract	v_fact_smtpth_cs_attack
9	EXTRACT_PMPC	Extract	fact_pmpc_proj_order
10	EXTRACT_TASK	Extract	fact_pmpc_task_details
11-13	Placeholders	Extract	Reserved: NRC, IDR MRC, Milestones (blocked)
14	SQL_TTI_ASSEMBLY	SQL Join	GROUP BY ord_num: MIN-concat 4 priorities + CATX regions + 12 MAX columns. Validated: 13,698→12,616
15	SQL_ENOC_ASSEMBLY	SQL Join	Same as TTI. Critical: ord_status_ctgry spelling. Prefix: enoc_
16	SQL_INO_ASSEMBLY	SQL Join	ml_priority MIN-concat + MAX for all other fields
17	SQL_EO_ASSEMBLY	SQL Join	GROUP BY + MAX only (no ranks in view for EO)
18	SQL_FIO_ASSEMBLY	SQL Join	MAX + province-to-region CASE: ON,QC→Central; MB..BC→Western; NL..NB→Atlantic
19	SQL_SMTPTH_ASSEMBLY	SQL Join	CATX pivot for req_ids + MIN/MAX dates + MAX flags
20	SQL_PMPC_ASSEMBLY	SQL Join	CATX pipe-delimited newest-first comments (up to 20 slots)
21	SQL_TASK_ASSEMBLY	SQL Join	COMPRESS('kw') + CATX top 5 task descriptions. RENAME order_id→ord_num
22	CIRCUIT_APPEND	Append	UNION ord_num + ckt_num from TTI, ENOC, INO, FIO
23	SORT_CKT_DEDUP	Sort	NODUPKEY by ord_num + ckt_num
24	AGG_CKT_COUNT	SQL Join	GROUP BY ord_num, COUNT(*) AS circuit_count
25	PREP_CKT_DETAIL	Extract	Pass-through for circuit fan-out
26	WATERFALL_JOIN	SQL Join	13+ inputs: OPTY anchor + all assemblies. COALESCE columns. result_source CASE.
27	APPEND_UNION	Append	WATERFALL + OPTY_ONLY
28	LOAD_FACT	TD Loader	Truncate & Reload → fact_cs_attack (PI: ord_num)
29	CKT_FANOUT+LOAD	SQL+Loader	Fan out → fact_cs_attack_ckt (PI: ord_num + ckt_num)

Build Order & Validation Gates

Step	Build	Validation Gate (must pass before proceeding)
1	All 13 Extracts (DONE)	Row counts match source tables. Priority columns populated.
2	Add priorities to TTI/ENOC/INO	bt_priority 1-9, sc_priority 1-5, ml_priority 1-7, rr_priority 1-29. Spot-check 5 orders.
3	SQL_TTI_ASSEMBLY	1 row per ord_num. COUNT(*)=COUNT(DISTINCT ord_num). 5 orders match view.
4	SQL_ENOC_ASSEMBLY	Same. Watch ord_status_ctgry spelling.
5	SQL_INO, EO, FIO	1 row per ord_num. MAX columns correct.
6	SMTPTH, PMPC, TASK	Concatenation output correct. No CHR(26). French accents preserved.

Step	Build	Validation Gate (must pass before proceeding)
7	Circuit pipeline	circuit_count per order correct.
8	WATERFALL_JOIN	CRITICAL: row count = EXTRACT_OPTY count. 5-order spot-check against live view.
9	APPEND + Loaders	Final row count = WATERFALL + OPTY_ONLY. Target table loaded.

9. ENGINEERING METHODOLOGY & KEY LEARNINGS

Systematic 'Trace the Data' Methodology

- **Start from the base table:** Work backwards through pipeline layers. Don't assume — verify with dbc.ColumnsV and SHOW VIEW DDL.
- **Check population rates before AND after joins:** A 99% populated source can drop to 0% for your specific filter (cust_comments for WIP).
- **Validate join keys and assess granularity:** Is it 1:1 or 1:many? Will the join multiply rows? (fact_pmpc_task_details is 1:many with ~6 tasks per order.)
- **Build diagnostic queries first:** Understand data shape and edge cases before writing production code. The ASCII() scan for CHR(26) was diagnostic-first.
- **Test incrementally:** Validate each node in isolation (row count + spot-check), then connect. The SQL_TTI_ASSEMBLY was validated at 12,616 rows before building ENOC.
- **Document everything:** JIRA comments follow structured template: Changes Implemented → Engineering Findings → Next Steps → Risks/Blockers.

