Protocol – Bronze Layer Build Database

Protocol: Building the Bronze Layer (Database Ingestion)

> Purpose: Step-by-step checklist + commented template to follow whenever building a new “Bronze” ingestion layer in a data warehouse (raw, 1-to-1 copy of sources, no transformations).

---

1. Analyze and Document Source Systems

1.1 Business context & ownership

- **Action:** Schedule an “interview” with source-system SMEs (or yourself if you are both). Capture answers in a markdown note.
- **Questions to ask / document:
  - What business domain is this system? (CRM, ERP, billing, logistics, finance, etc.)
  - What business processes does it support? (customer transactions, supply chain, reporting, etc.)
  - Who owns the system and data? (IT team, vendor, data steward, contact person)
  - What is the criticality of this data for reporting / analytics / downstream processes?

> Comment: This step makes sure you understand *why* the data exists and *who* to talk to when something breaks.

1.2 Source documentation & data model

- **Action:** Collect all available technical documentation.
- **Checklist:
  - ERDs, schema diagrams, field descriptions, existing data catalog entries.
  - Any business glossary mapping columns to business concepts.
  - Existing integration specs: API docs, file format specs, DB schemas.

> Comment: Treat these docs as “training material” for your data; link them into your project note and reference them whenever you design tables.

1.3 Architecture & technology stack

- **Action:** Identify *where* data lives and *how* you will reach it.
- **Questions:
  - Is the source on-prem (SQL Server, Oracle, etc.) or in the cloud (Azure, AWS, etc.)?
  - What integration options are available? (APIs, CSV/flat-file exports, direct DB connection, SFTP, etc.)
  - Are there existing ETL tools/pipelines already reading from this system?

> Comment: This drives your ingestion method (e.g., `BULK INSERT` from CSV vs. CDC vs. API pulls).

1.4 Load strategy, scope, performance constraints

- **Decide and document:**
  - **Load type:** full load vs incremental load (or hybrid). For the Bronze layer in this project, it is **full load** (truncate then reload).
  - **Historical scope:** keep all history or only last N years? Is history already handled in the source or to be built in the warehouse?
  - **Expected volume:** MB / GB / TB per extract, frequency (daily/ hourly), and growth expectations.
  - **Source limitations:** any query caps, performance constraints, allowed windows for heavy extraction, rate limits.
  - **Security:** Authentication and authorization (DB users, tokens, keys, passwords, VPNs, IP whitelists).

> Comment: This step protects you from overloading fragile legacy systems and shapes your scheduling and infrastructure choices.

---

2. Design the Bronze Layer Schema

2.1 Define Bronze layer principles

- **Key rules for Bronze:
  - 1-to-1 copy of the source structure (column names mirror the source; no renaming).
  - No business transformations or modeling; only minimal technical adjustments if absolutely required.

- Tables are grouped by source (e.g., `bronze.crm_*`, `bronze.erp_*`).
- Full-load pattern: truncate then reload. (Incremental will usually appear in Silver.)

> Comment: Bronze = raw and traceable. Later layers are free to reshape, but Bronze keeps the “truth as delivered.”

2.2 Naming conventions

- **Schema:** `bronze` (or similar) to clearly separate from `silver`, `gold`
- **Tables:**
  - Format: `<schema>.<source_system>_<original_table_name>`
  - Example: `bronze.crm_customer_info`, `bronze.erp_sales_details`
- **Columns:**
  - Exactly as in source CSV / DB (same names, same order where reasonable)

> Comment: Good naming lets you immediately see where each table came from and makes setting up data lineage easier

2.3 Derive table structure from incoming data

- **Action:** For each file / source table:
  - Open sample data (e.g., CSV) and inspect the header row and value types
  - Infer data types (INT, VARCHAR(n), DATE, DECIMAL, etc.)
  - Decide on nullability and any technical keys (if needed)

> Comment: When headers exist, you get column names “for free”; types require judgment based on sample values and documentation.

