

Informatica TDM (2): masking, generation, plans and workflows

Every masking technique and rule, mapplets and transformations, data generation, and how plans become workflows you monitor.

Informatica: masking data, rules and mapplets

Data masking replaces real sensitive values with realistic fake ones, so a tester sees "Divya Iyer" where production has "Asha Menon". The general idea is in the masking concepts chapter; this section covers how Informatica TDM does it.

5.1 Words to know first

Masking technique

The logic that replaces a value, such as "pick a name from a list" (substitution) or "swap values between rows" (shuffle).

Masking parameters

The options of a technique, such as the dictionary to use, the seed, or the range of dates allowed.

Masking rule

A technique plus its parameters, saved under a name. "FirstName_Substitution" is a rule. The same technique with different parameters is a different rule.

Data domain

A named group of columns that mean the same thing (for example "Social Security Number"), found by patterns in the data or in column names. You can attach preferred masking rules to it.

Policy

A bundle of data domains (and so their rules) that you reuse across projects. It holds no data source itself.

Deterministic / repeatable

The same input always gives the same masked output, so "Asha" becomes the same fake name in every table and in every run.

Seed

A starting number that makes repeatable output reproducible. Same seed, same rule, same input, same result.

Dictionary

A flat file or database table of replacement values used by substitution, for example a list of first names.

Mapplet

A reusable group of PowerCenter transformations with input and output ports. TDM can import one as a custom masking rule.

Value cascade / auto cascade

Ways to make related columns receive the same masked value: a value cascade for similar columns you group by hand, an auto cascade for primary and foreign keys.

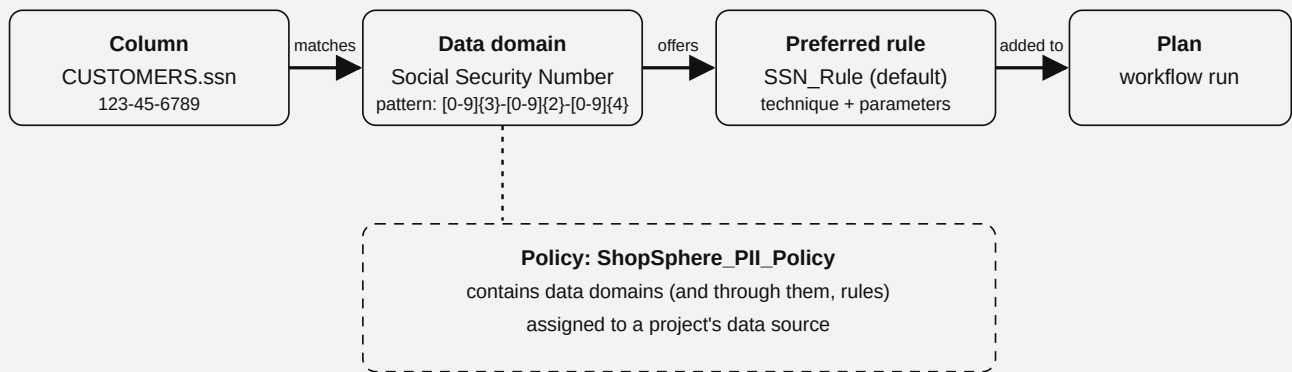
5.2 The big picture: rules, domains, policies and assignments

5.2.1 Six objects and how they connect

Start with the ShopSphere problem. Your **CUSTOMERS** table has first name, last name, email, phone, SSN, date of birth, zip code and tax ID, all sensitive. You do not want to configure "how to mask an SSN" nine times in nine projects. TDM lets you define it once and reuse it. The guide's building blocks (User Guide 10.5.10, p.71 to 73, p.135 to 138, p.149 to 150) :

Object	What it is	ShopSphere example
Rule	A masking technique, an optional rule qualifier and masking parameters. Can have an override option.	"SSN_Rule" using SSN masking
Data domain	The functional meaning of a column, identified from data patterns and column-name patterns; holds one or more preferred rules, one of them the default.	"Social Security Number Domain": data pattern for 999-99-9999 and column names matching SSN or SOCIAL
Policy	A collection of data domains. Contains no data source; can be applied to many projects.	"ShopSphere_PII_Policy"
Assignment	A rule attached to a column, either directly (column assignment) or through a data domain (domain assignment).	<code>CUSTOMERS.ssn</code> uses SSN_Rule
Plan	Defines the masking operation and whether to mask data in place in the source database or in stream into a target.	"ShopSphere_Mask_Plan"
Value cascade	Masks similar columns in different tables with the same rule so the values match.	A <code>customer_ref</code> column in another system masked like <code>customer_id</code>

You can create rules, domains and policies in any order. The guide's high-level policy task flow (p.72): create masking rules, define data domains, assign the rules to the domains, create a policy, assign the domains to the policy. You can apply the rules to source columns, data domains, policies and plans (p.136).



How a column ends up with a masking rule. A column is matched to a data domain (by data pattern or by column name), the domain offers a preferred rule, and the rule is added to a plan. The policy groups domains so the setup can be reused across projects. The Social Security pattern shown is the one the guide gives (User Guide 10.5.10, p.74 to 75).

5.2.2 Data domains and preferred rules

A data domain (User Guide 10.5.10, p.73 to 76) "represents the functional meaning of a column based on the column data or the column name". The guide's SSN example: social security numbers may sit in a column called `SSN`, in another table's `SOCIAL_SECURITY` and even in a `COMMENTS` column. You define a **data expression** for the format (regular expression `[0-9]{3}-[0-9]{2}-[0-9]{4}`) and **metadata expressions** for column names such as `.*SSN*` and `.*SOCIAL*`. Data discovery profiles (see the Informatica discovery and profiling chapter) use these patterns to assign domains to columns; without Data Discovery you assign columns by hand (p.73). Link back to discovery: profiling finds the sensitive columns, domains carry the rule to them.

When you attach masking rules to a domain they are called **preferred rules**. If you attach several, one is the **default rule**, applied to all columns of the domain, and you can change an individual column to another preferred rule (p.73). The guide's example: a `Last_Name` domain with shuffle as the default and substitution as another preferred rule for a different table.

TDM ships three policy packs, **PII**, **PHI** and **PCI**, each listing domains with a default rule (p.77 to 80). The PII pack, for example, pairs First Name Domain with First Name Rule, Phone Domain with Phone Number Rule and Social Security Number Domain with Social Security Number Rule. The guide does not say which technique each default rule uses, so open a rule in your install to see.

5.2.3 The task flow

A masking plan must contain at least one rule or one policy (User Guide 10.5.10, p.135). The PowerCenter Integration Service does the masking when the workflow runs. If a plan has both subset and masking

components, subsetting happens first (p.275), as in the Informatica data subset chapter. Plans are explained in the Informatica plans, workflows and warehouse chapter, and the screens in the Informatica TDM tour.

5.3 Creating masking rules

A rule is created in the Policies view, then added to a project, then assigned to columns. The guide recognises three kinds (User Guide 10.5.10, p.136 to 138) :

Kind	What it is	Columns	Use it when
Standard	A built-in technique with parameters. One masking technique per rule.	One input column, one output column	Most cases: mask a name, an SSN, a date
Advanced	A combination of techniques; masks multiple source columns, or one column based on more than one input.	Input, variable and output columns	A full name built from masked first and last name; conditional masking; dependent columns
Mapplet	Logic imported from a mapplet exported from PowerCenter or the Model repository.	Multiple input and output columns	You need a PowerCenter masking technique TDM does not offer natively

5.3.1 Creating a standard rule

1

Click Policies, then Actions > New > Masking Rule.

The Rule Wizard opens (User Guide 10.5.10, p.139).

2

Enter a name and optional description. Pick the datatype of the column the rule will be applied to.

Techniques depend on datatype: you cannot blur a string by a percentage. A rule can be created for string, numeric and date types (p.72).

3

Select the Standard masking rule type. Tick Override Allowed if users may change parameters when they assign the rule. Click Next.

Override is described below. The next screen changes depending on the masking technique you select (p.139).

4

Enter the masking parameters.

These decide exactly how values are masked: dictionary, seed, range, mask format and so on. Section 5.6 lists them per technique.

5

Enter the exception handling options.

Decide what to do with nulls, empty spaces and errors (14.5). Click Finish (p.139 to 140).

Expected result: The rule appears in the Policies view and can be edited, copied (Actions > Duplicate) or deleted there (p.139, p.146).

5.3.2 Previewing a rule: the Rule Simulator

The guide describes a built-in preview so you do not have to run a full workflow to see what a rule does (User Guide 10.5.10, p.137, p.140) . It works for **standard** rules only (the text says "simulate the output of a standard rule").

1

Policies view, click a masking rule name to open its Rule Properties, then click Rule Simulator.

This opens the simulator tab.

2

Choose a source in Source Details: Default, My Test Data or Connection, then click Edit.

Default uses sample data supplied by TDM. My Test Data lets you type rows (up to 100).

Connection reads real rows (the Configure Source Details page opens).

3

If Connection: open the Connection tab, pick connection and owner, Browse to the table, choose the column.

By default up to 10 random columns are listed; type the column name if yours is missing. Enter the number of rows: default 20, maximum 100. Click OK.

4

Click Go.

Expected result: The Rule Simulator tab lists original values beside masked values.

5

Adjust the rule and run the simulation again on the same data.

The latest simulation configuration is stored in the browser cache, and clearing the cache deletes it (p.137).

5.3.3 Override

With Override Allowed ticked, whoever assigns the rule may change its parameters for one column (p.136, p.146): select the column, Actions > Override, edit, Save. The Override column then reads Yes–Overridden. Different overridden seeds on columns that must match will break the match.

5.3.4 Creating a mapplet rule

The wizard path (p.140 to 141): Policies, Actions > New > Masking Rule, name, datatype, Mapplet masking type, browse to the exported XML (for Hadoop, select Hadoop and browse to a Developer tool mapplet), Next, mark *at least one input and one output column as mandatory*, Finish. Details and limits are in 14.9, which also walks through building a small mapplet in PowerCenter first.

5.3.5 Creating an advanced rule

1 Policies, Actions > New > Masking Rule. Enter a name and description, choose Advanced, click Next.

Advanced rules combine techniques (User Guide 10.5.10, p.141).

2 In Input Columns click Add New. Give the name, datatype, precision and scale. Choose whether assigning it is mandatory in every project.

You cannot enter masking properties on input columns, because input columns just receive the source value.

3 In Variable Columns click Add New. For each, enter an expression, a masking rule or a dependent column.

Variable columns are scratch fields for intermediate results. If you apply a masking rule, say which input column feeds it.

4 In Output Columns click Add New. Enter an expression, a masking rule or a dependent column.

The output column receives the final value written to the target.

5 Click OK and finish.

The column names in the rule do not have to match source column names; you map them at assignment time (p.156 to 157).

The full parameter tables for advanced rules are in 14.8.

5.4 Assigning rules to columns

5.4.1 Add rules to the project first

Rules live in the Policies area; a project only sees them after you add them. Projects, open the project, Overview > Policies, Actions > Add Additional Rules, tick your rules, OK; they appear under Additional Rules (User Guide 10.5.10, p.141 to 142) .

5.4.2 The Define | Data Masking view

Assignment happens in the project's Define | Data Masking view (p.138 to 139). Each row is one column of the source.

Column in the view	Meaning (p.138 to 139)
Name / Owner / Columns	Table name, schema name, column to mask
Data Type	Datatype of the column
Domain	Domain assigned by discovery or by hand
Sensitive	Yes or No: whether the column is marked sensitive
Similar Value Columns	Number of other columns in a cascade with this one
Policy	The policy the column is assigned to
Masking Rule	Click to choose a rule; a domain's preferred rule shows an asterisk before its name
Override	Yes if parameters may be overridden; Yes-Overridden once you have

5.4.3 Assigning a standard rule, step by step

- 1 Open the project, click Define | Data Masking, select a column.**
Choose the column that needs masking, for example `CUSTOMERS.first_name` .
- 2 If Domain is blank, click the Policy column and pick a policy that contains the rule you want.**

The list of rules comes from the data domain's preferred rules, then the policy's rules (p.142, p.145).

3

Click inside the Masking Rule column and pick a rule.

Domain preferred rules are at the top; the rest of the policy's rules are below. For a substitution rule you can set the rule assignment parameters (lookup condition, unique substitution column, see 14.6). For a mapplet or advanced rule you map columns (below).

4

Click Save for each column.

If you move to another row without saving, TDM discards the assignment (p.145).

5

If you changed an assignment that other objects use, the Impacted Objects dialog lists plans, columns and domains.

Click Export for a CSV, Continue to save. Regenerate and rerun the affected plans (p.142).

NO RULES APPEAR IN THE LIST?

No rules listed? Check that a rule exists for the column datatype, that the project has a policy with masking rules, and that the domain's preferred rule is in a policy assigned to the project (p.145). The Integration Service does not mask integers longer than 28 characters (p.145, p.152).

5.4.4 Assigning a mapplet or advanced rule (port mapping)

These rules have named input and output ports, so you map source columns to them (User Guide 10.5.10, p.142 to 143) . Pick the rule in the Masking Rule column; a Custom Rule Assignment or Advanced Rule Assignment dialog opens. Select a source column and a same-datatype rule input port, click Link (or Show Simple), then Next, link rule output ports to source columns, Save. For an *expression cascade* in an advanced rule, Add Associated Table lets you combine two tables with a join condition. Sybase to Hadoop limits a custom rule to 20 columns (p.143).

5.4.5 Many columns, many rules

- **One rule on many columns:** search for similar columns, for example all ID columns, tick them and click Single Rule Assignment. Pick a policy and a masking rule, click OK (p.147 to 148). The guide's figure on p.148 shows this dialog.
- **Clearing:** select the column and click Clear Assignment, confirm, and Continue if the Impacted Objects dialog appears (p.148).

5.4.6 Cascades: keeping values consistent between tables

The ShopSphere `customer_id` is a primary key in `CUSTOMERS` and a foreign key in `ORDERS`. If you mask the key but the foreign key keeps its old value, the join breaks. TDM handles this with cascades (User Guide 10.5.10, p.90 to 91) :

- **Auto cascade:** masks a primary key in a parent table and the related foreign keys in child tables with the same value. Applied by default to tables with key constraints; can be switched off or on. Needs key constraints between parent and child. Not available for XSD, MongoDB and Cassandra sources.
- **Value cascade:** for similar columns *not* linked by key constraints (the guide's example: `employee_num` in one table and `EmpID` in another). Select the columns in Discover | Columns, Actions > Set as similar value columns, and choose a driving column. A rule assigned to the driving column is assigned to all the group. The columns must have the same type. In the masking view the driving column shows the number of other columns in the cascade.
- **Not for computed columns:** you cannot apply masking or generation rules to computed columns or build cascades on them (p.90).

5.4.7 Changing or deleting rules after assignment

Editing or deleting a rule shows an Impacted Objects dialog, but changes do *not* flow into plans: regenerate and rerun them. Deleting a rule does not remove it from the columns, domains and plans using it. If you change a custom rule's field names or types, import the mapplet again (p.144 to 146) . For XSD sources you apply rules to XML elements and attributes in Discover > Files (p.144); shuffle is not allowed for XSD (p.135).

5.5 Parameters that most techniques share

Before the techniques, three ideas appear again and again (User Guide 10.5.10, p.153 to 155) .

5.5.1 Repeatable output and the seed

Repeatable output returns deterministic values: the same input gives the same masked value, every time you generate and run the workflow. Use it when the same value occurs in several tables and the target tables must show the same masked value. The guide's example: customer John Smith has account numbers 1234 and 5678 in many tables; masking turns John Smith into Frank Martinez everywhere, 1234 into 6549, 5678 into 3214, on every run.

The **seed** is the generator's starting point: 1 to 999, default 1 (p.154; the key masking string table says 1,000, p.168). Give the *same* seed to columns that must match, such as `Cust_ID` in four tables. You can put the

seed in a parameter file variable. A TDM administrator can set a **central seed** that overrides every other seed (p.155).

Substitution and expression masking keep a *storage table* of source and masked value pairs; you name the storage connection in the plan (p.286: `IDM_SUBSTITUTION_STORAGE` , `IDM_EXPRESSION_STORAGE`), and expression masking fails without it (p.163). Key masking is deterministic from source, rule and seed (p.164). Repeatable means "same in, same out", not "reversible".

5.5.2 Exception handling

Every rule has an Exception Handling section that decides what happens with nulls, blanks and errors (p.155):

Option	What it does
Preprocessing Expression	Optional expression applied to the data before masking
Post processing Expression	Optional expression applied to the masked data before it is written
Null and Empty Spaces	Constant (mask with a value you give); Log error and continue; Treat as value (mask the space or null as a valid value); Ignore (leave it unmasked). For dependent masking columns there is no separate rule: they follow the column they depend on.
Error Handling	Constant; Log exception and continue; Ignore and continue; Error (log and stop)
Trim Leading or Trailing Spaces	When on, " Jones", "Jones" and "Jones " are the same value

5.6 The masking techniques, one by one

The guide lists these techniques (User Guide 10.5.10, p.152 to 153): Advanced, Credit card, Email address, Encryption, Expression, IP address, Key, Mapplet, Nullification, Phone, Random, Shuffle, SIN, SSN,

Substitution and URL. Advanced and Mapplet are in 14.8 and 14.9; the other fourteen follow.

5.6.1 Substitution masking

What it does. Replaces a value with a "similar but unrelated" value looked up in a **dictionary**, a flat file or database table of replacement values (User Guide 10.5.10, p.176 to 177) . Works on date, numeric and string columns. The guide calls it an effective way to replace production data with realistic test data; its examples are a dictionary of addresses, or the `InvalidSSN` dictionary of Social Security numbers that are not valid.

Where	Parameter	Meaning
Rule assignment	Lookup Condition (optional)	Column of the source table to match against a column of the dictionary, for example match on gender so a female first name gets a female replacement
Rule assignment	Unique Substitution Column (optional)	Source column to substitute with unique data
Rule	Repeatable Output	Same result between sessions; masked values saved in the storage table
Rule	Seed	Start number for deterministic results. Required for deterministic output
Rule	Unique Substitution Data	Every unique source value gets a unique masked value. Fails if the source has more distinct values than the dictionary. Default is nonunique, "for security"
Rule	Dictionary Information (required)	Dictionary (the file or table); Masked Value (column returned); Lookup Column (source column used in the lookup); Serial Number Column; Sort Column (to keep results deterministic if a relational dictionary is moved and its row order changes)

ShopSphere example. A flat file dictionary `FIRSTNAMES.csv` :

```
SNO,GENDER,FIRSTNAME
1,F,Divya
2,M,Arjun
3,F,Latha
4,M,Karthik
```

SUBSTITUTION ON FIRST_NAME (EXAMPLE OUTPUT)

customer_id	first_name before	first_name after
1001	Asha	Divya
1002	Ravi	Arjun
1003	Meera	Latha

Gotchas.