Key Engineering Learnings

Learning	Detail	From
Build from actual DDL	All early plan errors came from building against assumed view structure. SHOW VIEW DDL is the only authoritative source.	ETL Build
XMLAGG is fragile across charsets	Implicit LATIN→UNICODE conversion fails on CHR(26) even though SELECT displays fine. Always test with full dataset.	PMPC RCA
View is not ETL	Views are presentation layers. Data quality (encoding) belongs in ETL. View-layer sanitization fails under aggregation.	PMPC RCA
Same pattern ≠ same result	XMLAGG on comments works; XMLAGG on task_description fails. Column CONTENT determines success.	PMPC RCA
CHR(26) is invisible	Control characters don't display. Need ASCII() scan to detect. TRANSLATE_CHK confirms but doesn't fix.	PMPC RCA
UNION beats waterfall for circuits	Priority waterfall drops circuits when higher-priority sources are less complete. UNION preserves maximum coverage.	Circuits
PROC RANK unsuitable for dedup	Assigns rank 1 to ALL tied rows within a group. ROW_NUMBER required for single-row selection.	ETL Build
0.2% overlap = separate pipelines	Cross-referencing CS Attack vs BB staging proved they're completely different populations. Can't reuse existing MRC pipeline.	MRC
Z2. format is critical	MIN-concat without zero-padding: '9'>'10' alphabetically. PUT(priority, Z2.) produces '09'<'10'. Wrong format = wrong winners.	ETL Build
Correlated subqueries can't go in Extracts	Row-numbering references other rows in result set. Must be in separate SQL Join node or embedded in Assembly.	ETL Build

Tools & Technologies

Tool	Usage in This Project
SAS Data Integration Studio 4.905	Primary ETL build: Extract, SQL Join (GROUP BY), Sort (NODUPKEY), Append, TD Loader. 29-node pipeline design.
Teradata SQL	50+ queries across GRP_JARVIS_ANALYSIS, GRP_JARVIS_STAGING, GRP_JARVIS_USFDC, GRP_BDM_TABLE, GRP_BDM_ETL, GRP_SHR, SMBBI_VIEW.
Oracle (via SAS LIBNAME)	IDR billing access: LIBNAME skyidr ORACLE PATH=SKY_IDR SCHEMA=IDR AUTHDOMAIN='SKY_IDR_AUTH'. For MRC per circuit.
JIRA	8+ tickets: DEBBM-18705, 18787, 18788, 18822, 18823, 18830, 18920-23, 19015. Sprints 60-62.
Confluence	Technical documentation: view DDL analysis, architecture decisions, ETL patterns.

Tool	Usage in This Project
Excel	Investigation runbooks: multi-tab format (Queries, Results, Samples, Comparisons, Recommendations).

Project Summary

Deliverable	Status	Impact
PMPC Comments	DEPLOYED	85.8% WIP orders have timestamped fulfillment logs. XMLAGG newest-first, pipe-delimited.
Task Descriptions	DEPLOYED	72% WIP orders show latest task. CHR(26) encoding issue resolved via 10-step RCA.
Three-Column Alt (v2)	AVAILABLE	Top 3 task descriptions in separate columns. No XMLAGG = no encoding risk.
Circuit-Level View	DEPLOYED	~95.5% circuit coverage. UNION across 4 sources beats priority waterfall.
Won-to-Order Age Fix	DEPLOYED	Running age for pending EOM lines. Previously returned NULL.
NRC Investigation	COMPLETE	4 sources mapped across 3 schemas. Primary: V_USF_ACCOUNT_OPTY. Blocked on access.
MRC Investigation	COMPLETE	Two MRC sources documented. 0.2% pipeline overlap proved. SAS ETL pattern captured.
29-Node ETL Pipeline	DESIGNED	Zero-UWC SAS DIS build plan. MIN-concat + CATX pivot + Waterfall COALESCE.
ETL Encoding Fix	IMPLEMENTED	SAS COMPRESS('kw') strips CHR(26) at source. French characters preserved.

Sabbir Hossain

Data Engineer I | SABRE / DEAI Team | Bell Canada
January – March 2026 | Sprints 60-62