---

3. Write DDL Scripts for Bronze Tables

3.1 “Create-if-not-exists” pattern with drop

Use a pattern that safely recreates tables so you can evolve schemas:

```
-- PURPOSE:
--   (1) Drop existing bronze table if it exists
--   (2) Recreate it with the current structure
--   This keeps your DDL idempotent: you can run it many times safely.
```

```
IF OBJECT_ID('bronze.crm_customer_info', 'U') IS NOT NULL
BEGIN
    DROP TABLE bronze.crm_customer_info;  -- Remove old version if it
exists
END;
GO

CREATE TABLE bronze.crm_customer_info (
    -- Column definitions mirror the source exactly (names and meaning)
    Id          INT,          -- Customer unique identifier from CRM
    Key         NVARCHAR(50), -- Business key / natural key from CRM
    CreateDate  DATE          -- Record creation date in the source
    -- Add more columns as needed, always checking data types against
the source
);
GO
```

> Comment: Wrap every Bronze table definition in this pattern so your DDL script can be rerun without manual cleanup

3.2 DDL script organization

- Place all Bronze table DDLs in a single file, e.g. `scripts/bronze/bronze_ddl.sql`
- Group by source system with clear comment banners:

```
-- =========================================
-- CRM SOURCE TABLES (BRONZE)
-- =========================================
-- 1) bronze.crm_customer_info
-- 2) bronze.crm_product_info
-- 3) bronze.crm_sales_details

-- =========================================
-- ERP SOURCE TABLES (BRONZE)
-- =========================================
-- 4) bronze.erp_customers
-- 5) bronze.erp_orders
-- 6) bronze.erp_categories
```

> Comment: This makes it obvious which section to touch when a specific source changes.

---

4. Implement Data Load Scripts (BULK INSERT from CSV)

4.1 Basic `BULK INSERT` pattern

```
-- PURPOSE:
--   Full load of bronze.crm_customer_info from CSV file.
--   Steps:
--     (1) Empty the target table (TRUNCATE)
--     (2) Bulk load full content from CSV
--     (3) No transformations, just 1-to-1 loading

-- 1) Make table empty so we avoid duplicates
TRUNCATE TABLE bronze.crm_customer_info;

-- 2) Load CSV file using BULK INSERT
BULK INSERT bronze.crm_customer_info
FROM 'C:\sql-data-warehouse-
project\dataset\source_crm\customer_info.csv'  -- Absolute path to the
source file
WITH (
    FIRSTROW = 2,               -- Skip header row; data starts from row
2
    FIELDTERMINATOR = ',',      -- CSV delimiter; change if using ';' or
'|'
    ROWTERMINATOR   = '\n',     -- End-of-line marker; platform-
dependent
    TABLOCK                       -- Lock table during load for
performance
);
```

> Comment: Always set `FIRSTROW` and `FIELDTERMINATOR` correctly; many ingestion bugs come from misaligned delimiters or headers

4.2 Validate load quality for each table

Immediately after each load:

```
-- Quick data sanity checks after BULK INSERT

-- 1) Sample data
SELECT TOP 100 *
FROM bronze.crm_customer_info;

-- 2) Row count in table
SELECT COUNT(*) AS row_count
FROM bronze.crm_customer_info;
```

And outside SQL, compare `row_count` to number of rows in the file (minus the header row).

> Comment: You are checking both completeness (row counts) and alignment (values in the correct columns)