- Choose Unique Substitution Data only if the dictionary is bigger than the number of distinct values in the column. A dictionary of 4 names cannot uniquely replace 20 customers; the operation fails.
- Two customers with the same first name get the same replacement only if Repeatable Output is on (the storage table remembers pairs). Without repeatability, each row can get a different value.

Repeatability. Tick Repeatable Output and set a seed. Name the storage connection in the plan.

5.6.2 Shuffle masking

What it does. Swaps values of one column between rows of the same table. The guide's example (p.173) uses four rows: Tom Bender, Sue Slade, Bob Bold, Eli Jones. After a shuffle of the first-name column they read Bob Bender, Eli Slade, Tom Bold, Sue Jones. You can restrict which values may be swapped by a lookup condition or a constraint. Works on date, numeric and string columns .

Parameter	Meaning (p.174)
Shuffle Type: Random	Values move from row to row without checking that target values are unique per source value. 12345 may become 65432 in one row, and 33333 may become 12345 in another.
Shuffle Type: Representative	All source rows with the same value get the same shuffle value (if 12345 becomes 65432 once, it does everywhere). Does not save values between workflow runs; use repeatable masking to get the same values across runs.
Seed	1 to 999, default 1. Enabled only when Representative is selected.
Constrained	Restrict shuffling to rows constrained by another column, such as shuffle employee names within gender, or addresses within the same city. You choose the constraint column when you assign the rule to columns.

SHUFFLE ON LAST_NAME (EXAMPLE OUTPUT; THE VALUES COME FROM THE SAME COLUMN)			
customer_id	first_name	last_name before	last_name after
1001	Asha	Menon	Nair
1002	Ravi	Kumar	Menon
1003	Meera	Nair	Kumar

Gotchas.

- It cannot be used for XSD sources, and not when both source and target are Hadoop HDFS connections (p.135, p.173). A Spark engine cannot do it (p.62).
- Empty strings: set Null and Empty Spaces to Treat as Value, or the empty columns are skipped (p.173).

5.6.3 Key masking

What it does. Produces **deterministic** results: the same source value, rule and seed always give the same masked value. Works on date, numeric and string columns (User Guide 10.5.10, p.164 to 168) .

Parameter	Meaning
Seed	Start number that makes output deterministic. Apply the same seed to columns that must match (string table: 1 to 1,000; see the note in 14.5)
Seed value is a parameter	Tick if a plan parameter supplies the seed
Mask Format (string)	What kind of character may replace each input character (table below)
Source String Characters (string)	Which characters to mask. Options: Mask Only (mask just these) or Mask all except (mask everything but these). Blank means all characters. Case sensitive
Result String Replacement Characters (string)	Characters allowed in the output. Options: Use only or Use all except
Case Insensitive (string)	Treat upper and lower case as the same input character
Delimited String Masking and Delimiters (string)	Also mask the value when it appears as a substring in other columns, splitting on delimiters you list

The **mask format** characters (shared with random masking; p.165):

Char	Means
A	Alphabetical, ASCII a to z and A to Z
D	Digit 0 to 9
N	Alphanumeric
X	Any character
+	No masking at that position
R	Remaining characters can be any type; must be the last character of the mask

If no mask format is given, each source character is replaced with any character. A mask format longer than the input is truncated; **shorter than the input means the characters at the end are not masked**. Lowercase letters you type are converted to uppercase (p.164).

KEY MASKING OF AN ALPHANUMERIC TAX_ID WITH MASK FORMAT AAAAADDDDA (EXAMPLE OUTPUT)		
customer_id	tax_id before	tax_id after
1001	ABCPM1234K	QWZRT5820M
1002	ZZTPK9087L	HNBXC4413D

Other forms the guide describes.

- *Delimited strings*: the guide's example masks "John Donne" consistently when it appears as the strings John and Donne in two columns and as "Donne, John" or "John Donne, General Manager" in two more. Assign delimited string masking with comma and space as delimiters to the last two (p.166 to 167).
- *Dates*: output dates fall between 1753 and 2400 and are always valid. A leap year maps to a leap year; a 31-day month to a 31-day month; February stays February (p.167). Give the same seed as another column to get matching dates.
- *Numbers*: to keep a primary-key to foreign-key relationship intact between two tables, give the key column in each table the **same seed**. Deterministic output means 1001 in CUSTOMERS and 1001 in ORDERS both become the same new number (p.167).

Gotchas. Output is not guaranteed unique when there are fewer source characters than result characters (p.165). By default a masked character may differ from the original only in case ("x" to "X"); Case Insensitive treats both cases as one character so it cannot. To get identical output for "John Brown" and "JOHN BROWN", convert to one case first with a preprocessing expression (p.166).

5.6.4 Random masking

What it does. Produces random, non-repeatable output: the same source value and rule can give a different result each run. No seed. Works on date, numeric and string columns (User Guide 10.5.10, p.169 to 173) .

Option	For	Meaning
Range	date, numeric, string	Output between a minimum and maximum. For strings it is a range of lengths. For dates the maximum must be later than the minimum
Blurring	date, numeric	Output within a fixed or percent variance of the source value
Mask Format	string	Same A, D, N, X, +, R characters as key masking
Source String Characters	string	Which characters to mask: Mask Only or Mask all except
Result String Replacement Characters	string	Use only or Use all except

Numeric blurring with a source value of 66 (guide table, p.170):

Type	Low	High	Result
Fixed	0	10	between 66 and 76
Fixed	10	0	between 56 and 66
Fixed	10	10	between 56 and 76
Percent	0	50	between 66 and 99
Percent	50	0	between 33 and 66
Percent	50	50	between 33 and 99

Bounds must be zero or greater. **Date blurring** takes a unit (year by default), a low and a high bound: unit year, bounds 2 and 2 on 02 February 2006 gives a date from 02 February 2004 to 02 February 2008 (p.170). The unit list differs between p.170 (to hour) and p.172 (to second); check your drop-down.

RANDOM MASKING ON SHOPSPHERE COLUMNS (EXAMPLE OUTPUT)

Column	Setting	Before	After
date_of_birth	Blurring, unit year, low 2, high 2	1990-04-12	1991-03-30 (somewhere in 1988-04-12 to 1992-04-12)
zip_code	Mask format <code>+++DDD</code>	560001	560417 (first three digits kept)

Gotchas. No seed, so reruns differ and the same customer will not match across tables. A short mask format leaves trailing characters unmasked: `++DD` on a 6 digit zip leaves the last digit as it was. If you need repeatability use key masking.

5.6.5 Encryption masking

What it does. Applies an encryption algorithm to a string column (User Guide 10.5.10, p.161 to 162) . Two families:

Type	Options	Notes
Standard Encryption	AES (Advanced Encryption Standard, 128-bit), CRC (cyclic redundancy check, a checksum for finding transmission errors), MD5 (one-way hash, 128-bit value)	You enter an encryption key of 16 characters or fewer
Format Preserving Encryption	Preserve Format and Metadata; Preserve Metadata; Change Metadata	Can be <i>decrypted</i> later with the same pass phrase and the same rule configuration. Not usable if the data has UTF-8 four-byte characters

The three format preserving options in the guide's own words (p.161 to 162):

- **Preserve Format and Metadata:** keeps format and length. Upper case becomes upper case, lower case lower case, digits digits, special characters special characters. Example: `Abc123@xyz.com` may become `Mpz849#dje!kuw` ; with "@" and "." set as Do Not Encrypt Characters, `Mpz849@dje.kuw` .
- **Preserve Metadata:** keeps only length. `Alexander` may become `j16#HB91v` .
- **Change Metadata:** length and format change. `London` may become `Xuep@8f5` or `6ky#ke33h*we` . Before using it you must raise the column precision in the database (and in TDM) using: Required

Precision = (1.33 x Original Precision) + 24, rounded up. After changing the database column, reimport metadata or edit the precision in Test Data Manager.

Parameter	Meaning (p.162)
Encryption Type	Standard Encryption or Format Preserving Encryption
Encryption Technique	The technique to encrypt or decrypt the data
Encryption Key	Input key for standard encryption, 16 characters or fewer
Publish Rule	Makes a format preserving rule read-only; only domain administrators can make it editable again or delete it
Do Not Encrypt Characters	Characters left as they are. A leading or trailing space is respected but not displayed in the field

Use it when you must be able to get the original back (decryption is described only for format preserving encryption) or the application checks the shape of a value. Reversibility is also a risk: anyone with the key or pass phrase and the rule can recover production values The guide does not say where the pass phrase is entered. Same key and configuration give the same output for the same input.

5.6.6 Expression masking

What it does. Applies a PowerCenter-style expression to a column to create or change data. You build the expression in the Expression Builder by clicking columns, functions, variables and operators. Any datatype (User Guide 10.5.10, p.162 to 163) .

Guide example: to join first names with the text ABC, enter `CONCAT(FirstName, 'ABC')` . On ShopSphere Asha becomes AshaABC, a poor mask on its own but it shows the idea; a better use builds a value from other pieces, such as `'CUST' || customer_id` .

Parameter	Meaning
Repeatable Output	Keeps the masked value for a given column value; you must enter a seed if you enable it
Alphanumeric Seed	Any text. Use the same seed in several rules that must return the same masked values
Expression	A PowerCenter expression working on values in a single row

Gotchas (p.163): use it for one column (use advanced masking for several); make the expression return the column datatype, or you get zero (numeric) or NULL (string); one expression's output cannot feed another; repeatable output needs a storage table in the plan or the operation fails; no repeatable expression masking for Hadoop.

5.6.7 Nullification masking

What it does. Replaces the value with NULL. Works on binary, date, numeric and string data. No parameters (p.168) .

Use it for columns the test environment does not need: the strongest privacy result. The target column must allow NULL, and the application may expect an SSN to exist.

5.6.8 SSN masking

What it does. Applies a built-in mask for Social Security numbers on string columns (User Guide 10.5.10, p.175) . It generates valid-looking numbers and tries to avoid numbers the US Social Security Administration has actually issued, using the High Group List. The service reads that file from the PowerCenter installation's `infa_shared/SrcFiles` directory; you can download the latest version from the Social Security Administration. Any nine-digit input is accepted, with any delimiters.

Parameters. Repeatable Output with a seed value; an SSN randomization option, based on the method the Social Security Administration developed to remove the geographic meaning of the first three digits. No other parameter names are documented.

SSN MASKING (EXAMPLE OUTPUT)		
customer_id	ssn before	ssn after
1001	123-45-6789	612-73-4410
1002	234-56-7890	478-09-2231
1003	345-67-8901	559-21-7068

Gotchas. TDM "cannot return all unique" SSNs, because it cannot return numbers already issued, so test for duplicates if `ssn` has a unique constraint. Any delimiter style is accepted; check the output style suits your validation. **Repeatability.** Tick Repeatable Output, tick Seed, enter a value.

5.6.9 SIN masking

SIN is the Canadian Social Insurance Number (general background). The technique handles a nine-digit SIN with any delimiters on string columns (p.175) : no delimiters in gives none out, otherwise `xxx-xxx-xxx` . Start Digit forces the first digit; Repeatable Output takes a seed. ShopSphere has no SIN column.

5.6.10 Phone masking

What it does. Changes phone numbers while keeping the original format; string columns only (p.169) . The guide's example: `(408)382 0658` becomes `(408)256 3106` . Input may contain digits, spaces, hyphens and parentheses; alphabetic and special characters are not masked. Example: 9876543210 might become 9876012345. No parameters are listed, and which digits are preserved is not documented; in the guide's example the first three stayed, but do not assume that holds for every format.

5.6.11 Credit card masking

What it does. Applies a built-in format to credit card numbers on string columns, producing a logically valid number with a valid checksum (User Guide 10.5.10, p.158 to 159) . The source must be 13 to 19 digits with a valid checksum under card industry rules. It may contain numbers, spaces and hyphens. Invalid characters or length: the service logs an error in the session log and applies a default credit card mask. The six-digit Bank Identification Number (BIN) at the front is **not** masked. The guide's example: `4539 1596 8210 2773` may become `4539 1516 0556 7067` .

Parameter	Meaning
Repeatable	Same masked value across runs and across tables
Seed	Start for repeatable output, 1 to 999, default 1
Keep card issuer	Output has the same card type (Visa in, Visa-format out)
Replace card issuer	Output has a different type: choose AMEX, VISA, JCB, MASTERCARD or ANY (default ANY)

5.6.12 Email masking

What it does. Generates realistic email addresses on string columns. It can build the user name from the *masked* first and last names, and use a constant or a dictionary-chosen domain (p.159 to 161) . Output is always ASCII. It can be repeatable between workflows or within one workflow.

Group	Parameter	Meaning
General	Repeatable output, Seed	Same email masked the same way; seed 1 to 999, default 1
General	Standard	Replaces the string with characters; user and domain names are not realistic
General	Advanced	Builds from names and domains in a dictionary or mapping
General	Columns to form email	From mapping (assign columns when you assign the rule) or From dictionary
User name	Dictionary for columns	A dictionary containing first and last names
User name	Expression	For From mapping: build the user name from parts of first and last name; names in the expression need not match source columns
User name	First name column, First name length	Dictionary column and how many characters to use (first letter, first 4); default 10
User name	Delimiter	Period, underscore or hyphen between first and last name
User name	Last name column, Last name length	Same for the last name; default 10
Domain	Constant	Every address gets the same domain
Domain	Random, Domain lookup dictionary, Domain name column	Pick domains at random from a flat file or relational dictionary, naming the column

ADVANCED EMAIL MASKING, USER NAME FROM MASKED NAMES, DELIMITER PERIOD, CONSTANT DOMAIN
(EXAMPLE OUTPUT)

Masked first/last name	email before	email after
Divya Iyer	asha.menon@example.com	divya.iyer@example.com
Arjun Das	ravi.k@example.com	arjun.das@example.com

Gotchas. The From mapping option needs the masked name columns, which usually means one advanced rule that masks the names and builds the email (14.8). An email still containing "asha.menon" defeats the name mask.

5.6.13 IP address masking

Splits an IP address into four numbers and masks the network number within its range; string columns (p.164) . Class A stays Class A and 10.x.x.x stays 10.x.x.x : 11.12.23.34 may become 75.32.42.52 , 10.23.24.32 may become 10.61.74.84 . No parameters are listed.

5.6.14 URL masking

Masks URLs on string columns by finding :// and masking what follows (p.177) ; the protocol is not masked. http://www.yahoo.com may return http://MgL.aHjCa.VsD/ , ASCII only and possibly invalid. No parameters are listed.

5.7 Comparison and decision table

From the guide's technique sections (User Guide 10.5.10, p.152 to 177); "Not described" means the guide does not say.

Technique	Datatypes	Output looks real?	Repeatable?	Reversible?	Good ShopSphere fit
Substitution	date, numeric, string	Yes, from a dictionary	Optional (seed, storage table)	No	first_name, last_name, city
Shuffle	date, numeric, string	Yes, real values from the same column	Representative plus seed	No	last_name, zip_code distribution
Key	date, numeric, string	Looks random but keeps character types	Always (same seed)	Not described	tax_id, customer_id (same seed on PK and FK)
Random	date, numeric, string	Controlled by range or blur	No	No	date_of_birth (blur), zip_code (format)
Encryption	string	Format preserving: shape kept; standard: not	Same key gives same output	Format preserving: yes, with passphrase	email, when originals must be recoverable
Expression	all	Whatever you build	Optional (seed, storage table)	Depends on expression	Building a code from other values

Technique	Datatypes	Output looks real?	Repeatable?	Reversible?	Good ShopSphere fit
Nullification	binary, date, numeric, string	No, NULL	n/a	No	A column nobody needs, such as ssn in a UI-only test
SSN	string	Valid-looking SSNs	Optional (seed)	No	ssn
SIN	string	Valid-looking, xxx-xxx-xxx	Optional (seed)	No	(not in ShopSphere)
Phone	string	Same format, new digits	Not described	No	phone
Credit card	string	Valid checksum, BIN kept	Optional (seed)	No	(not in ShopSphere)
Email	string	Yes (advanced)	Optional (seed)	No	email
IP address	string	Yes, same class	Not described	No	(not in ShopSphere)
URL	string	Looks random, may be invalid	Not described	No	(not in ShopSphere)

Technique	Datatypes	Output looks real?	Repeatable?	Reversible?	Good ShopSphere fit
Advanced	all	Whatever you combine	Per inner rule	Per inner rule	Full name from first and last
Mapplet	per mapplet	Per mapplet	Per mapplet	Per mapplet	A rule only PowerCenter can do

5.7.1 Choosing a technique: questions to ask

- **Does the test environment need this column at all?** If not, nullification is the simplest and safest.
- **Does the application check a format (SSN pattern, card checksum, phone layout, URL)?** Use the matching built-in technique (SSN, SIN, credit card, phone, email, IP, URL) so the checks pass.
- **Does it need real-looking text such as names and places?** Substitution (new values from a dictionary) or shuffle (existing values moved between rows).
- **Is the value an identifier that appears in several tables and must stay joinable?** Use key masking with the same seed, or rely on auto cascade for keys with constraints, or repeatable substitution with the same seed (14.4, 14.5).
- **Is it a number or date where only "near the original" matters?** Random masking with blurring.
- **Do you need to undo the masking later?** Format preserving encryption, knowing the privacy trade-off.
- **Does a masked value depend on more than one column?** An advanced rule, or a mapplet if PowerCenter logic already exists.

5.8 Advanced masking rules in depth

An advanced rule is a small program built from columns of three types (User Guide 10.5.10, p.137 to 138, p.156 to 158) : **input** (receives the source value), **variable** (a scratch column that holds an intermediate result from an expression or a masking technique, so you can combine several techniques) and **output** (the target column that receives the final value). Column names inside the rule are your own; at assignment time you map them to real columns.

