

THE DATA SERIES, FROM TALLY TO TOKEN

From Tally to Token

The Record, Its Readers, and What Each Era Broke

A record is a promise from the writer to a future reader, and every era changed the reader: the clerk, the tabulator, the program, the query, the person at the desk, the analyst, the platform and now the model. Most failures of meaning happened at the hops between them. This book follows one insurance record from an ink register in 1898 to an AI agent in 2026 and gives a CTO the Data Lineage Ledger: a plain way to see where each record was born, what changed at every hop, who can change it, what it costs and who reads it now.

Juan Cardena

Enterprise AI & Agentic Systems Architect · blog.jcardena.com

CONTENTS

BEFORE YOU START

Reader's Brief: One Record, Six Questions

PROLOGUE

One Record, Since 1898

PART I · THE RECORD

01 What a Record Is

02 The Lineage Ledger

PART II · THE ERAS

03 Before the Computer: Tallies, Tokens and Ledgers

04 Cards and the Unit-Record Shop

05 Tape, Disk and the First Databases

06 The Relational Bet

07 Data on Every Desk

08 Warehouse to Lakehouse

09 Meaning Moves to the Reader: Web, JSON, NoSQL, Logs and Streams

10 The Model as Reader

PART III · THE FORCES

11 Meaning: the Same Value in Every Representation

12 Control and Law: Who Could Change It, Who Could Read It

13 Cost and Death: Store, Move, Query, Keep, Delete

14 Proof: What Shows a Record Is True and Current

PART IV · THE WORK

15 Filling the Ledger in 90 Days

CLOSING

Epilogue: The Record Outlives the System

APPENDICES

A The Data Lineage Ledger: Template, Loss Codes and Mappings

B Timeline

C Failure Library

D Glossary

E Sources

BEFORE YOU START

Reader's Brief: One Record, Six Questions



THE ERAS OF THE RECORD

This book follows one record for 128 years, from a line of ink in 1898 to an AI service agent that misread it in 2026. Along the way it asks the same six questions of every record and gives you one tool, the Data Lineage Ledger, to write the answers down. This brief tells you who is in the story, what the questions are, and which chapters to read first for your role.

Three companies, all illustrative

The companies are invented. The practices they use are real, and each chapter sources them.

- **The insurer:** a mutual fire insurer founded in 1898. It owns the book's thread, policy 1898/0417, a perpetual fire policy on a brick warehouse, paid once by a deposit.
- **The health system:** regional hospitals and clinics with message feeds, APIs and images. It carries consent, time zones on lab results and health privacy law.
- **The hotel and retail group:** about 60 hotels and 400 stores, with bookings, point of sale, loyalty, event streams and a pricing agent. Its EU hotels bring European data protection and AI law into scope.

The six questions

Question	What a good answer names
Where is it written?	The system, format and place of every copy, not only the master
What does it mean?	The definition, the code tables and the unit, and where they are kept
Who can change it?	People, service accounts and agents with write access, including to the definition
What does it cost?	Store, move, query, keep and delete, with the people included
Who reads it now?	People, programs, models and agents, and for what purpose
What proves it is true and current?	Reconciliation, audit and tests; lineage shows the path, not the truth at birth

The Ledger in one row

The Ledger has one row per **hop**: any move or change of a record between people, systems or formats. Each row has ten columns: the record and hop, from, to, mechanism, what was lost (with one of eight loss codes), accountable owner, who can change it, classification, who reads it and what it costs. The first row of the thread reads: in 1898 a clerk copied the application and survey into register line 417 by hand, and the surveyor's notes on the building stayed in a separate book (loss code CONTEXT). Chapter Two teaches the columns.

Reading paths by role

Role	Read first	The decision you leave with
CTO	Prologue, Chapters 2, 10, 13, 15	Which five datasets get a Ledger this quarter, and what it costs
Chief data officer	Chapters 1, 2, 8, 14, 15	Who owns each hop, and which records have no owner
Data or AI architect	Chapters 2, 9, 10, 11, Appendix A	Which hops are AI risks, and where to capture lineage automatically
CISO	Chapters 2, 10, 12, 14	Which records an agent can already read that it should not

ERA CARD

Reader. You, with a record that matters and nobody who remembers its first writer.

Security. Every hop can add people and accounts who can change the record.

Cost. The Ledger costs hours by hand; Chapter Two states them.

AI. An agent reads whatever the hops left behind, meaning included or not.

Monday rule: pick one record your company could not afford to misread, and ask the six questions about it before you read further.

PROLOGUE

One Record, Since 1898



THE ERAS OF THE RECORD

On 12 February 2026 a logistics firm that owns a brick warehouse on the river front asked its fire insurer a plain question: is the building still covered? The insurer's service agent, an AI assistant with read access to the policy data, looked up the policy and answered within seconds. The policy, it said, had lapsed for non-payment: no premium had been received since 1898. The answer was wrong. The next morning, Friday 13 February 2026, a claims handler gave the customer the right answer. By then the wrong answer was already a record of its own, saved as a note on the customer's case. This prologue walks that answer back to the line of ink it came from.

The answer, and why it sounded right

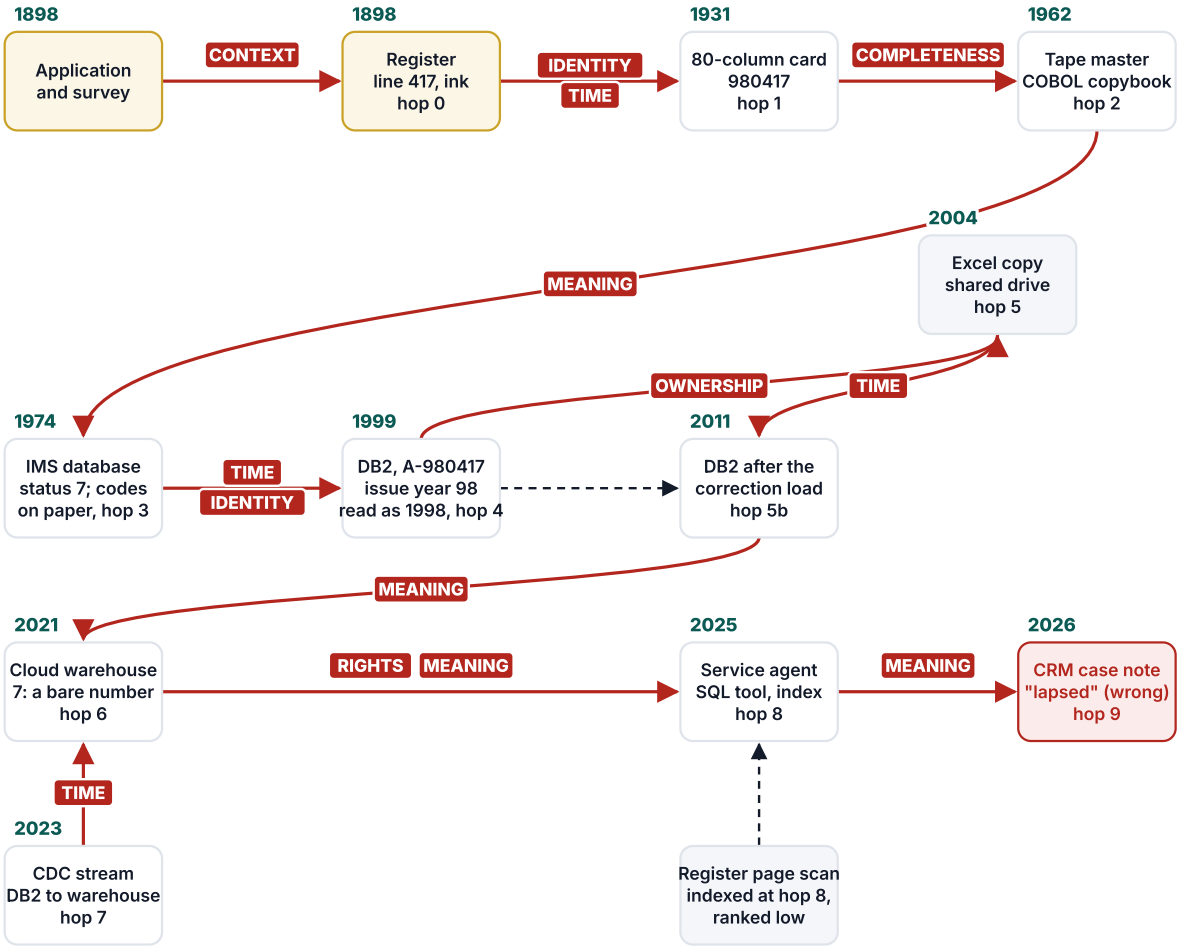
The insurer in this book is invented, a mutual fire insurer founded in 1898, and so is policy 1898/0417. The kind of policy is real. Policy 1898/0417 is a **perpetual** fire policy, issued on 14 March 1898 to a wool merchant who owned the warehouse. A perpetual policy is paid once, by a deposit, carries no annual premium and stays in force, unless cancelled, while the building stands; this one passes to a new owner with the insurer's consent. The deposit on this one was 412.50 dollars, held by the insurer and refundable if the policy is cancelled.

The Philadelphia Contributionship, a fire insurer founded in 1752, made its policies perpetual in 1810, according to both its own digital archive and the Library Company of Philadelphia's guide to its records; the guide describes a refundable deposit that "serves to provide coverage until the policyholder or the company cancels the policy."