---

5. Wrap Load Logic in a Stored Procedure

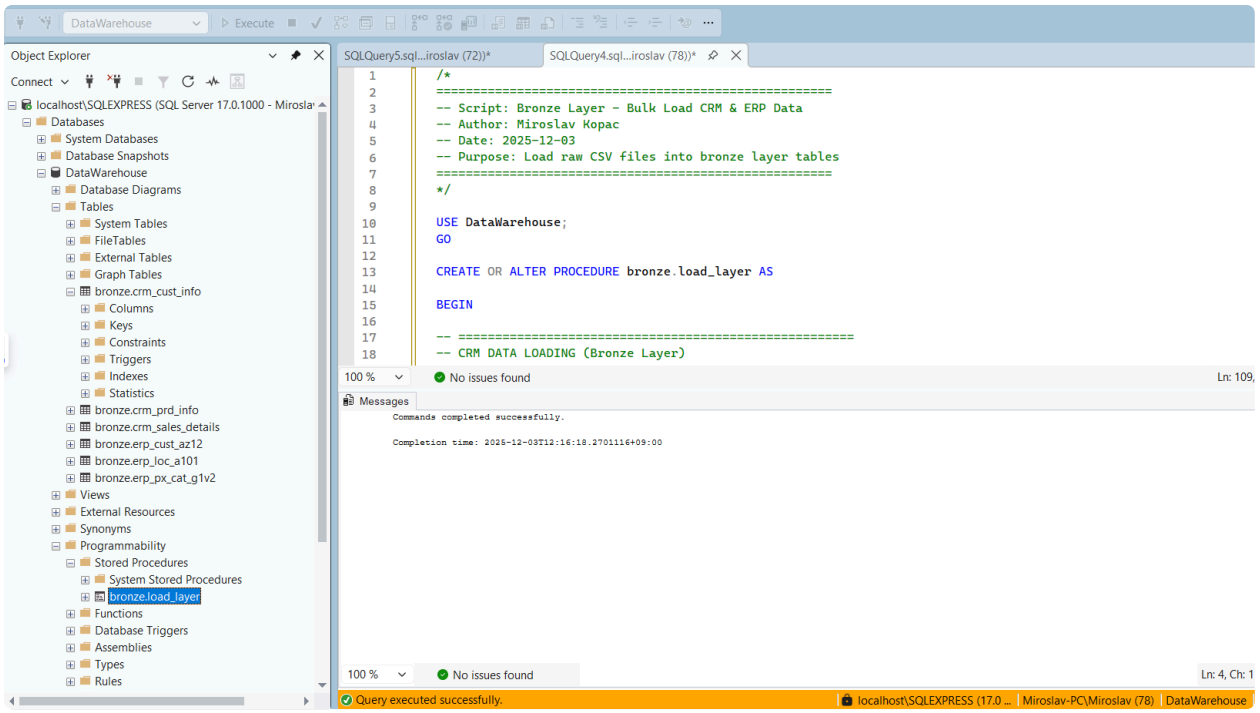
5.1 Create a Bronze load procedure

```
-- PURPOSE:
--   Stored procedure to fully load all Bronze tables from CSV files.
--   Usage:
--     EXEC bronze.load_bronze;
--   Behavior:
--     - Truncates each Bronze table
--     - Bulk inserts from configured file paths
--     - Prints progress messages

CREATE OR ALTER PROCEDURE bronze.load_bronze
AS
BEGIN
    -- Group messages: overall process start
    PRINT '========================================';
    PRINT 'LOADING BRONZE LAYER';
    PRINT '========================================';

    -- ============================
    -- SECTION: Load CRM tables
    -- ============================
    PRINT '--- Loading CRM tables ---';

    -- Example: CRM customer_info
    PRINT '>> Truncating bronze.crm_customer_info';
    TRUNCATE TABLE bronze.crm_customer_info;

    PRINT '>> Inserting data into bronze.crm_customer_info';
    BULK INSERT bronze.crm_customer_info
    FROM 'C:\...\source_crm\customer_info.csv'
    WITH (
        FIRSTROW = 2,
```

```
        FIELDTERMINATOR = ',',
        ROWTERMINATOR   = '\n',
        TABLOCK
    );

    -- Repeat same pattern for:
    -- - bronze.crm_product_info
    -- - bronze.crm_sales_details

    -- ============================
    -- SECTION: Load ERP tables
    -- ============================
    PRINT '--- Loading ERP tables ---';

    -- Example: ERP customers
    PRINT '>> Truncating bronze.erp_customers';
    TRUNCATE TABLE bronze.erp_customers;

    PRINT '>> Inserting data into bronze.erp_customers';
    BULK INSERT bronze.erp_customers
    FROM 'C:\...\source_erp\customers.csv'
    WITH (
        FIRSTROW = 2,
        FIELDTERMINATOR = ',',
        ROWTERMINATOR   = '\n',
        TABLOCK
    );

    -- Repeat for other ERP tables

END;
GO
```

> Comment: This procedure becomes your daily “bronze refresh” job; you can hook it into a scheduler (SQL Agent, Airflow, etc.).