Property	For	Meaning
Column Name, Column Type, Datatype, Precision, Scale, Mandatory	input, variabl e, output	General properties. Mandatory means the column must be assigned to a source column in every project (input and output only)
Expression	variabl e, output	An expression built in the Expression Builder
Masking Rule	variabl e, output	Apply an existing rule to an input column and write the result to this column. Sub-options: Condition (if true, mask the column), Rule , Override properties (if the rule owner allowed it), Input column
Condition Input	output only	Pick a condition applied earlier on an input column
Dependent	variabl e, output	Dependent masking: replaces a column's value with the value from the <i>same dictionary row</i> that another column's substitution returned. You must define substitution on another column first. Sub-options: Input column (the column configured for substitution) and Dictionary column

5.8.1 Worked example: a masked full name

The guide's example (p.158) builds a masked full name from first and last name, and it fits ShopSphere. Say the target has a `full_name` column derived from `first_name` and `last_name` :

Column	Type	Technique	Expression
FIRST_NAME	Input	none	none
LAST_NAME	Input	none	none
FIRST_MASKED	Variable	Substitution on FIRST_NAME	none
LAST_MASKED	Variable	Substitution on LAST_NAME	none
FULL_NAME	Output	Expression	<code>FIRST_MASKED ' ' LAST_MASKED</code>

Result for customer 1001: input Asha and Menon; variables Divya and Iyer; output "Divya Iyer". The guide's advice: build expressions by point-and-click to minimise errors, and check that the expression returns the output column's datatype.

5.8.2 Worked example: first names by gender and nationality (guide)

The guide's name example (p.177 to 182) keeps gender and nationality: a dictionary with `SNO`, `COUNTRY`, `GENDER`, `FIRSTNAME`; a string substitution rule with Lookup column `GENDER`; then an advanced rule with inputs `in_Country`, `in_FirstName`, `in_Gender`, variable columns `var_FirstName_us` (condition `in_Country='us'`) and `var_FirstName_jp` (condition `in_Country='jp'`) that each apply the substitution rule, and output `o_FirstName`. A condition on a variable column lets one rule choose a different substitution per row.

5.8.3 Worked example: shuffle addresses, keep ZIP (guide)

The shuffle address example (p.183 to 189) shuffles address lines together and leaves ZIP unmasked: a Random, Constrained shuffle rule, then an advanced rule where `v_AddressLine3` applies the shuffle with lookup column `i_ZIP`, and `v_AddressLine1` and `v_AddressLine2` use **dependent masking** on it so they follow the same row.

5.9 Custom rules: mapplets and the Data Masking transformation

Three parts: what a mapplet is, a worked build in PowerCenter, and how Test Data Manager uses a mapplet once it exists. The build follows the LearningMart video "How to create mapplet in informatica";

the TDM side follows the User Guide.

5.9.1 The general idea

A *transformation* in PowerCenter is one processing step on rows of data (calculate, look up, mask). In the LearningMart video "How to create mapplet in informatica", the presenter defines a *mapplet* as a reusable object that you create in the Mapplet Designer. His motivating case: many mappings must join first name and last name into a full name, so you build that logic once and reuse it everywhere, like a function you write once and call from many programs. A mapplet therefore has input ports, some transformations inside, and output ports. A *Data Masking transformation* applies masking techniques to a port; a mapplet that contains one is how you bring a PowerCenter masking technique into TDM.

5.9.2 Build a mapplet: full name from first and last name

This follows the video's build step by step. It is a PowerCenter task done in PowerCenter Designer, not in Test Data Manager. The video does not state the PowerCenter version.

WHAT THE FINISHED MAPPLET DOES TO SHOPSPHERE CUSTOMERS ROWS (EXAMPLE DATA)		
first_name (in)	last_name (in)	full_name (out)
Asha	Menon	Asha Menon
Ravi	Kumar	Ravi Kumar
Meera	Nair	Meera Nair

1

Open the Mapplet Designer and name the mapplet.

The Mapplet Designer is the part of PowerCenter Designer where mapplets are built. Use a clear prefix such as `MLT_` so mapplets are easy to tell from mappings.

Expected result: An empty mapplet workspace with your name on it.

2

Add a Mapplet Input transformation with two input ports, `first_name` and `last_name`.

The video says a mapplet has two edge pieces: an input transformation whose ports carry data in, and an output transformation whose ports carry data out. It opens the Transformation tab at the top of the Designer, chooses the Mapplet Input transformation, names it and adds the two ports.

Expected result: A Mapplet Input box listing `first_name` and `last_name`.

3

Add an Expression transformation and drag the two input ports into it.

Joining two values needs a calculation, and the video uses an Expression transformation for that (a step that computes a new value for every row). Dragging the ports copies them in as inputs.

Expected result: The Expression transformation shows `first_name` and `last_name`, linked from the Mapplet Input.

4

Create a new output port in the Expression transformation called `full_name`, with a size above 40.

The video explains the size: this port holds two values joined together, so it must be longer than a single name. Make it at least the two input sizes plus one for the space, because a smaller port risks cutting off long names.

Expected result: A `full_name` port flagged as an output port.

5

Write the expression: `first_name || ' ' || last_name`.

You can use the concatenation function or the pipe operator `||`; the video picks the pipes, with a space between the two names so they do not run together. It then validates the expression and clicks Apply and OK.

Expected result: The expression editor reports that the expression is valid.

6

Add a Mapplet Output transformation, name it, and drag `full_name` into it.

Without this step nothing leaves the mapplet. The output port name is what a mapping or a TDM rule will see.

Expected result: A Mapplet Output box with the `full_name` port, linked from the Expression.

7

Validate from the Mapplets menu, then save.

Validation catches broken links and bad expressions before a mapping depends on the mapplet. The video does it at the Mapplets menu, Validate.

Expected result: A message that the mapplet is valid, then a saved mapplet.

8

Create a mapping with an EMPLOYEE source and a target table, and drag the saved mapplet in between.

This proves the mapplet works. The video's target has a full name column. The mapplet appears with its input and output ports.

Expected result: Source, mapplet and target on one canvas.

9

Connect first and last name from the source to the mapplet inputs, and the mapplet's `full_name` to the target's full name column. Save.

Each connection tells PowerCenter which column feeds which port.

Expected result: The mapping validates ("mapping is valid").

10

Create the workflow in Workflow Manager. Check the source and target connection values, and choose Normal load when the target has a primary key. Save.

A workflow is what actually runs the mapping. The video gives the Normal-load rule without a reason; a common reason is that bulk loading skips some database checking and logging, so it may not suit a constrained target.

Expected result: A saved workflow with valid connections.

11

Run the workflow and check the target data.

Reading the rows is the real test.

Expected result: The run succeeds and the full name column shows first and last name together, as in the table above.

MISTAKES THIS VIDEO'S STEPS GUARD AGAINST

- Forgetting the Mapplet Output: the mapplet has nothing to return.
- Using a mapplet without validating it first.
- Making the `full_name` port too small for a joined value.
- Choosing the wrong load type when the target has a primary key (the video says Normal).
- Leaving out the space, which gives "AshaMenon".

5.9.3 From PowerCenter mapplet to TDM rule

Steps 8 to 11 above build a mapping and workflow by hand. TDM does not need those, because the PowerCenter Integration Service creates the mappings, transformations, mapplets, sessions and workflows itself when a plan runs, and the guide warns that editing generated components in the PowerCenter Client may give unexpected results (User Guide 10.5.10, p.141, p.150). What TDM needs is the mapplet. So the sequence is: build and validate the mapplet in PowerCenter (steps 1 to 7), export it to an XML file from the PowerCenter repository, then import it in Test Data Manager as a rule (p.137, p.140).

In the wizard you mark at least one input and one output column as mandatory, so for the video's mapplet you might mark `first_name` and `full_name` (p.141). When you assign the rule you link source columns to the input ports and the output port to a column (p.142 to 143).

The video's mapplet contains only an Expression, so it combines values and masks nothing. For masking, the guide says to include one or more Data Masking transformations in the mapplet and that all transformations must be passive (p.150); it does not show how to configure one. The guide's own full-name example is on the generation side: a custom generation rule from a mapplet that combines first and last name from an EMPLOYEES table (User Guide 10.5.10, p.196, p.211; see the Informatica data generation chapter). That is the closest fit for this mapplet.

5.9.4 What the guide says

- **Purpose.** Mapplet rules let you use PowerCenter masking techniques that are "not available in Test Data Manager" (p.150). The mapplet holds the logic to mask the input columns and return data to the target columns (p.137, p.152).
- **How it gets in.** You export the mapplet to an XML file from the PowerCenter repository or the Model repository, then import it as a rule (p.137, p.140). To mask Hadoop data you import mapplets built in the Developer tool (p.137). The TDM repository stores the mapplet logic, and **you cannot change the mapplet inside Test Data Manager** (p.137).
- **What it must contain.** One or more **Data Masking transformations**, and all transformations must be **passive** (they must not add or remove rows) (p.137, p.150). It can have multiple input and output columns (p.137).
- **Required ports.** Mark at least one input and one output column as mandatory; mandatory columns need source and target assignments (p.137, p.141, p.155).
- **Custom masking parameters** (p.156): Column Name, Column Type (input or output), Data Type, Precision, Mandatory, Group Name (input, output or another name you gave in the mapplet).
- **Exporting the other way.** You can also export rules as PowerCenter mapplets (p.150).
- **Assignment.** Select the rule in the Masking Rule column, then link source columns to rule input ports and rule output ports to source columns (p.142 to 143). Sybase to Hadoop limit: 20 columns.
- **Hadoop engine limits.** A Blaze engine allows Expression, Data Masking, Case Converter, Comparison, Decision, Labeler, Merge, Parser, Weighted Average, Standardizer and Java Passive transformations in a mapplet rule; Spark allows Expression, Data Masking and Java Passive; Hadoop mapplets cannot use sequence generator, lookup or classifier (p.62, p.137).
- **Changes.** If you change the name or type of a field of a custom rule, existing assignments do not see it; import the mapplet again (p.144).

- **Data Masking transformation session properties** (chapter 3, p.52 to 53): SSN High Group File path, DMO Cache Directory, Cache Size (default 8 MB), Shared Storage Table (default disabled) and Storage Commit Interval (default 100,000).

The architecture behind it, with PowerCenter running the workflows that TDM generates, is in the Informatica overview and architecture chapter. Video: "How to create mapplet in informatica" by LearningMart.

5.10 Putting it together: a masking design for ShopSphere

Column	Domain (policy pack name if one exists)	Technique and key parameters	Reason
first_name	First Name Domain	Substitution, dictionary FIRSTNAMES, Repeatable Output, seed 1	Realistic names; same customer gets the same name in every run
last_name	Last Name Domain	Shuffle, Representative, seed 1	Real surnames, detached from people; same surname always maps the same way
email	Email Address Domain	Email (advanced), columns from mapping, delimiter ".", constant domain example.com	Address matches the masked name, no real domain leaks
phone	Phone Domain	Phone masking	Keeps the number layout the app expects
ssn	Social Security Number Domain	SSN masking, Repeatable Output, seed 7	Valid-looking, avoids issued numbers, stable across runs
date_of_birth	Birth Date Domain	Random, blurring unit year, low 2, high 2	Keeps age band, hides the real date
zip_code	ZIP Domain	Random, mask format <code>+++ DDD</code>	Keeps the postal area, randomises the rest

Column	Domain (policy pack name if one exists)	Technique and key parameters	Reason
tax_id	(custom domain, found by column name)	Key, mask format AAAAADDDDA , seed 5	Keeps the letter-digit- letter shape, deterministic
customer_id (PK) and orders.customer _id (FK)	none	Left unmasked here. If you chose to mask them, rely on the auto cascade, or key masking with the same seed in both	A surrogate id is not personal data by itself; this is a project decision

CUSTOMERS ROWS AFTER MASKING (EXAMPLE OUTPUT)

customer_id	first_name	last_name	email	phone	ssn	date_of_birth	zip_code
1001	Divya	Nair	divya .nair @exa mple. com	98760 12345	612 -73 -4 41 0	1991-03-30	560417
1002	Arjun	Menon	arjun .men on@e xam ple.c om	91238 7046 5	47 8- 09 -2 231	1986-01-19	600358
1003	Latha	Kumar	latha .kum ar@e xam ple.c om	99881 20473	55 9- 21- 70 68	1998-09-05	682940

Compared with the originals, emails match the masked names, zip codes keep three digits, dates stay within two years, and the surnames are the same three moved around.

5.10.1 Running the masking operation: the guide's steps

The guide's list for performing a masking operation (User Guide 10.5.10, p.148 to 149) , with reasons:

- 1 Create a data masking rule.**
The rule is the recipe (14.3). Preview it in the Rule Simulator.
- 2 Create a project and import metadata.**
The project knows the tables and columns. Use separate projects for XSD and relational sources.
- 3 Optional: create parameters or import global parameters.**
For example a seed variable in a parameter file, so you can change the seed without editing the rule (p.155).
- 4 Add the rule to the project.**
Projects only see rules you add (14.4).
- 5 Assign the rule to a target column (or to an XML element).**
This is where you say "mask ssn with SSN_Rule". Save each row.
- 6 Create a plan, add the rule, and configure source and target connections.**
A plan holds at least one rule or policy. It states whether data is masked in place in the source database or in stream into a target (p.150). Add a storage connection if any rule is repeatable substitution or expression masking (p.286).
- 7 Generate and run the workflow.**
PowerCenter creates mappings, sessions and a workflow and runs them (p.150). Use Actions > Generate and Execute, then pick parameter values (p.310 to 311).
- 8 Monitor the workflow.**
Check Workflow Executions for Success, then open the logs if not (p.311 to 312).

Expected result: Target columns hold masked values; source untouched (when masking in stream into a target).

Plans and workflow settings are in the Informatica plans, workflows and warehouse chapter. If you also subset, masking runs on the subset result (p.275), as in the Informatica data subset chapter.

5.11 Hands-on labs

Lab 1: profile ShopSphere and apply masking rules by domain

Goal: find the sensitive columns of `CUSTOMERS` with a data domain profile (see the Informatica discovery and profiling chapter), let the domains suggest rules, and mask them.

1

Create a project and import the ShopSphere source metadata.

The project is where discovery results and rule assignments live (p.149).

Expected result: The four tables are listed under the project.

2

Run a data domain discovery profile.

Domains with data and column-name patterns (SSN, phone, email, zip) tag columns automatically (p.73 to 75).

Expected result: Columns such as `ssn` , `phone` , `email` show a Domain value in Define | Data Masking; where it is blank, you assign a policy or domain by hand.

3

Mark sensitive columns and add the PII policy pack (or your own policy) to the project.

The policy decides which preferred rules appear in the rule list (p.142, p.145).

4

Preview two rules in the Rule Simulator.

Open an SSN rule and a phone rule, Source Details > My Test Data, type the three sample rows, click Go (p.140).

Expected result: Original and masked values appear side by side.

5

In Define | Data Masking, select the sensitive columns and click Rule Assignment > Default Assignments, then Save.

Default assignments give each column its domain's default rule (p.145, p.147).

Expected result: The Masking Rule column fills in; rule names from domains show an asterisk.

6

Create a plan with the rules, generate and execute it.

Pick a target database that is not production (p.148 to 149).

Expected result: Status Success.

7

Verify.

Query the target and compare with the source (see the SQL after Lab 3).

► **Show a worked answer**

Lab 2: mask FirstName with substitution

Goal: replace every `first_name` with a value from a dictionary, the same way every run.

1

Create the dictionary file `FIRSTNAMES.csv` with a header row and 30 or more names, for example `SNO,FIRSTNAME` then `1,Divya`, `2,Arjun` and so on (the guide's sample dictionary also has `COUNTRY` and `GENDER` columns, p.178).

Substitution needs a list to pick from. For a unique substitution the list must have at least as many distinct names as the column has distinct values.

2

Add it in Administrator | Dictionaries: Actions > New Dictionary, type Flat File, name it, Choose File, pick the encoding, Save.

(p.178).

Expected result: The dictionary appears in the Dictionaries view.

3

Create the rule: Policies, Actions > New > Masking Rule, string datatype, Standard, Override Allowed, Next.

(p.139, p.179).

4

Choose Substitution. Set Repeatable Output and Seed 1. Fill Dictionary Information: Dictionary, Masked Value `FIRSTNAME`, Serial Number Column `SNO` and the lookup column if your version requires one.

The guide's example used a lookup column of GENDER; this dictionary has no gender column, so check which lookup columns your version requires (p.177, p.179).

5

Leave the exception handling at its defaults. Click Finish.

The defaults ignore nulls and continue on error (shown on p.154).

6

Open the rule, Rule Simulator, Connection source, pick `CUSTOMERS.first_name` , 20 rows, Go.

(p.140).

Expected result: Each original name beside a dictionary name.

7

Add the rule to the project, assign it to `CUSTOMERS.first_name` , Save.

(p.141 to 142).

8

Create a plan with this rule, set the storage connection (required for repeatable substitution), generate and execute.

(p.176, p.286).

Expected result: Success. Rows now show names from your dictionary.

9

Run the same plan again.

Because Repeatable Output is on with the same seed, row 1001 should get the same name as in the first run.

► **Show a worked answer**

Lab 3: mask alphanumeric text, SSN and telephone number

Goal: mask `tax_id` (letters and digits), `ssn` and `phone` so each keeps the shape the application expects.

1

Create a key masking rule for `tax_id`.

Policies, Actions > New > Masking Rule, string, Standard, Key. Seed 5, Mask Format `AAAAADDDDA` (five letters, four digits, one letter). Leave Source String Characters blank so every character is masked. Finish (p.164 to 165).

Expected result: The rule is listed. In the simulator ABCPM1234K becomes five letters, four digits, one letter.

2

Create an SSN rule.

String, Standard, SSN, tick Repeatable Output, Seed 7. Finish (p.175).

3

Create a phone rule.

String, Standard, Phone. Finish (p.169).

4

Preview all three in the Rule Simulator with My Test Data.

Type the sample values: ABCPM1234K , 123-45-6789 , 9876543210 (p.140).

5

Add the rules to the project and assign them: tax_id, ssn, phone. Save each row.

(p.141 to 142, p.145).

6

Create a plan, generate and execute.

(p.148 to 149).

Expected result: Status Success.