Now look at what the agent saw. Its SQL tool returned the warehouse row for policy A-980417, the key the policy has carried since 1999, with an issue date of 1898, no premium payments on record since then, and a status column holding the number 7. Nothing in the row said what 7 meant, so the agent supplied a plausible meaning. No payments for 128 years and an unexplained code read, to most readers, like a lapse. The truth, written in a paper code book in the archive and in a DB2 lookup table that never reached the warehouse, is that status 7 means **perpetual, deposit held, no premium due**.

FIGURE P.1

One record, since 1898: eleven hops, numbered 0 to 9 with a branch 5b, and what each one lost



Red arrows and chips mark a hop where something was lost (loss codes, Chapter 2).

Policy 1898/0417 from the ink register to the case note, row by row: the 2004 spreadsheet leaves DB2 and flows back in 2011, the 2023 stream feeds the warehouse, and each red chip is a Ledger loss code.

Author's diagram based on the book's running example; the insurer, numbers and people are illustrative.

Walking back: from the case note to the warehouse

A **hop** is any move or change of a record between people, systems or formats. Figure P.1 shows the eleven hops of policy 1898/0417, numbered 0 to 9 with a branch, 5b. Read it backwards: from the case note at the end of the last row, where the wrong answer sits, to the register at the top left.

2026, hop 9. The conversation and the agent's answer were saved to the customer's case in the CRM, with the agent's tool calls. The wrong fact now has its own lineage, and the next agent that reads the case can repeat it with a citation.

2025, hop 8. The agent reached the policy two ways. A read-only SQL tool returned rows from the warehouse. A retrieval index held scanned documents as vectors, among them the 1898 register page with the word "perpetual" on it. It ranked below newer documents and never reached the answer. The purpose of the processing, why the agent may read this policy at all, was not attached to the tool either.

2023, hop 7. Change data capture began to stream policy changes from DB2 into the warehouse within seconds. Endorsements arrived ordered by processing time, not event time: when an original change reached the stream late, the correction that followed it could appear first.

2021, hop 6. This is where the meaning stopped travelling. A nightly job unloaded the policy tables from DB2 for z/OS, converted EBCDIC text and packed decimal numbers to UTF-8 text and decimals, and loaded them into a cloud warehouse. The status column arrived as an integer; the STATUS_CODES table that had sat beside it in DB2 since 1999 did not. In the warehouse, 7 is a bare number.

2011, hop 5b, and 2004, hop 5. The date window had given every policy written from 1900 to 1949 a future issue date. In 2004 a branch clerk exported those to an Excel workbook on a shared drive, typed corrected years by hand from the registers, and added the 1898 and 1899 policies the window had made a century too young, this one among them. The workbook had no owner and was copied and emailed. In 2011 it was loaded back into DB2 as a one-off correction, which put 1898 back in this policy's issue date; the key stayed A-980417. For some 3,400 policies (an illustrative count) hand-typed dates overwrote the windowed ones, and a few hundred were wrong. The agent saw the correct year in 2026 because a spreadsheet nobody owned happened to be right about it.

1999, hop 4. The insurer merged, and both companies' policies were converted into one DB2 relational database on the mainframe. The same project applied a Y2K date window: years 00 to 49 read as 20xx, 50 to 99 as 19xx. The issue date's two-digit year, 98, was read as 1998, and from 1999 to 2011 the new DATE column said 1998 in four confident digits. The key is not a date and was left alone. Since 1931 new policies had a separate seven-digit series, so 980417 had stayed unique; now the other company's numbers collided with it, so keys got a prefix: the policy became A-980417. The same project keyed the paper code book into a DB2 lookup table, STATUS_CODES, for the first time.

1974, hop 3. Conversion programs loaded the tape file into an IMS hierarchical database, and the status digit of the 1931 card became a field in the policy segment. The code book that said what 7 meant, by now kept by the policy department, stayed on paper. The value entered a shared database without its meaning, as it had sat on the card; clerks could still walk over and ask. Hop 6 is where nobody could.

1962, hop 2. A conversion run read every punched card onto magnetic tape, described by a COBOL copybook. The remarks field, card columns 61 to 80, was dropped as "unused". For some policies it held remark code 9, the card's second mark of a perpetual policy; from then on status 7 was the only one.

1931, hop 1. Keypunch operators transcribed the register onto 80-column cards, one card per policy. A slash cannot be punched in a numeric field, and columns were scarce, so 1898/0417 became 980417: a two-digit year, no slash. The card also gave the policy a one-digit status code, 7, whose meaning was typed in a code book kept by the tabulating department.

1898, hop 0. A clerk wrote line 417 of the policy register by hand, in ink, from the application and the surveyor's report: number, date, insured, property, sum insured, deposit, remarks. The surveyor's notes on the building, its brick walls, slate roof and the distance to the next warehouse, stayed in a separate survey book. The first hop already left something behind.

What the walk back shows

Count the losses. At hop 0 the context stayed in another book. At hop 1 the key was renumbered and the century dropped. At hop 2 the second mark of a perpetual policy was deleted as unused. At hop 3 the value entered a shared database without its meaning. At hop 4 the date was misread and the key renumbered again. At hops 5 and 5b a copy without an owner corrected one error and created others. At hop 6 the code table that DB2 held was left behind. At hop 7 order in time was lost. At hop 8 the purpose was not attached, and the one document that held the truth ranked too low to be read. At hop 9 the wrong answer became a record with a citation.

No single system failure explains the answer. Records wrong at birth are Chapter Fourteen's subject. This one failed between systems, a little at each hop, and each reader trusted what the previous one handed over.

> *The agent did not lie. It read the record it was given, and that was no longer the record that was written.*

What changed is the reader. The model **interprets instead of executing**. Where meaning is missing it can supply a plausible one and say it in fluent prose to a customer, unless the system around it makes it look the meaning up or stop. That is why hops that people and programs lived with since 1931 are a risk now.

What this book gives you

The tool is the **Data Lineage Ledger**: one row per hop, ten columns, and a loss code for what was lost. Filled in for policy 1898/0417, it would have shown a red mark at hop 6 in 2021, four years before the agent was connected. Lineage of this kind is evidence that the record behind an answer is the record that was written, and of who changed it on the way. It does not prove the record was true when it was born.

The chapters that follow take the eras in order, then the forces that cut across them; the last chapter fills the Ledger in 90 days and closes hop 9 with a correcting note.

> **ERA CARD**
>
> **Reader.** Every reader from the clerk to the model read or handled this record; the model is the newest.
>
> **Security.** By 2026, service accounts, three data engineers, a stream team and an agent platform could change or read it.
>
> **Cost.** The 2011 load took a DBA one day; reconciling it took years (illustrative).
>
> **AI.** The agent read a bare code and gave it a meaning, and its answer became a new record.

Monday rule: before an agent reads a table, list every coded column it returns and check that the code table travels with it.

CHAPTER ONE

What a Record Is



In 1768, according to the Library Company of Philadelphia's guide to its archive, the directors of the Philadelphia Contributionship, a fire insurer founded in 1752, "apparently felt the need for additional safeguards. They required that the Surveyors record their surveys." From then on, what a surveyor saw on a house was written down, kept, and could be read by someone who had never met him. That is the whole subject of this book in one decision: a fact becomes a record when it is written so that a future reader can rely on it. This chapter defines the record and the things that are often confused with it, names the parts of a record that are older than any machine, sets out the six questions the book asks of every record, and introduces the readers who have read records since the clerk. The decision it changes: **before you govern data, decide which of it is a record, and at what grain.**

A record is a promise to a future reader

A **record** is information written down, kept, and relied on as evidence of something that happened or was decided. The writer makes a promise: this was true when I wrote it, and you may act on it later without asking me. US federal law draws the same line for its own agencies. Under 44 U.S.C. 3301, federal records include "all recorded information, regardless of form or characteristics" made or received by an agency and "preserved or appropriate for preservation" as evidence of its activities or "because of the informational value of data in them." Form does not decide it. Keeping it as evidence does.

Five neighbours are easy to confuse with a record, and the difference matters when you decide what to govern.

Term	What it is	Example from the insurer
Assertion	A claim someone makes, not yet kept as evidence	The applicant says the roof is slate
Document	A container for content; it may hold records, or none	The scanned register page
Event	A fact that something happened at a time; a record if it is kept	An endorsement added to the policy, with the time it took effect
Dataset	Many records, or parts of them, held together for a reader	The policy table in the warehouse
Derived artifact	Something computed from records: a total, a score, a model, an answer	The agent's reply in 2026, saved as a case note

The last row is the one that surprises teams. A derived artifact can itself become a record the moment someone keeps it and relies on it. That is what happened to the service agent's wrong answer in the prologue: a derived artifact became a case note, and the case note became a record with its own readers.

Field, key and audit trail are older than machines

Three ideas carry every record in this book, and none of them was invented by a computer.

A **field** is a named place for one kind of value. The ruled columns of the 1898 register (number, date, insured, property, sum insured, deposit, remarks) are fields. When the Census Office tabulated the 1890 census with Herman Hollerith's machines, the Census Bureau's history explains that, on the pantograph punch the clerks used, "each hole in the template corresponded to a specific demographic category"; the card's layout was the schema. The field moved from a ruled column to a hole position to a column in a table, and its meaning stayed outside it every time.

