Data Conversion Mapping Standards

Practical Standards for Data Migration Mapping

Version 1.0 | January 1, 2026

Bridger DMA | bridger-dma.com

Revision History

Version	Date	Description
1.0	January 1, 2026	Initial public release

License and Usage

These standards were developed over years of real-world data migration engagements across multiple industries. We are sharing them as general knowledge — the kind of discipline that any experienced migration team should already have in place, or should build toward. Use them freely, adapt them to your organization, and make them your own. No attribution is required.

1. Introduction

This document describes a methodology for planning and documenting data conversion mappings. It is not a software manual. It is a guide to the *process* of recording how legacy data transforms into a target system, structured so that human analysts, business reviewers, and AI agents can all work from the same source of truth.

These standards are tool-agnostic in principle. The methodology applies regardless of what software you use to document your mappings — a spreadsheet, a dedicated tool, or a custom system. If you are using Bridger, these standards are implemented as the default starting configuration. Everything described here ships ready to use and is fully configurable; treat it as a battle-tested starting point, not a mandate. Organizations should adapt these standards to match their review workflows, compliance requirements, and team preferences.

The audience is deliberately broad. A mapping analyst uses this to know what transformation logic to write. A business stakeholder uses it to understand what will happen to their data and to approve or challenge decisions. A developer uses it to build conversion programs from the documented rules. An AI agent uses it to suggest mappings, generate validation scripts, and flag inconsistencies — but only if

the data is clean, consistent, and unambiguous.

1.1 First Principles

Two principles underpin every decision in this methodology. They are worth internalizing before engaging with the specifics that follow.

The Dual Completeness Rule. A mapping effort is not complete until every column on both sides has been accounted for. Every source column must be mapped to a target column or documented with an exclusion reason explaining why it was not converted. Every target column must be mapped from a source column or documented with an explanation of how it will be populated. This ensures nothing is overlooked, and it gives auditors, reviewers, and downstream developers a complete picture.

The No Unilateral Decisions Rule. The migration team does not make decisions unilaterally. The conversion team brings experience and methodology. It uses that experience to *suggest* transformation logic, default values, exclusion reasons, and mapping strategies. Those suggestions are then reviewed and approved — first by internal subject matter experts (business analysts, product developers, architects), then by the client or business stakeholders.

This two-tiered review process is deliberate. The internal SME review catches technical issues and ensures the mapping aligns with the target system's design intent before it reaches the client. The client review confirms that business rules and data handling meet their expectations. This structure minimizes the burden on both groups — SMEs are not reviewing raw first-draft mappings, and clients are not being asked to evaluate technically incomplete work.

The conversion team documents what *should* happen. The stakeholders decide what *will* happen. Mapping statuses, review sessions, and approval workflows exist to enforce this discipline.

2. Organizing a Migration Project

Before any mapping work begins, the project itself needs structure. How tables are grouped, named, and organized has a direct effect on review quality and stakeholder engagement.

2.1 Data Domains as Project Boundaries

Large migrations involve dozens or hundreds of tables. Mapping everything in a single flat workspace becomes unmanageable quickly. The standard practice is to organize mapping work into projects by data domain — groupings of related tables that a business stakeholder can review as a coherent unit.

For example, a retail migration might organize projects around customers, inventory, orders, and employees. Each domain contains the source and target tables that are functionally related, allowing focused review sessions where the right stakeholders are in the room.

How projects are named matters less than being consistent about it. Develop a naming convention early — before the first project is created — and enforce it throughout the engagement. A lack of consistency at this level may impact a customer's confidence in the quality of the project at other levels.

Naming conventions are a team discipline, not a tool constraint.

2.2 Schema Libraries and Mapping Projects

A sound organizational pattern separates the full schema inventory (the library) from the active mapping workspace (the project). A table exists once in the library and can be referenced by multiple projects. This prevents duplication and ensures that schema changes propagate consistently.