7

Check the results in the target with SQL.

```
-- General SQL; adjust syntax to your database.
-- 1. SSN keeps its pattern (expect same count as total rows):
SELECT COUNT(*) FROM customers WHERE ssn LIKE '___-__-____';
-- 2. Phone keeps its length (expect all rows; null phones excluded):
SELECT COUNT(*) FROM customers WHERE LENGTH(phone) = 10;
-- 3. tax_id keeps length 10 (expect all rows):
SELECT COUNT(*) FROM customers WHERE LENGTH(tax_id) = 10;
-- 4. Nothing unchanged (expect 0), joining target to a copy of source on
id:
SELECT COUNT(*) FROM customers t JOIN src_customers s
  ON t.customer_id = s.customer_id
 WHERE t.ssn = s.ssn OR t.phone = s.phone OR t.tax_id = s.tax_id;
```

► **Show a worked answer**

5.12 Common mistakes and gotchas

WATCH OUT FOR THESE

- **Forgetting to click Save** after each assignment; TDM discards unsaved assignments (User Guide 10.5.10, p.145).
- **Not rerunning plans.** Edits to rules, assignments or domains do not change existing plans until you regenerate and run them (p.144 to 148).
- **Different seeds on columns that must match.** The same seed gives the same output across columns (p.154, p.167). Different seeds give different values, and joins break.
- **Repeatable substitution or expression masking with no storage table.** Expression masking fails; substitution cannot remember pairs (p.163, p.176, p.286).
- **Unique substitution with a short dictionary.** The operation fails (p.176).
- **Short mask formats.** Trailing characters stay unmasked (p.164).
- **Masking the primary key but not the foreign key.** Use auto cascade, value cascade or the same seed (p.90 to 91, p.167).
- **Shuffle on XSD or Spark, expression repeatable on Hadoop, flat-file dictionaries on Hadoop.** Documented as unsupported (p.62, p.135, p.163, p.176).
- **Expecting reversibility from the wrong technique.** Only format preserving encryption is described as decryptable (p.161).

5.13 Recap

KEY POINTS

- A rule is a technique plus parameters. Rules attach to columns directly or through data domains, which a policy groups for reuse. Always preview a standard rule in the Rule Simulator.
- Three rule kinds: standard (one input, one output), advanced (input, variable, output columns with expressions, conditions and dependent masking) and mapplet (imported PowerCenter logic made of passive transformations including Data Masking transformations).
- Repeatable output and a seed make masking deterministic. Substitution and expression masking also need a storage table.
- Substitution gives realistic values from a dictionary; shuffle moves real values between rows; key masking keeps character types deterministically; random masking uses ranges and blurring; encryption can be reversible; nullification empties the column; SSN, SIN, phone, credit card, email, IP and URL are built-in formats.
- Choose by what the application needs: realism, format validity, joinability across tables, reversibility, or nothing at all.
- A mapplet is built and validated in PowerCenter (the video's example joins first and last name), exported to XML, then imported and assigned in TDM. The guide covers the TDM side only.
- Masking happens after subsetting when both are in one plan.

Informatica: generating test data

Data generation builds test data from nothing but rules, so no production row is read, copied or at risk. This differs from discovery, subsetting and masking, which all start from real production data. The general idea is in the generation concepts chapter.

The plan, workflow and monitoring side is shared with masking and subsetting and is covered in the plans, workflows and warehouse chapter.

5.14 Words to know first

Data generation

Creating new, realistic but fictional rows by following rules, instead of copying real rows. Also called synthetic data creation.

Generation rule

A saved recipe for one kind of value, for example "ten random digits starting with 6 to 9". It names a technique and sets its parameters.

Assignment

The link between a rule and a target column: "column `phone` uses rule `Phone_10_digits`".

Global rule

A named rule created once in the Policies area and reusable in every project. Standard, custom and advanced rules are global.

Ad hoc rule

An unnamed rule created inside one project directly on a column. It cannot be moved to another project.

Default rule

The rule TDM applies to a column you did not assign anything to, chosen per data type.

Entity

A set of tables linked by keys (a parent with its children) that TDM treats as one unit. You met entities in the Informatica subsetting chapter.

Dictionary

A file or table of ready-made values, such as first names, that a rule can pick from. It must be added to TDM by a user with the right permission before a rule can use it.

Exception test data

Deliberate nulls and deliberately invalid values mixed into the output, so that tests also see bad data.

5.15 What data generation is for

Picture ShopSphere's performance team. They want to test the "Customer Orders" app with 500,000 customers, ten times more than production holds. Subsetting can only shrink what exists. Masking keeps the same number of rows. Neither can *invent* rows. Generation can, and since it never touches production, there is nothing to leak.

The guide defines it plainly: use data generation to create realistic test data for test environments, using rules that define the logic (User Guide 10.5.10, p.190). You import the target table definitions (the *metadata*) into a project to say what shape of data you want, assign rules to columns, then run a plan.

Three ways to get test data, side by side (concept view; the Informatica terms are from the guide)

	Subset	Masking	Generation
Reads production rows?	Yes	Yes	No
Number of rows	Fewer than production	Same as input	Whatever you ask for
Real values at risk?	Yes, unless masked	Disguised	None exist
Realism comes from	The real data	The real data plus substitutes	Your rules (dictionaries, patterns, distributions)
Typical ShopSphere use	A small regression database	A realistic UAT copy	Load tests, new features with no production data yet

WHY IT MATTERS

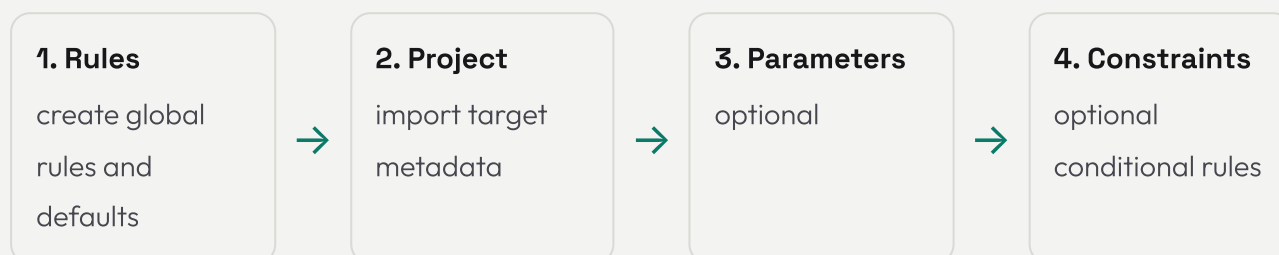
Generated data is the safest test data there is, and it is the only kind that can reach any volume you like, or hit awkward cases (nulls, invalid values, boundary dates) on purpose. Its weakness is realism: it only looks as real as your rules. A rule that makes every email "name@example.com" will not exercise your duplicate-email checks. Most of this section is about writing rules that are realistic enough.

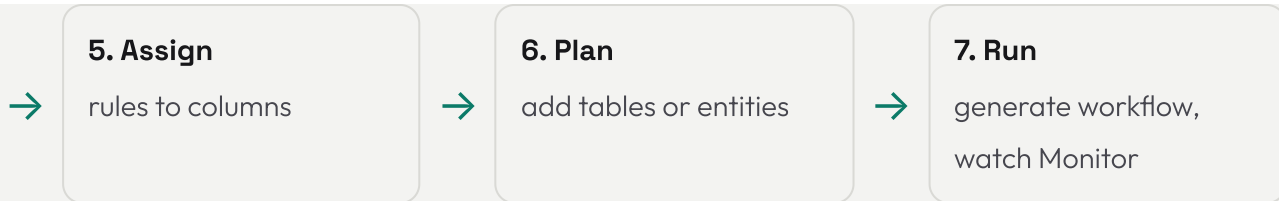
5.16 The building blocks and the order of work

The guide names five components you create or use for generation (User Guide 10.5.10, p.191):

Data generation components		
Component	What it is	ShopSphere example
Rule	A technique plus its parameters. May allow overriding.	Phone_10_digits
Assignment	The allocation of a rule to a column.	CUSTOMERS.phone gets Phone_10_digits
Table	The target table where rules are assigned.	CUSTOMERS
Entity	Tables related by physical or logical constraints.	CUSTOMERS with its children ORDERS and ORDER_ITEMS
Plan	Defines the generation operation: which tables or entities, how many records, which connections.	PLAN_GEN_CUSTOMERS_50

The order matters, because each step needs the one before it. The guide's task flow (User Guide 10.5.10, p.192):





Steps 1 to 5 are *design time*: you are only describing data, and no row exists yet. Step 6 packages the description, and step 7 is the first time anything is written. This mirrors the ETL and TDM chapter: the plan is the specification, and the workflow that TDM generates from it is the ETL job.

One detail from the overview is easy to miss: if a table name or column name contains special characters, the data generation workflow fails (User Guide 10.5.10, p.190). Check your target names before you import metadata.

WHERE YOU CLICK

In the Test Data Manager web client, global rules live under **Policies**. Inside a project, the tabs are Overview, Discover, Define, Execute, Monitor and Permissions (older screenshots say Security). Generation rules are assigned under **Define | Data Generation**, plans are built under **Execute** (User Guide 10.5.10, pp.50, 203).

5.17 Rule types: standard, custom, advanced and ad hoc

A rule can be created in four ways. The difference is mostly *how many columns* it touches and *where it lives* (User Guide 10.5.10, pp.192-193).

The four generation rule types

Type	Scope	What it does	Created in
Standard	Global, one output column	Uses one predefined technique. You can assign the same rule to many columns.	Policies
Custom	Global	Generates data from the logic of a PowerCenter mapplet.	Policies
Advanced	Global, several columns	Combines techniques and uses more than one input column.	Policies
Ad hoc	One project, no name	Created straight on a column. Can use extra techniques that global rules cannot.	Define Data Generation of a project

Global rules appear in every project and carry a name. Ad hoc rules have no name, cannot be moved, and are defined by the target columns you pick when you create them (p.193). The three techniques that exist *only* as ad hoc are **reference lookup**, **effective dates** and **expression**; **conditional** is also listed as ad hoc in the technique table (p.209).

5.17.1 Editing and deleting rules: the "impacted objects" check

If you edit or delete a global rule that is already assigned somewhere, TDM shows an **Impacted Objects** dialog listing affected columns, plans and data domains. You can export that list as a `.csv` file, then click Continue to save. **The change does not reach a plan until you generate and run that plan again** (User Guide 10.5.10, pp.195-196). The same dialog appears when you edit an ad hoc rule, assign a rule, or clear an assignment (pp.199, 204-206).

5.17.2 Creating a standard rule

1

Open Policies and choose Actions > New > Generation Rule.

Creating it in Policies, not inside a project, makes the rule global and reusable.

2

Enter a name, pick the column's data type, select the standard type and optionally tick Override Allowed.

The data type decides which techniques the wizard offers, so choose it first.

3

Click Next, fill in the parameters, set the exception test data options and click Finish.

The options choose whether to allow nulls or empty spaces and whether to produce invalid data (see exception test data).

Expected result: The rule appears in the Policies list and can be assigned in any project (p.195).

5.17.3 Override: one rule, small variations

If a rule has **Override Allowed** set, whoever assigns it can change its parameters for that one column without touching the global rule. The guide's example: a rule that makes random dates from 2011 to 2012 with override on; a user assigns it and adds Saturday and Sunday as excluded days, so that column gets weekday dates only (p.193). To do it, select the column in Define | Data Generation, choose Actions > Override, edit the values and Save. The Override column then reads **Yes-Overridden** (p.196).

5.17.4 Custom and advanced rules, in brief

A **custom** rule imports a PowerCenter mapplet. You must mark at least one input and one output column as required. The guide's use: combine first and last name into a full name, or use PowerCenter transformations such as an SQL or Stored Procedure transformation (User Guide 10.5.10, p.196). An **advanced** rule has input, variable and output ports and can mix techniques. Both get their own sections in the technique tour.

5.18 Default rules: what happens to columns you ignore

Columns without an assigned rule receive the **default rule for their data type** (User Guide 10.5.10, p.193). The defaults (Numeric, String, Date and Binary, plus separate primary key and unique key defaults for each type) are visible on a project's **Overview | Properties** tab (pp.193-194).

Keys get their own defaults because a primary key must never repeat or be empty. The administrator configures the initial defaults (p.193), and in the plan's Data Generation Properties you can adjust them again and tick **Apply Default Rule to** columns without rules (p.285).

Two more shortcuts the guide mentions (pp.194, 205):

- **Default data domain rule.** You set this when you create and run a data domain profile (the Informatica discovery and profiling chapter), so discovery can feed generation.

- **Auto rule.** After a column profile you review and finalise results. Columns whose **Inferred Rule** shows Yes can take the Auto Rule, which fills the column from the approved profile results. Do not apply an auto rule to a primary key column, because a primary key cannot contain nulls.

DEFAULT RULES ARE NOT A PLAN

The guide notes that default rules are not filled in if the rules are not part of the project. If the default boxes in the plan are blank, you must choose the default rules yourself (p.284). Random defaults are fine for a column nobody tests, not for one with business meaning such as email.

5.19 Assigning rules to columns

An assignment is made on one screen: **Define | Data Generation** inside the project. It is a grid with one row per target column and these fields (User Guide 10.5.10, p.203):

Fields of the Data Generation view

Field	Meaning
Table	Table name
Column	Column the data will be generated into
Data Type	Data type of that column
Is Null	Whether the column can hold nulls
Generation Rule	Click inside it to pick a rule. The list shows only rules valid for the column's data type. "New Generation Rule" creates an ad hoc rule.
Override	Yes if the rule allows override. Yes-Overridden once you changed it.

VISUAL REFERENCE

Informatica TDM 10.5.10 User Guide, p.203, figure "Define | Data Generation view in a project". The figure shows the project tab bar (Overview, Discover, Define, Execute, Monitor, Security), the sub-tabs Data Subset, Data Masking and Data Generation, and a grid of a customer table where two string columns already show a rule named StringRule and Override "No". What to look for: the grid has exactly the six columns in the table above, and unassigned rows have an empty Generation Rule cell.

Table	Column	Type	Null	Generation Rule	Override
CUSTOMERS	customer_id	int	no	Seq_1001	No
CUSTOMERS	first_name	varchar	no	Dict_FirstName	No
CUSTOMERS	last_name	varchar	no	Dict_LastName	No
CUSTOMERS	email	varchar	no	Adv_Email	No
CUSTOMERS	phone	varchar	yes	Phone_10_digits	Yes-
Overridden					
CUSTOMERS	ssn	varchar	yes	SSN_like_pattern	No
CUSTOMERS	date_of_birth	date	yes	Rand_Date_1950_2005	No
CUSTOMERS	zip_code	varchar	yes	Set_ZipCodes	No
CUSTOMERS	tax_id	varchar	yes	(default string rule)	

Schematic illustration, not an official screenshot.

SAVE AFTER EVERY ASSIGNMENT

In this view you must click the **Save** icon each time you assign a rule. If you pick a rule for the next column without saving, Test Data Manager discards the earlier assignment (User Guide 10.5.10, p.203).

5.19.1 How each kind of rule is assigned

- **Standard rule:** click inside Generation Rule for the column, choose the rule, click Save (p.204).
- **Custom (mapplet) rule:** choose it, and a Custom Rule Assignment dialog shows your table's columns on one side and the mapplet's input ports on the other. Select a column, select a port of the same data type, click the Link icon. Click Next, then link each rule output port to a target column and Save (p.204).
- **Advanced rule:** the same linking dialog, with a second step for outputs. If the variable column groups child records by a foreign key, you must also select that foreign key column (pp.204-205). You need not map an input port that you parameterised, nor a rule with no input ports.
- **Auto rule:** select the column, choose Auto Rule, Save. To apply it to many columns, select several rows and use Generation Rule Assignment, then Apply (p.205).
- **Remove an assignment:** select the column and choose Actions > Clear Assignment (p.205).

5.20 Tables, entities, relationships and keys

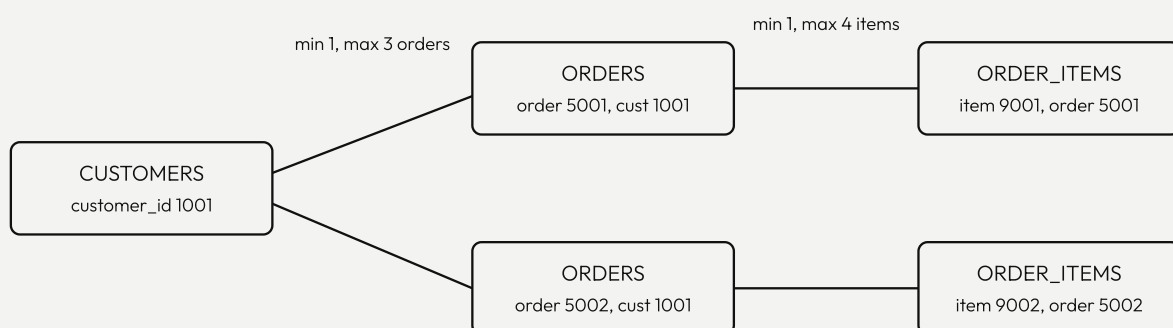
Real data is not a pile of independent tables. Every ShopSphere order must point to a customer that exists, and every order item to an order that exists. If you generate each table separately with

random numbers, the foreign keys will point nowhere and the app will fail for reasons unrelated to what you wanted to test. TDM gives you two tools for this.

5.20.1 Entities: generate a parent with its children

An **entity** defines tables related by physical or logical constraints, and keeps the hierarchy (User Guide 10.5.10, p.277). You can add entities (and plain tables) to a generation plan. You create the entity in the project's Define area, pick a driving table, and TDM finds related tables from the constraints. If your database has no declared foreign keys, add a **logical foreign key** on the Discover | Tables | Constraints tab, mapping the child column to the parent column (the Informatica subsetting chapter; User Guide 10.5.10, pp.110, 116).

For each relationship in an entity map there are **Minimum** and **Maximum** properties: the fewest and most child records to generate for each parent record (p.117). The plan-level Data Generation Properties repeat this as Minimum and Maximum (guide default 1 for each), plus **Parent Records Without Children** and **Child Records Without Parent** percentages for self-cyclic tables or reference lookups (p.285). A link between XML elements works the same way. A range set on an individual relationship takes priority over the plan-level ratio (p.207).