In SQL Server



6. Add Logging, Error Handling and Duration Metrics

6.1 TRY…CATCH for ETL robustness

```
CREATE OR ALTER PROCEDURE bronze.load_bronze
AS
BEGIN
    BEGIN TRY
        -- All normal loading logic lives here
        -- (all TRUNCATE + BULK INSERT statements)
    END TRY
    BEGIN CATCH
        PRINT '========================================';
        PRINT 'ERROR DURING LOADING BRONZE LAYER';
        PRINT '========================================';

        -- Print error details to help debugging
        PRINT 'Error message: ' + ERROR_MESSAGE();
        PRINT 'Error number : ' + CAST(ERROR_NUMBER() AS NVARCHAR(10));
        PRINT 'Error state  : ' + CAST(ERROR_STATE()  AS NVARCHAR(10));

        -- (Optional) Insert into a logging table here

        -- Optionally rethrow if needed:
        -- THROW;
    END CATCH;
END;
```

```
GO
```

> Comment: Centralizing error handling makes it easier to understand what went wrong without scrolling through raw SQL errors

6.2 Measure per-table and batch durations

```
CREATE OR ALTER PROCEDURE bronze.load_bronze
AS
BEGIN
    DECLARE @batch_start_time DATETIME,
            @batch_end_time   DATETIME;

    SET @batch_start_time = GETDATE();  -- When the entire run starts

    BEGIN TRY
        PRINT '========================================';
        PRINT 'LOADING BRONZE LAYER';
        PRINT '========================================';

        -- Example: duration tracking for one table
        DECLARE @start_time DATETIME,
                @end_time   DATETIME;

        PRINT '--- Loading CRM tables ---';

        SET @start_time = GETDATE();

        TRUNCATE TABLE bronze.crm_customer_info;
        BULK INSERT bronze.crm_customer_info
        FROM 'C:\...\source_crm\customer_info.csv'
        WITH (
            FIRSTROW = 2,
            FIELDTERMINATOR = ',',
            ROWTERMINATOR   = '\n',
            TABLOCK
        );

        SET @end_time = GETDATE();

        PRINT '>> Load duration (crm_customer_info): '
              + CAST(DATEDIFF(SECOND, @start_time, @end_time) AS
NVARCHAR(10))
              + ' seconds';

        PRINT '----------------------------------------';
```

```
        -- Repeat same pattern for every Bronze table

        SET @batch_end_time = GETDATE();

        PRINT '======================================';
        PRINT 'LOADING BRONZE LAYER COMPLETED';
        PRINT 'Total load duration: '
              + CAST(DATEDIFF(SECOND, @batch_start_time,
@batch_end_time) AS NVARCHAR(10))
              + ' seconds';
        PRINT '======================================';

    END TRY
    BEGIN CATCH
        PRINT '======================================';
        PRINT 'ERROR DURING LOADING BRONZE LAYER';
        PRINT 'Error message: ' + ERROR_MESSAGE();
        PRINT 'Error number : ' + CAST(ERROR_NUMBER() AS NVARCHAR(10));
        PRINT 'Error state  : ' + CAST(ERROR_STATE()  AS NVARCHAR(10));
        PRINT '======================================';
    END CATCH;
END;
GO
```

> Comment: Duration metrics help you spot bottlenecks (which table suddenly takes 10x longer) and track performance regressions