The workflow is: import schemas into the library first, then add tables from the library into projects as mapping work begins on each data domain. In Bridger, this separation is built into the application structure — the library holds every known source and target table, and projects draw from it.

2.3 Source-to-Target and Target-to-Source

Every mapping can be read in both directions. Source-to-target shows where each legacy field ends up. Target-to-source shows where each new system field comes from. Both perspectives matter. A developer building the conversion program thinks source-to-target. A business reviewer validating the target system thinks target-to-source. Both views should be derivable from the same underlying mapping data.

3. Mapping Patterns

Not all mappings are simple one-to-one column pairings. The relationship between source and target columns can take several forms, and understanding these patterns is essential for writing clear transformation logic.

Regardless of the pattern, the principle is the same: an effective mapping connects a single source endpoint to a single target endpoint. The source endpoint is either a real source column, a Source Group (Section 3.3), or one of the special source tables. The target endpoint is a single target column or the special target table. For a description of the special tables, see Section 4 below.

The one-to-one relationship between endpoints is what makes it possible to track status, specify transformation logic, and attach notations cleanly. When a mapping tries to serve multiple purposes or document multiple relationships at once, status tracking becomes ambiguous and transformation logic becomes muddled.

3.1 One Source to One Target

The most common pattern. A single source column maps directly to a single target column. The transformation may be a direct move, a type conversion, a format change, or a code-to-value expansion, but the relationship is one column in, one column out.

Example: `CUST_MSTR.CUST_FST_NAM` maps to `customer.first_name` with transformation logic "Move source to target."

Example with transformation: `CUST_MSTR.CUST_TY_CD` maps to `customer.customer_type` with a Case/Switch expanding single-character codes (R, W, F) to full values (Retail, Wholesale, Fleet).

3.2 One Source to Many Targets

A single source column maps to multiple target columns. This occurs when the target schema breaks data out into more granular fields than the source, or when the same source value is needed in multiple places.

Example: A source `CUST_MSTR.CUST_ID` maps to both `customer.external_id` (for cross-system traceability) and is also used as a lookup key during order conversion to resolve `sales_order.customer_id` via a foreign key mapping table.

Each mapping is documented individually. The source column will appear multiple times in the source-to-target view — once for each target column it feeds. The transformation logic on each mapping describes what happens for that specific target.

3.3 Many Sources to One Target (Source Groups)

Sometimes a target column's value can only be derived by combining multiple source columns. Those columns may live in the same source table or span different source tables. A customer's full mailing label might require the name from one table and the address from another. A classification code in the target might depend on three columns in the same legacy table evaluated together.

This pattern is handled through Source Groups — named groupings of multiple source columns that collectively map to a single target column. Rather than forcing the analyst to pick one "primary" source and document the rest in free text, Source Groups make the multi-column dependency explicit and visible in the mapping structure.

When a mapping uses a Source Group, the transformation logic should document how the grouped columns combine to produce the target value — which column drives the primary value, which columns contribute filtering or conditional logic, and what happens when one of the source columns is null or invalid.

3.4 Target Columns Without a Source

Target columns that do not map to any real source data are documented using the special source tables described in Section 4. Every unmapped target column gets one of these designations — nothing is left undocumented.

3.5 Source Columns Without a Target

Source columns that do not map to any target column are mapped to the Excluded special target table with an exclusion reason code (see Section 7). This documents the team's deliberate decision not to convert the data and provides the rationale for auditors.

4. Special Tables

Not every column maps to a real counterpart. Some target columns have no source data. Some source columns have no target equivalent. Both cases must still be documented. Five special source-side table types and one special target-side table type serve as structured placeholders. Each answers a specific question about how the real column's data will be handled.

4.1 Source-Side Special Tables

Excluded — *This target column will not be populated during conversion.*

Use for target columns that are populated by application logic after go-live, managed by interfaces or batch processes, reserved for future phases, or simply not applicable to this implementation.

Examples: auto-generated surrogate keys (target manages its own identity), application-calculated fields, columns reserved for post-conversion configuration.

