Appendix A: Technical Environment and Lessons Learned

A.1 Technical Architecture

The data warehouse implementation used the following technical stack:

- **Source System**: MySQL 8.0 (handlebarhaven operational database)
- **ETL Tool**: SQL Server Integration Services (SSIS) in Visual Studio 2022
- **Target System**: SQL Server 2019 (HandlebarHavenDW data warehouse)
- **Analysis Tools**: SQL Server Management Studio (SSMS), Power BI Desktop
- **Connections**: ADO.NET Source for MySQL extraction, OLE DB Destination for SQL Server loading

A.2 Implementation Challenges and Resolutions

Challenge 1: Fact Table Duplication (819,804 vs 121,000 Expected Rows)

Problem: The initial FactSales implementation produced 819,804 rows, nearly seven times the expected 121,000 transactions.

Root Cause: The source query joined to products_history, which is a Slowly Changing Dimension Type 2 table. This table contains multiple historical price versions per Product_ID with different End_Dates. When joined without filtering, this created a product with approximately 26 historical price versions matching each transaction record.

Resolution: I modified the source query to join directly to the products table instead of products_history, retrieving only current pricing. This eliminated the duplication while maintaining accurate cost and margin calculations.

Challenge 2: Customer Dimension Lookup Failures (62 Missing Customers)

Problem: FactSales loading failed with 62 lookup errors. These customers appeared in sales_orders but were missing from DimCustomer.

Root Cause: The DimCustomer source query used INNER JOINs through the customer_address, state, and country_region_codes hierarchy. Customers with incomplete address data were excluded entirely, even though they had valid transaction history in the system.

Resolution: I converted the INNER JOINs to LEFT JOINs and used COALESCE functions to assign 'Unknown' values where geography attributes were missing. This increased DimCustomer from 19,758 to 19,820 rows, capturing all customers while flagging incomplete data that the operational team could potentially remediate.

Challenge 3: Data Type Mismatches in Lookup Transformations

Problem: TimeKey lookup transformations failed despite TimeKey existing in both the source and dimension tables.

Root Cause: MySQL's CAST(DATE_FORMAT(Order_Date, '%Y%m%d') AS UNSIGNED) function produces a BIGINT (8-byte unsigned integer), while SQL Server's DimTime.TimeKey was defined as INT (4-byte signed integer). SSIS lookup transformations require exact data type matching between source and target fields.

Resolution: I added a Data Conversion transformation component before the TimeKey lookup, explicitly converting the BIGINT to DT_I4 (four-byte signed integer) to match the SQL Server INT type.

Challenge 4: Inverted Seasonal Logic in DimTime

Problem: Initial seasonal assignments showed January in the USA as "Summer" instead of "Winter". The Season_North and Season_South columns had been swapped.

Root Cause: The SSIS Derived Column transformation formulas for Season_North and Season_South were accidentally inverted during implementation.

Resolution: I applied a post-load correction using SQL UPDATE statements with temporary columns to swap the values without triggering MySQL's safe update mode restrictions. A verification query comparing Australia (territory_code='AU') against other territories confirmed the proper hemisphere assignment.

A.3 Technical Lessons Learned

SSIS Best Practices:

- Use OLE DB Destination (not ADO.NET Destination) for SQL Server targets. OLE DB provides better bulk loading performance and proper handling of square bracket identifiers.
- Create target tables manually in SSMS before configuring SSIS destinations. The SSIS "New..." button generates DDL immediately rather than during package execution, which complicates iterative development.
- Use "SQL command" mode (not "Table or view" mode) for MySQL sources. This avoids SSIS auto-generation of problematic double-quote syntax.
- Configure Lookup transformations with "Full cache" mode and "Fail component" error handling during development. This enables rapid identification of dimension/fact referential integrity issues.

Data Quality Strategies:

- Apply DISTINCT in dimension source queries to prevent duplicate surrogate keys.
- Use LEFT JOINs with COALESCE for incomplete hierarchical data. This captures all relevant records while flagging data quality issues for potential resolution.
- CAST dates explicitly to DATE type to strip timestamps that can cause lookup mismatches.
- TRUNCATE target tables before reloading during iterative development to prevent accumulation of duplicate records across test runs.

Cross-Platform Data Type Management:

- MySQL UNSIGNED integers do not directly map to SQL Server signed integers and require explicit Data Conversion transformations.
- SSIS string literals default to Unicode (DT_WSTR/NVARCHAR). Dimension columns should use NVARCHAR unless specifically requiring VARCHAR for compatibility reasons.
- Currency and financial calculations require DECIMAL(19,4) format, never FLOAT or REAL, due to precision loss that affects margin calculations.

Source Table Selection:

- Use current operational tables (products) rather than historical audit tables (products_history) for fact table extraction, unless specifically implementing Slowly Changing Dimension logic.
- Validate expected row counts before and after ETL. Significant discrepancies indicate join issues or data quality problems requiring investigation.