A **key** is the value that tells one record from another and lets a reader find it again. Policy 1898/0417 was a register year and a line number. It became 980417 on a card, then A-980417 after a merger. Each renumbering was reasonable. Each one is a place where a later reader could join the wrong record or miss the right one.

An **audit trail** is the record of changes to records: who wrote, who corrected, who approved. In the 1898 register, corrections were initialled by the company secretary. Double-entry bookkeeping, first printed by Luca Pacioli in 1494 as the method used in

Venice, writes every amount on two sides so that the books can be checked against themselves. Chapter Three tells that history. The point here is narrow: the questions an auditor asks of a database today were asked of ledgers for centuries before it.

> **WHAT HASN'T CHANGED**
>
> The Contributionship's archive, as the Library Company's guide lists it, holds survey books and, from 1810 to 1918, policy registers in separate volumes. The notes on a building and the line that insured it lived in different books in the nineteenth century, and they live in different systems now. Chapter Two calls that loss CONTEXT.

The six questions

Every chapter asks the same six questions of a record. They are short on purpose: a team can ask them in a meeting about any table, file, stream or model input.

- **1. Where is it written?** Every copy, not only the master: the warehouse, the stream, the spreadsheet, the retrieval index.
- **2. What does it mean?** The definition, the code tables, the unit and the time basis, and where those are kept.
- **3. Who can change it?** People, service accounts and agents with write access, to the value and to its definition.
- **4. What does it cost?** To store, move, query, keep and delete it, people included.
- **5. Who reads it now?** People, programs, models and agents, and for what purpose.
- **6. What proves it is true and current?** Reconciliation, audit, tests. Lineage shows the path; it does not prove the record was true when written.

Asked of policy 1898/0417 in 2025, the second question would have had an awkward answer: "status 7 means something; the meaning is in a DB2 lookup table and a paper code book, and neither is in the warehouse." The fifth would have added "and an AI agent reads it now." Put together, those two answers were the warning.

The six questions at the three companies

The questions are not only for old records. Here they are asked of one record at each of the book's three companies, as each stands in 2026. The companies are illustrative; the patterns are common.

Question	The insurer: policy 1898/0417	The health system: a potassium result	The hotel group: a booking
Where is it written?	DB2, the warehouse, the stream, a retrieval index, a CRM note	Lab system, message feed, patient record, an API, the analytics store	Booking engine, payment provider, CRM, stream, warehouse
What does it mean?	Status 7 is defined in a DB2 lookup table and on paper, not in the warehouse	The value needs its unit, reference range and the time the sample was taken	The rate code needs its cancellation rule
Who can change it?	DBAs, pipeline accounts, data engineers, CRM users and agents	Lab staff through an amendment; interface accounts	Reservations staff, the booking engine, a pricing agent
What does it cost?	Store, move and query across five systems, plus reconciliation	Interfaces, storage of images and results, retention for years	Seconds of compute, and long retention of payment-related data
Who reads it now?	Actuaries, dashboards, the service agent	Clinicians, billing, researchers, a summarizing assistant	Front desk, revenue managers, the pricing agent
What proves it true and current?	Reconciliation with the deposit ledger; the register scan	The lab's amendment history and its timestamps	The payment record and the guest's confirmation

Read the table by row, not by column. The second row is where an AI reader is most exposed at every company: a code without its table, a number without its unit or time, a rate without its rule. The third row grows with every hop, and in 2026 it includes agents. The last row is the one most teams leave blank, because the honest answer is often "nobody checks."

Two warnings keep the questions honest. First, a record can be wrong at birth: the applicant can misdescribe the roof, the lab can mislabel the sample, the guest can mistype a name. No amount of lineage repairs that; Chapter Fourteen deals with proof. Second, a record can be stale but traceable: every hop is documented, and the value is still out of date. Lineage tells you how a value arrived. Whether it is still true is a separate question, and the sixth question exists to keep it separate.

The questions also give a team a shared vocabulary. A data engineer hears question one as "which tables", a lawyer hears question three as "who is accountable", a finance lead hears question four as "which budget line", and a security lead hears question five as "which

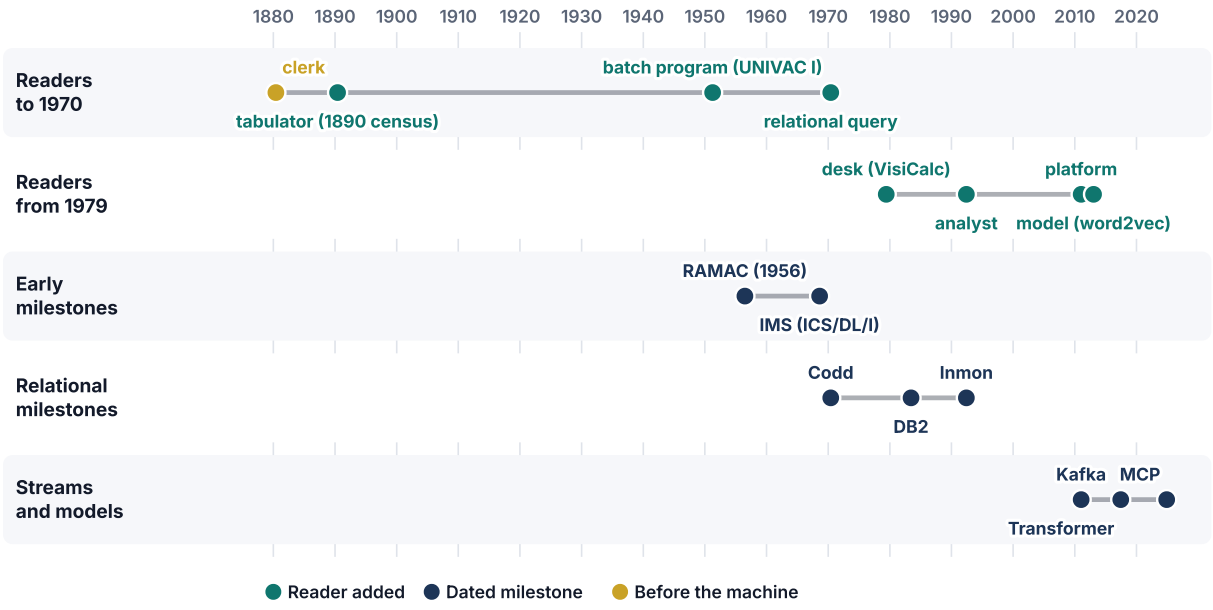
identities". When all four answer about the same record in the same meeting, the gaps between their answers are usually where the next incident is.

The reader ladder

A record is written for a reader, and every era changed the reader. Figure 1.1 places each new reader on the time line at a milestone this book uses to mark its era, not at the reader's first appearance, and Figure 1.2 stacks them as a ladder. Each rung added a reader; none of the older readers went away. A clerk still reads ledgers, and a batch job still runs every night beside the agent.

FIGURE 1.1

Each era added a new reader of the record: the milestones this book uses



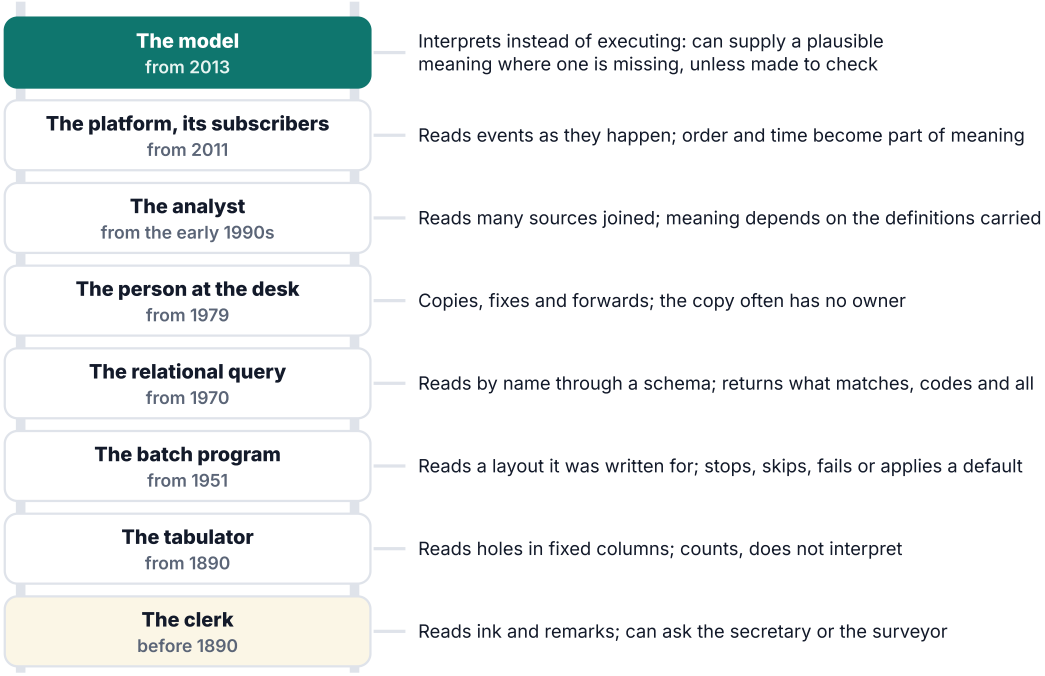
Before 1890 the reader was a clerk with a pen (Chapter 3); from 1890 each era added a reader, and none of the older readers fully went away. Dates are milestones that mark each era, not the first appearance of each kind of reader: 2013 marks word2vec, an influential step in word embeddings (earlier neural language models existed); the agent that answers customers arrives in the 2020s. The era strip at each chapter's opening marks when the technology arrived; this ladder marks the milestone this book uses for each reader.