Conversion Default — *The migration team supplies a specific value.*

Use when the conversion program needs to insert a known constant that differs from the database default or when no database default exists. The actual value is documented in the transformation logic notation.

Examples: `created_by` set to `'CONV'` to mark records as conversion-originated, status fields set to an initial state like `'Active'`, date fields set to the cutover date using a substitution variable like `&CONVERSION_DATE`.

Database Default — *The database schema already defines the right default.*

Use when the target column has a DEFAULT constraint that provides the correct value without any conversion logic. The conversion program simply omits the column from the INSERT statement and lets the database handle it.

Examples: `created_date` defaulting to `CURRENT_TIMESTAMP`, boolean flags defaulting to `false`, counter fields defaulting to `0`.

Stored Value — *The value was computed or looked up during conversion.*

Use when the target column's value comes from a prior step in the conversion process rather than directly from source data or a static default. This is the key management mechanism — when a surrogate key is generated for a parent record, child records reference that stored value for their foreign keys.

Examples: a foreign key to a parent record whose surrogate key was generated during an earlier conversion step, a computed classification value derived from multiple source fields that is reused across several target tables.

Source Groups — *The value is derived from multiple source columns grouped together.*

Use when the target column requires more than one source column to derive its value. The grouped columns may come from the same source table or span multiple source tables. Source Groups make multi-column dependencies explicit in the mapping structure rather than burying them in free-text notes. See Section 3.3 for details on the many-sources-to-one-target pattern.

4.2 Target-Side Special Table

Excluded — *This source column will not be converted to the target system.*

Use for legacy columns that have no target equivalent — deprecated fields, duplicates of other mapped fields, or data that the new system does not capture. Every excluded source column must include an exclusion reason code (see Section 7) documenting why it was not converted.

4.3 Choosing the Right Special Table

The decision tree is:

1. Does the column require data from multiple source columns? If yes → **Source Groups**.
2. Will this column be populated during conversion at all? If no → **Excluded**.
3. Does the database schema already define an appropriate default? If yes → **Database Default**.
4. Is the value a static constant the migration team has decided on? If yes → **Conversion Default**.
5. Is the value dynamic, computed or looked up during a prior conversion step? If yes → **Stored Value**.
6. Does the column map to actual source data from a single table? If yes → map it to a real source column.

5. Status Progression

Each mapping has an associated status that tracks its position in the review workflow. Statuses are color-coded for visual scanning and drive the cadence of review sessions. Use them faithfully — a status should reflect reality, not optimism. If a mapping has open questions, it is not ready for review. If an approved mapping is affected by a schema change, it must be reopened. Misrepresenting status undermines the entire workflow and erodes stakeholder trust.

Remember the first principle: the migration team suggests, stakeholders decide. Statuses enforce this discipline by making the current state of every mapping visible to everyone involved.

5.1 Recommended Statuses

The following statuses reflect a two-tier review process (internal subject matter expert review, then client review) which is common practice on large migration engagements. In Bridger, these ship as the default starting configuration. Organizations with different review structures should modify these to match their workflow — add tiers, rename stages, change colors, or reorder the progression.

Two structural requirements should hold regardless of how statuses are customized: there must be exactly one default status for newly created mappings, and there must be at least one approval status representing a locked, final state. Once a mapping reaches the approval status, it signals formal acceptance and should not be changed without deliberate reopening.