The fan-out an entity describes. Each generated parent receives a random number of children between the minimum and maximum, and every child carries its parent's key.

5.20.2 Reference lookup: borrow values from a table that already exists

The second tool is the **reference lookup** technique (see Reference Lookup). Instead of making values, it reads them from an existing parent table, so a child column can only receive keys that exist there. The plan then needs a **Reference Lookup Connection**, a connection to the database that holds the parent tables (User Guide 10.5.10, p.280). Use it for static reference data: ShopSphere's `PRODUCTS` table is the natural example for `ORDER_ITEMS.product_code`.

NOT DOCUMENTED

The guide says entities keep parent-child hierarchy and lets you set child counts per parent. It does not spell out exactly how TDM copies the parent key into the child's foreign key column, or whether a reference lookup may read a parent table *generated in the same plan* rather than one already loaded in the database. After a first run, check keys with a quick join query (the lab does).

5.20.3 Primary keys and unique values

- A computed column as primary key blocks generation for an entity: you cannot run generation that includes an entity whose primary key column is a computed column (p.191).
- For multirecord IMS sources imported from PowerCenter, the table sequence is not imported, so you must supply it with the **Manage Sequence** button on the entity's Entity Map tab (pp.191-192). This is a mainframe case.
- Many techniques have a **Generate Unique Values** checkbox. It cannot be combined with regular-expression patterns, and cannot be used for binary or flat-file dictionary rules (pp.216, 222-223).
- For ids, the cleanest choice is the Sequence technique. A sequence never repeats within a run, and "Start Sequence from Last Value" lets a second run continue where the first stopped (p.231).

5.21 How many records? And exception data

The number of rows is not part of a rule. It is set in the **plan**: the field **Number of Records to Generate** is marked Required for generation plans (User Guide 10.5.10, p.280). In the plan wizard it is step 14: "enter the number of records that you want to generate" (p.305). You can also give criteria per entity on the generation criteria step (p.305).

For a parent-child set, the Minimum and Maximum child settings decide how many children each parent gets, so total rows in the child tables are not a round number. The guide gives no worked example of totals; with 50 customers and 1 to 3 orders each, expect between 50 and 150 orders. Confirm with the Row Count report (the plans, workflows and warehouse chapter).

5.21.1 Exception test data: nulls and invalid values

Almost every technique ends with the same two settings, **Null Values** and **Invalid Values** (User Guide 10.5.10, pp.210-211). Each takes a rule (an existing global rule, or "Default") and a percentage, which may have up to two decimal places.

- **Null Values:** the percentage of rows that will contain a null in this column. Cannot be generated for MongoDB sources (p.210).
- **Invalid Values:** the percentage of rows that deliberately break the rule, where "invalid" means whatever does not conform to the rule: outside the numeric range, wrong length, not in the dictionary and so on.

The guide's trick: make a rule "Recent_Dates" for 2012 to 2013 and, in its Invalid Values setting, choose a second saved rule "Dates_2005" (2005 to 2010) at 15 percent. Now 85 percent of dates are recent and 15 percent are old (p.210). So "invalid values" is really a way to mix in a second population.

WHY GENERATE BAD DATA?

Good testing needs negative cases. If ShopSphere's checkout rejects a customer with no phone number, you need rows with no phone number. Ten percent nulls in `phone` and two percent invalid `zip_code` values give your tests something to reject, without hand-crafting rows.

5.22 The generation techniques, one by one

A technique is the method a rule uses to produce values. The guide lists eleven (User Guide 10.5.10, p.209). Which ones you may choose depends on the column's data type, and some are **ad hoc only**, meaning you can use them only when you create a rule directly on a column inside a project (pp.208-209).

All eleven generation techniques (User Guide 10.5.10, pp.209-235)

Technique	Kind	Data types	One-line purpose
Sequence	Standard	Numeric, date	Values in order with a start and an increment
Random	Standard	String, numeric, date	Random strings, numbers or dates, optionally from a pattern
Set of Values	Standard	String, numeric, date	A short list of values with a percentage for each
Dictionary	Standard	String, numeric, date, binary	Values picked from an imported dictionary
Credit Card	Standard	String only	Card numbers by network, passing the Luhn check
Reference Lookup	Ad hoc	String, numeric, date	Values taken from an existing parent table
Expression	Ad hoc	String, numeric, date	Values computed from other columns
Conditional	Ad hoc	String, numeric, date	Different rule depending on another column's value
Effective Dates	Ad hoc	Date	Consecutive start and end dates in two columns
Advanced	Combination	String, numeric, date	Several techniques and several columns in one rule
Custom	Mapplet	String, numeric, date	Logic imported from a PowerCenter mapplet

5.22.1 Sequence

A global or ad hoc rule that makes numeric or date values in a sequence. You pick a beginning value and an increment (User Guide 10.5.10, p.230).

When: ids and anything that must be ascending: customer ids, ticket numbers, consecutive dates. The guide's example is a TICKET_NUMBER column with start 1000 and increment 1 (p.231).

Parameters (pp.231-232): numeric rules take Start Value, Increment Value and **Start Sequence from Last Value**, which makes the next run continue from the previous output (on a string column it restarts from the beginning). Date rules take Start Date and Increment by (a number plus year, month, day, hour, minutes or seconds). Null Values and Invalid Values apply, where invalid means below the start for a positive increment.

ShopSphere: `customer_id` with start 1001, increment 1 gives 1001, 1002, 1003 and so on. A date sequence for `order_date` with start 2026-01-01 and increment 1 day gives one order per consecutive day.

Limits: you cannot add a conditional constraint to a sequence column (p.201) or put conditions on a sequence rule (p.234).

5.22.2 Random

A global or ad hoc rule that creates random strings, numbers and dates, for things like salary and birth dates (p.220). For strings you give a minimum and maximum length; for numbers a minimum and maximum value; for dates a range. You can instead supply a **regular expression** (a compact pattern language for text shapes) and say what percentage of rows follow it. If you use several patterns the percentages must total 100.

VISUAL REFERENCE

Informatica TDM 10.5.10 User Guide, p.210, figure "random string generation rule". The figure shows a wizard step titled "New Generation Rule: Step 2 of 2" with a Random option (Minimum Length, Maximum Length), a From Regular Expression option with a Pattern box and a Distribution % box, a Generate Unique Values checkbox, and an Exception Test Data area with Null Values and Invalid Values drop-downs set to "Default" beside percentage boxes showing 0. What to look for: the two radio choices are mutually exclusive ways of making a string.

Parameters (pp.221-223): strings take Minimum Length (default 1) and Maximum Length (default the column maximum); numbers take Minimum Value (default 0) and Maximum Value (default the column maximum); dates take Start and End Date/Time and Exclude (up to two different weekdays). Strings and numbers can instead use **From Regular Expression** with a Pattern and Distribution. **Generate Unique Values** is not available with a pattern, and neither are invalid values. Null Values and Invalid Values apply as usual.

Writing patterns

For strings the usable operators are `.`, `\d`, `\w`, alternation `(a|b|c)`, `{n}` repeat counts and `[]` sets. To make numbers that contain special characters, use the *string* technique. To make plain numbers use the *numeric* technique, which allows `\d`, alternates like `(1|2|3)` and `[0-9]`, and alternates cannot be nested (p.223).

SHOPSPHERE PATTERN EXAMPLES (EXAMPLE PATTERNS USING THE OPERATORS LISTED ON P.223)

Goal	Technique	Pattern	Example output
ShopSphere mobile, 10 digits starting 6 to 9	Numeric	<code>(6 7 8 9)\d{9}</code>	9123456780
ShopSphere SSN-like	String	<code>\d{3}-\d{2}-\d{4}</code>	234-56-7890
ShopSphere tax id	String	<code>[A-Z]{3}\d{6}</code>	KQM481926
ShopSphere test email	String	<code>[a-z]{8}@example\.(com org)</code>	qkdmpzra@example.org

Read a pattern left to right as slots: `\d` is one digit, `{n}` repeats the previous piece, `[A-Z]` is one character from a range, `(a|b)` is a choice and `\.` is a literal dot. So `(6|7|8|9)\d{9}` is a first digit 6 to 9 followed by nine digits. Test a pattern by generating a few rows into a scratch table first.

PATTERNS CANNOT MAKE VALID CHECKSUMMED NUMBERS

For credit card, Social Security and Social Insurance numbers the guide warns that pattern-made values might not be valid, because real numbers follow algorithms that a pattern cannot reproduce (p.224). That is fine for "SSN-like" test data, and it is why the Credit Card technique exists.

Date example: a random date rule from 1950-01-01 to 2005-12-31 for `date_of_birth` gives values such as 1990-04-12 and 1999-07-23. Set Exclude to Saturday and Sunday and no value falls on a weekend; "invalid" for dates means outside the range or on an excluded day (p.221).

5.22.3 Set of Values

You type the values you want and the percentage of rows for each; the percentages must total 100 (decimals allowed). Works for string, numeric and date, and can be global or ad hoc (pp.232-233). Best for columns with a handful of values, such as membership status.

Parameters: **Value** (one row per value, add rows as needed), **Distribution%**, and the usual Null Values and Invalid Values. For this technique "invalid" means anything not in the list (p.233).

The guide's example is a PAYMENT_TYPE column with Visa 60, MasterCard 30, Paypal 10.

ShopSphere: zip_code with 560001 at 40, 600017 at 35 and 682001 at 25. Of 50 customers, about 20, 17 or 18, and 12 or 13 will get each zip code. Percentages are targets, so expect small rounding differences.

5.22.4 Dictionary

Loads values from a **dictionary** (a flat file or relational table of values) into a column. Good for columns with realistic but finite values: names, states, countries. Before you create the rule, someone with permission must add the dictionary under **Administrator | Dictionaries**. Works for string, numeric, date and binary data; flat file or relational source; binary only from relational (p.215).

Parameters (p.216): Dictionary Name; Dictionary Column (longer values are truncated to the target size); an optional Dictionary Lookup Column plus Table Column to match rows; Serial Number Columns (relational dictionaries, needed for binary); Generate Unique Values (not for flat-file dictionaries or binary); Null and Invalid Values (invalid means not in the dictionary).

If both lookup and table columns are of double data type the process generates nulls (p.216). The guide's example makes two string rules from one names dictionary: one picks the first-name column, one the last-name column (p.215).

ShopSphere: first_name uses dictionary NAMES column FIRST, last_name uses column LAST. Sample output: Meera, Nair; Ravi, Kumar; Asha, Menon. Dictionary and column names here are placeholders. Because the two rules pick independently, "Ravi Menon" is as likely as "Ravi Kumar", which is fine for testing and wrong for anything that needs a believable family name.

5.22.5 Credit Card

Creates card numbers by issuing network. Output is a **string** (not numeric, date or binary), and numbers must pass verification using Luhn's algorithm, the check-digit formula real cards follow (pp.226-227). Formatted output looks like XXXX-XXXX-XXXX-XXXX .

Parameters (pp.227-228): Issuing Network (required: American Express, Discover, JCB, Mastercard or Visa); an optional Issuer Identification Number of up to six starting digits (a value invalid for the network makes the workflow fail; blank lets TDM use valid ones); Distribution % (must total 100); Generate Formatted Values; Generate Unique Values; Null and Invalid Values. Valid starting digits include Visa 4, Mastercard 51 to 55 and American Express 34 and 37.

ShopSphere: the four tables hold no card numbers, so imagine a later `PAYMENTS.card_no` column. A rule with Visa at 60 percent and Mastercard at 40 percent, formatted, gives values shaped like `4666-1760-3137-2159` and `5122-3307-9244-0268`. Those two pass the Luhn check; TDM would produce its own digits.

5.22.6 Reference Lookup

An **ad hoc** rule that takes column values from an existing table, for string, numeric and date columns. You name the parent table and map parent columns to child columns. The same rule can fill several child columns from the same parent table (p.229). The guide's example: use the same customer ids that exist in another table.

Parameters (p.230): Minimum (default 1) and Maximum (default 3) child records per parent; Data Source (connection to the parent table); Parent Table; pairs of Child Column and Parent Column; a Parent Table Filter; the percent of parent records without children and of child records without a parent (if that is 0 you must give a Default Value to write instead of nulls).

ShopSphere: `ORDERS.customer_id` reads `CUSTOMERS.customer_id` with minimum 1 and maximum 3, so each existing customer gets one to three orders. Set "Parent records without children" to 10 and about one customer in ten has no orders, a useful edge case. Remember the plan needs a *Reference Lookup Connection* (p.280), and see the note on reference lookup about where the parent rows must already live.

REFERENCE LOOKUP IS NOT FOR XML TARGETS

You cannot use reference lookup or effective dates rules for an XSD (XML) source (p.206).

5.22.7 Expression

An **ad hoc** rule that computes a value from an expression you build in the Expression Builder, choosing columns and functions from lists. Works for string, numeric and date (p.219). Parameters: Expression, Null Values, Invalid Values (invalid = does not follow the expression) (p.220). If you tick "Override Data Type to Numeric" on a string column, you cannot enter a numeric expression (p.219).

The guide's example builds a Name column from FirstName and LastName with `CONCAT` (`FirstName+' '+LastName`). **ShopSphere:** imagine a `display_name` column built as `CONCAT(CONCAT(first_name, ' '), last_name)`, giving "Asha Menon". Check the exact function list in your Expression Builder.

When: any column that is a function of other columns in the same row. The dependent column must still be assignable after its inputs, so assign rules to the inputs too.

5.22.8 Conditional

An **ad hoc** rule with one or more rows of "*if this condition, then use this generation rule*". Conditions are applied in the order listed (reorder with the up and down arrows). Works for string, numeric and date, but you cannot apply conditions to effective dates or sequence rules (p.234).

The guide's example: DEPT_ID 11 gets a 5 percent hike, 22 gets 10, 33 gets 15. The column that the condition reads (DEPT_ID) must itself have a rule, and TDM generates it first, then the dependent column (p.234).

ShopSphere (hypothetical city column): if `zip_code = '560001'` then city is "Bengaluru", if `'600017'` then "Chennai", if `'682001'` then "Kochi". Each "then" is a one-value Set of Values rule. The guide does not describe a catch-all "else" row, so make sure every possible input value has a condition.

5.22.9 Effective Dates

An **ad hoc** date rule that fills two related columns, an effective start date and an effective end date, so that periods follow one another without overlap. It makes a random start in a range, an end after it, and the next start right after that end. A **grouping column** decides how many dates are in each sequence: the number of rows sharing the same grouping value (pp.217-218).

You create the rule on either the start or the end column and name the other. Parameters: Start Date/Time, End Date/Time, Exclude (up to two different weekdays), Effective Start Date column, Effective End Date column, Grouping Column, Invalid Values (percent of rows breaking the sequence; the rule is fixed to Default) (pp.218-219).

ShopSphere: a hypothetical `CUSTOMER_ADDRESS_HISTORY` table with grouping column `customer_id` gives each customer a chain of addresses with no gaps. Good for testing "address on the order date" logic.

5.22.10 Advanced

A combination of techniques that fills one or several target columns from one or more input, variable and output ports (pp.199, 212). Use it when one column needs two techniques, or the value depends on other columns. You can generate string, numeric and date data.

An **input port** (optional) takes a target column or a parameter. A **variable port** holds intermediate values and uses either *Conditional* or *Generation Rule Set* (rules with distribution percentages, optionally grouping records by foreign key). An **output port** receives the data and needs an Expression built from the inputs and variables. Port names need not match column names; you map them when you assign the rule (pp.199, 214-215).

VISUAL REFERENCE

Informatica TDM 10.5.10 User Guide, p.200, figure "advanced rule parameters". The figure shows "New Generation Rule: Step 2 of 2" with a Columns pane listing one input column (lp_1, string, 10), a variable column (Var_1) with Configure Rule and two default numeric sequence rules, and one output column (Out_1), plus a Properties pane showing the output expression CONCAT(lp_1,Var_1). What to look for: the three port groups stacked in one list.

The guide also shows 50 "Invoice-n" plus 50 "Order-n" ids built from a conditional sequence (pp.199–200).

SHOPSPHERE: REALISTIC EMAILS (ADAPTED FROM THE GUIDE'S EXAMPLE)

Input `in_FirstName` mapped to `first_name`. Variable `var_Domain` with a Generation Rule Set: `example.com` 50 percent, `example.org` 50 percent. Output `out_Email` with the expression above. Result: `Asha@example.com`, `Ravi@example.org`.

Two customers called Asha share the same name before the @. The guide does not say whether an input port reads the value *generated* for another column in the same run, so check that `first_name` is generated before the email.

5.22.11 Custom

A global rule built from a PowerCenter **mapplet**. You import it, and mark at least one input and one output column as required. Works for string, numeric and date (pp.196, 211). Use it when you need PowerCenter transformations (SQL, Stored Procedure) or logic that the built-in techniques cannot express.

You map your table columns to the mapplet's input ports and its output ports to target columns (p.204).

ShopSphere: a mapplet that takes first and last name and returns "Asha Menon" for a `display_name` column. A worked mapplet build (this same full-name idea) is in the mapplet walkthrough in the Informatica masking chapter.

5.23 Conditional constraints: rules between columns

Techniques fill one column at a time. A **conditional constraint** is a table-level rule that restricts what may appear in a column based on another column. It becomes part of the target table's metadata (User Guide 10.5.10, p.201).

The guide's example: a flight table with `From_City` and `To_City` both filled by the same dictionary rule, plus the constraint `To_City  $\neq$  From_City` so no ticket flies from a city to itself (p.202). A ShopSphere analogue with hypothetical columns: `delivery_date > order_date` .

To create one, open Discover > Tables, click the table, open the Constraints tab, choose Create New Constraint, set the type to Conditional and build the expression in the Expression Builder (pp.110-111).

Where the constraint cannot be used: numeric sequence, date sequence and random data patterns (p.201). On XSD sources you cannot constrain an element that allows more than one child per parent (p.201).

WHEN THE RULE AND THE CONSTRAINT DISAGREE, THE RULE WINS

The guide's example: both `Created_Date` and `Modified_Date` use a random date rule ending 31 July 2014, with the constraint `Modified_Date > Created_Date` . When `Created_Date` is 31 July 2014 no later date is allowed, so TDM writes a date within the rule's range anyway and **generates warning messages** (p.202). The constraint was broken, quietly. Read the warnings in the Monitor logs (the plans, workflows and warehouse chapter) and loosen the rule's range, not just the constraint.