Source: US Census Bureau (Hollerith, UNIVAC I); IBM history (RAMAC, IMS, DB2); Codd, CACM, June 1970; Computer History Museum (VisiCalc, 1979); Inmon 1992 (publisher record); LinkedIn (Kafka, 2011); arXiv 1301.3781 and 1706.03762; Anthropic (MCP, 2024). Details in Appendix E.

Each reader fails differently when meaning is missing. The clerk could ask the secretary what a remark meant. The tabulator counted holes and did not interpret them. The batch program read the layout it was written for; on anything else it stopped, skipped, failed, or applied a default and carried on, as the date windows of 1999 did. A relational query, after Codd's 1970 paper, reads by name through a schema and returns whatever matches, codes and all; the person reading the report has to know what the codes mean. The person at the desk copies, fixes and forwards, and the copy often has no owner. The analyst joins many sources and inherits every definition that did and did not travel. The platform and its subscribers read events as they happen, so order and time become part of meaning.

FIGURE 1.2

The reader ladder: who reads the record, and what each reader can do when meaning is missing



Each rung added a reader without removing the ones below; the top rung interprets, and where meaning is missing it can supply a plausible one and carry on unless the system makes it check.

Author's diagram.

Then the model. The batch program of 1962 could not ask anyone what status 7 meant, any more than the agent of 2026 could. What is new is how the model behaves when meaning is missing: it **interprets instead of executing**. A program given an unknown code follows its code path. A model given an unknown code can supply a plausible meaning, write it in fluent language and carry on, unless the system around it requires a lookup or lets it say it

does not know. That is what the service agent of the prologue did. It reads documents as well as rows, and it can be connected to tools that write. So the cost of a meaning lost at a hop changed: it usually surfaced as a wrong total in a report a person reviewed; now it can surface as a confident sentence to a customer.

Grain: say what one row is

Ralph Kimball, writing in March 2003 about dimensional design, put the discipline in one line: "Declaring the grain means saying exactly what a fact table record represents." He added that "the grain declaration is like a contract." This book borrows the idea for lineage. Before you trace a record, say what one unit of the trace is. Five grains appear in the chapters:

- **Record:** one policy, one patient encounter, one booking. The thread of this book is at record grain.
- **Field:** one column of one record, such as the status code or the issue date. Many losses happen at field grain.
- **Event:** one change at one time, such as an endorsement in the stream.
- **Dataset:** a table, a file or a topic as a whole. Most lineage tools work here.
- **Derived artifact:** a report, a feature, an embedding, a model or an agent's answer.

Grain mismatch is the most common reason lineage diagrams look complete and answer nothing. A dataset-level graph can show that the warehouse table came from DB2 every night. It cannot show that one column lost its code table at that hop, or that one policy's date was hand-typed in 2004. Chapter Two sets the Ledger's grain at one row per hop of a record or dataset, and says where tools that work at other grains fit.

TYPE CONTRACT: STATUS CODE ON POLICY 1898/0417

Type, precision and rounding. a code, not a quantity: one decimal digit, 0 to 9; value 7; no arithmetic and no rounding apply.

Encoding and serialization. one punched digit on the 1931 card, a six-bit BCD digit on the 1962 tape, a one-digit field in IMS from 1974 and in DB2 from 1999, a bare integer column in the warehouse from 2021.

Unit or currency. none; it is a code.

Time basis. current status, not dated.

Identity scope. one policy: 1898/0417 in the register, 980417 on the card, A-980417 from 1999.

Absence. never stated.

Authority. the code book: "perpetual, deposit held, no premium due", in a typed paper code book from 1931 and in the DB2 lookup table STATUS_CODES from 1999.

The definition was written down twice, and from 1999 it sat beside the value in DB2, but it did not travel with the value into the warehouse, so every reader of the warehouse got the digit alone.

ERA CARD

Reader. All eight readers, from the clerk to the model; each added, none removed.

Security. Who can change a record has grown from a clerk and a secretary to accounts, teams and agents.

Cost. Deciding what is a record, and at what grain, is the cheapest governance step there is.

AI. The model interprets instead of executing, so missing meaning turns into plausible answers.

Monday rule: name the five records your company would least like an agent to misread, and write down the grain at which you will trace each one.

CHAPTER TWO

The Lineage Ledger



THE ERAS OF THE RECORD

Hop 6 of the thread happened in 2021. A nightly job moved the insurer's policy tables from DB2 for z/OS into a cloud warehouse, and the status code table did not come with them. Nobody noticed for four years, because nothing broke: dashboards showed counts by status, and a count by status can be computed without knowing what each status means. In 2025 an AI agent was connected to the warehouse, and in February 2026 it told a customer that a perpetual policy had lapsed. A single row in a plain table, written in 2021, would have carried the warning: **hop 6, status code table not carried, loss code MEANING, read by dashboards and data scientists**, and later, **read by the service agent**. This chapter teaches that table, the Data Lineage Ledger: its ten columns, its eight loss codes, its grain, how to fill one row by hand in thirty minutes, what it costs, when it is not worth it, and how it maps to two open specifications for lineage and provenance. The decision it changes: **lineage starts as a short, owned list of hops for the records that matter, not as a platform.**

One row per hop, ten columns

A **hop** is any move or change of a record between people, systems or formats: a copy, a load, a conversion, an export, a correction, a retrieval, an answer saved as a note. The Ledger has one row per hop. Its ten columns answer the six questions of Chapter One for that hop.

The Data Lineage Ledger: one row per hop

Record or dataset, hop	From	To	Mechanism and schedule	Format change; what was lost or assumed	Accountable owner	Who can change it	Classification; retention or erasure duty	Read by (people, apps, models, agents)	Cost per month; basis

Loss codes

MEANING	a code or definition did not travel with the value
CONTEXT	notes or surrounding knowledge stayed behind
PRECISION	type or rounding changed the value
TIME	century, time zone, or event versus processing time changed
IDENTITY	the key was renumbered, merged or split
OWNERSHIP	a copy has no accountable owner
RIGHTS	purpose, consent or legal basis did not travel
COMPLETENESS	rows or fields were dropped

Columns 1 to 5 describe the movement and what it did to meaning. Columns 6 to 10 describe control, readers and cost after the hop.

- **1. Record or dataset, hop.** What moved, and the hop's number in its graph. Hops can branch and flow back, as hops 5 and 5b of the thread do, so the number names a place in a graph, not a step in a line.
- **2. From** and **3. To.** The system, platform and location on each side, or the person, if a person is the source.
- **4. Mechanism and schedule.** How it moved and how often: a clerk by hand once, a nightly unload at 02:00, change data capture within seconds, a tool call per question.
- **5. Format change; what was lost or assumed.** The type and encoding change, then one or more loss codes and one sentence on what was lost. Neither OpenLineage nor W3C PROV, the two specifications this chapter maps the Ledger to, has a standard field for it, and it is the reason the Ledger exists.
- **6. Accountable owner.** One role that answers for the record after the hop. "None recorded" is a valid and alarming answer.
- **7. Who can change it.** People, service accounts and agents with write access, to the values and to the definition.
- **8. Classification; retention or erasure duty.** How sensitive it is, and how long it must be kept or when it must be deleted.
- **9. Read by.** People, applications, models and agents, and whether it is used to train anything.
- **10. Cost per month; basis.** One figure and how it was reached: store, move and query, plus people where they are material.

Some teams add optional columns: the birth date and reason of the record, its legal basis, a contract or service level, freshness, an exit path, and **own or rent**. Keep the core ten; add others only when a decision needs them.

Eight loss codes

"What was lost" needs a controlled vocabulary, or every row says it differently and nobody can count. The Ledger uses eight codes. A hop can carry more than one; a hop with no material loss carries none, and saying so is useful.

