CHAPTER 01

Welcome: why TDM exists and the data basics you need

Start from zero: what test data is, why copying production is dangerous, how TDM fits into software testing, and the database and file basics every later chapter relies on.

What TDM is, the three tools, and the example used throughout

1.1 Words to know

Each term is explained properly in the chapter named next to it.

Test data

The data you put into an application so you can check that it works. Think of the customers, orders and products you type in while testing an online shop. (the why-TDM chapter)

Production

The live system real customers use, holding real data. Also called "prod". (the why-TDM chapter)

Non-production environment

Any copy of the application that is not live: development, QA, UAT, staging. Testing happens here. (the why-TDM chapter)

TDM

Test Data Management: the planning, tools and habits used to give testers safe, realistic, right-sized data when they need it. (the why-TDM chapter)

Masking

Replacing real sensitive values (a real name, a real ID number) with realistic but fake ones. (the masking concepts chapter)

Subsetting

Taking a small, consistent slice of a big database instead of all of it. (the discovery and subsetting concepts chapter)

Synthetic data

Data created by a program from rules, not copied from real people. (the classification and generation concepts chapter)

1.2 What TDM is

Test Data Management is the discipline of getting the right data to the people who test software, at the right time, without exposing anyone's private information. Testers need data that looks and behaves like real data. The easy way to get it is to copy the live database, but that copy contains real people's names, phone numbers and ID numbers, it is huge, and it goes stale. So TDM teams find the sensitive values (*discovery*), cut the data down to a usable size (*subsetting*), replace sensitive values with fake ones (*masking*), create brand-new data from rules where no real data fits (*generation*), and deliver fresh copies on request (*provisioning*). the why-TDM chapter explains each of these properly, starting from why testing needs data at all.

The Informatica documentation frames the same idea from the tool side: an organisation creates a smaller copy of production data, masks the sensitive data, finds the sensitive columns, and can create test data that never contained sensitive values (Informatica TDM 10.5.10 User Guide, p.21).

1.3 The three tools and why there are three

This guide covers three commercial products: software licensed from a vendor, not free open-source programs. The why-TDM chapter explains the term *COTS*. Each one is described below in the words of its own documentation.

Tool	What its documentation says it does	Where it appears
Informatica TDM	Works with PowerCenter and other Informatica products to manage non-production data. In one web client (Test Data Manager) you configure data discovery, data subset, data masking and data generation, and you can keep versions of test data in a test data warehouse (User Guide 10.5.10, pp.21-22).	Informatica chapters
GenRocket	A platform for synthetic test data, meaning data manufactured by an algorithm rather than copied from real events. Its help site says the whole platform is built on five key components: Domain, Attribute, Generator, Receiver and Scenario, and that its methodology is "Model, Design, Deploy, Manage".	GenRocket chapters
Delphix	A data platform whose masking capability finds sensitive data (profiling), replaces it with fictitious but realistic values (masking), and which can also create lightweight virtual copies of databases (virtualization). The notes here cover the masking engine, called Continuous Compliance in the docs.	Delphix chapters

Sources: Informatica User Guide pp.21-22; the GenRocket pages Why Synthetic Test Data?, GenRocket Five Key Components and The GenRocket Methodology; the Delphix page Introduction to Delphix Masking.

1.3.1 Why three tools and not one?

Each product puts its weight in a different place. Informatica presents discovery, subset, masking and generation together in one client. GenRocket is built around generating data from scratch. Delphix is built around masking plus fast delivery of copies. Seeing the same ideas in three products shows the idea rather than a menu of buttons, and a TDM engineer who knows what masking is can pick up a fourth tool quickly.

> **WHY THIS MATTERS**
>
> Learning "click here, then here" for one tool leaves a person lost when the project uses another. The concepts come first (Part I); each tool is then one way to carry them out.

1.4 Meet ShopSphere: the running example

Every chapter uses the same fictional company.

ShopSphere is a fictional online retailer. Its engineers are testing the "Customer Orders" application: customers register, browse products and place orders. The live database is called `SHOPSPHERE_PROD`. It is never copied into testing as it is. That sentence is the whole reason this subject exists, and the next chapter explains why.

The database has four tables. A table is like a sheet in a spreadsheet; the data basics chapter explains them properly.

Table	What a row is	Columns
`CUSTOMERS`	One customer	`customer_id`, `first_name`, `last_name`, `email`, `phone`, `ssn`, `date_of_birth`, `zip_code`, `tax_id`
`ORDERS`	One order placed by a customer	`order_id`, `customer_id`, `order_date`, `total_amount`
`ORDER_ITEMS`	One product line within an order	`item_id`, `order_id`, `product_code`, `quantity`, `unit_price`
`PRODUCTS`	One product in the catalogue	`product_code`, `product_name`, `price`

Three example customer rows follow. The "SSN" values are made-up patterns, not real identifiers.

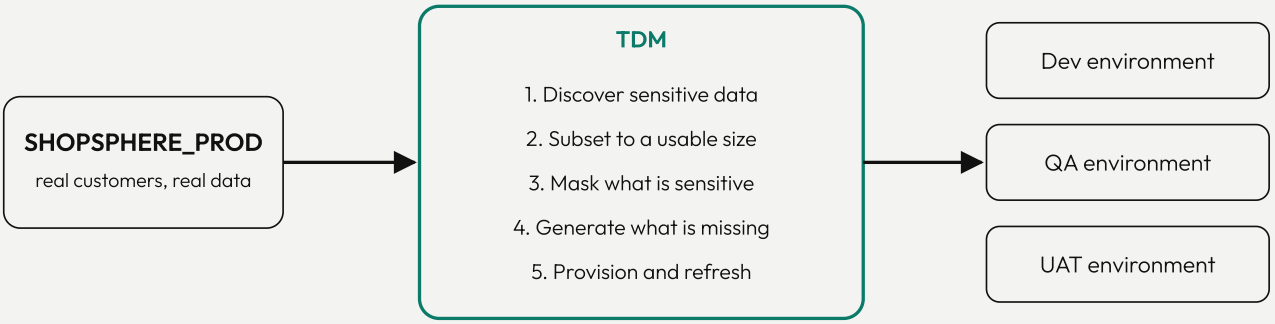
SHOPSPHERE CUSTOMERS (EXAMPLE DATA)

customer_id	first_name	last_name	email	phone	ssn	date_of_birth	zip_code
1001	Asha	Menon	asha.menon@example.com	9876543210	123-45-6789	1990-04-12	560001
1002	Ravi	Kumar	ravi.k@example.com	9123456780	234-56-7890	1985-11-02	600017
1003	Meera	Nair	meera.n@example.com	9988776655	345-67-8901	1999-07-23	682001

1.4.1 The ShopSphere test team

- **QA engineers** who write and run test cases.
- **Developers** who build the app and need data to debug with.
- A **TDM engineer**, who prepares and delivers the test data.
- A **data steward**, who knows what each piece of data means and who owns it.
- A **compliance officer**, who makes sure no private data leaks.

1.4.2 The picture you will see again and again



Original diagram: the TDM idea in one picture. Production data goes in; safe, right-sized, realistic data comes out in each test environment. The five steps are explained in the why-TDM chapter.

When a later chapter says "ShopSphere", it means this company. When it says "the CUSTOMERS table", it means the table above. In the data basics chapter you will see how the four tables relate to each other.

Why Test Data Management exists

Software has to be tested, testing needs data, and the obvious source of data (the live system) is the one place that must not be copied carelessly. That is why a whole discipline, Test Data Management, grew up around what looks like a small chore. ShopSphere, the fictional company used throughout, is introduced in the start-here chapter.

1.5 Words to know first

Software testing

Checking, before real users rely on it, that a program does what it should and does not break when something unexpected happens.

Test case

One specific check: "do this, with this data, and expect that result". Example: place an order for a customer with no previous orders and expect a confirmation.

Test data

The data a test case needs: the customers, products and orders that exist before the test starts, and the values typed in during it.

Environment

A complete running copy of the application with its own servers and its own database. Different environments serve different purposes.

Production (prod)

The live environment used by real customers. It holds real data.

Non-production

Every environment that is not production: dev, QA, UAT, staging and so on.

Sensitive data

Information that would harm a person or the company if leaked: names with ID numbers, health records, card numbers. The sensitive data and regulations chapter covers it fully.

Refresh

Replacing the data in a test environment with a newer copy.

Edge case

An unusual but possible input, such as an empty name, a very long address or a date on 29 February.

COTS

Commercial off-the-shelf: software you buy ready-made from a vendor and configure, instead of building it yourself.

TDM

Test Data Management.

1.6 What software testing is, and why it needs data

1.6.1 Testing in brief

Imagine ShopSphere's developers have just changed how discounts are calculated. Before the change reaches real shoppers, someone has to check it. They log into a copy of the app, place an order, and look at the total. If the discount is wrong, they report a bug. That checking is *software testing*.

Teams run many kinds of testing, from a developer checking one function to a performance tester sending thousands of orders at once. They all share one need.

1.6.2 An application without data does nothing

An online shop with an empty database has no customers to log in, no products to buy, no orders to total. A test that says "check the discount for a returning customer" needs a returning customer to exist. So before the test starts, the environment must already contain:

- customers with the right attributes (a returning customer, a new one, one with an expired card);
- products with prices;
- earlier orders for those customers, linked correctly to them.

That pre-loaded information is test data. It is as essential to a test as the code under test.

> **WHY THIS MATTERS**
>
> Bad test data wastes effort two ways. Tests that should pass fail because the data is wrong (a false alarm that costs a developer an afternoon). Or tests pass when they should fail because the data never contained the tricky case (a bug reaches customers). Good data decides whether a test result can be trusted.