5.24 Choosing a technique for every ShopSphere column

One reasonable rule set for the four ShopSphere tables (example rule set)

Column	Technique	Parameters	Why
first_name, last_name	Dictionary	one names dictionary, two columns	Looks like a name
ids, dates, amounts, quantities	Sequence or Random	keys from a sequence; ranges for amounts and dates	Valid, plausible values
CUSTOMERS.c ustomer_id	Sequence (numeric)	start 1001, increment 1	Primary key: never repeats
email	Advanced	first name + domain rule set	Depends on another column
phone	Random (numeric pattern)	(6 7 8 9)\d{9} , 10 percent null	Right shape, some nulls for negative tests
ssn	Random (string pattern)	\d{3}-\d{2}-\d{4}	SSN-like, not guaranteed valid
zip_code	Set of Values	three codes, 40/35/25	Small, known list
tax_id	Random (string pattern)	[A-Z]{3}\d{6}	Fixed shape
ORDERS.custo mer_id	Entity relationship (or Reference Lookup)	min 1, max 3 children	Must match a real customer
order_id	Entity relationship	min 1, max 4 children	Must match a real order
product_code	Reference Lookup or Set of Values	codes that exist in PRODUCTS	Must match a real product

5.25 From rules to a running job: the generation plan

Rules and assignments are only a description. A **plan** turns them into something that runs. The plan lists which entities and tables to generate, how many records, and which connections to use; then TDM generates a workflow from it and the workflow does the work (User Guide 10.5.10, p.207). The plan screens are shared with masking and subsetting, so the full tour is in the plans, workflows and warehouse chapter. Here is the generation-specific part.

5.25.1 Creating a generation plan

From the guide's wizard (User Guide 10.5.10, p.305), with the reason for each step:

1

In the project, click Execute, then New Plan, and name it.

Plans live in Execute, the project's "run things" tab.

2

Click Next twice to skip the masking and subset pages.

The wizard offers all three operation types on one path. You are skipping the two you must not use.

3

Click Add Generation Components and choose entities and tables (or the XSD file for XML output).

Only components with generation rule assignments make sense here.

Expected result: The chosen entities and tables appear in a list.

4

Review table names, set generation criteria for an entity if you want, click Next.

Criteria are where you can restrict which rules apply for a test.

5

Enter the number of records to generate and set connections and target properties.

The count is required (p.280).

6

Optionally edit data generation, error and recovery, and advanced settings. Click Next and Finish.

Defaults are usually fine for a first run.

Expected result: The plan appears in the project's Execute tab. It has not run yet.

You can also make a **quick ad hoc generation plan** for a single column: in Define | Data Generation select the column, choose Actions > Generate and Execute, name the plan, pick the Integration

Service, enter the number of records and answer Yes at the prompt (p.306). It is the fast way to try one rule before you commit to a full plan.

5.25.2 Connections and settings that matter for generation

The settings that matter for generation (pp.280-285): the target connection (a relational database, a flat file or the test data warehouse); **Number of Records to Generate** (required); a Reference Lookup Connection if any reference lookup rule is assigned; a Dictionary Lookup Connection if the plan has a dictionary rule or a mapplet rule with a transformation; the default Numeric, String, Date and Binary rules with Apply Default Rule to; Minimum and Maximum child records (default 1); and the percentages of parents without children and children without a parent. Target options such as Truncate Table (off by default) and Disable Constraints are covered in the plans, workflows and warehouse chapter.

If the target is a flat file you can use the test tool integration properties to copy the output to an integrated HP ALM server. If the target is the **test data warehouse**, you fill in data set properties (name, application, version, tags) and the generated data is stored as a versioned data set (pp.190, 281). That is how generated data gets reused, see the plans, workflows and warehouse chapter.

HEAP SPACE AND SPECIAL CHARACTERS

If a "Java heap space" exception occurs while setting an ad hoc rule, ask the administrator to increase the heap size (p.198). Remember also that special characters in table or column names make the generation workflow fail (p.190); the plan setting "Enable special characters in metadata" defaults to No (p.289).

5.26 Why a generation plan stands alone

The guide is firm: when you add data generation components to a plan, you **cannot** add data masking or data subset components, and the reverse holds too (User Guide 10.5.10, pp.274-276, 305). It repeats the rule in the plan overview, the task list, the component table and the creation steps. A Hadoop plan likewise cannot do subset or generation (p.306).

The guide states the restriction but does not explain it. Masking and subsetting both *read existing* rows from a source, then change or filter them. Generation has no source rows; it builds new rows from rules. A plan that tried to do both would have to decide what the masking rule should act on. Keeping them apart avoids that.

One plan: subset and masking

Source connection to target connection. The guide says you can combine them in one plan, or keep separate plans (p.304). If both are present, subset runs first and masking applies only to the subset rows (p.275).

Another plan: generation

Only entities, tables or XSD files with generation rule assignments. Number of records to generate. No source data is masked or subsetted in this plan.

What to do when you need both. Use two plans:

- **Masked copy of old customers plus brand-new ones:** run a masking plan into the test database, then a generation plan into the same schema. Make the generation sequences start above the existing ids, or continue with "Start Sequence from Last Value", so keys do not collide.
- **Generate once, then make small slices:** send the generation plan to the test data warehouse, then create a *related data set*, a subset of that data set built from an entity or group (pp.248-251). That is subsetting generated data without breaking the rule.

WHY IT MATTERS

This restriction shapes how a data pipeline is designed: generation is its own track, not a masking option.

Lab A: generate 50 ShopSphere customers with valid orders

Goal: end with 50 rows in `CUSTOMERS` plus linked `ORDERS` and `ORDER_ITEMS` in an empty test schema, then prove the keys are valid. The steps follow the guide's task flow (pp.192, 195-205, 305). You need an installed TDM, an empty test schema `SHOPSPHERE_TEST` with the four tables, and a `PRODUCTS` table already holding a few products (P-100, P-200, P-300).

1

Ask an administrator to add a names dictionary under Administrator | Dictionaries.

Dictionary rules refuse to be created until a dictionary exists (p.215). A file with a `FIRST` and a `LAST` column is enough.

Expected result: The dictionary appears in the list.

2

In Policies, create standard rules with Actions > New > Generation Rule.

Make the rules from the table in 15.11: Seq_Cust (Sequence from 1001), Dict_First , Dict_Last , Phone_10 ((6|7|8|9)\d{9} , 10 percent nulls), SSN_like , DOB_Range , Zip_Set (40/35/25) and TaxId_Pattern .

Pick the data type first, since it decides which techniques you see.

Expected result: The rules appear in the Policies list.

3

Create a project and import metadata for the four tables.

Click Projects, Actions > New, name it, open it, then Actions > Import Metadata and choose Datasource Connection to the test schema (pp.53, 66). Metadata tells TDM what columns exist; it copies no rows.

Expected result: The four tables appear under the project's data sources.

4

Create an entity for the parent-child tables.

In Define, create an entity with CUSTOMERS as driving table so TDM pulls in ORDERS and ORDER_ITEMS (pp.116, 277). If the foreign keys are not declared in the schema, add logical foreign keys first on Discover | Tables | Constraints (p.110).

Set the relationship minimum and maximum to 1 and 3 for CUSTOMERS to ORDERS, and 1 and 4 for ORDERS to ORDER_ITEMS (p.117).

Expected result: A diagram shows the three tables joined.

5

Add the rules to the project and assign them in Define | Data Generation.

Assign the rules to the matching CUSTOMERS columns. Click Save after every assignment (p.203). Also give the order and item tables sequences for ids, Random ranges for dates, amounts and quantities, and a Set of Values rule (P-100, P-200, P-300) for product_code .

Expected result: Every row has a rule name, and nothing is left to the default rule except columns you chose to ignore.

6

Add the advanced email rule.

In Policies create an advanced rule with input in_FirstName , variable var_Domain (Generation Rule Set: example.com 50, example.org 50) and output out_Email with the expression

`CONCAT(CONCAT(in_FirstName, '@'), var_Domain)` . Assign it to `email` and map the input port to `first_name` (pp.200-201, 204-205, 213).

Expected result: The email cell shows the advanced rule's name.

7

Create the plan.

Execute > New Plan, name it `PLAN_GEN_CUSTOMERS_50` , skip masking and subset, add the entity as a generation component, enter 50 records, choose the target connection `SHOPSPHERE_TEST` , and set the Reference Lookup Connection only if you used a lookup rule (p.305). Tick Truncate Table (p.282) if you may want to rerun.

Expected result: The plan appears in the Execute list.

8

Generate and execute the workflow, then watch it.

Open the plan's properties, Actions > Generate and Execute, choose the Integration Service and run immediately (pp.310-311). Open Monitor and wait for the Execute Workflow job to read Success (p.317).

Expected result: Job status moves from In Queue to Running to Success. If it shows Error, click the job ID and read the log (see the plans, workflows and warehouse chapter).

9

Check the data with SQL.

Run `SELECT COUNT(*) FROM customers;` (expect 50) and `SELECT COUNT(*) FROM orders;` (expect 50 to 150, see the record count section). Then prove the keys: `SELECT COUNT(*) FROM orders o LEFT JOIN customers c ON o.customer_id = c.customer_id WHERE c.customer_id IS NULL;` must return 0. Repeat for `order_items` against `orders` and `products` .

Expected result: Counts as described and zero orphan rows. A non-zero count means the foreign keys were not honoured; see the verify note on reference lookup.

► Show a worked answer

5.27 Common mistakes and gotchas

THINGS THAT GO WRONG

- **Forgetting to Save each assignment.** The earlier assignment is discarded (p.203).
- **Editing a rule and expecting a finished plan to change.** Regenerate and rerun the plan (pp.195-196).
- **Combining masking or subset components with generation components in one plan.** Not allowed (pp.274-275). Use two plans.
- **Using a pattern and "Generate Unique Values" together.** The guide says uniqueness is unavailable with a regular expression (pp.222-223). Use sequences for keys.
- **Re-running into a table that already has ids from 1001.** Key collisions. Truncate the target, or continue the sequence from its last value (p.231).
- **Believing pattern-made SSNs or card numbers are valid.** They might not be (p.224). Use the Credit Card technique when validity matters.
- **Ignoring warnings.** A rule that conflicts with a conditional constraint wins quietly and only logs a warning (p.202).

5.28 Recap

RECAP

- Generation builds new rows from rules and never reads production data. It is set up in a project, run from a plan, and executed as a generated workflow.
- Rules come as standard, custom, advanced (global) and ad hoc (per project). Unassigned columns get the default rule for their data type.
- The eleven techniques are Sequence, Random, Set of Values, Dictionary, Credit Card, Reference Lookup, Expression, Conditional, Effective Dates, Advanced and Custom. Reference Lookup, Expression, Effective Dates and Conditional are ad hoc.
- Null and invalid values are mixed in by percentage through the Exception Test Data settings.
- Entities with minimum and maximum child counts, and reference lookups, keep parent-child keys consistent. Always verify keys with a join.
- The number of records is a plan setting. A generation plan cannot contain masking or subset components.
- Conditional constraints restrict one column using another, but a conflicting rule wins and only produces a warning.

Informatica: plans, workflows, monitoring and the Test Data Warehouse

The discovery, subsetting, masking and generation work all ends in a **plan** that packages it. TDM generates a **workflow** from the plan, the **Monitor** shows its progress, **reports** prove what happened, and the **Test Data Warehouse** can store, version and restore the result.

In ETL terms (see the ETL from scratch chapter and the ETL and TDM chapter): **a TDM workflow is an ETL job that TDM writes for you.** You describe the data and the rules; TDM builds the mappings and runs them.

5.29 Words to know first

Plan

The saved definition of one operation: which components, which source and target connections, which settings. It does nothing until you generate and run a workflow from it.

Workflow

The runnable job generated from a plan. It contains PowerCenter mappings and sessions and is run by the PowerCenter Integration Service.

Parameter

A named placeholder (for a connection, an owner or a filter value) whose value you supply at run time, so one plan serves many runs.

Test Data Warehouse (TDW)

A central store for versions of test data, so teams can save a known-good state and restore it later.

Data set

A named collection of one or more tables stored in the warehouse. Each time you store it again, a new *version* is created.

Reset

Copying a data set version from the warehouse back to a target so a test environment returns to a known state.

Data pack

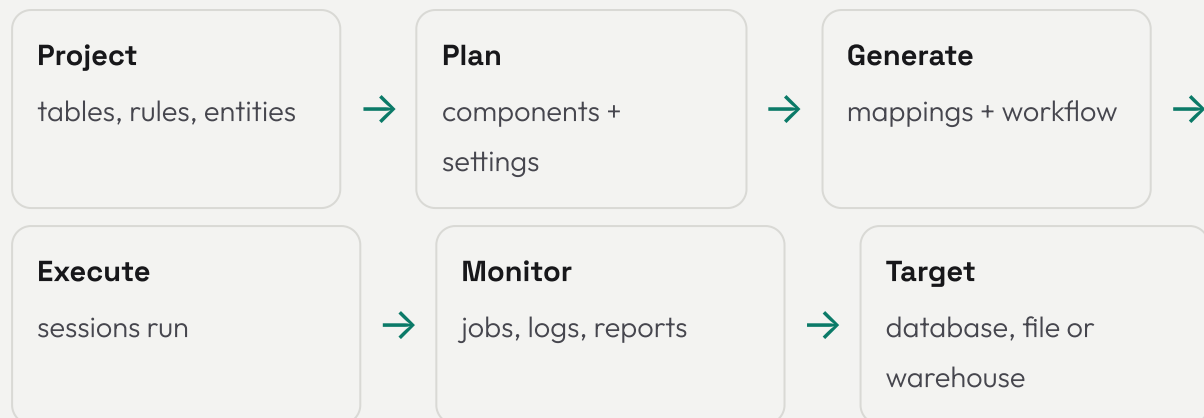
A data set published to the self-service portal for testers who do not use Test Data Manager.

Data coverage

A check of whether a data set has enough rows for each combination of values your test cases need.

5.30 Plans and workflows: the big picture

A plan defines a data subset, data masking or data generation operation, and includes the components you need to generate a workflow (User Guide 10.5.10, p.274). When you start a workflow, the PowerCenter Integration Service completes the operations in the plan (p.275).



ETL ideas and their TDM names. TDM terms and roles: pp.275, 308.

ETL idea (earlier chapters)	In Informatica TDM	Who creates it
Job specification	Plan	You, in Test Data Manager
Transformation logic	Mappings with masking, subset or generation logic	PowerCenter Repository Service generates them from the plan
Job run	Workflow and its sessions	PowerCenter Integration Service runs them
Source and target connections	Connections named in the plan	Administrator creates, you choose
Run-time inputs	Parameters and parameter files	You
Job scheduler and monitor	Run now or schedule; Monitor view	You
Staging area	Staging connection (temporary tables)	Chosen in plan settings

The analogy stops at one point. A hand-written ETL job is a thing you maintain. A TDM workflow is disposable: change the plan, generate again. The guide says so directly: when you update or delete a plan component you must generate and run the plan again to apply the change (p.275).

5.30.1 Plan types and what each can contain

Plan components by operation (User Guide 10.5.10, pp.274-277, 306)

Component	What it is	Used in
Rule	A masking rule, assigned to a policy or standalone	Masking plan
Policy	A set of data domains, each with rules	Masking plan
Entity	Related tables with a parent-child hierarchy and filter parameters	Subset or generation plan
Group	A set of tables to copy to a target subset database	Subset plan
Template	A bundle of subset components; see the Informatica subsetting chapter (named with entities and groups on p.293)	Subset plan
Table	Target table columns that have generation rule assignments	Generation plan
XSD file	An imported XML source definition	Subset, masking or generation plan

The practical plan types are:

- **Masking, subset, or masking plus subset in one plan.** If both kinds of component are present, subset has priority: TDM subsets first, then masks columns in the subset data that have assignments. It does *not* apply masking to source rows outside the subset, even if they are part of a cascade (p.275).

- **Generation plan.** Only entities, tables or XSD files. You cannot add masking or subset components to the same plan, and the reverse (pp.274, 276). See the Informatica generation chapter for the reasoning.
- **Hadoop plan.** Groups and masking components for Hadoop sources and targets. No subset or generation (p.306).
- **Data set (related data set) plan.** Created from a data set in the warehouse to cut a smaller data set from it (p.251).
- **Reset and restore plans** appear in the guide's notes about pre and post workflows (p.277); resets are covered in the warehouse section below.

5.31 The ShopSphere story

ShopSphere's QA team needs a safe, small, realistic copy of `SHOPSPHERE_PROD` for the next release of Customer Orders. They want to refresh it quickly after every destructive test run. The sequence:

1

Discovery.

A profile has already tagged `ssn`, `tax_id`, `phone`, `email`, `first_name` and `last_name` as sensitive data domains (the discovery chapter).

2

Masking plus subset plan.

An entity with `CUSTOMERS` as driving table pulls in its orders and items. A limit keeps about 5 percent of customers. Masking rules disguise the sensitive columns in that small slice.

3

Workflow.

Generate and run it. The target is the Test Data Warehouse.

4

Monitor and reports.

Watch the job turn to Success, read the session counts, run the Row Count report for the audit file.

5

Store.

The warehouse now holds data set `SHOPSPHERE_MASKED`, version 1.

6

Reuse.

After a test run damages the data, testers reset version 1 back into the test database, cut a smaller related data set for a hotfix, and check coverage.

5.32 Creating plans

5.32.1 A data masking and data subset plan

You may combine subset and masking in one plan or keep separate plans; generation needs its own (User Guide 10.5.10, p.304). The wizard, with reasons:

1

Open the project, click Execute, then Actions > New. Enter a plan name and optional description.

Plans belong to a project so they can use its rules, entities and parameters.

2

Click Add Masking Components and select policies and rules. Click Next.

Pick only what this run needs. You may add rules assigned to a policy and rules that are not.

3

Click Add Subset Components and select entities, groups, templates or XSD files. Click Next.

This is where you say which tables to copy. If an entity is already inside a chosen template, you cannot add the same entity again (p.277).

4

Skip the generation step with Next.

Generation cannot be combined here.

5

Switch off any masking rule you want to skip, using the check box and Off.

A cheap way to test one rule at a time.

6

Filter subset components: choose a basic or advanced expression, or click Limit and choose percentage, absolute value or an interval of rows. Click Next.