Code	What did not travel	In the thread
MEANING	A code or definition did not travel with the value	Hop 6: status 7 arrives as a bare number
CONTEXT	Notes or surrounding knowledge stayed behind	Hop 0: survey notes stay in the survey book
PRECISION	Type or rounding changed the value	Not on this policy; Chapter Eleven shows a float that is wrong
TIME	Century, time zone, or event versus processing time changed	Hop 4: 98 read as 1998
IDENTITY	The key was renumbered, merged or split	Hop 1: 1898/0417 becomes 980417
OWNERSHIP	A copy has no accountable owner	Hop 5: the Excel workbook on a shared drive
RIGHTS	Purpose, consent or legal basis did not travel	Hop 8: the agent's tool carries no purpose
COMPLETENESS	Rows or fields were dropped	Hop 2: remarks field dropped as "unused"

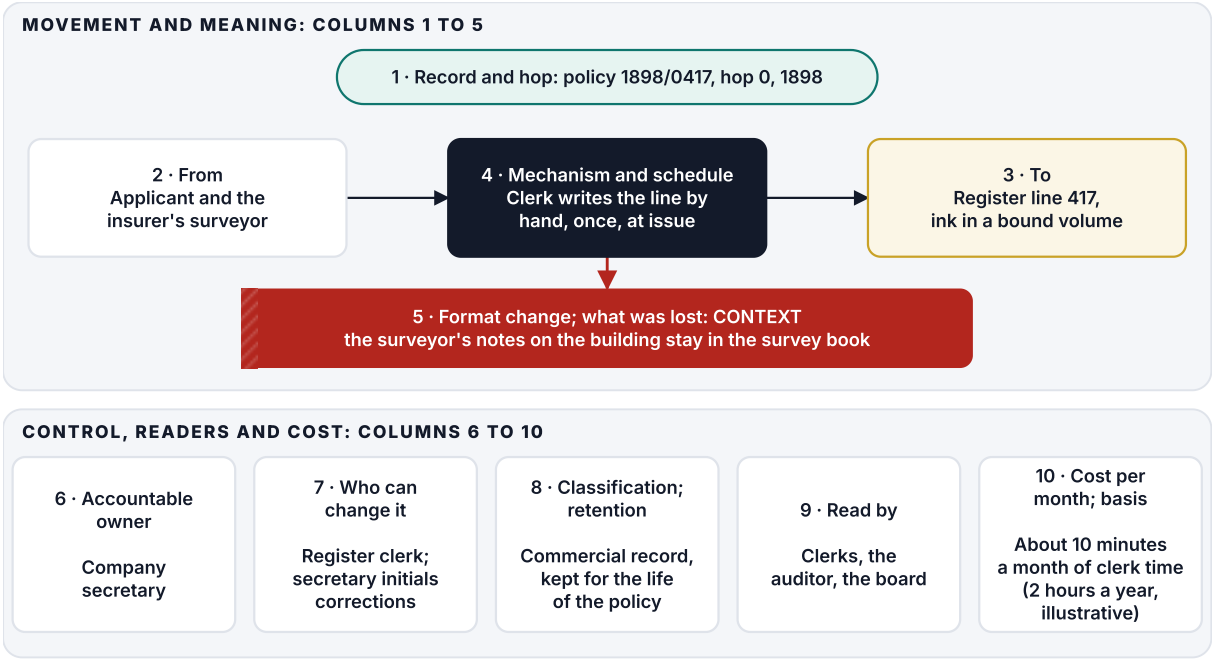
The codes are about meaning, not about technology. A migration from one database to another can carry no loss at all; a spreadsheet copy made in five minutes can carry two. The codes also make the Ledger countable: a CDO can ask how many hops into the agent platform carry MEANING or RIGHTS, and get a number.

Grain, and the anatomy of a hop

Chapter One named five grains: record, field, event, dataset and derived artifact. The Ledger's grain is **one row per hop of a named record or dataset**. For a single important record, such as policy 1898/0417, the rows follow that record. For most enterprise work the rows follow a dataset, such as the policy table, and column 5 names any field that lost something, such as the status column. A nightly job that runs 365 times a year is one hop, one row, not 365 rows; the run history belongs to the tools that record runs. That choice is what keeps a hand-filled Ledger small enough to finish.

FIGURE 2.1

Anatomy of a hop: ten columns, one row, filled here with hop 0



A hop is any move or change of a record between people, systems or formats; columns 1 to 5 describe the movement and what it lost, columns 6 to 10 who controls, reads and pays for the result.

Author's diagram; values from hop 0 of the book's thread (illustrative insurer).

Figure 2.1 shows a hop's anatomy with the first hop of the thread. The movement is above: from the applicant and the surveyor, by a clerk's hand, to line 417 of the register. The loss sits under the mechanism, in red, because that is where it happens: the copying step chose what to write and what to leave in the other book. Below are the five columns that say who controls the result. Here is the same hop as the Ledger records it.

THE HOP · 0 · 1898 · APPLICANT AND THE INSURER'S SURVEYOR TO POLICY REGISTER, LINE 417 (INK, BOUND VOLUME)

Record	Policy 1898/0417 (perpetual fire policy), hop 0, 1898
From	Applicant and the insurer's surveyor
To	Policy register, line 417 (ink, bound volume)
Mechanism and schedule	Clerk writes the line by hand from the application and the survey report
Format change; what was lost or assumed	CONTEXT Ink; fields in ruled columns: number, date, insured, property, sum insured, deposit, remarks. **The surveyor's notes on the building (brick, slate roof, distance to the next warehouse) stay in a separate survey book**
Accountable owner	Company secretary
Who can change it	Register clerk; corrections initialled by the secretary
Classification; retention or erasure duty	Commercial record; kept for the life of the policy
Read by (people, apps, models, agents)	Clerks, the auditor, the board
Cost per month; basis	Clerk time, about 10 minutes a month (2 hours a year) for the register line and the annual audit (illustrative)

Notice what the row does not say. It does not say the register was wrong; it was right. It does not say the survey book was lost; it was kept. It says that a reader of the register alone will not know the building had a slate roof and stood close to its neighbour. In 1898 that reader was a clerk who could walk to the shelf. In 2025 that reader was a retrieval index that had a scan of the register and no scan of the survey book.

A 30-minute exercise: fill one row by hand

The Ledger is learned by filling it. Pick one record or dataset your company could not afford to misread: a policy table, a patient's allergy list, a price list an agent reads. Book thirty minutes with the person who knows the system best. Use the blank above, on paper or in a spreadsheet.

Minutes	Do this	Columns
0 to 5	Name the record and the one hop into the system an agent or analyst reads today	1, 2, 3
5 to 10	Write the mechanism and schedule as the job or person actually does it, not as the diagram says	4
10 to 18	Compare one real value on both sides. List every coded column and check where its code table lives. Pick loss codes	5
18 to 24	Name one accountable owner by role; list who can write, including service accounts	6, 7
24 to 28	Write the classification, the retention duty, and every reader you can name, agents included	8, 9
28 to 30	Write one monthly cost figure and its basis, even if rough; mark it as an estimate	10

Most teams find something in minutes 10 to 18. A coded column whose table is in a wiki, a PDF or someone's memory is a MEANING loss waiting for a reader that interprets. If the row ends with "none recorded" in column 6 or an agent in column 9 that nobody approved, you have your first finding. Chapter Fifteen turns one row into five datasets in 90 days.

Once a few rows exist, read them for three patterns. A **chain of MEANING losses**, where the same code or definition is missing at several hops in a row, says that nobody ever owned the definition; fix it at the source and require it at every hop after. A **hop with no owner** in column 6, like the 2004 workbook, says that a copy is making decisions nobody answers for; give it an owner or delete it. A **branch that flows back**, like hop 5b, says that a side copy has become a source of truth; record the load as a hop of its own, with who signed it, or the next reader will never know why some values differ from their history.

Why lineage programs fail, and what the Ledger costs

The honest counterargument comes first. Lineage programs have a long record of disappointment: metadata repositories and catalogs bought, populated once, and left as shelfware. A reasonable CTO can say that a contract and a test at the last hop, the table the agent reads, is cheaper than history. Often it is.

> **OUR VIEW, AS OF NOVEMBER 2026**
>
> Lineage programs tend to fail for four reasons. They start at the wrong grain, capturing every job run and column for every dataset, so the graph is complete and no one can read it. They have no owner per hop, so nobody keeps it current. They record movement but not loss, so the graph shows that data moved but not what it no longer means. And they are sold as a platform before anyone has asked a question the platform should answer. The Ledger is meant to avoid all four: hand-filled first, scoped to a few records, with an owner and a loss code in every row.

The cost is mostly people. As an estimate for planning, not a measured figure: one row takes 30 minutes to two hours with the right person in the room, and a dataset that matters usually has five to twelve hops from birth to its readers. Five datasets is therefore about 12 to 120 hours, from two days to three weeks of one person's time spread across a month, plus an hour a quarter per dataset to keep the rows current. Automated capture, covered below, lowers the cost of columns 2 to 4 and part of 9. It does not fill column 5.

The last-hop contract and the Ledger are not rivals. A contract on the warehouse table could have said "status is an integer from 1 to 9" and passed every day after 2021, because the values were valid. Only the meaning was missing. What the Ledger adds is the hop where meaning left, so the contract can be written to require it: "status arrives with its code table and the definition of each value."

Not worth it when:

- The data is disposable: scratch tables, test extracts, caches rebuilt every hour, with no reader that acts on them.
- There is one hop: the system that writes the record is the only system that reads it, and no agent reads it.
- A strong contract already exists at the reader, and it covers meaning (code tables, units, time basis), not only types.
- The record will be retired within months, and the decision is to delete, not to explain.
- Nobody will own the rows. A Ledger without owners becomes the shelfware it was meant to replace; spend the time on the contract instead.

How the Ledger maps to OpenLineage and W3C PROV

The Ledger is not a private standard. Its columns map to two open specifications, and teams that automate should use them.

OpenLineage, which its project site describes as "an open framework for data lineage collection and analysis", models three things. A **Dataset** is "an abstract representation of data", named within a namespace. A **Job** is "a process that consumes or produces Datasets." A **Run** is "an instance of a Job that represents one of its occurrences in time", identified by a runId. Runs emit events with a run state: START, RUNNING, COMPLETE, ABORT, FAIL or OTHER. Extra metadata travels as **facets**: a schema facet, an ownership facet, a tags facet, and a column-level lineage facet that records, for each output column, its input fields and the kind of transformation (DIRECT or INDIRECT, with subtypes such as IDENTITY, TRANSFORMATION, AGGREGATION, JOIN or FILTER). The specification lets producers define custom facets with their own schema.

