Protocol – Silver Layer Build Database

Overall silver-layer workflow

1. Explore & understand sources (in bronze)

1.1. Explore tables (bronze schema)

For each table, always:

- Use `SELECT TOP 1000` (or similar) to avoid scanning millions of rows.
- Take notes of column types and keys.

Table: `bronze.crm_cust_info`

1. Run:

```
  SELECT TOP 1000 * FROM bronze.crm_cust_info;
   ```

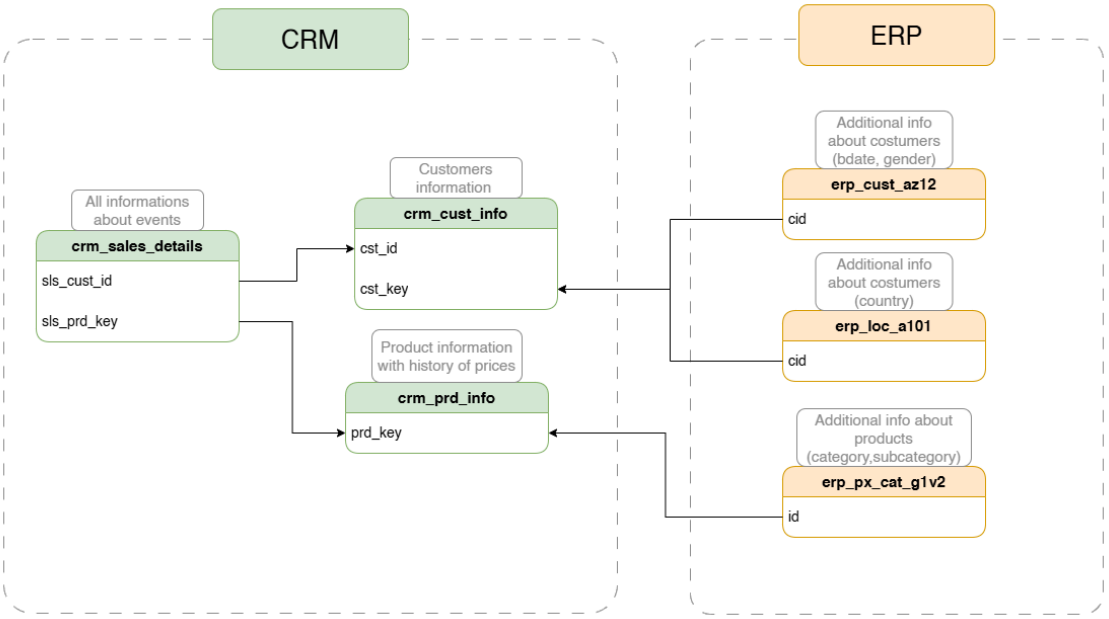

2. Identify key fields:
   - Technical PK: `cust_id` (example name; use actual from video).
   - Business key: `cust_key` (customer number).

3. In your diagramming tool (e.g., PowerPoint, Miro, Draw.io):

   - Draw a table box “`crm_cust_info`”.
   - Inside list only **primary key** (technical ID).
   - Add a friendly label above it like “Customer information”.

4. Do for all tables
```

- Explore each source table, understand columns, relationships, and business meaning.

2. Design & document integration model

#### Build Data integration diagram

1. Draw arrows between the primary keys which are referring to each other
2. Group tables based on the source
```

RESULT



- Draw tables and relationships (per source system), including join keys. (Draw.io)

3. Create silver DDL (schemas & tables)

Define silver layer specs & DDL
1.1. Silver layer specifications
Write these rules explicitly:

- Goal: **clean & standardized data** prepared for the gold layer (analytics).
- Loading pattern: **full load**
  - For each silver table: `TRUNCATE` then `INSERT` from bronze (no incremental logic yet).
- Scope:
  - Apply heavy transformations: cleansing, trimming, normalization, standardization, derived columns, enrichment, type casting.
  - **Do not** introduce new business data models here; structure remains close to source.