Status	Color	Purpose
New	White	Mapping has been created but no logic has been documented yet
Ready for SME Review	Yellow	Analyst has completed the mapping, awaiting internal team review

Status	Color	Purpose
Ready for Client Review	Orange	Internal review is complete, awaiting business stakeholder review
Approved by Client	Green	Business stakeholder has approved the mapping
Question/Issue	Red	A question or issue has been raised that blocks progress
Reopened After Approval	Blue	A previously approved mapping has been reopened due to schema changes, requirement changes, or conversion testing findings
AI Generated	Purple	Mapping was created by an AI agent and has not yet been human-reviewed

5.2 Status as Workflow

Statuses are not just labels — they are workflow states. The expected forward progression is:

New → Ready for SME Review → Ready for Client Review → Approved by Client

Lateral and backward transitions are normal:

- Any status can move to **Question/Issue** when a blocker is identified. The question must be documented (typically in a mapping note) before the status is changed.
- **Approved by Client** can move to **Reopened After Approval** when the target schema changes, business rules are revised, or conversion testing reveals a problem. This status explicitly signals "this was signed off, but something changed."
- **AI Generated** must transition to another status before it enters the normal review flow. This forces a human to acknowledge AI-produced work rather than letting it silently enter the approval pipeline.

6. Transformation Logic

Transformation logic describes what happens to source data on its way to the target. A well-organized transformation logic library uses a consistent pseudocode style that is readable by both humans and AI agents.

Of all the notation types in a mapping system, transformation logic is the only one that must exist for every mapping. Other annotation types can be added, removed, or renamed freely, but every mapping needs a place to document what happens to the data. Without transformation logic, a mapping is just a line connecting two column names — it tells you *what* is related but not *what to do about it*.

6.1 Configurability

Every logic template — its name, content, category, usage guidance, and default values — should be configurable. Organizations can modify existing templates, add project-specific patterns, or remove templates that do not apply. The templates described here have proven effective across many engagements and are the starting defaults in Bridger.

6.2 Data Moves

The simplest transformations. Data moves from source to target with minimal or no alteration.

Move source to target — Direct 1:1 mapping. Source and target have compatible types and lengths. No transformation needed.

Number to Char / Char to Number — Type conversion with a simple cast. Consider formatting needs (leading zeroes, decimal places) and what happens with invalid data.

Char to Varchar — Trim fixed-length CHAR padding before moving to variable-length target. Specify left trim, right trim, or both.

Source to Smaller Target — Target column is shorter than source. Truncate and document which end is trimmed. Always report truncations as warnings — data loss may indicate a design problem.

Source to Larger Target — Target column is longer than source. Specify justification and pad character if applicable.

Crosswalk Lookup — Translate source values to target values using a reference table. Always handle lookup failures with a default or error.

6.3 Defaults

Static values applied to target columns that have no source equivalent or need a conversion-specific constant.

Common defaults include: space(s), zero, one, 'Y', 'N', -1, NULL, a specific value, the conversion user identifier (`'CONV'`), and the conversion date (using substitution variable `&CONVERSION_DATE`). Each has specific usage guidance — for example, -1 as a default is appropriate for ID fields where 0 might be a valid identifier.

The "Database Default" template documents that no conversion logic is needed because the target schema's DEFAULT constraint is appropriate. This pairs with the Database Default special source table.

6.4 Common Routines

Reusable validation functions that follow a consistent pattern:

```
Perform FunctionName on &lt;&lt;SOURCE_FIELD&gt;&gt;
If ( TRUE ) Then
   Move source to target
Else
   Report exception EXCEPTION_CODE
   Move &lt;&lt;DEFAULT&gt;&gt; to target
   [OR Bypass row]
End If
```

This pattern is intentionally uniform. Every common routine validates, branches on the result, and handles failure with either a default value or row bypass. The consistency makes these templates both human-scannable and machine-parseable — an AI agent can reliably extract the function name, exception code, and fallback behavior from any common routine.