1.6.3 What makes test data ”good”?

Testers usually want data that is:

Quality	Meaning	ShopSphere example
Realistic	Looks and behaves like real data, so the app reacts as it would in production.	Phone numbers have 10 digits; emails contain "@"; order totals match their line items.
Consistent	Related records agree with each other.	Every order points to a customer that exists.
Complete enough	Covers the cases the tests need, including odd ones.	A customer with zero orders; an order with 100 items; a customer born on 29 February.
Right-sized	Big enough to be meaningful, small enough to load and refresh quickly.	Thousands of customers for functional tests, millions only for performance tests.
Safe	Cannot hurt anyone if it leaks.	No real names with real ID numbers.
Available	There when the tester needs it, not three weeks later.	A fresh dataset the morning a release starts testing.
Repeatable	Can be put back to a known state so a test gives the same result twice.	Re-running yesterday's test on yesterday's data.

Notice that these seven qualities pull against each other. "Realistic" and "safe" fight: the most realistic data is real data, which is the least safe. "Complete" and "right-sized" fight too. TDM is the work of balancing them deliberately instead of by accident.

1.7 Test environments in plain words

An *environment* is a full copy of the application with its own servers and its own database. Teams run several, each with a job. Names vary between companies, but the pattern is common.

Environment	Who uses it	What it is for	Typical data
Dev (development)	Developers	Building and debugging new code. Breaks often, and that is fine.	Small, often hand-made; developers need to reset it freely.
QA / Test	QA engineers, automation	Systematic testing of features and bugs.	Broad and varied; needs edge cases.
UAT (user acceptance testing)	Business users	The business confirms the app does what they asked.	Must look believable to business people.
Staging / pre-prod	Release team, performance testers	A rehearsal as close to production as possible.	Large volumes; production-like shape.
Production	Real customers	The live business.	Real, sensitive, valuable.

> **SECURITY IS USUALLY WEAKER OUTSIDE PRODUCTION**
>
> The Informatica User Guide points out that organisations typically keep strict controls on production systems, while data security in non-production systems is often not as strong (User Guide 10.5.10, p.21). That gap is exactly where leaks happen.

1.7.1 Who needs data, and when?

In ShopSphere, one sprint can ask for data in several ways at once. A developer fixing a rounding bug needs one customer with a specific order total. A QA engineer testing refunds needs hundreds of orders in different states (paid, shipped, returned). A performance tester needs millions of orders. A UAT analyst needs data that "looks like our real customers". An automation engineer needs the same data every night.

Five people, five requirements. If nobody manages this, each one solves it alone, usually in the least safe way. That is the next section.

1.7.2 Types of testing and the data each one wants

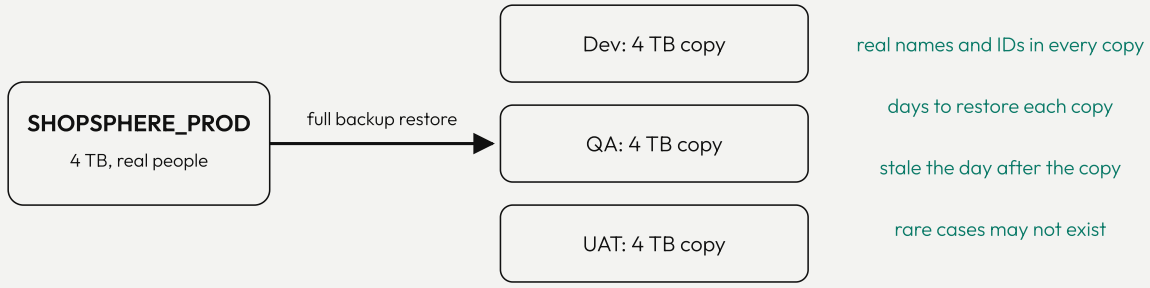
The word "testing" covers several activities. They need different data, which is why one dataset rarely suits everyone.

Kind of testing	What it checks	Data it wants
Unit testing	One small piece of code, by the developer.	A handful of hand-made values, often created inside the test and thrown away.
Integration testing	Whether separate parts (web app, database, payment service) work together.	Consistent data across all the systems involved, so the same customer exists everywhere.
System / functional testing	Whole features from the user's point of view.	A broad mix of customers, products and order states.
Regression testing	That old features still work after a change.	The same data every run, so results can be compared.
Performance / load testing	Speed and stability under heavy use.	Very large volumes with realistic shape.
User acceptance testing	Whether the business accepts it.	Believable data that business users recognise as "like ours".
Security testing	Whether the app resists attacks.	Data that includes hostile input such as unusual characters; it must not include real people's data.

1.8 The naive approach: copy production

The quickest way to get realistic, consistent data is to take a backup of the production database and restore it into the test environment. It is what many teams do first, because it works on day one and needs no special tool. This is the *production copy* approach.

It feels right: the data is as real as data can be, and every relationship between tables is already correct. The problems appear later, and they are serious. Here are eight, each shown with ShopSphere.



Original diagram: what "just copy production" produces. The 4 TB size is example data.

1.8.1 Problem 1: privacy and security leaks

Production holds real names, phone numbers, ID numbers and dates of birth. A full copy puts all of that where far more people have access: developers, testers, contractors, sometimes outside vendors, usually with weaker access controls and monitoring.

ShopSphere example. The row for customer 1001 (Asha Menon, her email, phone and SSN) is now in the dev, QA and UAT databases, in a test engineer's laptop dump and in a bug-report screenshot. Any one of those is a breach waiting to happen.

It is also often a legal problem. Laws and standards such as GDPR, HIPAA and PCI restrict how personal data can be used. The Delphix introduction names exactly these three (HIPAA, PCI and GDPR) as mandates organisations must stay compliant with, and notes that growing pressure from regulators and consumers makes protecting sensitive data across the enterprise essential (Introduction to Delphix Masking). the sensitive data and regulations chapter explains the regulations.

1.8.2 Problem 2: huge size

Production databases grow for years. A full copy needs the same disk space as production, multiplied by the number of environments, plus time to move and back up each copy.

ShopSphere example. If production is 4 TB (example figure), five test environments need 20 TB, and most of the content is orders from five years ago that no current test will ever read.

1.8.3 Problem 3: slow refreshes

Copying terabytes takes hours or days, so teams refresh rarely, queue for the operations team, and sometimes freeze the data for weeks.

ShopSphere example. QA breaks the data while testing refunds. Restoring it takes two days. Everyone waits, or carries on with broken data and gets false test results.

1.8.4 Problem 4: stale data

A copy is a snapshot of one moment. The day after, production has moved on, so tests of the newest code may run against data that does not contain it.

ShopSphere example. A "gift card" payment type launched last week. The copy taken last month has no orders that use it, so the gift-card tests find nothing to test.

1.8.5 Problem 5: missing edge cases

Real data contains only the cases that happened. Tests need values at the boundary: the largest allowed order, an empty field, a name with an apostrophe, a negative quantity that the app must reject.

ShopSphere example. Nobody in production has a 300-character last name, so no test checks that the screen handles one. The first customer who does crashes the checkout page.

This is where synthetic data comes in. GenRocket's help site defines synthetic test data as data manufactured by an algorithm rather than collected from real-world events, and says companies can maximise test coverage by using it to test all permutations and combinations (Why Synthetic Test Data?).

1.8.6 Problem 6: broken relationships when people shrink or edit it

Finding the copy too big, someone cuts it down by hand: "delete orders older than 2023". But orders are linked to customers and order items to orders. Delete the wrong rows and you leave order items pointing to orders that no longer exist, or orders for removed customers. The application then crashes or gives nonsense, for reasons unrelated to the code being tested.

ShopSphere example. A script deletes customers who have not logged in since 2022. It forgets their rows in `ORDERS`. Now there are orders with a `customer_id` that matches nobody. The "my orders" page fails for the wrong reason.

The same breakage happens if hand-made replacements differ between tables: the customer is "Asha" in one and "Priya" in another. The data basics chapter explains the keys that tie tables together. The Informatica documentation describes the goal directly: a data subset keeps primary and foreign key relationships so that the data stays referentially intact (User Guide 10.5.10, p.23).

1.8.7 Problem 7: cost

The costs are not only disk space. Add the time of people who run restores, the delay while testers wait, and the risk cost of a breach or regulator fine. None of it appears as a "test data" line on a budget, which is partly why the problem lingers.

1.8.8 Problem 8: no repeatability and no control

If every team makes its own copy at its own time, nobody knows which data version a bug was found on. Tests that passed Monday fail Tuesday because the data under them changed. Nobody can say who has a copy of what.

#	Problem	Symptom you would see	TDM capability that addresses it
1	Privacy leaks	Real names and IDs in dev databases	Discover and mask
2	Huge size	Disk full; slow restores	Subset (or virtual copies)
3	Slow refreshes	Waiting days for new data	Provision and refresh
4	Stale data	New features have no test data	Refresh; generate
5	Missing edge cases	Bugs only appear in production	Generate
6	Broken relationships	Orders with no customer	Subset and mask while keeping keys
7	Cost	Big storage and waiting bills	All of the above, plus governance
8	No repeatability or control	"It passed on my copy"	Govern: catalogue, versions, access

1.8.9 The same problems, as a vendor described them in 2020

In a 2020 webinar, GenRocket's CEO used an industry quality report to argue that test data lags behind test automation. He said roughly two thirds of organisations still use spreadsheets to build test data and over half still create it by hand. He added two customer stories: a large financial firm with a TDM tool could get the data a test case needed from its production copies only about a third of the time, and a large health insurer waited many days for data. These are a vendor's claims from 2020, not measured facts.

In the same webinar a testing-services partner listed the challenges its clients keep raising. Most match problems above, and one adds something new:

- **Business-process coverage.** Tests need negative cases and many permutations, not only the happy path (problem 5).
- **Speed.** Data must arrive as fast as the team deploys (problems 3 and 4).