1.2. Build DDL for silver tables

1. Start from bronze DDL script.
2. Replace schema: `"bronze."` → `"silver."`.
3. Add metadata column to each table for later troubleshooting
   - For each `CREATE TABLE silver.<table>` add at the end:

```
dwh_create_date  DATETIME2 DEFAULT GETDATE()
-- dwh = created by data_enginner
-- it helps in a case of troubleshooting
```

1.3 Adjust silver DDL after creating new data columns or changing data types in table

> **Danger**
> Reflect this change in your repo DDL file for silver.

- Start from bronze DDL, adjust schema to `silver` and add metadata columns.

4. Detect data-quality issues in bronze

Data-quality checks for `bronze.crm_cust_info`

Now build a **quality-check protocol** before writing transformations.

Primary key uniqueness & nulls

1. Check duplicates and nulls in PK:

```
SELECT cust_id,
COUNT(*) AS cnt
FROM bronze.crm_cust_info
GROUP BY cust_id
HAVING COUNT(*) > 1 OR cust_id IS NULL;
```

2. Interpretation:
   - Any rows returned = data-quality issue (duplicates or missing PK).

Focus on one problematic key

1. Pick one `cust_id` from the bad list and inspect:
2. Examine rows

Use window function to keep only latest version

1. Build a ranked query:

```
SELECT *,
RANK() OVER(PARTITION BY cst_id ORDER BY cst_create_date DESC) as
last_date
FROM bronze.crm_cust_info
WHERE cst_id = 29466
```

Logic:

- For `cust_id`, row with `flag_last = 1` is the most recent; `2,3,…` are older duplicates.

2. Check duplicate rows to be discarded:

```
SELECT
*
FROM
(SELECT *,
RANK() OVER(PARTITION BY cst_id ORDER BY cst_create_date DESC) as
last_date
FROM bronze.crm_cust_info
WHERE cst_id = 29466)t
WHERE last_date = 1
```

3. This pattern will be embedded into the silver SELECT later without `WHERE cst_id = 29466`

Detect unwanted spaces on strings

1. For `first_name`:

```
SELECT
cst_firstname
FROM bronze.crm_cust_info
where cst_firstname != TRIM(cst_firstname);
```

- Any row returned = leading or trailing spaces you should remove.

Coded low-cardinality values (gender, marital status)

1. Check distinct values for gender and marital status
2. Decide on project standard:
   - No abbreviations in silver.
   - Map to full friendly values, handle nulls as `N/A` or `Unknown`.

- Run targeted checks per table: PK uniqueness, spaces, coded values, invalid dates, etc.

5. Write cleansing transformation SELECT for each table

Build cleansing SELECT for `crm_cust_info` (silver) EXAMPLE
Goal: one big SELECT that both **cleans** data and is used for insertion into `silver.crm_cust_info`.

Write code where you avoid all [issues]

EXAMPLE:

```sql
SELECT
    cst_id,
    cst_key,

    /*
    -- Name Cleaning
    -- Removes unnecessary leading and trailing whitespace from
names.
    */
    TRIM(cst_firstname) AS cst_firstname,
    TRIM(cst_lastname) AS cst_lastname,

    /*
    -- Marital Status Standardization
    -- Maps single-character codes to descriptive text.
    */
    CASE
        WHEN UPPER(TRIM(cst_marital_status)) = 'M' THEN 'Married'
        WHEN UPPER(TRIM(cst_marital_status)) = 'S' THEN 'Single'
        ELSE 'n/a'
    END AS cst_marital_status,

    /*
    -- Gender Standardization
    -- Maps single-character codes to descriptive text.
    */
    CASE
        WHEN UPPER(TRIM(cst_gndr)) = 'M' THEN 'Male'
        WHEN UPPER(TRIM(cst_gndr)) = 'F' THEN 'Female'
        ELSE 'n/a'
    END AS cst_gndr,

    cst_create_date

FROM
    (
        /*
        -- Deduplication Logic
        -- The source data may contain duplicate records for the
same customer (cst_id).
        -- We use the Window Function RANK() to identify the most
recent record.
        -- PARTITION BY cst_id: Groups data by customer.
        -- ORDER BY cst_create_date DESC: Orders them by date
(newest first).
        -- Result: The row with 'last_date = 1' is the most
recent entry.
        */
        SELECT
```

```
            *,
            RANK() OVER (PARTITION BY cst_id ORDER BY
cst_create_date DESC) AS last_date
        FROM
            bronze.crm_cust_info
    ) t
WHERE
    -- Filter to keep only the most recent record per customer
    last_date = 1
    -- Ensure the creation date is valid (not NULL)
    AND cst_create_date IS NOT NULL;
```