W3C PROV is a family of documents published on 30 April 2013; its data model, PROV-DM, is a W3C Recommendation. It has three core types. An **entity** is "a physical, digital, conceptual, or other kind of thing with some fixed aspects." An **activity** is "something that occurs over a period of time and acts upon or with entities." An **agent** is "something that bears some form of responsibility for an activity taking place." Relations join them: an activity *used* an entity, an entity *wasGeneratedBy* an activity, one entity *wasDerivedFrom* another, an entity *wasAttributedTo* an agent, an activity *wasAssociatedWith* an agent, and an agent *actedOnBehalfOf* another.

FIGURE 2.2

The Ledger's ten columns mapped to OpenLineage, W3C PROV and the Medallion Data Bill of Materials

Ledger column	OpenLineage	W3C PROV	Medallion Data BOM
1 Record or dataset, hop	Dataset (namespace and name); a recurring hop is about one Job, a one-off hop one Run	Entity; each version a new entity, linked by wasDerivedFrom	Dataset or answer
2 From	Input datasets of a run	used (the activity used an entity)	Source
3 To	Output datasets of a run	wasGeneratedBy	No column
4 Mechanism and schedule	Job, and each Run (runId; START to COMPLETE events; nominalTime facet)	Activity, with optional start and end time	Freshness (the result, not the mechanism)
5 Format change; what was lost	Schema facet; column-level lineage facet (DIRECT or INDIRECT, with subtypes). No field for what was lost: a custom facet	No standard term: an attribute on the derivation	Contract (what should arrive)
6 Accountable owner	Ownership dataset facet (lists owners; accountability is a company rule)	Agent, wasAttributedTo (attribution, not accountability)	Accountable owner
7 Who can change it	No standard facet	Shows who did act (wasAssociatedWith, actedOnBehalfOf), not who may	Who can change it
8 Classification; retention or erasure duty	Tags dataset facet; no retention field	No standard term: an attribute	Classification

Ledger column	OpenLineage	W3C PROV	Medallion Data BOM
9 Read by	Jobs that list the dataset as input, if they emit events	Later activities that used the entity	Exposed to agents?
10 Cost per month; basis	No standard facet	No standard term	Own or rent (related, not the same)

OpenLineage and PROV can carry most columns; neither specification has a standard field for what was lost, who may change a record, or what it costs.

Source: OpenLineage specification (object model, run cycle, dataset facets), openlineage.io; W3C PROV-DM Recommendation, 30 April 2013; Medallion column: the Data Bill of Materials in From Bronze to Gold.

Two gaps and one mismatch matter.

The gaps. Neither specification has a standard field for column 5's "what was lost", for column 7's "who can change it", or for column 10's cost. OpenLineage's column-level facet would record that the status column was copied unchanged (its subtype for this is named IDENTITY, which is unrelated to the Ledger's IDENTITY loss code). That is a true description of hop 6. It is also why the facet is silent on the code table: the value did travel, and the meaning was never a column. PROV tells you which agents did act, not which may. Teams that automate can carry loss codes as a custom OpenLineage facet or a PROV attribute; the vocabulary of eight codes is the part worth standardizing inside a company.

The grain mismatch. OpenLineage records lineage at dataset, job and run grain, with column-level detail as a facet: about sixty run events a month for one nightly job (a START and a COMPLETE per run), and thousands across a platform. PROV has no fixed grain; you choose what an entity is. The Ledger records one row per hop of a named record or dataset. So one Ledger row summarizes many OpenLineage runs of one job, and points to them as evidence. Use the tools to keep columns 2 to 4 current and to prove the hop still runs as described; use the Ledger to say what each hop means for the reader.

Same names as the Medallion Data Bill of Materials

The Data Bill of Materials in *From Bronze to Gold*, the first Medallion book, describes each dataset at its latest stage of trust. The Ledger describes every hop that led there. Where they meet, the shared columns carry the same names, so one can be read against the other: **Accountable owner**, **Who can change it**, **Classification**, and the optional **Own or rent**. The Bill of Materials' "Source" is the Ledger's "From" on the last hop, and its "Exposed to agents?" is the agent part of "Read by". Think of the Bill of Materials as a snapshot of the last row, and the Ledger as the history behind it.

ERA CARD

Reader. Whoever reads after the hop: in 2021, dashboards; from 2025, an agent.

Security. Column 7 names every account that can change the record after each hop.

Cost. A row takes 30 minutes to two hours by hand (estimate); automation lowers columns 2 to 4 and part of 9.

AI. Rows marked MEANING or RIGHTS on a path to an agent are the first ones to fix.

Monday rule: fill one Ledger row by hand this week for the table your newest agent reads, and give column 5 a loss code.

CHAPTER THREE

Before the Computer: Tallies, Tokens and Ledgers



On 16 October 1834 two workmen spent the day feeding two cart-loads of old wooden tally sticks into the underfloor stoves beneath the House of Lords. The Exchequer had stopped using tallies in 1826 and wanted them gone. Visitors that afternoon noticed the floor was hot and smoke was seeping through it; the furnaces were put out at five, and at six the House of Lords was on fire. By morning both Houses of Parliament and most of the buildings on the site were destroyed. The records of a retired system had outlived the system, nobody owned their end, and their disposal burned down the building that had kept them. This chapter is about records before machines: what the clerk, the tally cutter and the bookkeeper already knew about keeping a promise to a future reader, and the one decision it changes: **every record you keep needs a stated reader, a stated owner and a stated end**.

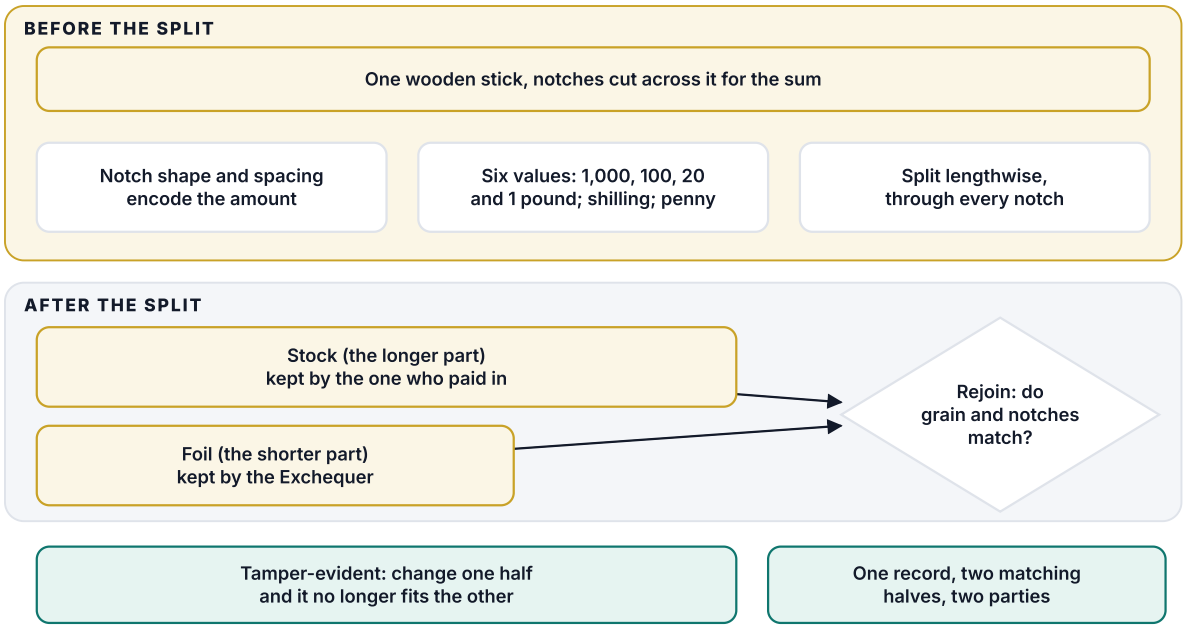
The split tally: a record two parties hold

A **tally** was a wooden stick that recorded a sum paid or owed. The English Exchequer, the royal treasury, used tallies to record deposits from the mid-twelfth century until 1826, according to the Science Museum Group, which holds a stock dated 1822. A clerk cut notches across the stick for the sum, with notches of different shapes and spacing standing for a penny, a shilling, a pound, twenty pounds, a hundred pounds and a thousand pounds. Then the stick was split lengthwise through the notches.

The two halves had names. The longer part, the **stock**, went to the one who had paid in. The shorter part, the **foil**, stayed with the Exchequer. When the debt was settled, the halves were brought together. UK Parliament's own account puts it in one line: split in two on issue of a debt, when confirmation of payment was made, the parts were married up again. If the grain and the notches matched, the two records agreed.

FIGURE 3.1

The split tally: one record, two halves, two keepers



The notches were cut before the split, so each party held half of the same record, and an altered half would normally fail to match the other.

Author's diagram based on the Science Museum Group collection (Exchequer tallies; stock dated 1822, object 1860-30/1) and UK Parliament, The Great Fire of 1834.