Available validation functions include:

Function	Validates	Exception Code
HasValue	Field is not NULL or empty	MISSING
IsAllNumeric	String contains only digits	NONNUMERIC
IsValidInteger	String can cast to integer	BADINTEGER
IsValidDecimal	String can cast to decimal	BADDECIMAL
IsAlpha	String contains only letters	NONALPHA
IsAlphanumeric	String has no special characters	SPECIALCHAR
IsCleanString	No leading/trailing whitespace	PADDING
IsEmptyOrSpaces	String has no meaningful content	EMPTYDATA
CheckStringFit	Source fits in target length	TRUNCATION
IsValidDate	String parses as valid date	BADDATE
IsDateInRange	Date within business bounds	DATERANGE
IsReasonableDate	Date not absurdly past/future	UNREASONABLEDATE
IsInNumericRange	Number within specified bounds	NUMRANGE
IsPositive	Number is greater than zero	NEGATIVE
IsValidEmail	Basic email format check	BADEMAIL
IsValidPhone	Phone format check	BADPHONE
IsValidSSN	SSN format check	BADSSN
IsValidZipCode	US ZIP format check	BADZIP

Organizations can add custom common routines for domain-specific validation (e.g., NPI validation in healthcare, VIN validation in automotive).

6.5 Control Statements

For complex transformations that require branching or iteration.

Case/Switch — Map discrete source values to different target values. Better than nested IF statements when there are three or more value mappings. Always include a "When Others" clause for unexpected values.

If-Then-Else — Conditional logic based on expressions or ranges. Provide an Else clause for safety.

For Each Loop — Iterate over related child records to accumulate values (sums, counts, min/max).

Build Delimited String Loop — Concatenate multiple related values into a single delimited string. Specify the delimiter and consider target field length limits.

6.6 Foreign Key and Primary Key Values

Key management is critical in migrations where the target system uses surrogate keys that do not exist in the source.

Generate New PK (Sequence) — For Oracle-style sequences. Obtain the next value and store the source-to-target key mapping for child record FK lookups.

Generate New PK (Identity) — For SQL Server or PostgreSQL-style identity/serial columns. The database generates the value on INSERT; capture it and store the mapping.

Pre-Defined PK Lookup — When target keys are pre-planned in a key mapping table before conversion runs. New source records discovered during conversion are assigned the next available key.

FK Lookup — For child records referencing converted parents. Look up the source FK in the key mapping table to find the target FK. Handle orphans explicitly: bypass the row, set NULL, or point to a catch-all parent record.

Parent tables must be converted before child tables. This ordering is a fundamental constraint that should be documented at the project level.

7. Exclusion Reason Codes

When a column is mapped to Excluded, the mapping must include a reason code in the transformation logic. These codes provide traceable documentation for auditors and reviewers explaining *why* something was not converted. Free-text explanations are not acceptable — reason codes enforce consistency and enable reporting.

7.1 Source Exclusion Codes

Code	Meaning
S01	Source field used for conversion logic only — does not exist as a discrete element in the target
S02	Source field not used by source system (deprecated, empty, or abandoned)
S03	Source field is a duplicate of another field that is mapped
S04	Corresponding target field uses a conversion default instead of this source data
S05	Source field is mapped in a different project
S06	Corresponding target field is populated by a non-conversion process

7.2 Target Exclusion Codes

Code	Meaning
T01	Target field does not exist in source system
T02	Target field is a system-assigned key (surrogate) not present in source
T03	Target field populated with a conversion marker value
T04	Target field populated by a non-conversion process (application, interface, manual)
T05	Target field not needed for this implementation
T06	Target field populated by a database default
T07	Target field explicitly set to NULL
T08	System-assigned key generated by conversion for referential integrity

7.3 Table-Level Exclusion Codes

Code	Meaning
X01	Source table not required for conversion after evaluation
X02	Target table will be empty at go-live
X03	Target table loaded entirely by application process

Code	Meaning
X04	Target table loaded entirely by manual process

These codes are a starting point. Organizations may add domain-specific codes (e.g., "regulatory data handled by compliance import" or "historical archive excluded per retention policy").

8. Notations

Notations are structured annotations attached to mappings, tables, or columns. Each notation type has a defined purpose and format. Notation types should be fully configurable — organizations can add, rename, or remove them to match their documentation practices. The one exception is transformation logic: every mapping needs exactly one designated place to document what happens to the data.

8.1 Recommended Notation Types

The following notation types are the defaults in Bridger. Each has a short name, a full descriptive name, a format (Text, Code, or SQL), and a scope that defines where it can be applied.