An XSD subset component cannot have its criteria overridden. Details in the Informatica subsetting chapter.

7

Configure connections and other properties.

For a flat-file target you can set test tool integration. If you choose the test data warehouse as the target you enter the data set properties. With an XSD source you cannot add other source types. Click Next.

Optionally override plan settings for one data source, or data source settings for one table. Click Finish.

Overrides handle the one table that needs special treatment.

Expected result: The plan appears in the project's Execute list. Nothing has run.

Steps from User Guide 10.5.10, pp.304-305.

5.32.2 A data generation plan

The wizard is the same, but you skip the masking and subset pages, add generation components (entities and tables), enter the number of records and set connections (p.305). The full walk-through, with ShopSphere examples, is in the Informatica generation chapter. There is also a one-column ad hoc generation plan reached from Define | Data Generation with Actions > Generate and Execute (p.306).

5.33 Plan components in practice

5.33.1 Masking and subset components

Masking components are policies and rules. You may restrict a run to some of them, and turn masking on or off per column in the plan criteria. A plan that decrypts with a Format Preserving Encryption rule cannot include other masking rules (p.292). **Subset components** are entities, groups, templates or XSD files (p.293).

Component criteria. For each column of a subset component, choose a filter method in the Plan Components panel: *Custom* (a Basic expression with an operator and operand, or an Advanced expression with several columns, functions and parameters) or *Limit* (percentage, absolute value or interval of rows). To disable masking for a column click the masking component, select the column in the Criteria panel and click OFF (or ON) (p.294).

SHOPSPHERE: SHRINK TO ABOUT 5 PERCENT

The plan holds one entity (driving table CUSTOMERS , children ORDERS and ORDER_ITEMS) and one policy PII_Policy . For the entity's customer criteria choose **Limit** by percentage and enter 5. Subset runs first, so only the customers in the slice, and their orders and items, are masked (p.275). Check the Row Count report after the first run to see how the limit applied to the entity.

What subset-then-mask does to three rows

Illustration of the rule on p.275: subset runs first, so customer 1002 never reaches the target, and masking then changes only the rows that arrived.

EXAMPLE: BEFORE AND AFTER A PLAN WITH AN ENTITY AND A PII POLICY

customer_id	In production	In the subset?	In the target
1001	Asha Menon, 123-45-6789	Yes	Meera Joshi, 456-78-9012 (masked)
1002	Ravi Kumar, 234-56-7890	No	not present
1003	Meera Nair, 345-67-8901	Yes	Kiran Rao, 567-89-0123 (masked)

5.33.2 Connections

A workflow connects to the Integration Service, the TDM repository and one or more connection objects. You choose them in the plan (p.276):

Connections you can set in a plan (User Guide 10.5.10, p.276)

Connection	Purpose
Source	Where data is read from
Target	Where data is written: relational, test data warehouse, flat file or Hadoop HDFS, or a variable
Staging	A database for temporary staging tables; a default comes from Preferences. Recommended to be the source connection for cyclic subsets and similar cases (p.286).
Lookup	A database that holds lookup tables
Storage	A database for storage tables used for repeatable masking (substitution and expression); default from Preferences
Teradata	An ODBC connection to Teradata
Dictionary	A database with a dictionary table for substitution, email masking and data generation

Plans also need connections to repositories, and this has a hard limit. An offline job needs three connections to the TDM repository, each workflow in a multi-workflow plan needs one more, and you can run at most **eight workflows at a time** (eight workflows need 11 connections). Extra workflows wait in the queued state (p.276).

The guide also states that **Use Source Connection as Target Connection** is for in-place masking, where data is changed in the same database it came from (p.279). That is risky.

5.33.3 Pre and post workflows, and target SQL

You can run another existing workflow in the project's PowerCenter folder *before* the TDM workflow (it must finish successfully first) or *after* all TDM workflows complete successfully. Example from the guide: copy files from production area to test area, mask them, then run database statistics. You can disable triggers before and enable them after. Not available for Hadoop, reset and restore plans (p.277).

You can also run **target pre SQL and post SQL** in the plan's Advanced Options, using the variables `$OWNERNAME` , `$SCHEMA` and `$TABLENAME` . Example from the guide: `TRUNCATE`

TABLE \$OWNERNAME.\$TABLENAME; deletes all rows before the load (pp.277-278). Use these variables so one statement works across environments.

5.34 Plan settings you will actually touch

5.34.1 Target options

Target options (User Guide 10.5.10, pp.282-283)		
Option	What it does	ShopSphere advice
Truncate Table	Truncates the table before loading. Off by default.	On for a private test schema, off for a shared one
Disable Indexes	Faster loading	Safe if nobody else uses the tables during the load
Disable Constraints	Disables physical constraints before the load and enables after. If enabling fails, the workflow fails and you must enable them by hand and rerun.	Needed for parallel loads, but tests on those tables may fail during the window
Skip Validation of Constraints	Skips validation when enabling (Oracle and similar targets)	Only if you trust the data
Disable Triggers	Disables DML triggers; selected automatically with Truncate for relational targets	Remember: only DML triggers
Optimize Relationship Type for Entities	Optimizes relations at plan level; needs plan-level criteria	Leave default

Two platform notes from the guide: on a DB2 for z/OS target, disabling constraints leaves the tablespace in Check Pending state until a Check Data utility is run; on PostgreSQL, exclusion constraints are not disabled (p.283).

5.34.2 Error and recovery

Error and recovery options (User Guide 10.5.10, pp.285-286)

Option	Meaning and default
Stop on Error	Number of non-fatal errors before the session stops. Zero means never stop for them. Default 0.
Suspend on Error	Suspend the workflow so you can fix and restart. Default No.
Rollback Transactions on Error	Roll back at the next commit point on a non-fatal write error. Default Yes.
Recover Strategy	Resume from last checkpoint; Restart task; or Fail task and continue workflow (default)
Enable High Availability Recovery	Allows recovery from a checkpoint. Default No.
Commit Type and Interval	Target-based by default; commit every 10,000 rows by default
Error Log Type and Directory	Flat file or none; default none

5.34.3 Advanced options (selection)

- **Tables Per Workflow:** number of tables per workflow, default 10 (p.287). More tables than that means several workflows from one plan.
- **Max Parallel Sessions:** sessions that may run at once, relevant when indexes and constraints are disabled (p.287).
- **Data Integrity for Subset:** Referential only; Transactional Integrity for Major Relation Type (default); One-way; Two-way (p.287). See the subsetting chapter.
- **Email:** notification address, subject and text for when the plan finishes (p.287).
- **Target Load Type:** Normal (default) or Bulk. Bulk is faster but limits recovery because no database logging happens (p.288).

- **Source and Target Schema Name:** use when the tables sit in a different schema; may be a parameter (p.288).
- **Batch Update and Batch Size:** for in-place masking (p.288).
- **Shared Storage Table, Cache Size, Storage Commit Interval:** tuning for repeatable masking (pp.289–290).
- **Enable special characters in metadata:** default No (p.289).
- **Reset Recovery prefix:** used only to restore after a failed reset (p.289).
- **Locale, Datetime Format String, Enable High Precision, DTM Buffer Size, SSN High Group File, Use SoftHSM:** specialised settings (pp.287–290).

5.35 Source, table and file overrides

Many plan settings can be overridden for one data source or one table; where a property is not offered at that level, the plan-level value applies (p.295). Examples: connections, flat-file name, delimiter and encoding, schema names, pre and post SQL, batch size, table name overrides and partitioning (pp.295–301).

For flat-file sources you can supply a **file list**, a text file naming one source file per line; all listed files must share the same properties (p.303).

5.36 Parameters and parameter files

Parameters let one plan serve many runs. You may use **project parameters** and **global parameters** (created by an administrator, shown with an asterisk) in connections, owner names and filter criteria (pp.275, 279). In an entity or plan you can use a parameter instead of a fixed value in a criterion.

A **parameter file** lists every parameter used in a plan with its default value. Criteria parameters carry an *ignore flag* that defaults to "N" (include); set it to "Y" and save to ignore that criterion in a run. Parameters you cannot override do not change, and the workflow logs an error (pp.309–311). Create one with Actions > Create Parameter File on the plan. If you use parameters for connections, the parameter file must be on the node that runs the Test Data Manager Service, in the same path on every node (p.279).

When you execute the workflow you choose how values are supplied (pp.310–311):

Parameter value options at execution

Option	Behaviour
Default	Use the parameter file named in the plan settings
Use Parameter File	Browse to a different file for this run
Add Parameter Values	Type values in a list. Criteria parameters are not listed by default; use Select Parameter to add them or the workflow ignores them.

SHOPSPHERE: ONE PLAN, THREE REGIONS

Make the subset criterion on `zip_code` a criteria parameter `p_zip`. Run the same plan with `560001` for the Bengaluru team, `600017` for Chennai, `682001` for Kochi. No plan edits, three different data sets. This is the guide's "test data on demand" idea (p.309).

5.37 Plan management

After you create a plan you can edit it, copy it, export it to XML, import one exported from another TDM repository, or delete it (User Guide 10.5.10, p.304).

Plan management tasks (pp.304-308)

Task	How	Notes
Copy	Execute tab, click the plan's Description or Status field (do not open it), Actions > Duplicate	Default name "Copy of <original name>". Edit the copy.
Export	Select the plan the same way, Actions > Export, save the XML	Default file name starts with "Plan_" plus date and time
Import	On the Projects list select the project (do not open it), Actions > Import, browse to the XML, Finish	For moving plans between TDM repositories
Delete	Select the plan, Actions > Delete, confirm Yes	Deleting the plan also deletes its workflow

WHY EXPORT AND IMPORT?

ShopSphere has a development TDM and a production-grade TDM. A plan proved on the first is exported to XML and imported into the second, with connections and parameters adjusted. The command line tool `ilmcmd`, covered below, can do the same export and import without the web page (pp.330-332).

5.38 Generating and running a workflow

You generate a workflow from a plan to perform subset, masking and generation. When you generate, the **PowerCenter Repository Service** creates the mappings. When you start, the **PowerCenter Integration Service** performs the work. You can generate and start in a single step (User Guide 10.5.10, p.308).

1

Generate: in the project click Execute, select a plan, Actions > Generate Workflow.

This builds mappings and stores them. Do it again whenever the plan changes.

2

Choose whether to run now or later.

You may generate now and execute on a schedule.

3

Execute: Actions > Execute Workflow from the Plan properties or Workflow view.

If the plan has several workflows, open the Workflow view, select one and execute it. A workflow name contains the project, plan, connection and table names (p.309).

4

Pick the Integration Service and run now or schedule.

You must select the service configured in the Test Data Manager Service.

5

Choose parameter values if the plan has parameters.

See the parameters section.

6

If the target is the warehouse, optionally publish to the self-service portal, then click Generate and Execute.

Publishing creates a data pack for testers who cannot use Test Data Manager.

Expected result: The workflow appears in the Workflow Executions view and in Monitor.

Steps from User Guide 10.5.10, pp.308-311. To do both in one go use Actions > Generate and Execute on the Plan / Properties page.

CONSTRAINTS THAT WILL NOT TURN BACK ON

If you ran the workflow with Disable Constraints and TDM then fails to re-enable them after loading, the workflow fails. Enable the constraints in the database by hand and run the workflow again (p.311).

5.38.1 The Workflow Executions view

This view shows the current workflow for the plan and all earlier ones, even for workflows that no longer exist in the plan. Click the Job ID to see the workflow log. Fields: Job ID, Name, Description (profiling, import or workflow), Status (In Queue, Running, Success, Error), Start and End Date/Time, User (pp.311-312). The Sessions tab lists each session, with Source Rows Succeeded and Failed and Target Rows Succeeded and Failed. Click the session name for Session Details, with the tables processed, affected rows, rejected rows, throughput and last error message (pp.313-314).

Tasks you can perform on a workflow here and in Monitor:

Workflow tasks (pp.312, 318)

Task	What it does	Use it when
Auto Refresh	Refreshes every ten seconds	Watching a run
Workflow Stop	Stops after the next commit	You want a clean stop
Workflow Abort	Stops immediately without waiting for a commit	Something is badly wrong
Workflow Recover	Recovers a stopped or aborted workflow. With high availability recovery, restarts from the last checkpoint; otherwise follows the plan's recovery strategy.	After fixing the cause
Abort (a job)	Stops a non-workflow job immediately	A stuck profile or import. Workflows are stopped or aborted with the two tasks above, not this one.
Unschedule	Removes a job you scheduled for later	Cancelling a scheduled run

5.39 The Monitor: watching and reading jobs

In the Monitor view you check the status of jobs you start in Test Data Manager, stop jobs and view job and session logs. The global Monitor shows jobs for all projects; open a project and click Monitor to see only that project's jobs. For a data set version, open it and click Monitor (User Guide 10.5.10, p.315).

VISUAL REFERENCE

Informatica TDM 10.5.10 User Guide, p.316, figure "Monitor view". The figure shows the Test Data Manager window with the Monitor tab active, a Jobs grid with columns Job ID, Name, Description, Status, Project, Start Date/Time, End Date/Time and User, rows of Execute Workflow, Generate Workflow and Import from Source jobs all with status Succeeded, and a lower panel with Properties and Logs tabs showing the selected job's

General details (Job ID, Name, Description, Project, start time, elapsed time, status, user). What to look for: each job has a consecutive Job ID, and one plan run produces a Generate Workflow row and an Execute Workflow row.

5.39.1 Job types

Job types shown in Monitor (User Guide 10.5.10, pp.316-317)

Job type	What it does
Import from Source	Imports source data from a source file
Import from PowerCenter	Imports a source from the PowerCenter repository
Profiling	Discovery of primary keys, entities and data domains
Generate Workflow	Generates a workflow from a plan
Execute Workflow	Runs a workflow after generation; may run several workflows
Workflow	Runs the subset, masking or generation operation
Upload to test tool	Uploads flat files to the integrated HP ALM location
Reset Data Set from Test Data Warehouse	Resets target data from a data set
Data Coverage Analysis	The analysis in a data coverage task
Fill Cell	Moves data from one cell to another in a coverage task
Session	One task within a workflow; click a workflow job ID to see its sessions

5.39.2 Job states

Job statuses (User Guide 10.5.10, p.317)

Status	Meaning	Your move
In Queue	Waiting to run	Wait. If it never moves, check the Integration Service and the eight-workflow limit.
Running	In progress	Watch, or turn on Auto Refresh
Success	Ran successfully	Verify the data, run reports
Error	Did not run successfully. Click the job ID for the log.	Read the log
Terminated	The job was terminated	Find out who or what stopped it; recover if it is a workflow

Job details you can sort and filter by: Job ID, Name, Description (plan, profile, folder or connection), Status, Project, Start and End Date/Time, User (p.317).

5.39.3 Reading the logs

Select a job and open the **Logs** tab. The TDM server writes TDM logs; the PowerCenter Integration Service writes session and workflow logs (a Data Integration Service does for Hadoop). TDM always writes **console logs** when a job is triggered, so if job logs are missing, read the console log (User Guide 10.5.10, p.318). Each message has a Date/Time, a Severity and a Description that explains the error, its cause and the solution.

Severity levels (p.319): **Error** (the TDM server failed an operation; start here), **Warning** (an operation that might cause an error), **Info** (normal), **Debug** and **Trace** (increasingly detailed server operations, mostly for support).

1

Click Monitor, then the job ID, then the Logs tab.

The log belongs to that job only.

2

Filter by severity Error, and by date range or a keyword.

Real logs are long; the first Error line is usually the cause and the rest are knock-on effects.

3

Select a message and click Download to keep the log file.

Attach it to a ticket. To see console logs, select the job ID and use Actions > View Console Logs.

4

For a workflow, click its job ID to open the workflow details, then the Sessions tab.

Look for a session whose Target Rows Failed is not zero.

5

Click the Session ID to see tables processed, affected and rejected rows, error messages and the session log.

This is where "which table broke" is answered.

Expected result: You can name the failing session, table and message.

Steps from User Guide 10.5.10, pp.319-320.

For Hadoop plans you see mapping logs only: no workflow logs and no row counts (p.320).

On a clean copy run the Source and Target Rows Succeeded counts match and both Failed counts are zero; a non-zero Failed count means rejected rows, so read the session's last error message.

5.40 Reports

Reports show detailed information about recent activity, a project or a plan. They open in a new window, so allow pop-ups for the site (User Guide 10.5.10, p.321).

Reports in Test Data Manager (pp.321-325)

Report	Run it on	What it contains	How to run
Audit Trail Report	The Overview (Recent Activities)	All recent activities: type of activity, type and name of object, user, date and time. Ignores permissions. Filter by date range. Download as CSV.	Click the Audit Trail Report button in the Recent Activities tab
Data Masking Report	A project	All table columns with the rules and policies assigned	Define > Data Masking > Print Masking Report
Plan Audit Report	A plan with at least one successful run	Policy assignments (data source, table, column, data type, data domain, policy, rule, rule type, masking properties, exception handling) and plan execution (connections, dates, total rows processed, status)	Execute, click the plan, Actions > Plan Audit Report
Plan Detail Report	A plan	Masking assignments, entities with filter criteria, groups, templates and template components	Execute, click the plan, Actions > Plan Detail Report
Row Count Report	An Execute Workflow job	Tables that took part and the number of rows the plan affected	Monitor, click an Execute Workflow job, Actions > Row Count Report

Pick the report by question. *Who changed what and when?* Audit Trail. *Which columns have which masking?* Data Masking or Plan Detail. *Prove to an auditor which rules ran and on how many rows?* Plan Audit. *How many rows did this run touch?* Row Count (the lab and the generation chapter both use it).

5.41 Data coverage analysis, at overview level

Having a data set is not the same as having *enough of the right data*. Data coverage analysis checks whether a data set holds a minimum amount of data for specific test cases. It is

pairwise: you choose two columns and see how many rows exist for every combination of their values, drawn as a grid (User Guide 10.5.10, p.259). Cell colour shows density: white means no data, light blue is below the minimum threshold you set, dark blue is above it (p.261).

VISUAL REFERENCE

Informatica TDM 10.5.10 User Guide, p.266, figure "data coverage analysis page". A grid with one column's values along the bottom and another's up the side, mostly white cells with some light and dark blue, a green-bordered selected cell and a side panel of task columns. What to look for: white cells are the gaps.

1

Open the data set, click Data Coverage, Actions > New, name the task and set a minimum threshold.