Look at the tally as a data architect would. It is one record held by two parties, each with a full copy of the amount and half of the physical evidence. A party that changed its half would normally be found out when the halves met: the record is **tamper-evident**, not tamper-proof. The check is cheap and needs no trusted third party. It also has a clear limit. The notches say how much; they do not say why the money was owed. Even this tidy record depended on context kept somewhere else.

The end of the tally is the other lesson. The system was retired in 1826, yet in 1834 the sticks were still in the building, waiting. When the Clerk of Works, an officer responsible for the buildings, was asked to burn them, he judged the two underfloor stoves of the House of Lords a safe and proper place. The method of destruction was chosen by whoever held the problem, not by a rule. Westminster Hall survived, UK Parliament records, because of firefighting and a change in the wind; so did the Jewel Tower, the Undercroft Chapel and the Cloisters and Chapter House of St Stephen's. Deletion is a hop too, with an owner, a method and a risk. Chapter Thirteen returns to it.

Tokens and tablets: counting before writing

Much older records were clay. Denise Schmandt-Besserat, an archaeologist at the University of Texas, argues that small clay **tokens** were used to count goods in the Near East from about 7500 BCE. In her account, tokens for unpaid debts were sealed inside clay envelopes, such as one from Susa dated about 3300 BCE, with the tokens pressed into the outside first so the contents could be checked without breaking the seal. Those impressions then replaced the tokens on flat tablets, around 3200 BCE, and incised signs followed at Uruk around 3100 BCE. She calls the move from token to impression the invention of writing, and holds that accounting came first.

The thesis is disputed, and the dispute is worth stating. Lucy Bennison-Chapman, writing in the *Cambridge Archaeological Journal* in 2019 after studying some 3,000 Neolithic clay objects, argues that the common reading is wrong: the objects were multifunctional, served several roles even within one site, and show no evidence of a unified symbolic system in the Neolithic. She accepts that simple counting was probably among their uses. This book takes no side on the origin of writing. It needs only the part both readings allow, that people counted goods with physical tokens, and one point of Schmandt-Besserat's reading: the clay envelope with its outside impressions let a later reader check the contents without trusting the person who held it.

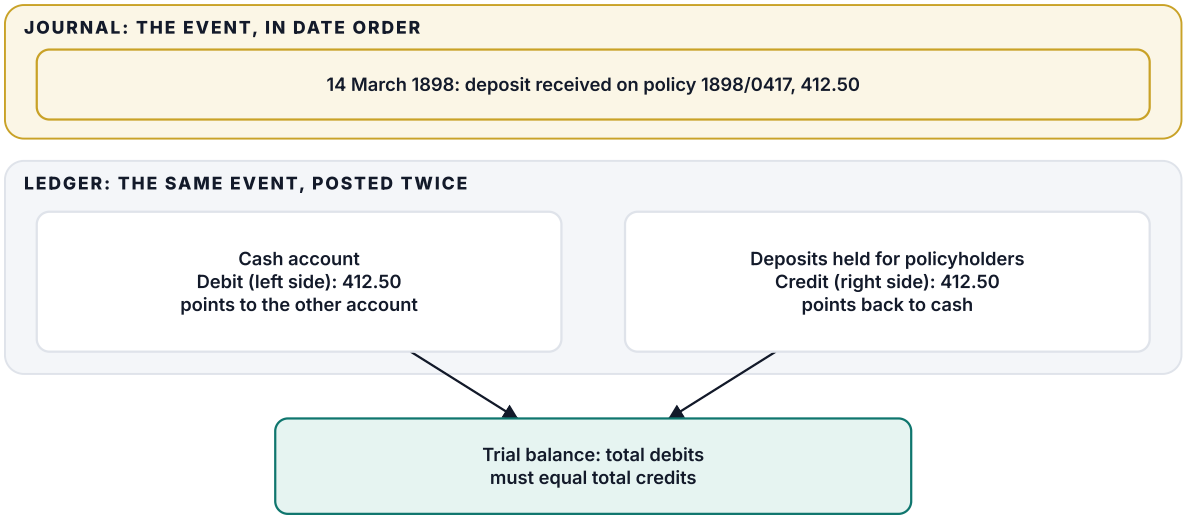
Double entry: the method of Venice, in print

The ledger is the record most of this book descends from. In 1494 Luca Pacioli, a teacher of mathematics, published the *Summa de arithmetica* in Venice, printed by Paganino de Paganini. One section, *Particularis de computis et scripturis*, describes double-entry bookkeeping as the method of Venice. The library of the Institute of Chartered Accountants in England and Wales holds a copy. Pacioli gave the earliest printed description known to historians of accounting; he did not invent the method: Benedetto Cotrugli wrote on double entry in a manuscript of 1458, which was printed only in 1573. Alan Sangster and Franco Rossi, publishing a new translation of Cotrugli's chapter in *De Computis* in 2018, note that the 1573 printed text had been severely altered, and that serious study of the chapter rests on manuscript copies.

The method is simple to state. Every event is written once in date order, in a journal, and then twice in the ledger: as a debit in one account and a credit in another, for the same amount. Because every entry has two sides, the sum of all debits must equal the sum of all credits. The **trial balance** tests that. A one-sided posting, or a debit and a credit that differ, shows as a difference. A transaction left out entirely, an equal error on both sides or a posting to the wrong account still balances.

FIGURE 3.2

Double entry: one event, two accounts, a check that must balance



Each transaction is written once in the journal and twice in the ledger, on opposite sides, so total debits must equal total credits; a one-sided or unequal posting shows, while a transaction left out entirely or posted to the wrong account may not.

Author's diagram of the double-entry principle printed by Pacioli (Venice, 1494); amounts from the book's illustrative thread.

Double entry is reconciliation built into the record. The tally checked two halves held by two people; the ledger checks two entries held in one set of books. Neither proves that the event was true. A fictitious sale entered on both sides balances. That limit is the subject of Chapter Fourteen, which starts from double entry as one of the oldest proofs in business and asks what it cannot prove.

Registers, audit and the census

A **register** is a bound book of numbered lines, one per thing: a policy, a birth, a ship, a deed. Its strength is order. Each line has a place, the next number is obvious, and a missing or inserted line is hard to hide in a bound volume. Corrections are made in the open, struck through and initialled, so the book carries its own audit trail. An **auditor** is a reader who does not trust the writer and checks the record against something else: the cash, the counterfoils, the other half.

The census is the register made mass. The first United States census took Monday 2 August 1790 as its official day. The National Archives records how it was done: the United States Marshal in each judicial district ran the count with assistants he appointed, and outside

Massachusetts there were no standard printed forms, so assistants used whatever paper they had. Only the head of each family was named. Everyone else became a number in a column: free white males of 16 and over, free white males under 16, free white females, all other free persons, and enslaved people. No schedules survive for the areas that are now Delaware, Georgia, Kentucky, New Jersey, Tennessee and Virginia.

Two features of that census return in every later era. The schema decided who was a person on the record and who was a count, which is a decision about meaning made by the form, not the clerk. And the record outlived its purpose: the 1790 schedules were made to apportion seats in the House (the Constitution apportioned direct taxes the same way) and are read now by historians and families, readers the marshals never imagined.

The perpetual policy and the register line

The thread of this book is a perpetual fire policy, and the practice is real. The Philadelphia Contributionship for the Insurance of Houses from Loss by Fire held its first organizational meeting on 13 April 1752. The company's own timeline dates the conversion of its policies to perpetual coverage to 1810; some later accounts give earlier dates. The Baltimore Equitable Society, incorporated by the Maryland Legislature in 1794, began with seven-year policies whose deposits were returned at expiry. The Maryland Center for History and Culture, which holds its records, says the Society moved to a perpetual plan shortly after 1857: a member paid one deposit, usually about 2% of the cover for a dwelling, owed no further premiums, and got the full deposit back on cancellation, whether or not claims had been paid.

A perpetual policy is a record built to be read for a century or more. That is why the insurer in this book could hold a 19th-century date in 2026, and why "no premium since 1898" is the correct state of the account, not a warning sign.

THE HOP · 0 · 1898 · APPLICANT AND THE INSURER'S SURVEYOR TO POLICY REGISTER, LINE 417 (INK, BOUND VOLUME)

Record	Policy 1898/0417 (perpetual fire policy), hop 0, 1898
From	Applicant and the insurer's surveyor
To	Policy register, line 417 (ink, bound volume)
Mechanism and schedule	Clerk writes the line by hand from the application and the survey report
Format change; what was lost or assumed	CONTEXT Ink; fields in ruled columns: number, date, insured, property, sum insured, deposit, remarks. **The surveyor's notes on the building (brick, slate roof, distance to the next warehouse) stay in a separate survey book**
Accountable owner	Company secretary
Who can change it	Register clerk; corrections initialled by the secretary
Classification; retention or erasure duty	Commercial record; kept for the life of the policy
Read by (people, apps, models, agents)	Clerks, the auditor, the board
Cost per month; basis	Clerk time, about 10 minutes a month (2 hours a year) for the register line and the annual audit (illustrative)

On 14 March 1898 the insurer issued policy 1898/0417 to a wool merchant, on a brick warehouse on the river front, against a single deposit of 412.50 dollars. The register clerk wrote it on line 417 of the 1898 register, in ink, from the application and the surveyor's report: number, date, insured, property, sum insured, deposit, remarks. The company secretary owned the book and initialled corrections. The surveyor's notes on the building, brick, a slate roof, the distance to the next warehouse, went into a separate survey book. Nothing was wrong with that; any reader of the register who needed them knew where the survey book was. The loss code is CONTEXT because the knowledge that joined the two books lived in the clerks, and the next hop would not carry it.