Transformation Logic — The transformation rule itself. The core mapping specification that developers code from and reviewers approve. Format: Code. Applies to: Mappings.

Mapping Note — Supplementary notes, context, or rationale for a specific mapping. Use for information that does not belong in the transformation logic itself — background on a business decision, a reference to a requirements document, or context for why a particular approach was chosen. Format: Text. Applies to: Mappings.

New Row Logic — Logic for determining when a new row should be created in the target table. Relevant when a single source record may produce multiple target records or when records from multiple source tables are merged. Format: Code. Applies to: Target Tables.

Table Note — Notes applicable to an entire table. Use for table-level context that applies across all columns — data volumes, refresh frequency, known quality issues, relationships to other tables. Format: Text. Applies to: Source and target tables, in both library and project contexts.

Column Note — Notes applicable to a specific column regardless of how it is mapped. Use for column-level context — known data quality issues, business rules that affect all mappings involving this column, or clarification of cryptic legacy column names. Format: Text. Applies to: Source and target columns, in both library and project contexts.

Historical Pattern — Historical context or change tracking notes. Use to document why a mapping was changed, what it used to be, or decisions made during earlier review cycles. Format: Text. Applies to: All levels.

Test Case: MINUS Query — Validation SQL using MINUS or EXCEPT to compare source and target datasets. Format: SQL. Applies to: Mappings.

Test Case: Count Reconciliation — Validation SQL for count-based comparison between source and target. Format: SQL. Applies to: Mappings.

Test Case: Default Value Check — Validation SQL to verify that default values were applied correctly. Format: SQL. Applies to: Mappings.

Test Case: Manual Review — Documentation of validation checks that require human judgment rather than automated queries. Format: SQL. Applies to: Mappings.

8.2 Writing Effective Transformation Logic

Transformation logic is the most important notation. It is the authoritative specification that conversion developers code from and that reviewers approve. Poorly written logic creates ambiguity, rework, and defects.

Do:

- Use the logic template library. Templates exist for a reason — they enforce consistent structure.
- Reference source fields explicitly by table and column name: `CUST_MSTR.CUST_TY_CD`.
- Use the established pseudocode patterns: `If/Then/Else`, `Evaluate/When`, `Perform FunctionName`.
- Specify exception handling in every branch: what happens when the validation fails?
- Use substitution variables for values that vary by environment: `&CONVERSION_DATE`, `&CONVERSION_USER`.

Do not:

- Write "same as above" or "see previous mapping." Every mapping's logic must stand alone.
- Use ambiguous pronouns: "convert it to the right format" — convert *what*, to *which* format?
- Leave the Else/When Others clause empty. Every branch must have an explicit outcome.
- Mix actual SQL with pseudocode in the same logic. Transformation logic describes *what* to do. Developers decide *how*.

9. Indicators

Indicators are metadata flags attached to tables or columns that track categorical properties across the entire mapping effort. Unlike statuses (which track workflow position), indicators track *characteristics* of the data or mapping that are relevant to planning and reporting.

9.1 Recommended Indicators

Indicator	Values	Purpose
Required for Conversion	Yes / No / N/A	Is this table or column in scope for the current conversion phase?
Dependent on Configuration	Yes / No / N/A	Does the mapping depend on target system configuration that may not be finalized?
Required by Client	Yes / No / N/A	Has the client explicitly identified this as a requirement?

9.2 Using Indicators

Indicators serve multiple purposes:

Scope management — "Required for Conversion" allows the team to mark tables and columns as in-scope or out-of-scope without deleting them from the library. Deferred tables stay visible for future phases.

Dependency tracking — "Dependent on Configuration" flags mappings that cannot be finalized until the target system is configured. These mappings should not be submitted for approval prematurely.

Prioritization — "Required by Client" highlights business-critical items that need early attention in the mapping schedule.