- **Quality and security.** Good testing needs good data, and the data must be protected (problem 1).
- **Many formats and sources.** Data arrives in files, databases and other forms, and a tool must deliver to all of them. This is the one the eight problems do not cover.
- **Sharing and reuse.** Projects and environments want to share the same data (problem 8).

GenRocket's 2022 overview video presents customer results in the same spirit, for example a cycle time cut from 16 days to 2 hours. Those are again vendor-reported figures for specific customers.

> **"BUT WE ONLY USE IT INTERNALLY" IS NOT A DEFENCE**
>
> A breach through a test environment is still a breach of the people whose data it is. Internal staff, contractors and vendors can all misuse or lose data, and many regulations make no exception for non-production systems. The sensitive data and regulations chapter goes through this carefully.

> **WORKED EXAMPLE: ONE BUG REPORT, THREE FAILURES**
>
> A ShopSphere tester finds that the "order history" screen shows an empty page for customer 1002 in the QA environment. She files a bug. The developer cannot reproduce it on his machine because his data is different. After a day of investigation they discover that the QA database was a partial manual copy in which customer 1002's orders were not copied. The bug is not in the code at all.
>
> That one incident shows three problems from the list: broken relationships (6), no control (8) and slow refresh (3). It also cost two people a day. Multiply by a team and a year and the cost becomes visible.

1.9 So what is Test Data Management?

Test Data Management (TDM) is the set of processes, roles and tools that plan, create, protect, deliver and retire the data used in non-production environments. In simpler words: it is how you make sure the right testers get the right safe data at the right time, without copying production carelessly.

Some people use "TDM" for the software and some for the practice. Here "TDM" is the practice, and "a TDM tool" is software that supports it.

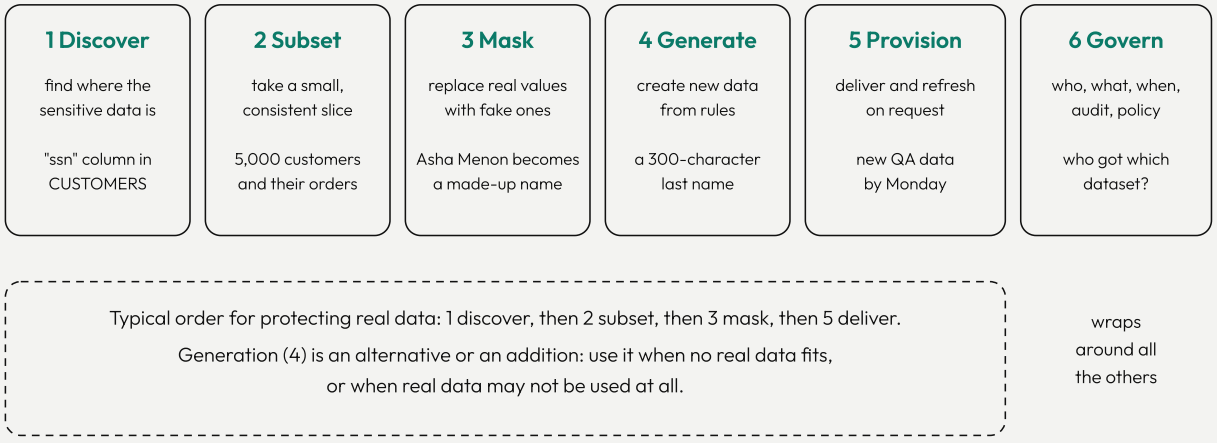
1.9.1 Goals of TDM

1. **Protect people.** No real sensitive data in non-production.

2. **Fit the test.** Data that is realistic, consistent and covers the cases the tests need.
3. **Right size.** Smaller than production where possible, large only where tests need it.
4. **Be fast.** Data on request, refreshed in minutes or hours, not weeks.
5. **Be repeatable and traceable.** Known data versions, known owners, known copies.
6. **Lower cost.** Less storage, less waiting, less rework, lower breach risk.

1.10 The six capabilities of TDM

TDM splits into six capabilities. A project may use all six or only a few. The first four change the data; the last two deliver and control it.



Original diagram: the six capabilities of TDM, each with a ShopSphere example. This is a teaching model, not a product architecture.

1.10.1 1. Discover

You cannot protect what you cannot find. **Discovery** means scanning a database or a set of files to locate sensitive data: which columns hold names, ID numbers, phone numbers. Large databases can have thousands of columns, and sensitive values hide in odd places such as a "notes" column. Discovery also finds the *relationships* between tables, which later steps need.

Why it matters: if discovery misses a column, that column goes into test unmasked.

ShopSphere: discovery flags `ssn`, `phone`, `email`, `date_of_birth` and `tax_id` in `CUSTOMERS`. The discovery and subsetting concepts chapter teaches this in depth.

1.10.2 2. Subset

Subsetting builds a smaller database from a big one: for example "only customers from the last two years, and everything linked to them". Done properly, every order in the subset still has its customer and its items. Smaller means cheaper, faster to load and easier to refresh.

ShopSphere: pick 5,000 customers and bring their orders and items along, leaving the other millions of rows behind. The discovery and subsetting concepts chapter covers it; the Informatica data subset chapter shows it in Informatica.

1.10.3 3. Mask

Masking replaces sensitive values with realistic but fake ones. The application still works because the fake values look right (a fake phone has ten digits), but nobody can learn about a real person from them. Good masking is consistent: the same real value always becomes the same fake value, so relationships between tables survive.

The Delphix documentation describes masking as replacing confidential information such as social security numbers, patient records and card numbers with fictitious yet realistic data, and contrasts it with encryption: masking is irreversible, while encryption can be bypassed through schemes to obtain user credentials (Introduction to Delphix Masking).

ShopSphere: Asha Menon, 9876543210, 123-45-6789 becomes, say, a made-up name, a made-up ten-digit number and a made-up SSN pattern. The masking concepts chapter teaches the techniques.

> **BEFORE AND AFTER: WHAT MASKED DATA LOOKS LIKE**
>
> The first table is the original data; the second shows what a masked copy might look like (example values).

CUSTOMERS BEFORE MASKING (SHOPSPHERE FAKE ROWS)

customer_id	first_name	last_name	phone	ssn
1001	Asha	Menon	9876543210	123-45-6789
1002	Ravi	Kumar	9123456780	234-56-7890

CUSTOMERS AFTER MASKING (EXAMPLE DATA)

customer_id	first_name	last_name	phone	ssn
1001	Lakshmi	Pillai	9404418273	581-20-3347
1002	Arjun	Das	9051627384	672-31-4458

Notice what changed and what did not. Names, phones and SSNs are different, but they keep their shape: ten-digit phones, SSNs in the nnn-nn-nnnn pattern. The `customer_id` stayed the same, so every order that points to customer 1001 still finds its customer. If the same real SSN appeared in another table, it would have to receive the same fake value there too. That consistency requirement is the heart of the masking concepts chapter.

1.10.4 4. Generate

Generation creates data from rules instead of copying real data. You describe what you need ("fifty customers, each with a first name, a last name, a phone number") and a tool produces it. Use it when real data does not exist yet (a new feature), when you need edge cases, when you need very large volumes, or when real data cannot be used at all.

Informatica's documentation gives the second reason almost word for word: use data generation when you need test data but do not want to use production data, for example when test teams handle sensitive data and have no access to production (User Guide 10.5.10, p.23).

ShopSphere: generate a customer whose name is 300 characters long. The classification and generation concepts chapter covers the idea; Part III is GenRocket, a tool built around it.

1.10.5 5. Provision and refresh

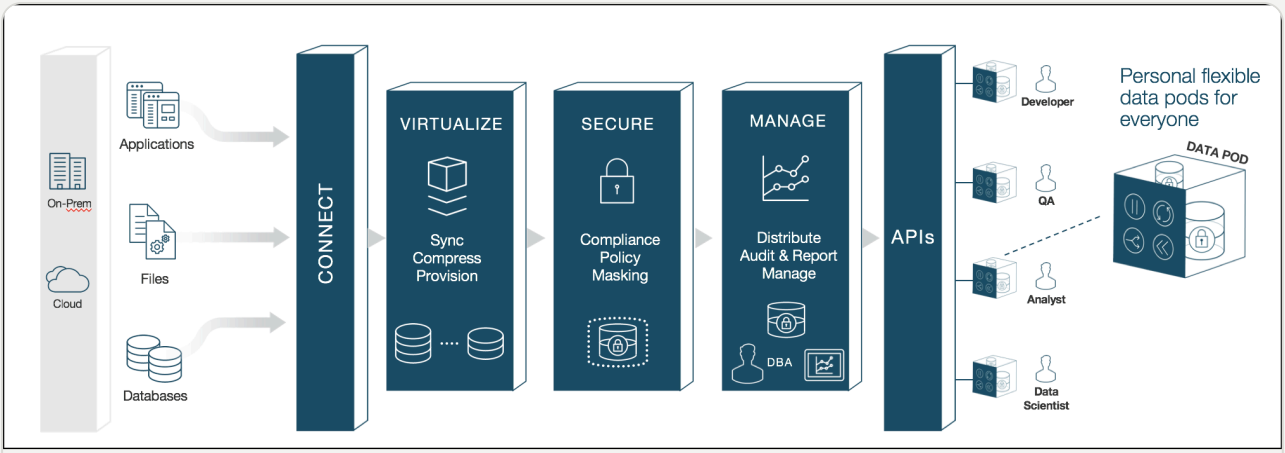
Provisioning means delivering a ready dataset into an environment: loading it into the QA database, or placing a file where a test tool expects it. **Refreshing** means doing it again later to get new or reset data. Speed and repeatability matter most here.