- Build one SELECT per source table that outputs “clean” rows for silver.

6. Insert clean data into silver tables
   1. Ensure `silver.crm_cust_info` DDL columns (without metadata):
      - `cust_id`, `cust_key`, `first_name`, `last_name`, `gender`, `marital_sta tus`, `create_date`, `dw_create_date`.
   2. Insert:

```
INSERT INTO silver.crm_cust_info (
cust_id,
cust_key,
first_name,
last_name,
gender,
marital_status,
create_date
)
SELECT
--the code created for avoiding quality issues
```

   - `dw_create_date` is auto-filled by default constraint.
- Use `INSERT INTO silver.table (…) SELECT …` pattern.

7. Re-run quality checks on silver

Create a second script where all `bronze` references are replaced by `silver`.

1. PK uniqueness:

```
SELECT cust_id,
COUNT(*) AS cnt
FROM silver.crm_cust_info
GROUP BY cust_id HAVING COUNT(*) > 1 OR cust_id IS NULL;
```

   - Expect no rows.

2. Spaces:
3. Distinct normalized values:
4. Visual check:

```
SELECT TOP 100 * FROM silver.crm_cust_info;
```

- Confirm `dw_create_date` populated and transformations applied.

- Confirm issues are fixed and constraints/assumptions hold.