9.3 Configurability

Organizations can define additional indicators with custom value lists. Examples: a "Data Quality Risk" indicator (High/Medium/Low), a "Requires Profiling" flag (Yes/No), or a phase indicator (Phase 1/Phase 2/Phase 3) for multi-wave conversions. Indicators can be scoped to apply at the table level, column level, or both.

10. Exception Codes

Exception codes define the vocabulary for error handling in transformation logic. When a common routine or validation check fails, the exception code identifies *what went wrong* in a consistent, reportable way.

Each exception code has a severity level:

- **Info** — Non-critical. Logged for audit purposes. No action required. (Example: PADDING — whitespace was trimmed.)
- **Warning** — Potential issue. Data was converted using a fallback. Review recommended. (Example: DATERANGE — date was outside expected bounds.)
- **Error** — Significant issue. Data loss, integrity risk, or business rule violation. Investigation required. (Example: ORPHAN — foreign key reference not found.)

Exception codes are referenced in transformation logic and drive conversion reporting. A conversion summary showing "4,200 TRUNCATION errors across the Customer project" tells the team exactly where to focus data quality remediation.

Organizations can add custom exception codes for domain-specific validation failures.

11. Handling Schema Changes

Target systems evolve during the life of a migration project. The vendor releases a new version. The application team adds columns, changes data types, or renames fields. These changes must be detected, assessed, and reflected in the mappings.

11.1 Detection

When a new schema is received, import it alongside the existing library and compare it against the previous version to identify differences: added columns, removed columns, type changes, length changes, and renamed tables or columns. Bridger includes a schema comparison tool for this purpose; any capable mapping tool should provide similar functionality.

11.2 Impact Assessment

For each detected change, determine whether existing mappings are affected. A new column on a target table needs a mapping decision (source data, special table, or exclusion). A removed column invalidates any mappings that reference it. A type or length change may require updated transformation logic.

11.3 Re-Review Workflow

Affected mappings should be set to "Reopened After Approval" and re-enter the review cycle. This ensures that previously approved work is not silently invalidated by schema drift. The Historical Pattern notation type is useful for documenting what changed and why the mapping was reopened.

12. Validation and Quality

Mapping documentation is only valuable if the resulting conversion is correct. Validation is the bridge between documented intent and actual outcome.

12.1 Test Case Notations

Well-structured mapping documentation supports several categories of validation:

- **MINUS/EXCEPT queries** that compare source and target datasets. If the query returns rows, something was lost or transformed incorrectly.
- **Count reconciliation** between source and target. Total rows, rows by category, rows by status.
- **Default value checks** that verify Conversion Default and Database Default mappings produced the expected results.
- **Manual review** documentation for validation checks that require human judgment.

12.2 Auto-Generation and AI Assistance

When transformation logic is written using a consistent template library, simple validation scripts can be generated automatically. A direct "Move source to target" mapping produces a straightforward MINUS query. A mapping with a Case/Switch produces a count-by-value reconciliation. Bridger supports this auto-generation as well as AI-assisted validation for complex mappings — but the underlying principle applies regardless of tool: consistent, structured logic enables automation; free-text logic does not.

For complex mappings — multi-step transformations, Source Group derivations, conditional logic with multiple branches — AI assistance can generate validation SQL. The quality of that output depends directly on the quality of the transformation logic. Clean, template-based logic produces useful validation. Ambiguous free-text logic produces guesswork.

12.3 How Clean Mapping Drives Validation

When transformation logic is written consistently using the template library, validation can be generated or suggested with high confidence. Conversely, free-text transformation logic that says "convert the data appropriately" gives neither a human nor an AI agent enough information to generate a meaningful validation.

This is the practical payoff of standards discipline. The investment in writing structured transformation logic and using the template library pays dividends not just in clarity for reviewers, but in automated validation that catches defects early.

13. Review Sessions

Mapping review is iterative. The initial mapping is a starting point for discussion, not a finished product. And it is *always* a discussion — the migration team proposes, the stakeholders dispose.