Both Informatica and Delphix address this in their own way. Informatica describes a test data warehouse that stores multiple versions of test data and can restore a version to put an environment back into a particular state (User Guide 10.5.10, p.23). Delphix describes virtual copies that are readable and writable, can be created or removed in minutes, and take a fraction of the storage of physical copies; in its example ten virtual copies fit in the space of one physical copy (Introduction to Delphix Masking).

1.10.6 6. Govern

Governance is the rules and records around the other five: who may request what, which data is approved for which environment, which masking rules are in force, an audit trail of who received which dataset. It is the least glamorous capability and the first one auditors ask about.

Delphix says the platform lets operators audit, monitor and report against access and usage, and generates reports and email notifications confirming sensitive data has been masked. Informatica's administrator manages roles and privileges that decide what each TDM user can do (User Guide 10.5.10, p.23).



Official screenshot: Delphix's own platform diagram. Sources (applications, files, databases, on-premises or cloud) enter through "Connect", then pass through Virtualize (sync, compress, provision), Secure (compliance, policy, masking) and Manage (distribute, audit and report), and leave through APIs as "data pods" for developers, QA, analysts and data scientists. It is a vendor view of capabilities 3, 5 and 6 above.

`Source page ↗`

> **WHY THE SIX ARE SEPARATE IDEAS**
>
> Beginners often use "masking" to mean everything. It is only one of six. A team that masks but never subsets still has a huge, slow copy. A team that subsets but never discovers may mask the wrong columns. Learn the six as separate tools in a toolbox, then pick the ones a project needs.

1.11 What the three tools say about their own use cases

The idea of TDM is tool-independent. Here is how each vendor's documentation frames its product, so you recognise the vocabulary later.

1.11.1 Informatica TDM

The User Guide lists use cases under two headings, performance and security/compliance, then a third about verified data (pp.23-24).

Use case (User Guide wording, paraphrased)	Capability from this section
Create subsets of production data that keep primary and foreign key relationships so the data stays referentially intact; manageable and scalable test data (p.23).	Subset
Generate data from rules when you need test data but do not want to use production data (p.23).	Generate
Store multiple versions of test data in one shared test data warehouse; restore a version to return the test environment to a particular state; edit data and apply row-level tags (p.23).	Provision, refresh, govern
Copy flat-file test data directly to an integrated test tool to run test cases (p.23).	Provision
Run profiles to find sensitive columns and compare them against compliance standards such as PII, PCI or PHI (p.24).	Discover
Run data masking with standard formats such as credit card or Social Security numbers, or custom formats (p.24).	Mask
Use TDM with Data Validation Option to give test data with verifiable proof that it contains no sensitive values (p.24).	Govern

1.11.2 Delphix

The Delphix introduction frames a challenge (breaches, regulator pressure, and encryption being a poor fit for non-production data) and a solution: automated masking that replaces sensitive values with realistic fictitious ones, delivered with virtual data copies. It lists four services: Virtualize, Identify and secure, Manage and Self-service. In the self-service service, users such as developers and testers can refresh data to the latest state of production, rewind to an earlier point in time, bookmark, branch or share data copies (source). The terminology page defines the masking part as

the Continuous Compliance product, sometimes called a Compliance Engine, which discovers sensitive data and replaces it with realistic but fictitious data (source).

1.11.3 GenRocket

The GenRocket help site opens with the idea of synthetic test data, data manufactured by an algorithm rather than collected from real events, and says it lets companies test all permutations and combinations (Why Synthetic Test Data?). It states that GenRocket follows a "Model, Design, Deploy, Manage" methodology (The GenRocket Methodology). It also notes that the Test Data Workflow starts with the Power User, a Test Data Engineer (TDE) (Test Data Workflow).

The details (domains, generators, receivers, scenarios) arrive in the GenRocket overview chapter.

1.12 Putting numbers on the cost of copying

Lab: price the cost of ”just copy it” (thinking lab)

Goal: see in numbers why unmanaged test data is expensive. The figures below are examples; replace them with a real project's.

1. **Storage.**
   Production is 4 TB and ShopSphere has five non-production environments, each holding a full copy. Compute the total storage for the copies.

   Expected result: An answer in TB.

2. **Waiting.**
   A restore takes 2 days. QA needs one restore per fortnight and the team of 6 testers is idle half of the restore time. Compute idle tester-days per year.

   Expected result: An answer in tester-days.

3. **Subset effect.**
   A subset keeps 5 percent of the data. Recompute step 1.

   Expected result: A much smaller number.

▶ **Show a worked answer**

1.13 Roles in a TDM effort

TDM is a team activity. Titles vary by company, and one person may wear several hats, but the responsibilities are consistent.

Role	What they do in TDM	ShopSphere example
QA engineer / tester	Defines what data the tests need and consumes it. Reports when data is wrong.	"I need 200 orders in a 'returned' state."
Developer	Needs small, resettable data to debug and build features.	"I need one customer with an order total of exactly 0.01."
TDM engineer	Builds and runs the process: discovers sensitive data, designs subsets, configures masking and generation, schedules refreshes, fixes failures. This is your target role.	Sets up the masking job that turns `SHOPSPHERE_PROD` into safe QA data every week.
Data steward / data owner	Knows what each data item means, which is sensitive, how it may be used; approves rules.	Confirms that `tax_id` is sensitive and must be masked.
Compliance / privacy officer	Ensures practice meets laws and internal policy; reviews evidence.	Asks: "Prove no real SSN exists in UAT."
Database administrator (DBA)	Gives access to source and target databases, helps with performance and storage.	Creates the account the TDM tool uses to read production.
Tool administrator	Installs, upgrades and manages the TDM product and its users.	Sets which users may run masking jobs.
Release / environment manager	Plans which environment gets which data and when.	Schedules the Monday refresh of QA.

The tool documentation uses role names too: GenRocket's workflow begins with the Power User, a Test Data Engineer (TDE); Delphix has separate interfaces for administrators and for project

teams; Informatica's administrator manages user roles and privileges (User Guide 10.5.10, p.23).

1.14 Real-world scenarios at ShopSphere

Three short stories, each choosing a different mix of the six capabilities.

SCENARIO 1: WEEKLY QA REFRESH

Situation. QA tests every Monday and needs realistic data, but production holds real customers.

Approach. Each Friday night a job picks the last 12 months of customers and their orders (*subset*), replaces names, emails, phones, SSNs and dates of birth with fake but consistent values (*mask*), and loads the result into QA (*provision*). Discovery was done once and is re-checked when the schema changes.

Outcome. QA gets fresh, safe, small data every Monday.

SCENARIO 2: A NEW LOYALTY-POINTS FEATURE

Situation. ShopSphere adds loyalty points. No production data has points yet.

Approach. Nothing to copy, so the team *generates* 10,000 customers with random point balances, including zero, the maximum and a negative balance that the app must reject.

Outcome. The feature is tested before launch, including the edge cases, with no personal data involved.

SCENARIO 3: PERFORMANCE TEST BEFORE THE HOLIDAY SALE

Situation. The company expects ten times normal traffic and needs 50 million orders in the performance environment.

Approach. Production does not have that many. The TDM engineer *generates* the volume and *masks* a real sample so that distributions (items per order, typical prices) match the real shape.

Lab: choose the capabilities (thinking lab, no tool needed)

Goal: practise mapping a testing problem to the six capabilities.

For each situation, write down which of discover, subset, mask, generate, provision and govern you would use, and in what order.

1. **The UAT team complains the database is too slow to refresh and takes 6 TB.**
   Think about size and speed.

2. **A developer wants to test what happens when a customer's email is blank.**
   Think about whether such a record exists in production.

3. **ShopSphere acquires another retailer whose database has unfamiliar columns and nobody knows what is sensitive.**
   Think about what must come first.

▶ **Show a worked answer**

1.15 Benefits and trade-offs

Benefits

- Lower privacy and compliance risk.
- Faster test cycles because data is ready.
- Smaller storage and infrastructure cost.
- Better test coverage through generated edge cases.
- Repeatable tests on known data.
- Fewer false bug reports from bad data.

Trade-offs and costs

- Set-up effort: discovery, rules and approvals take time up front.
- Licence and infrastructure cost for commercial tools.
- Masked data is never exactly real data; some production-only bugs may not reproduce.
- Generated data is only as good as its rules; it can miss real-world quirks you did not think of.
- New skills and a new process for the team to learn.
- Ongoing upkeep as the schema and the regulations change.

MASKED AND GENERATED DATA COMPARED

Masking starts from real data, so it keeps real-world shape and messiness. Generation starts from rules, so it is safe by construction but covers only the cases you thought of. Many teams use both. the classification and generation concepts chapter compares them in detail.

1.16 TDM maturity: from ad hoc to self-service

Organisations do not jump straight to full TDM. A common way to describe the journey is a ladder of stages. This is a teaching model, not an official standard; names and numbers of levels vary between authors.

Stage	How data is obtained	Typical pain
1. Ad hoc	Each team copies production or types data by hand, whenever it can.	Privacy risk, inconsistent data, long waits.
2. Scripted	Someone writes scripts to subset or mask. Works, but depends on a few people.	Scripts break when the schema changes; knowledge sits in heads.
3. Managed	A TDM team uses a tool with defined rules: discovery, masking, subsetting, scheduled refreshes, approvals.	Requests still go through the team, which becomes a bottleneck.
4. Self-service and automated	Testers request or refresh data themselves through a portal or API, often inside the build pipeline; governance is automatic.	Needs investment and discipline to keep rules current.