The threshold is the least number of rows you accept as "dense enough".

2

Select a master table (and optionally related tables), then create task columns.

Columns can be *Use as is* (individual values), *Range* (numeric or date ranges) or *Mapping* (group values, such as states into regions); binary columns are not allowed (pp.260-263).

3

Put one column on each axis, optionally add up to two filter columns, and click Analyze.

The grid shows density.

4

Fix gaps: mark meaningless cells invalid, or select a gap and use Fill to move rows from dense cells.

Fill runs a separate job per source cell. If a needed parent row is missing, a dialog asks you to type values (pp.267-269).

Coverage grid (schematic, ShopSphere)

threshold = 5	customers	total_amount band		
zip band	under 500	500-2000	over 2000	
56xxxx	9	7	0	← gap
60xxxx	6	2	1	← below threshold
68xxxx	0	8	5	← gap

Schematic illustration, not an official screenshot.

Here "large order from a Bengaluru customer" has no data. Fill moves rows in, or you generate more. Fill changes the data set itself (p.268), so keep a version first. Coverage jobs show in Monitor as Data Coverage Analysis and Fill Cell, run by the Data Integration Service (pp.26, 316-317).

5.42 Command-line tools: ilmcmd, tdwcmd and tdwquery

TDM ships three small command-line programs (User Guide 10.5.10, pp.326-350). Scripts and schedulers can call them, so a nightly refresh needs no human clicking. Names of commands, options and objects are case sensitive, and `-h` shows the syntax.

The three tools at a glance

Tool	Purpose
<code>ilmcmd</code>	A subset of Test Data Manager tasks: import and export XML objects, search, validate, delete, generate and run workflows, show workflow status, reset data set versions, and run data set subset plans
<code>tdwcmd</code>	List test data warehouse details: repository, warehouse, connections, data sets, versions and version tables (it lists and changes nothing)
<code>tdwquery</code>	Run SELECT queries (joins, aggregates, WHERE) on the tables of one data set in the warehouse

Examples printed in the guide for `ilmcmd` (pp.329-338): generate a workflow, get its status, run it, export a policy, and reset a data set version from a reset file. It cannot search data generation objects (p.333), and over HTTPS you must first set up a keystore (p.327).

```
ilmcmd -wf -g -Project Infa_Project -pn Plan_NameMasking
ilmcmd -wf -gs -JobId 360
ilmcmd -wf -ex -Project TDM -pn Plan_NameMasking -is PCInteg
ilmcmd -r -dvn "RESET Version1" -rf "C:\Documents\ResetObject.json" -
ris PIS
```

For `tdwquery` you first fill in `userConfig.ilm` (service URL, user, domain, warehouse and data set version), then run for example `tdwquery "SELECT * FROM td_empdetails"` (pp.348-349).

PASSWORDS ON THE COMMAND LINE

The guide's `tdwcmd` examples put the password in the command (`-pd Administrator`), where shell history can expose it. Follow your organisation's secret-handling rules.

5.43 The Test Data Warehouse

Test data goes stale and gets damaged. A tester deletes orders to check a refund flow; a batch job rewrites prices; another team starts testing on the same tables. Re-running the whole subset and masking plan each time is slow. The **Test Data Warehouse** solves this: it is a central location that stores multiple versions of test data, and multiple Test Data Manager services can use the same warehouse, so teams share one store (User Guide 10.5.10, p.236).

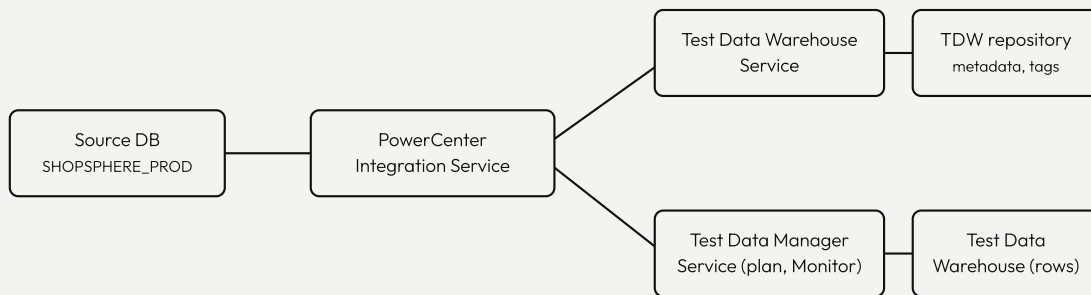
The guide's scenario: before any tests, store the original data as a data set. After each team finishes, store the modified data as another version. To return the environment to its original state, restore the version holding the original data. You can restore any version (p.237).

5.43.1 What is behind it

You create a **Test Data Warehouse Service** in the Informatica Administrator tool. That application service manages two relational databases (pp.26-28, 236):

- the **test data warehouse repository**, which holds project metadata (plan information, version, tags) and source table metadata of each data set;
- the **test data warehouse** itself, which holds the actual source data you put in data sets.

The service copies the metadata created when you make a data set and creates the data set tables in the warehouse (p.27). Workflows that store data in the warehouse need connections to the external source databases, the repository and the warehouse (p.28). Data coverage tasks run on the Data Integration Service (p.26).



Simplified from pp.26-28 and 236: a map of who holds what, not a network drawing.

A plain database backup

Restores a whole database, usually needs a DBA, and carries no test metadata. Hard for a tester to find "the data from the refund tests".

A warehouse data set (concept view)

Holds chosen tables, with application name, version and tags you can search. Testers with Execute permission can reset it, or pick it up from the portal. The analogy to a backup stops here: a data set can also be edited and sliced into related data sets.

5.44 Data sets, versions and tags

A **data set** is a collection of one or more tables from one or more data sources, and generally matches one or more test cases (User Guide 10.5.10, p.238). You create one when you subset production, make a masked copy, or generate data, by selecting the test data warehouse as the target when the plan runs. It stores source table metadata and plan metadata, and you can search by those properties.

5.44.1 Creating a data set

1

Check the tables first.

You cannot create a data set from tables with user-defined data types; mark such columns Restricted under Discover > Columns. Tables without primary keys need logical keys, or a reset might fail. More than two character columns with precision 4000 should be edited to 3,999 or less (p.239).

2

Build a plan (masking, subset or generation) and in the plan settings select Test Data Warehouse from the target connections.

The Data Set Properties section appears.

3

Enter a data set name (default: the plan name), application name, application version and tags.

A new name creates version 1. TDM fills in the version number (p.240).

4

Finish the plan and click Actions > Generate and Execute.

Optionally publish to the self-service portal in the same step.

Expected result: Monitor shows the job. On success the data set appears in the Data Sets view.

Condensed from User Guide 10.5.10, pp.239-240.

5.44.2 Versions and tags

Test data changes during testing. You make a new version in two ways (p.238):

- rerun the same plan without changing the data set name; or
- create and run a different plan and enter the same data set name.

Tags are comma-separated words (test case name, ticket number) used to filter and search data sets. **Row-level tags** mark individual rows, to track edits or to select rows for a related data set (p.239). Tag text may include spaces and special characters but not a comma (pp.247, 255).

SHOPSPHERE: NAMING

Data set `SHOPSPHERE_MASKED` , application `CustomerOrders` , application version `2.4` , tags `baseline, masked, 5pct` . After the regression run, rerun the plan or run a different plan with the same data set name to get version 2 tagged `after-regression` .

VISUAL REFERENCE

Informatica TDM 10.5.10 User Guide, p.237, figure "tasks for storing and using warehouse data". A left-to-right flow: create project, optional parameters, plan with the warehouse as target, generate and run, view the data set, then optional steps (edit

metadata, permissions or lock, edit data, coverage, reset, publish to the portal). What to look for: only the first four boxes are mandatory.

5.45 Restoring: the reset

Reset moves data from a data set version in the warehouse to a target, so a test environment goes back to a known state (User Guide 10.5.10, p.240). You can make the reset delete everything in the target and replace it, which suits tables you do not share. Or you can keep other testers' data in the target and replace only the part that overlaps the data set. In selective resets TDM runs an SQL script that backs up the deleted data in the target (p.241).

5.45.1 Master and transactional tables

Reset behaves differently per table, according to how you classify it (p.241):

Table classification and reset behaviour (pp.241-242)

Class	Meant for	Reset action by default	ShopSphere
Master	Records shared by all testers	No deletes. Upsert: update rows that exist in both, insert rows that exist only in the data set.	CUSTOMERS, PRODUCTS
Transactional	Descendants of master tables; rarely shared	Insert. If a record already exists the insert fails. Deletes cascade from top-level transactional tables (direct children of master tables).	ORDERS, ORDER_ITEMS

Unclassified tables are treated as transactional. To classify: open the data set, Plan Components tab, Tables tab, Edit Table Type. When you mark a table as master you must choose to mark all descendants as transactional, or all other tables (pp.241-242). Verify the classification before every reset.

5.45.2 Reset options

1

Open the data set version you want and click Actions > Reset.

The source is fixed: the warehouse.

2

Select the target connection.

Usually your test database.

3

Choose reset settings.

Truncate Tables deletes all target data and loads the data set, disabling constraints and indexes; no backup is made. *Disable Constraints and Indexes* speeds things up. *Upsert in Transactional Tables* avoids insert failures when parents changed. *Insert in Master Tables* skips the existence check but needs all tables to be master and no overlapping records.

4

Enter reset recovery properties if you are not truncating: a *Backup Recovery Prefix* and the *Backup Recovery User Name* (schema) for backup tables named `<prefix>_<tablename>` .

This is your undo.

5

Review Error and Recovery and Advanced Settings; optionally override at data source or table level; choose the PowerCenter Integration Service; run now or schedule.

Then follow the job in Monitor.

Expected result: A "Reset Data Set from Test Data Warehouse" job reaches Success and the target matches the version.

Steps from User Guide 10.5.10, pp.243-244.

VISUAL REFERENCE

Informatica TDM 10.5.10 User Guide, p.245, figure "reset process". It is a flowchart: Reset leads either to "Full restore" (truncate tables, insert tables from the data set, success message) or to "Reset with default settings", which validates table classification (not valid: exit with error), creates staging tables to find candidates for cascade delete, then asks whether to disable constraints and indexes. Yes goes to a parallel transactional cascade delete and insert; No goes to a sequential one. Both then upsert master tables in parallel, re-enable constraints, and show the success message. What to look for: a note says the backup tasks are not drawn.

5.45.3 Speed versus downtime

Disabling constraints lets TDM load several tables in parallel, so resets finish sooner. But constraints are disabled for the entire table, and while they are off, other testers' tests that

use those tables might fail. If tables are shared, a slower reset may be better than downtime; if no one else is using them, disable constraints (pp.242-243). As good practice, create an index on foreign keys in the target so child-record deletes are quick (p.243).

5.45.4 Failure and special cases

- **Failed reset:** if recovery was enabled, backups are kept. Restore with a separate plan: put the backed-up tables in a group, add it as a subset component, use the backup schema as source and the real database as target, disable constraints and indexes, set Treat Source Row As Update and Update As "Update else Insert", and put the backup prefix in the Reset Recovery Prefix advanced field (p.247). On success TDM deletes the backup tables (p.246).
- **Empty strings in non-null columns:** a reset fails if it must copy empty strings into a non-null column, for example from an Oracle warehouse to SQL Server. The project can supply default values, but a reset fails if the non-null target columns are unique (p.243).
- **PWX NRDB Batch target:** you must delete the target files yourself first; the reset only copies (p.243).

5.46 Managing data sets

- **Permissions.** Whoever runs the plan owns the data set. Owner and domain administrator can grant **Read** (view, monitor, create coverage tasks, run related data set plans), **Write** (create, edit metadata and data, classify, delete, create related plans) and **Execute** (reset). Users also need matching privileges such as View Data Set or Reset Data Set (pp.252-253).
- **Lock and unlock.** Owner and administrator can lock a data set so nobody edits or resets it, for example while you check it first. The owner and administrator can still reset it (p.254).
- **View and edit data.** From the Tables tab you can browse, or run SQL in the SQL Query field and edit the grid, add or delete rows and add row-level tags. DML only, no DDL. Restrictions: no aggregates, GROUP BY or joins in SELECT; no table aliases or owner names; dates in WHERE need TO_DATE; Oracle BLOB and RAW data cannot be viewed or edited (pp.255-258).
- **Related data sets.** A related data set is a subset of a data set, cut by an entity or group built inside the data set and run in a plan. A lineage window shows parent and child data sets (pp.248-252). ShopSphere use: a hotfix needs only the Kochi customers.
- **Self-service portal.** Publish a data set as a **data pack**. Testers without Test Data Manager can then view and restore the data. You cannot delete a published data set until you delete the data pack (pp.239, 247-248).

- **Delete.** From the Data Sets view, delete whole data sets or single versions, Actions > Delete. Data and metadata are removed (p.248).

5.47 Labs: the story in practice

Lab A: build a masking and subset plan that stores a data set

Goal: take the ShopSphere sample from production to a masked, small, versioned data set in the warehouse. Needs an installed TDM with a Test Data Warehouse Service, a project with the four tables imported, an entity with CUSTOMERS as driving table, and masking rules assigned (see the masking and subsetting chapters).

1

**In the project click Execute, Actions > New, name the plan
PLAN_MASK_SUBSET .**

Naming by purpose makes Monitor rows easy to find.

2

**Add Masking Components: select policy PII_PoLicy . Add Subset
Components: select the entity.**

Subset runs first, then masking on the subset (p.275).

Expected result: Both lists show one item.

3

Select the entity, choose Limit and enter 5 percent. Click Next.

Keeps the data set small.

4

**On the connection page set Source Connection to the production schema
connection and Target Connection to Test Data Warehouse.**

The Data Set Properties section appears.

5

**Enter data set name SHOPSPHERE_MASKED , application CustomerOrders ,
version 2.4 , tags baseline,masked . Finish.**

Metadata for later search (pp.239-240).

Expected result: The plan is listed in Execute.

6

**Open the plan and click Actions > Generate and Execute. Choose the
Integration Service, run immediately.**

Two jobs start.

7

Open Monitor and watch Generate Workflow then Execute Workflow until Success.

Turn on Auto Refresh (every ten seconds).

Expected result: Both rows show Success. A row in Error means open the job ID and read the log.

8

Select the Execute Workflow job and choose Actions > Row Count Report.

This is your evidence of how many rows each table received.

Expected result: A new window lists CUSTOMERS, ORDERS and ORDER_ITEMS with counts. ORDERS and ORDER_ITEMS should not exceed what the selected customers own.

9

Open the Data Sets view and the new data set. Check the Tables tab.

Run `SELECT * FROM customers` in the SQL Query field and confirm names are disguised (compare with production by sample, never copy production values out).

Expected result: Version 1 exists; sensitive columns look masked.

► **Show a worked answer**

Lab B: store a second version, then reset the first

Goal: undo the effects of a destructive test run. Uses the data set from Lab A and a private test database as the target.

1

Classify the tables.

Open the data set, Plan Components tab, Tables tab, select CUSTOMERS and click Edit Table Type, choose Master and then "Mark all descendant tables as transactional".

Expected result: ORDERS and ORDER_ITEMS show as transactional (pp.241-242).

2

Reset version 1 into the test database once.

Actions > Reset, target connection = test database, tick Truncate Tables (the schema is yours), run. That creates your working copy.

Expected result: A reset job reaches Success. Row counts in the test database match the data set.

3

Damage the data.

In the test database run `DELETE FROM order_items;` and `UPDATE customers SET zip_code = '000000';` This simulates a destructive test.

Expected result: Order items are gone and zip codes are wrong.

4

Reset again, this time without truncate and with recovery.

Actions > Reset, leave Truncate off, enter Backup Recovery Prefix BK and a backup schema user name, run.

Expected result: Order items return and the damaged zip codes are updated back from the data set (master rows are upserted). On success the backup tables are deleted (p.246); counts match version 1.

► Show a worked answer

Lab C: read a failed job

Goal: practise reading logs. The messages below are example log lines showing the structure the guide describes (date, severity, description).

```
10:02:11 Info      Workflow wf_PLAN_MASK_SUBSET started
10:02:15 Info      Session s_PLAN_MASK_SUBSET_1 started
10:02:19 Warning   1 row rejected in table ORDERS
10:02:23 Error     Session s_PLAN_MASK_SUBSET_1 failed: could not
enable constraint on ORDERS
10:02:24 Error     Workflow wf_PLAN_MASK_SUBSET failed
```

1

Find the first Error line.

Later errors are usually consequences.

Expected result: The 10:02:23 line.

2

Link it to a plan setting.

Which option disables and re-enables constraints?

Expected result: Disable Constraints in Target Options (p.283).

3

Decide what to do.

Read the earlier Warning too.

Expected result: Fix the rejected row, enable the constraint by hand, rerun the workflow (p.311). Consider whether Recover is better than a fresh run.

► Show a worked answer

5.48 Common mistakes and gotchas

THINGS THAT GO WRONG

- **Changing a plan and expecting the old workflow to follow.** Generate and run again (p.275).
- **Mixing generation with masking or subset in one plan.** Not allowed (p.274).
- **Forgetting the eight-workflow limit.** Extra workflows queue (p.276).
- **Criteria parameters silently missing.** They are not in the run list by default; add them with Select Parameter or the workflow ignores them (p.311).
- **Leaving tables unclassified before a reset.** They are treated as transactional (p.241).
- **Truncating a shared schema on reset.** You delete other testers' data (p.244).
- **Disabling constraints on shared tables.** Other tests may fail during the window (p.243).

5.49 Recap

RECAP

- A plan defines a masking, subset or generation operation; TDM generates a workflow (an ETL job) from it and the PowerCenter Integration Service runs it.
- Subset plus masking can share a plan (subset first); generation cannot share a plan with either.
- Connections, target options, update strategy, error and recovery and parameter files control where and how the run happens, with overrides per source or table.
- Monitor shows each job's status (In Queue, Running, Success, Error, Terminated), its logs by severity, and sessions with row counts. Start at the first Error.
- Reports: Audit Trail, Data Masking, Plan Audit, Plan Detail and Row Count.
- The Test Data Warehouse stores versioned data sets. Reset restores a version, with master and transactional classification deciding what is deleted or upserted.
- Data coverage analysis finds gaps in a data set; ilmcmd, tdwcmd and tdwquery script and inspect the same objects.