8. Create a Stored Procedure

Create Procedure:

```
/*
===============================================================================
==============
Stored Procedure: Load Silver Layer (Bronze -> Silver)
===============================================================================
==============
Script Purpose:
    This stored procedure performs the ETL (Extract, Transform,
Load) process to
    populate the 'silver' schema tables from the 'bronze' schema.

    Actions Performed:
        1. Truncates silver tables.
        2. Inserts data with transformations:
            - Standardization: Capitalization, trimming,
replacement of abbreviations.
            - Cleansing: Handling NULLs, invalid dates, and
negative values.
            - Enrichment: Deriving columns like 'cat_id' or
logical date calculations.

Parameters:
    None.
      This stored procedure does not accept any parameters or
return any values.

Usage Example:
    EXEC silver.load_silver;
===============================================================================
==============
*/

ALTER   PROCEDURE [silver].[load_silver] AS
BEGIN
```

```
    DECLARE @start_time DATETIME, @end_time DATETIME,
@batch_start_time DATETIME, @batch_end_time DATETIME;

    BEGIN TRY
        SET @batch_start_time = GETDATE();
        PRINT '================================================';
        PRINT 'Loading Silver Layer';
        PRINT '================================================';

        PRINT '------------------------------------------------';
        PRINT 'Loading CRM Tables';
        PRINT '------------------------------------------------';

        --
==================================================================
===
        -- Table: silver.crm_cust_info
        --
==================================================================
===
        SET @start_time = GETDATE();
        PRINT '>> Truncating Table: silver.crm_cust_info';
        TRUNCATE TABLE silver.crm_cust_info;

        PRINT '>> Inserting Data Into: silver.crm_cust_info';
        INSERT INTO silver.crm_cust_info (
            cst_id,
            cst_key,
            cst_firstname,
            cst_lastname,
            cst_marital_status,
            cst_gndr,
            cst_create_date
        )
        SELECT
            cst_id,
            cst_key,
            TRIM(cst_firstname) AS cst_firstname,
            TRIM(cst_lastname) AS cst_lastname,
            CASE
                WHEN UPPER(TRIM(cst_marital_status)) = 'M' THEN
'Married'
                WHEN UPPER(TRIM(cst_marital_status)) = 'S' THEN
'Single'
                ELSE 'n/a'
            END AS cst_marital_status,
            CASE
                WHEN UPPER(TRIM(cst_gndr)) = 'M' THEN 'Male'
                WHEN UPPER(TRIM(cst_gndr)) = 'F' THEN 'Female'
                ELSE 'n/a'
```

```
            END AS cst_gndr,
            cst_create_date
        FROM (
            SELECT
                *,
                RANK() OVER (PARTITION BY cst_id ORDER BY
cst_create_date DESC) AS last_date
            FROM bronze.crm_cust_info
        ) t
        WHERE last_date = 1 AND cst_create_date IS NOT NULL;

        SET @end_time = GETDATE();
        PRINT '>> Load Duration: ' + CAST(DATEDIFF(SECOND,
@start_time, @end_time) AS NVARCHAR) + ' seconds';
        PRINT '>> ------------';

--
==========================================================================
===
       -- ADD REST OF THE TABLES
--
==========================================================================
===



        SET @batch_end_time = GETDATE();
        PRINT '========================================='
        PRINT 'Loading Silver Layer is Completed';
        PRINT '   - Total Load Duration: ' +
CAST(DATEDIFF(SECOND, @batch_start_time, @batch_end_time) AS
NVARCHAR) + ' seconds';
        PRINT '========================================='

    END TRY
    BEGIN CATCH
        PRINT '========================================='
        PRINT 'ERROR OCCURED DURING LOADING SILVER LAYER'
        PRINT 'Error Message: ' + ERROR_MESSAGE();
        PRINT 'Error Number:  ' + CAST(ERROR_NUMBER() AS
NVARCHAR);
        PRINT 'Error State:   ' + CAST(ERROR_STATE() AS
NVARCHAR);
        PRINT '========================================='
    END CATCH
END
```

Execution

- Test the full pipeline:

```
EXEC silver.load_silver;
```

- Verify all tables are populated.

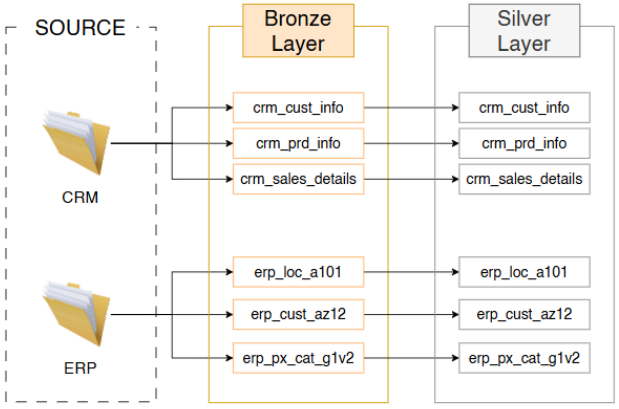
Does this note add something to discussion in Obsidian ?

- Is this note open to connection to other note or is it already connected ?
- Is this note enriching my Obsidian ?
- Am I focused only on this project or am I looking also on bigger picture ?
- Am I seeking for dis-confirming facts or only confirming previous notes?

- Instead of running 6 separate scripts every time, wrap the entire Silver Layer load in a single Stored Procedure.

9. Document data flow & commit to Git

DATA FLOW CHART



- Update diagrams (lineage sources → bronze → silver) and store scripts + quality checks.