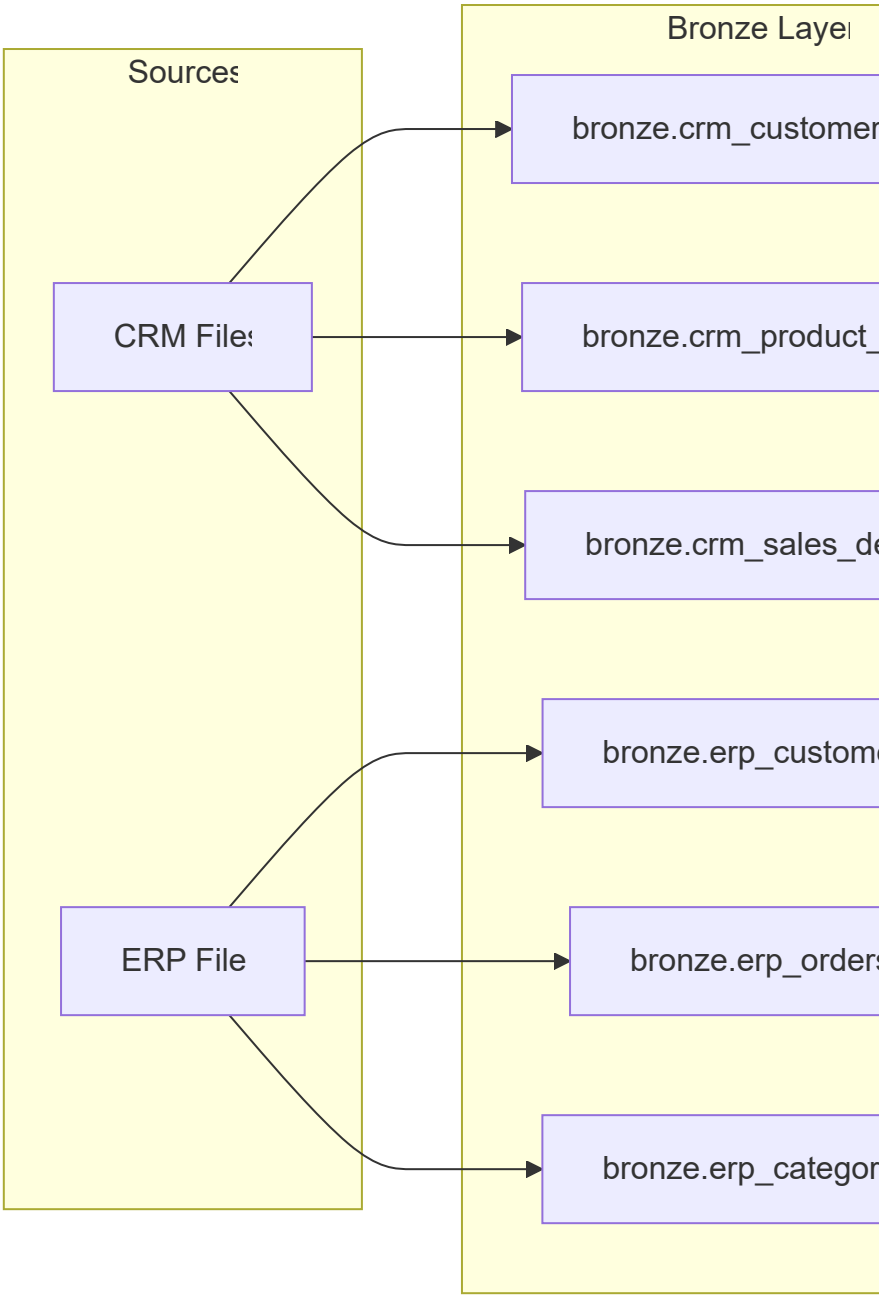
---

7. Create Data Flow / Lineage Diagram

- **Action:** Draw a simple diagram (e.g., in draw.io, Excalidraw, or Mermaid in Obsidian) showing:
  - Source systems (folders / icons for `CRM`, `ERP`, etc.)
  - Bronze layer box with each table (`crm_customer_info`, `crm_product_info`, `erp_customers`, etc.)
  - Arrows from each source to its corresponding Bronze tables

> Comment: This is your “map” when debugging or explaining lineage to others. For small/medium warehouses, keeping this updated pays off a lot.

Example Mermaid block you can paste into Obsidian:



---

8. Version Control and Documentation

8.1 Organize scripts in Git

- **Folder structure suggestion:
  - `scripts/bronze/bronze_ddl.sql` – all Bronze table definitions
  - `scripts/bronze/proc_load_bronze.sql` – stored procedure definition.
- **Comment headers:** At top of each script, clearly state:
  - Purpose
  - Behavior
  - Parameters (if any) and example usage.

> Comment: This ensures anyone (including future you) can quickly understand and run the correct script

8.2 Link to documentation in Obsidian

In your Bronze note, add links like:

- `[[Layer – Bronze Overview]]` – this protocol itself.
- `[[Diagram – Data Flow Bronze]]` – the diagram / Mermaid note.

> Comment: Treat each layer (Bronze, Silver, Gold) and each source system as separate notes; link them heavily to build a navigable knowledge graph.

---

9. How to Use and Reuse This Protocol

- When starting a new system ingestion:
  1. Duplicate this note.
  2. Replace “CRM/ERP” with the actual sources.
  3. Fill in load strategy, volumes, and paths.
  4. Implement the DDL and procedure using the templates above.
- For **Silver/Gold** notes: link back here as “Upstream: Bronze ingestion protocol”, and add downstream transformation protocols in new notes.