13.1 Ground Rules

- **Time-box discussions.** If a topic exceeds five minutes without resolution, create an action item and move on. Use the Question/Issue status to flag the mapping.
- **Do not guess.** If a question arises and the answer is unclear, mark it as an action item. Do not provide speculative answers that may be recorded as approved decisions.
- **Review material in advance.** Review sessions are for discussion and approval, not first-time reading. Distribute exported mappings before the meeting.
- **Mapping approval is iterative.** Business rules change. Conversion testing reveals issues. Schema changes occur. Expect multiple review passes over the life of the project.

13.2 Exported Reports

A mapping tool should be able to generate source-to-target and target-to-source exports for stakeholder review. These exports should include transformation logic, mapping notes, validation scripts, statuses, and approval dates, and should be scopeable to a single project or across all projects. Bridger's reporting system generates these exports in Excel format.

Stakeholders who do not have direct access to the mapping tool review and annotate these exports. Their feedback is then applied to the mappings, and statuses are updated to reflect the outcome of the review.

13.3 Feedback Loop

The cycle is: export → distribute → review → collect feedback → update mappings → update statuses → re-export if needed. Each pass narrows the set of open items until all mappings reach an approved state.

14. Writing for AI Collaboration

AI agents — whether local models or cloud-hosted LLMs — can meaningfully assist with mapping work. They can suggest transformation logic, generate validation SQL, flag data type mismatches, and identify patterns across large mapping sets. But they can only do this if the mapping data is clean, consistent, and unambiguous.

The first principle applies here with full force: the migration team does not make decisions unilaterally, and neither does an AI agent. AI-suggested content — whether it is a proposed transformation rule, a validation query, or an entire batch of auto-generated mappings — is always a *suggestion* that requires human review and approval. The same two-tier review process that governs human-authored mappings governs AI-authored mappings. An AI agent is a powerful assistant, but it is not an authority.

14.1 Patterns That Help

Use logic templates consistently. When every Case/Switch follows the same `Evaluate / When / When Others / End` structure, an AI agent can parse thousands of mappings reliably.

Reference columns explicitly. `CUST_MSTR.CUST_TY_CD` is unambiguous. "The type code field" is not. An AI agent resolving references across a 200-table migration needs exact identifiers.

Use exception codes from the defined vocabulary. An AI agent can summarize risk across an entire project when every error follows the same code taxonomy. Ad-hoc descriptions like "bad data, check later" are invisible to automated analysis.

Document exclusion reasons with codes. S01 through X04 are parseable. "Not needed" is not.

Use substitution variables for environment-specific values. `&CONVERSION_DATE` tells an AI agent "this is a parameter that will be resolved at runtime." A hardcoded date like `'2026-03-15'` looks like transformation logic that should be validated.

14.2 Patterns That Hurt

"Same as above." An AI agent processing mappings row by row has no concept of "above." Every mapping must be self-contained.

Ambiguous pronouns. "Convert it to the right format" — an AI agent cannot resolve "it" or "right format."

Mixing SQL with pseudocode. Transformation logic that contains `SELECT CASE WHEN` in one mapping and `Evaluate / When` pseudocode in another forces the AI to handle two different grammars. Pick one and stay consistent.

Undocumented assumptions. "Use the standard date routine" without specifying which routine, what the default is on failure, or which exception code to report.

14.3 The AI Generated Status

When an AI agent creates or suggests mappings, those mappings should be assigned a distinct status that flags them as unreviewed. In Bridger this is the "AI Generated" status (purple), which is locked from the normal approval workflow — a human must change the status to another value before the mapping can progress toward review and approval. This ensures that AI-produced work is always acknowledged and vetted before entering the pipeline.

This document describes a methodology, not a product. The standards, statuses, templates, and codes described here are implemented as configurable defaults in Bridger and represent years of refinement across real-world migration engagements. They are shared freely as general best practice. Adapt them to your organization's needs — but understand the reasoning behind each standard before changing it.