The product documentation points toward the top of the ladder. Informatica mentions a self-service portal for users who use test data warehouse data but do not create it (User Guide 10.5.10, p.22). Delphix describes a self-service service where testers can refresh, rewind, bookmark and share copies. Both are vendor views of what later stages look like, not proof of how any particular organisation works.

WHY MATURITY MATTERS TO YOU

When you join a project, find out which stage it is in. A stage 1 team needs convincing and a small pilot; a stage 3 team needs a good TDM engineer to keep the machine running. Planning and assessing a line of business (LOB) is about exactly this (the planning real engagements chapter).

1.17 The commercial TDM product landscape

1.17.1 What "COTS" means

COTS stands for commercial off-the-shelf. It means software that a vendor builds once and sells to many customers, as opposed to custom software written for one company. You buy a licence, install it, and configure it with your own rules.

In TDM there are three broad ways to get the capabilities from section 1.10:

Approach	Strengths	Weaknesses
Home-grown scripts	Cheap to start; tailored to your database.	Hard to maintain; usually weak on discovery, governance and audit evidence; depends on key people.
Open-source tools	No licence fee; community knowledge.	You assemble and support it yourself; coverage varies.
COTS products	Many capabilities in one supported product; connectors to many databases; auditing and role-based access; vendor support.	Licence cost; training; you follow the vendor's way of working; version upgrades.

The comparison is general; real products vary, so evaluate with a proof of concept (a small trial on your own data). The planning real engagements chapter shows how.

1.17.2 The three products covered here

This guide covers three COTS products: Informatica TDM (one web client for discovery, subset, masking and generation), GenRocket (synthetic data built from rules) and Delphix (profiling, masking and virtual copies). Other TDM products exist and are not covered.

1.18 Common mistakes and gotchas

"WE MASK IT, SO IT IS SAFE"

Masking only protects the columns you masked. If discovery missed a column, or a free-text "notes" field contains names, sensitive data survives. Discovery and verification matter as much as the masking itself.

TREATING TDM AS A ONE-OFF PROJECT

Production changes every day: new columns, new data, new rules. A TDM set-up that is never maintained goes stale just like a copied database.

USING ONE TECHNIQUE FOR EVERYTHING

Masking is not always right. If no real data exists for the case, or real data cannot be used at all, generate. If the problem is only size or speed, subset or virtualise.

IGNORING RELATIONSHIPS

Any change to the data (subset, mask, generate) must keep related tables consistent. The data basics chapter explains how; skipping it leads to the "orders with no customer" bug.

BELIEVING TOOL NAMES ARE THE CONCEPTS

"Plan", "rule set", "scenario" and "domain" mean different things in different products. Learn the concept, then learn each tool's word for it.

1.19 Recap

RECAP

- Testing needs data; good test data is realistic, consistent, complete, right-sized, safe, available and repeatable.
- Copying production gives real data but causes privacy leaks, huge size, slow refreshes, stale data, missing edge cases, broken relationships, cost and loss of control.
- TDM is the set of processes, roles and tools that deliver safe, right-sized, realistic test data on demand.
- Its six capabilities are discover, subset, mask, generate, provision and govern; most projects use several.
- Roles include testers, developers, the TDM engineer, data steward, compliance officer, DBA and administrators.
- Maturity runs from ad hoc, through scripted and managed, to self-service and automated.
- COTS means commercial off-the-shelf; the three tools covered are Informatica TDM, GenRocket and Delphix.

Data basics every TDM beginner needs

Everything a TDM tool does is done to data: it finds data, cuts it down, hides parts of it, makes more of it, and moves it. That takes a small, firm vocabulary about how data is organised: tables, keys, files and basic SQL.

These ideas return constantly. Subsetting, masking and generation all have one rule in common: **keep the relationships between records intact**. You cannot follow that rule if you do not know what a relationship is.

1.20 Words to know first

Data

Raw facts, such as "1001", "Asha" and "560001", before anyone says what they mean.

Information

Data with meaning: "customer 1001 is Asha, living in postal area 560001".

Database

An organised store of data that a program can search and change reliably.

Table

A grid of data about one kind of thing, like one sheet in a spreadsheet.

Row / record

One item in a table: one customer, one order.

Column / field

One attribute of every row: first name, phone, order date.

Primary key (PK)

A column (or columns) whose value is unique for every row and identifies it.

Foreign key (FK)

A column whose values point to the primary key of another table.

Referential integrity

The rule that every foreign key value must match an existing primary key value.

Schema / metadata

The description of the data (table names, columns, types, keys) as opposed to the data itself.

Flat file

A plain file holding records as lines of text, with no built-in database engine behind it.

SQL

The language used to ask a relational database questions and to change it.

1.21 Data and information

Look at this: `1001 | Asha | Menon | 560001`. On its own it is just data: four values. You can guess it is a person, but you are guessing. Add labels (*customer_id, first_name, last_name, zip_code*) and a table name (*CUSTOMERS*) and it becomes information: customer 1001 is Asha Menon in postal area 560001.

That distinction matters for TDM because tools often see only data and have to work out the meaning. Discovery (see the discovery and subsetting concepts chapter) is largely the job of recovering meaning: "this column, called `col_17`, contains phone numbers". Column names, data types and descriptions are the clues.

1.22 Databases and tables

1.22.1 The spreadsheet analogy

If you have used a spreadsheet you already know the idea. A *table* is one sheet: columns across the top with names, rows below, one item per row. A *database* is the whole workbook, with several sheets (tables) that belong together.

Here is the ShopSphere `CUSTOMERS` table:

`CUSTOMERS (FAKE SAMPLE ROWS, FIRST FIVE COLUMNS)`

customer_id	first_name	last_name	email	phone
1001	Asha	Menon	asha.menon@example.com	9876543210
1002	Ravi	Kumar	ravi.k@example.com	9123456780
1003	Meera	Nair	meera.n@example.com	9988776655

- Each **row** (also called a *record*) is one customer.
- Each **column** (also called a *field* or attribute) is one fact about every customer.
- Each **cell** is one value: the phone number of customer 1002.

Unlike a spreadsheet sheet, a database table enforces each column's declared type, can be linked to other tables by keys, can hold millions of rows, and is queried with SQL (later in this section). ShopSphere uses a *relational* database (Oracle, SQL Server, MySQL, PostgreSQL and DB2 are examples): tables relate through keys.

1.22.2 Data types

Each column has a *data type* that says what kind of value it holds. The type controls what is allowed, how much space it takes and how it can be used (you can add numbers but not names).

Type family	Typical names	ShopSphere example	Why a TDM person cares
Whole numbers	INTEGER, INT, NUMBER	`customer_id`, `quantity`	Generated or masked values must still be whole numbers.
Decimal numbers	DECIMAL(10, 2), NUMERIC	`unit_price`, `total_amount`	Precision matters: 10,2 means up to 10 digits, 2 after the point.
Text	VARCHAR(50), CHAR(10)	`first_name`, `email`	A masked name must fit the length limit.
Dates and times	DATE, TIMESTAMP	`date_of_birth`, `order_date`	A masked date of birth must still be a real date.
True/false	BOOLEAN, BIT	(none in ShopSphere)	Only two values exist.
Binary / large objects	BLOB, CLOB, images	(none in ShopSphere)	Special handling; many masking techniques do not apply.

Notice something about the ShopSphere sample data: `phone`, `ssn` and `zip_code` look like numbers, but they are stored as **text**. You never add two phone numbers, and a leading zero would be lost if they were numbers. That choice affects how you mask them: a text column can hold dashes (123-45-6789); a numeric column cannot.

WHY THIS MATTERS

A TDM operation must produce values the column accepts. A masked or generated name longer than the column's limit makes the load fail. A fake date such as 31 February is rejected. Many job failures in the tool chapters come down to "the new value did not fit the column's type or length".

1.22.3 Schema and metadata

Metadata means "data about data". For a database it is the list of tables, their columns, each column's name and type, which columns are keys, and which tables relate to which. The collection of all that is the *schema*.

Here is part of the ShopSphere schema written as a simple description (not the data):

```
CUSTOMERS
  customer_id    INTEGER       PRIMARY KEY
  first_name     VARCHAR(50)
  last_name      VARCHAR(50)
  email          VARCHAR(100)
  phone          VARCHAR(15)
  ssn            VARCHAR(11)
  date_of_birth  DATE
  zip_code       VARCHAR(10)
  tax_id         VARCHAR(20)
```

The metadata tells a tool a lot without reading a single row. A column called `ssn` of type `VARCHAR(11)` is a strong hint (11 characters fits 123-45-6789). TDM tools therefore read metadata first, then sample the actual data. The Informatica guide describes a TDM repository that stores the metadata you import into Test Data Manager from a source database (User Guide 10.5.10, p.26). The Delphix introduction describes two profiling methods, one scanning column names (metadata) and one scanning the data itself (Introduction to Delphix Masking).

1.23 Keys and relationships

A single table is easy. Real applications split data across many tables and connect them. Keys are the connectors. This section is the most important in the whole chapter.

1.23.1 Why data is split into several tables

Imagine ShopSphere stored everything in one big sheet: one row per ordered item, with the customer's name, phone and address repeated on every row. Customer Asha has two orders with three items in total, so her phone number would be typed three times. If she changes her phone number you must update every copy, and if you miss one the data disagrees with itself.

The fix is to store each fact once, in the table where it belongs, and refer to it from elsewhere. Customer facts live in `CUSTOMERS`. An order stores only the customer's *ID*, not the name and phone. To get the name, you look the ID up. The ID is a key.

1.23.2 Primary key

A *primary key* is a column (or a small group of columns) whose value is **unique** in every row and is **never empty** (null). It is the row's identity, like a library card number or a passport number.

- `CUSTOMERS.customer_id`: 1001, 1002, 1003. No two customers share an ID.
- `ORDERS.order_id`, `ORDER_ITEMS.item_id` and `PRODUCTS.product_code` are the primary keys of their tables.

A database enforces this: it refuses to insert a second customer with ID 1001.

Why not use the name as the key? Two customers can share a name; names change; spellings vary. A key must be stable and unique. That is why tables carry an artificial ID such as `customer_id`. (Keys made up of real-world data, like an email address, are called *natural keys*; made-up IDs are *surrogate keys*. Both appear in the real world.)

A key can have more than one column. Suppose a table `ORDER_ITEM_NOTES` allowed one note per item per language; then `(item_id, language)` together would be unique, though neither alone is. That is a *composite key*.

1.23.3 Foreign key

A *foreign key* is a column in one table whose values must match the primary key of *another* table. It is the link.

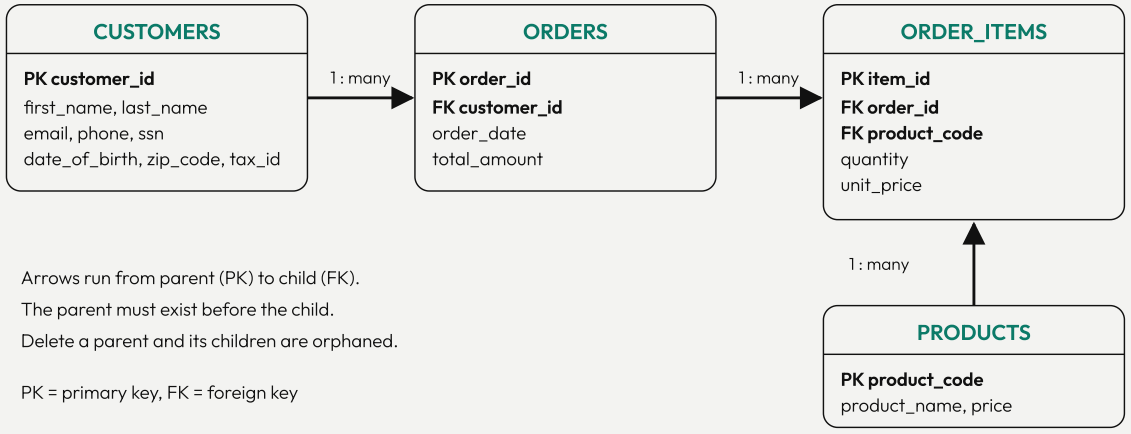
- `ORDERS.customer_id` is a foreign key to `CUSTOMERS.customer_id`: it says which customer placed the order.
- `ORDER_ITEMS.order_id` is a foreign key to `ORDERS.order_id`: it says which order this line belongs to.

- `ORDER_ITEMS.product_code` is a foreign key to `PRODUCTS.product_code`: it says which product was bought.

The foreign key column is allowed to repeat. Customer 1001 can appear in many order rows, because Asha can place many orders. The primary key side cannot repeat.

1.23.4 Parent and child tables, one-to-many

The table that holds the primary key is the *parent*; the table with the foreign key pointing at it is the *child*. One customer (parent) can have many orders (children), and one order can have many items. That shape is called *one-to-many*.



Original diagram: ShopSphere's four tables and the keys that connect them. Compare with the table definitions in the start-here chapter.

1.23.5 The linked sample data

Here are the other three ShopSphere tables with fake rows, linked to the customers. Trace the keys with your finger: order 5001 belongs to customer 1001, and its items 9001 and 9002 point to products P100 and P200.

`ORDERS`

order_id (PK)	customer_id (FK)	order_date	total_amount
5001	1001	2026-01-15	1098.00
5002	1001	2026-02-03	1499.00
5003	1002	2026-02-10	598.00

ORDER_ITEMS

item_id (PK)	order_id (FK)	product_code (FK)	quantity	unit_price
9001	5001	P100	1	799.00
9002	5001	P200	1	299.00
9003	5002	P300	1	1499.00
9004	5003	P200	2	299.00

PRODUCTS

product_code (PK)	product_name	price
P100	Wireless Mouse	799.00
P200	USB-C Cable	299.00
P300	Laptop Stand	1499.00

Customer 1003 (Meera) has no orders: a parent with no children is perfectly legal. The reverse is not.

1.23.6 Referential integrity

Referential integrity is the rule that every foreign key value must match a primary key value that exists. Every order must belong to a customer who exists; every item to an order that exists.

A child row whose parent is missing is called an *orphan*. Suppose someone deletes customer 1001 but leaves orders 5001 and 5002. Those orders now point to nobody. The "my orders" page crashes. When the database has the foreign key declared, it refuses such a delete, and it refuses to insert an order for a customer ID that does not exist. (In SQLite with foreign key enforcement switched on, both attempts are rejected with a foreign key constraint error.)

> **WHY KEYS MATTER SO MUCH TO TDM**
>
> Every TDM operation reshapes data. Each one can break referential integrity if done carelessly.

Operation	How it can break keys	What it must do instead
Subsetting	Pick 100 customers but leave out their orders (missing children), or pick orders but leave out their customers (orphans).	Follow the keys: choose a starting table, then bring along every related row in the other tables.
Masking	Mask `customer_id` in `CUSTOMERS` to a new value but leave `ORDERS.customer_id` unchanged, or mask them differently. Orders no longer match customers.	Apply the same masking to the parent key and every child foreign key so the values still match, or leave keys alone.
Generation	Generate 1,000 orders with customer IDs that no generated customer has.	Generate parents first and give each child a valid parent ID.
Provisioning	Load child tables before parent tables so foreign key checks fail.	Load parents first; delete children first.

Each tool has its own way to say this. The Informatica documentation says a data subset retains primary and foreign key relations so that the data stays referentially intact (User Guide 10.5.10, p.23). The Delphix introduction says it scans metadata and data to identify and preserve primary/foreign key relationships so that data is masked the same way across tables and databases (Introduction to Delphix Masking).

1.23.7 Logical keys: keys the database does not know about

Everything above assumed the keys are *declared* in the database. In real life they often are not. Older systems, performance-minded designs and imported data frequently have tables with no declared primary or foreign keys at all, even though the application treats certain columns as keys. The relationships exist in the developers' heads and in the code, not in the schema.

TDM tools cannot follow a relationship they do not know about. So they let you (or their profilers) describe the relationships separately. Keys defined this way are called *logical keys*: the tool is told "treat this column as the key" without any change to the real database.

Tool	What its documentation says
Informatica	Primary-key and foreign-key profiles can discover potential relationships (p.21). The foreign key profile adds the discovered logical constraints to the TDM repository but does not apply them in the source tables; the subset workflow uses them to produce referentially intact data (p.86). You can also add a logical primary key to a table (p.109) and a logical foreign key choosing a column from a parent table (p.110). In its worked example the database contains no primary or foreign keys, so the developer creates logical ones (p.34). For data sets it advises creating logical keys in transactional tables that have no primary keys (p.239).
Delphix	In a rule set, a logical key is a unique, non-null value identifying a row. If a table has no primary key defined and you use in-place masking, you must name an existing column or columns as the logical key. It does not change the target database; it only informs Delphix. Without one, an identity column may be created, which requires permission to modify the schema or the job may fail. The rule set "Checklist" validates that every table has a primary key, an indexed identity column or a logical key (Managing rule sets).

The idea is the same in both: **a key you declare to the tool, which the database may know nothing about**. Chapters on Informatica subsetting and Delphix rule sets and keys show the screens.

> **LOGICAL KEYS ARE ONLY AS GOOD AS YOUR UNDERSTANDING**
>
> If you declare a column as a key but it is not actually unique, masking or subsetting will misbehave. A logical key must be unique and never null for every row, exactly like a real primary key. You prove it by checking the data (for example by counting rows and counting distinct values), not by assuming.

1.24 Flat files

Not all test data lives in databases. Banks, insurers and retailers exchange huge amounts of data as *files*: one system writes a file overnight, another reads it in the morning. Masking text and .dat files is a common task.

A *flat file* is a file holding records as lines of text. "Flat" means there is no table engine, no keys, no data types: it is just text in a layout that both sides agree on. All the structure lives in

the agreement, not in the file.

1.24.1 .txt, .dat, .csv: what the extension tells you (little)

A file extension is only a label. `.txt` says "text", `.csv` says "comma-separated values", `.dat` says "data". None of them tells you how the records are laid out. A `.dat` file can be comma-separated, pipe-separated, fixed-width or something else; a `.txt` file can hold any of those too. You find the layout from the **file specification** (a document from the team that produces the file), or by opening the file and looking.

> **WHY THIS MATTERS**
>
> A tool that masks files has to be told the layout, because the file itself does not say. The Delphix documentation puts it directly: unlike databases, files typically have no built-in metadata describing the format of their fields, so you must provide that information (Managing file formats).

1.24.2 Records, fields, headers and footers

- A *record* is one line (one customer, one order), ending with an *end-of-record* marker, usually a line break.
- A *field* is one value within the record.
- A *header* is a record at the top that is not data: often the column names, or a label with a date.
- A *footer* (or trailer) is a record at the end, often a count or a total so the reader can check nothing was lost.

1.24.3 Delimited files

In a *delimited* file a special character, the *delimiter*, separates the fields. The most famous is the comma, giving the CSV (comma-separated values) format. Pipes (|), tabs and semicolons are also common, especially when data itself contains commas.

Here is a ShopSphere customer file in CSV with a header row. All values are fake:

```
customer_id,first_name,last_name,email,phone
1001,Asha,Menon,asha.menon@example.com,9876543210
1002,Ravi,Kumar,ravi.k@example.com,9123456780
1003,Meera,Nair,meera.n@example.com,9988776655
```

And an orders file, `.dat` extension, pipe-delimited, with a header record (H), data records (D) and a footer record (T) that carries the record count:

```
H|ORDERS|20260215
D|5001|1001|2026-01-15|1098.00
D|5002|1001|2026-02-03|1499.00
D|5003|1002|2026-02-10|598.00
T|3
```

Reading it: the first field of each line says what kind of record it is. This is a file with *several record types* in one file, a layout common in business exchanges. The footer says "3 data records", so a reader can detect a truncated file.

What if the data contains the delimiter? If a last name were "O'Neil, Jr." in a comma file the comma would split it into two fields. The usual answer is an *enclosure* (also called a qualifier or quote character), commonly the double quote: `1004,Liam,"O'Neil, Jr.",liam@example.com,9000000001`. Then a literal quote inside the field needs an *escape*, often by doubling it. The Delphix rule-set page documents settings for exactly this: an enclosure escaping strategy for delimited files, with a default of "Double enclosure" (the escape character is the same as the enclosure), and an option to use a custom single character (Managing rule sets).

1.24.4 Fixed-width files

In a *fixed-width* file there is no delimiter. Each field occupies an exact range of character positions, padded with spaces. The layout is defined by position:

LAYOUT SPECIFICATION FOR CUSTOMERS.TXT (FIXED-WIDTH)

Field	Start	Length	Positions
customer_id	1	4	1-4
first_name	5	10	5-14
last_name	15	10	15-24
phone	25	10	25-34

The data, with a ruler line on top so you can count columns (the ruler is not part of the file):

```
12345678901234567890123456789012345678901234
1001Asha      Menon     9876543210
1002Ravi      Kumar     9123456780
1003Meera     Nair      9988776655
```

Every record is exactly 34 characters long. "Asha" is followed by six spaces so that the last name starts at position 15. Without the layout table you could not tell where "Asha" ends and "Menon" begins.

Delimited

- Fields found by a separator character.
- Field length can vary.
- Needs rules for delimiters inside values.
- Easy to open in a spreadsheet.

Fixed-width

- Fields found by position.
- Every field has a set length; short values are padded.
- No delimiter issues, but a masked value must not exceed the field width.
- Needs the layout specification to read at all.

> **FIXED-WIDTH AND MASKING**
>
> If you mask a name in a fixed-width file and the new name is longer than the field, it either overflows into the next field (corrupting the record) or is cut off. Masked values must fit the field. More detail for Delphix is in the Delphix file masking chapter.

1.24.5 Line endings

The end of a record is marked by invisible characters that differ by platform. Windows traditionally uses two characters (carriage return then line feed, often written CR+LF); Linux uses one (line feed, LF). If a tool assumes the wrong one, records are read wrongly or every line gets an odd extra character. Delphix asks you to choose the end-of-record, Windows/DOS (CR+LF) or Linux (LF), when you assign a file format, and to enter the delimiter for a delimited file (Managing file formats). It also supports using a control

character as the delimiter or end of record, entered in a particular format (Managing rule sets).

1.24.6 Records of different shapes in one file

The orders file above mixed header, data and footer lines. Other files mix different *kinds* of data record. The Delphix documentation gives an example where some records carry an account number and a postal address while others carry an account number and an email address, told apart by the first field ("address" or "email"). It calls the differentiating field a record type qualifier, supports up to three qualifiers, and says a record gets exactly one record type. It also says that header and footer records can be configured so unmodified copies go into the masked output, and that these features apply only to delimited and fixed-width formats (Managing record types and header/footer records).

The point for you: a file is rarely "just rows". Before touching it, list the record types it contains. Masking a header line as if it were a customer would corrupt it.

1.24.7 Encodings: how text becomes bytes

Computers store characters as numbers; an *encoding* is the agreed table. Two matter to you:

- **ASCII**: an old, small table with 128 characters: English letters, digits, punctuation. One byte per character.
- **UTF-8**: a modern encoding that can represent essentially every writing system, including Tamil, Hindi, Chinese and emoji. It is ASCII-compatible: plain English text looks identical, but other characters take two, three or four bytes.

This causes two classic problems.

1. **Garbled text.** A file written in one encoding and read as another turns "Müller" into something like "MÃ¼ller". The data is not lost; it was read with the wrong table.
2. **Fixed-width surprises.** If a layout counts *bytes* but a name has an accented letter, "Müller" is 7 bytes in UTF-8 though it is 6 characters. The field after it shifts by one. Always ask whether widths are in bytes or characters.

The Delphix page on control characters says the control-character value supports the UTF-8 character set (Managing rule sets). No encoding-selection setting is described there, so check your version for how it handles encodings.

1.24.8 XML and JSON at a glance

Some files are not line-and-column shaped at all; they are *nested*. Two formats dominate.

XML wraps values in named tags (for example `<firstName>Asha</firstName>`); **JSON** uses braces, names and values:

```
{
  "customerId": 1001,
  "firstName": "Asha",
  "orders": [
    { "orderId": 5001, "total": 1098.00 },
    { "orderId": 5002, "total": 1499.00 }
  ]
}
```

In both formats the orders are *inside* the customer. The parent-child relationship is shown by nesting instead of by a foreign key. For TDM, the thing to remember is that sensitive values (here the name) hide inside the structure, and a tool must know the path to them. The Delphix rule-set page lists source types of Database, Fixed Width, Delimited, JSON, XML and DataSet, and the rule-set creation dialog offers Database, Semi-Structured, Flat Files and Copybook (Managing rule sets). A *copybook* describes mainframe record layouts.

1.24.9 How the Delphix documentation treats file formats and keys

A short orientation, so that later Delphix chapters feel familiar. All statements below are from the pages linked; details come later in the Delphix rule sets and keys chapter and the Delphix file masking chapter.

- **Formats are defined separately from files.** You add them under Settings, in a Data Formats area with a tab for File formats and one for Mainframe (Managing file formats).
- **A format is shared.** Changing a file format's inventory affects every data file that uses that format. This applies to Fixed-Width, Delimited, Mainframe, JSON and XML formats.
- **Rule sets group the tables or files** a job will work on, and a rule set can be of type database, flat file, semi-structured or copybook (Managing rule sets).
- **Keys appear in the rule set.** A database table's rule-set row shows a logical key column, and a table without a primary key needs one for in-place masking (see the logical keys section above).

- **A filter is a WHERE clause** that limits the rows from a table, which is how a rule set does a simple subset. You will see WHERE in the SQL section next.

1.25 Basic SQL you will see

SQL (Structured Query Language, often said "sequel") is the language for asking a relational database questions and changing its data. You will meet it in TDM tools in small doses: a WHERE filter for a subset, a custom query for a table, a check of results. You need to read it, not master it.

The statements below use the ShopSphere tables and sample rows from this section; SQLite is a free way to run them on a laptop. The create-table statements are in the lab at the end of this section. Details such as date handling differ slightly between databases.

1.25.1 SELECT: read data

`SELECT` chooses columns; `FROM` names the table.

```
SELECT first_name, last_name, zip_code
FROM customers;
```

RESULT

first_name	last_name	zip_code
Asha	Menon	560001
Ravi	Kumar	600017
Meera	Nair	682001

`SELECT *` means "all columns". Handy for a quick look, but on a big table it returns everything, which can be slow and can display sensitive columns you did not need.

1.25.2 WHERE: choose rows

`WHERE` keeps only the rows that satisfy a condition.

```
SELECT * FROM orders
WHERE customer_id = 1001;
```

RESULT: ASHA'S TWO ORDERS

order_id	customer_id	order_date	total_amount
5001	1001	2026-01-15	1098.00
5002	1001	2026-02-03	1499.00

Text values go in quotes; numbers do not. Because `zip_code` is text in our table, `WHERE zip_code = '600017'` returns Ravi Kumar. You can combine conditions with `AND` and `OR`, and compare with `<`, `>`, `LIKE` and others.

> **WHY THIS MATTERS**
>
> A WHERE clause is the simplest subset there is: "only customers in postal area 600017". Both the Informatica and Delphix documentation use filters like this to limit rows. Delphix describes a rule-set filter as a WHERE clause, and says to prefix the column with the table name, for example `customer.cust_id < 1000` (Managing rule sets). Note that a WHERE on one table does not by itself bring along the related rows of other tables; keys do that, which is why subsetting is more than a filter.

1.25.3 JOIN: combine tables through keys

A join follows a foreign key. Here, each order is matched to its customer by `customer_id`:

```
SELECT c.first_name, o.order_id, o.total_amount
FROM customers c
JOIN orders o ON o.customer_id = c.customer_id;
```

RESULT

first_name	order_id	total_amount
Asha	5001	1098.00
Asha	5002	1499.00
Ravi	5003	598.00

`c` and `o` are short nicknames (aliases) for the tables. The `ON` line states the key match. Meera does not appear: she has no orders, and a plain `JOIN` keeps only matches. A `LEFT JOIN` keeps every customer and shows empty (null) where there is no order:

```
SELECT c.first_name, o.order_id
FROM customers c
LEFT JOIN orders o ON o.customer_id = c.customer_id;
```

This returns Asha 5001, Asha 5002, Ravi 5003 and Meera with no order id. You can chain joins to walk the whole chain customer, order, item, product:

```
SELECT c.last_name, p.product_name, i.quantity
FROM order_items i
JOIN orders o    ON o.order_id = i.order_id
JOIN customers c ON c.customer_id = o.customer_id
JOIN products p  ON p.product_code = i.product_code
ORDER BY i.item_id;
```

The result has four rows: Menon with Wireless Mouse (1), Menon with USB-C Cable (1), Menon with Laptop Stand (1) and Kumar with USB-C Cable (2). A subsetting tool does something like this walk in reverse: start from the customers you chose and follow the keys out to every related row.

1.25.4 COUNT: how many?

```
SELECT COUNT(*) AS customer_count FROM customers;
```

Returns 3. With `GROUP BY` you can count per group:

```
SELECT customer_id, COUNT(*) AS order_count
FROM orders
GROUP BY customer_id;
```

Returns 1001 with 2 orders and 1002 with 1 order. TDM people use COUNT constantly: how many rows before and after a subset? Does the masked table have as many rows as the source? Does the number of distinct values in a candidate key equal the number of rows? (`COUNT(DISTINCT customer_id)` counts the unique values.)

1.25.5 INSERT and UPDATE: change data

`INSERT INTO customers (...) VALUES (...)` adds a row (the lab below has full examples); the column list and value list must line up. If you inserted an order with `customer_id` 9999 when no such customer exists, a database that enforces the foreign key rejects it.

```
UPDATE customers
SET zip_code = '560002'
WHERE customer_id = 1001;
```

After the UPDATE, a SELECT for customer 1001 returns zip_code 560002. **Never write UPDATE without a WHERE** unless you really mean to change every row: the statement above, without its last line, would give all three customers the same postal code.

Masking is, at its core, a very large controlled UPDATE: replace the value in a sensitive column in every row (with a consistent rule). Delphix's in-place masking updates the source table; a copy-style approach writes to a different target. The tool generates the SQL for you, but understanding that it is an UPDATE explains why a table with no key is a problem: the tool needs a way to say "this exact row". Delphix's documentation says the custom SQL for a table must include the primary key column(s) and the columns to be masked (Managing rule sets).

Lab: run the ShopSphere queries (database needed, any relational database)

Goal: create the four tables, load the fake rows and reproduce this section's results.

1. **Create the tables in the order parents first, children second.**
   A child table with a foreign key can only be created once its parent exists (in most databases), and rows must be inserted in the same order.

```
CREATE TABLE customers (
  customer_id INTEGER PRIMARY KEY,
  first_name VARCHAR(50), last_name VARCHAR(50), email
VARCHAR(100),
  phone VARCHAR(15), ssn VARCHAR(11), date_of_birth
DATE,
  zip_code VARCHAR(10), tax_id VARCHAR(20));
CREATE TABLE products (
  product_code VARCHAR(10) PRIMARY KEY,
  product_name VARCHAR(100), price DECIMAL(10,2));
CREATE TABLE orders (
  order_id INTEGER PRIMARY KEY,
  customer_id INTEGER NOT NULL,
  order_date DATE, total_amount DECIMAL(10,2),
  FOREIGN KEY (customer_id) REFERENCES customers
(customer_id));
CREATE TABLE order_items (
  item_id INTEGER PRIMARY KEY,
  order_id INTEGER NOT NULL, product_code VARCHAR(10)
NOT NULL,
  quantity INTEGER, unit_price DECIMAL(10,2),
  FOREIGN KEY (order_id) REFERENCES orders (order_id),
  FOREIGN KEY (product_code) REFERENCES products
(product_code));
```

Expected result: Four tables created with no errors.

2 **Insert the rows.**

Use the sample rows shown earlier in this section. Customers and products first, then orders, then items.

```
INSERT INTO customers VALUES
(1001,'Asha','Menon','asha.menon@example.com','9876543210'
45-6789','1990-04-12','560001','TAX-A1001');
INSERT INTO customers VALUES
(1002,'Ravi','Kumar','ravi.k@example.com','9123456780','23
56-7890','1985-11-02','600017','TAX-R1002');
INSERT INTO customers VALUES
(1003,'Meera','Nair','meera.n@example.com','9988776655','3
67-8901','1999-07-23','682001','TAX-M1003');
INSERT INTO products VALUES ('P100','Wireless
Mouse',799.00);
INSERT INTO products VALUES ('P200','USB-C
Cable',299.00);
INSERT INTO products VALUES ('P300','Laptop
Stand',1499.00);
INSERT INTO orders VALUES (5001,1001,'2026-01-
15',1098.00);
INSERT INTO orders VALUES (5002,1001,'2026-02-
03',1499.00);
INSERT INTO orders VALUES (5003,1002,'2026-02-
10',598.00);
INSERT INTO order_items VALUES
(9001,5001,'P100',1,799.00);
INSERT INTO order_items VALUES
(9002,5001,'P200',1,299.00);
INSERT INTO order_items VALUES
(9003,5002,'P300',1,1499.00);
INSERT INTO order_items VALUES
(9004,5003,'P200',2,299.00);
```

Expected result: Three customer rows, three products, three orders and four items.

3 **Run the orphan check.**

This query lists orders whose customer does not exist. In a healthy database it returns no rows. It is the check you will run after every subset and mask to prove integrity.

```
SELECT o.order_id
FROM orders o
LEFT JOIN customers c ON c.customer_id = o.customer_id
WHERE c.customer_id IS NULL;
```

Expected result: No rows returned.

4 **Write two queries of your own.**
(a) How many order items does each order have? (b) Which customers have no orders?

▶ **Show a worked answer**

1.26 Data volumes and why size matters

ShopSphere's sample tables hold ten rows between them. The real database holds millions of customers and tens of millions of orders. Size changes everything about TDM.

Sizes run from KB (a short text file) through MB (a CSV of tens of thousands of customers) and GB (a small production database) to TB (a large enterprise database).

Why size matters to a TDM engineer:

- **Time.** Masking updates every row of every sensitive column. Ten times the rows is roughly ten times the work unless you add parallelism. Delphix profile jobs have a setting for the number of parallel streams, for example two streams to run two tables of a rule set at once (Delphix masking terminology).
- **Storage.** Every environment that holds a full copy multiplies the cost. See the why-TDM chapter.
- **Memory and file limits.** A file of many gigabytes cannot be opened in a spreadsheet; you work on it with tools or by sampling.
- **Test fitness.** Performance tests need volume; functional tests do not. Matching size to purpose is part of the job.
- **Row count versus width.** A table with a hundred million narrow rows and a table with a million rows of large documents behave very differently. Ask for both numbers.

> **ALWAYS ASK FOR COUNTS FIRST**
> Before any TDM job, ask for the row count of each table. It tells you whether a subset is needed, how long a job should take, and gives you a number to verify afterwards (row counts before and after should reconcile with what you expect).

1.27 Hands-on: which keys exist in the ShopSphere files?

This exercise rehearses the real situation from the logical keys section: data arrives as files, no database has declared any keys, and you must work out the relationships. It needs no tool, only reasoning.

ShopSphere's partner system sends three files. None of them states any key.

```
customers.csv
customer_id,first_name,last_name,email
1001,Asha,Menon,asha.menon@example.com
1002,Ravi,Kumar,ravi.k@example.com
1003,Meera,Nair,meera.n@example.com
1004,Asha,Menon,asha.m2@example.com

orders.dat  (pipe-delimited, no header)
5001|1001|2026-01-15|1098.00
5002|1001|2026-02-03|1499.00
5003|1002|2026-02-10|598.00
5004|1004|2026-02-11|299.00

order_items.txt  (fixed width: item_id 1-4, order_id 5-8, product 9-
12, qty 13-15)
90015001P100  1
90025001P200  1
90035002P300  1
90045003P200  2
90055004P200  1
```

Splitting the first item line by position gives item_id 9001, order_id 5001, product P100 and qty 1 (padded with two spaces to fill its field). Without the layout line above the file, the run of digits at the start would be meaningless. **The layout specification is part of the data.**

Lab: find the keys (thinking lab, no tool needed)

Goal: decide the primary key of each file, the foreign keys between them, and any problems.

1. **For each file, name a candidate primary key.**
   A candidate must be unique and never empty in every row. Test it by looking for repeats.

2 **Find the foreign keys.**
Which column in one file should match the primary key of another? Write each as *child.column to parent.column*.

3 **Check integrity.**
For each foreign key, list the values and check each one exists in the parent. Note any orphans.

4 **Spot a trap.**
Customers 1001 and 1004 look similar. Is that a key problem?

5 **Decide how you would tell a tool.**
The files have no declared keys. What would you tell a TDM tool to treat as logical keys?

▶ **Show a worked answer**

1.28 Common mistakes and gotchas

TREATING A COLUMN AS UNIQUE BECAUSE THE SAMPLE LOOKS UNIQUE

Three rows with distinct emails do not prove emails are unique across three million rows. Test with counts of all rows against counts of distinct values.

ASSUMING THE DATABASE ENFORCES THE KEYS

Many databases, especially older ones and data warehouses, have no declared foreign keys. Relationships exist only by convention. That is why profiling for keys and logical keys exist.

MASKING KEYS INCONSISTENTLY

If a key is masked in one table and not in its child tables (or masked differently), relationships break. Mask keys the same way everywhere, or leave them alone.

UPLOADING REAL DATA TO DEFINE A FORMAT

A layout needs only structure. Delphix says not to upload production data, sample records or PII when defining file formats (Managing file formats).

1.29 Recap

RECAP

- A database holds tables; a table has rows (records) and columns (fields) with data types. Metadata and schema describe the data without containing it.
- A primary key uniquely identifies a row; a foreign key in a child table points at a parent's primary key; together they give one-to-many relationships.
- Referential integrity means every foreign key matches an existing parent. Subsetting, masking and generation must all preserve it.
- Logical keys are keys declared to the tool when the database does not declare them. Informatica and Delphix both support them.
- Flat files have no built-in schema: know the layout (delimited or fixed-width), header and footer records, record types, line endings and encoding.
- XML and JSON nest children inside parents instead of using foreign keys.
- SELECT, WHERE, JOIN, COUNT, INSERT and UPDATE cover most of the SQL you will read. The orphan check (LEFT JOIN where the parent is null) is a TDM staple.
- Size drives time, cost and test fitness: ask for row counts first.