ERA CARD · BEFORE 1890

Reader. The clerk, the auditor, the other party to the tally; a person who could ask the writer, or the writer's successor, what an entry meant.

Security. Possession of the book or the stick; bound pages and initialled corrections; two-party copies that show tampering when they meet.

Cost. People: clerk time to write, copy, balance and audit; shelf space; and the cost of an end nobody planned, as Westminster found in 1834.

AI. Today a model reads these records as scans and OCR text. It sees the line, not the survey book beside it, and not the clerk who knew both.

Monday rule: for each of your five most important datasets, write down its oldest surviving form, where that form is kept, who owns it and when it may be destroyed; if any answer is "nobody knows", that is your first Ledger row.

CHAPTER FOUR

Cards and the Unit-Record Shop



In 1887, the year the Census Bureau's history gives for the end of the hand count of the 1880 United States census, Herman Hollerith was testing his new tabulating machine on city health records. IBM dates his first tests to 1886, on mortality statistics for Baltimore, Jersey City and New York City; the United States Patent and Trademark Office records that in 1887 he counted health statistics for Baltimore and the state of New Jersey. A clerk had read the register; now a machine read a card. The machine could not ask what a hole meant. It counted what the layout of the card told it to count, and nothing else. This chapter follows that change from Hollerith's census to the 80-column card and the tabulating departments that ran business on it for decades, and the decision it changes: **a record's layout is a schema, and every schema is written for a reader that cannot ask**.

Hollerith and the census

The Census Bureau's own history states the problem plainly: after the 1880 census, the census office was collecting more data than it could tabulate. (The office was temporary then; it became the permanent Census Bureau in 1902.) Before the 1890 count it ran a contest. Contestants processed 1880 census data from four areas of St. Louis, and the fastest would win the work. In the Bureau's account, capturing the data took Hollerith 72.5 hours against 144.5 and 100.5 for the other two contestants, and preparing it for tabulation took him 5.5 hours against their 44.5 and 55.5; the page does not pair the other two contestants' times. He won the contract to process and tabulate the 1890 census. The Patent and Trademark Office's account adds that the work was done in under three years and five million dollars below projected costs.

The machine read holes in paper cards. In the Census Bureau's description, pins passed through each punched hole, touched mercury below, closed an electrical circuit and advanced a dial. Hollerith's patents, issued in January 1889 as US 395,781, 395,782 and 395,783, covered what he called the art of compiling statistics and the apparatus for compiling statistics. The Bureau's history says modified versions of his machines stayed in

use there until computers replaced them in the 1950s. Hollerith formed the Tabulating Machine Company in 1896, and the company line that grew from it was renamed International Business Machines, IBM, in 1924.

The census mattered to this book for one reason: it moved the reading of a record from a person to a machine, at national scale. A census clerk in 1790 could read a scrawl. A tabulator in 1890 could read only a hole in an agreed position. The meaning moved out of the reader's head and into the agreement about positions.

The card is the schema

A punched card holds no labels. Columns 7 to 12 are just columns 7 to 12. What makes them "the date of issue" is a layout sheet, kept by the supervisor of the tabulating department, that says which columns hold which field, how many digits each field has, where an implied decimal point sits and what each code means. The card is the data; the layout sheet is the **schema**, the description of structure and meaning that a reader needs to interpret the data. The machine did not need the layout sheet. It was wired to it. The control panel of a tabulator was set up to add these columns, print those, and sort on that one. So the schema lived in three places: the paper layout sheet, the wiring, and the heads of the people who wrote both.

IBM's history gives the dates. Until the late 1920s, IBM cards had 45 columns of round holes. In 1928 IBM introduced a card with 80 columns of smaller rectangular holes and 10 rows for coding numbers; a modified version in 1930 had 12 rows. The card measured 7 3/8 by 3 1/4 inches. Its 80 columns became a unit of thought that outlived the card itself: about 80 characters to a line of program code, 80 columns to a record. IBM also gives a sense of the scale. For the 1935 Social Security contract it needed millions of cards; by 1937, 32 presses in Endicott, New York, were producing 5 million to 10 million cards a day; and in the mid-1950s cards were roughly 20% of IBM's revenue and 30% of its profit.

> *A card holds no labels. Columns 7 to 12 mean "date of issue" only because a layout sheet, a wired panel and a supervisor agree that they do.*

The unit-record shop

The business form of the card was the **unit record**: one card per unit of business, a policy, an employee, an invoice line. A department built around it, often called the tabulating department, ran a fixed sequence of work. Operators punched cards from source documents.

Machines sorted the deck on a field, and the tabulator read the sorted deck and printed totals and listings. The master deck, one card per policy, lived in steel cabinets; the monthly run read it from front to back.

Three properties of this shop set patterns that the rest of the book keeps meeting:

- **Sequential reading.** To find one policy, you pulled its card by hand or ran the deck. A question that needed a different order needed a sort first. Tape, in Chapter Five, kept this habit.
- **Transcription as a hop.** Every card was keyed by a person reading a document. What the operator could not punch, did not fit, or was told to leave out did not reach the machine.
- **Ownership by department.** The head of the tabulating department owned the layout, the deck and the code book. The policy department owned the business the codes stood for. Those were two different people, often on different floors.

The six questions, asked of a card

The six questions of Chapter One work on a card as well as on a cloud table, and asking them of a 1931 card shows where the later trouble was planted.

Where is it written? In two places that must agree: the card, and the layout sheet that gives the card meaning. A third copy of the meaning sits in the wiring of the machines. The ink register still exists, in a vault, and it is now the older and richer record, but nobody reads it any more.

What does it mean? Only what the layout says. A digit in column 13 is a status code; what the code means is in a typed code book kept by the tabulating department, not on the card and not on the layout sheet. Meaning has already moved off the record.

Who can change it? Any keypunch operator with a source document. A changed policy gets a new card, and the old card is pulled from the deck. Control comes from batch totals and from the supervisor, not from the record. There is no trail on the card itself of who punched it; the register's initialled corrections have no equivalent.

What does it cost? Columns, first. Every field is a trade against another. Then machine rent, card stock and people: operators, a supervisor, the clerks who read the reports. Saving two columns on a date was cheap in 1931 and expensive in 1999.

Who reads it now? The sorter and the tabulator, which read only what they are wired to read, and clerks who read printed totals, not cards. The record's direct reader is a machine for the first time in this story.

What proves it is true and current? Control totals: the count of cards and the sum of a field must match the totals from the source documents and from last month's run. That catches lost cards and mistyped amounts. It cannot catch a field that was never punched.

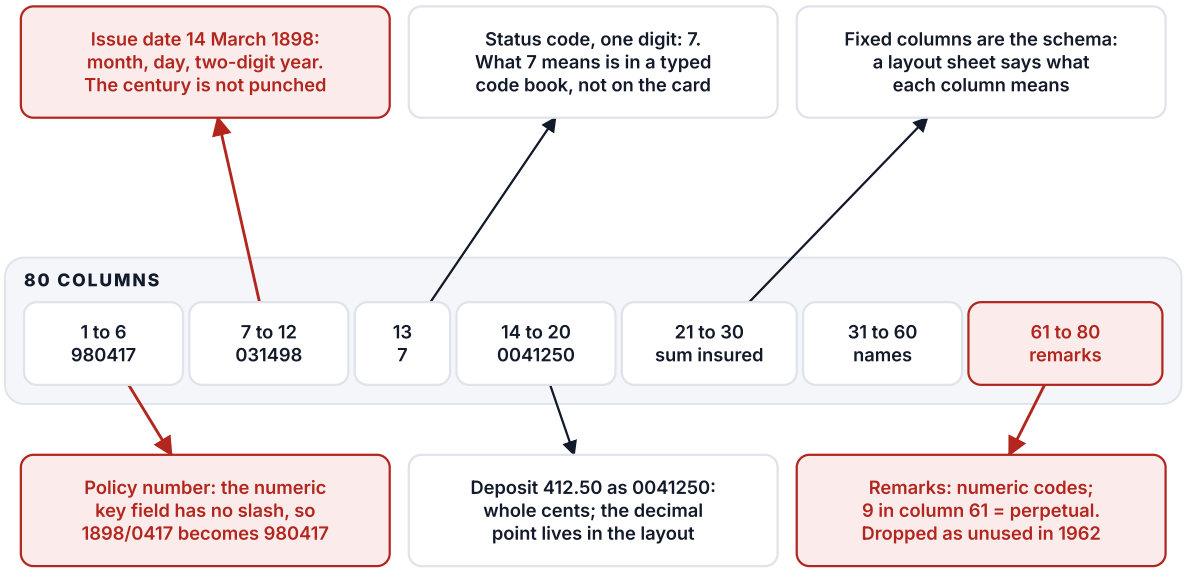
> **WHAT HASN'T CHANGED**
>
> Fixed-width records, where a position in a line carries the meaning and a separate document describes the positions, still move money every night: mainframe extracts, bank and payroll files, many government interchange formats. The layout sheet became a copybook or a PDF specification. The rule is the same as in 1928: the data is useless to a reader who does not have the layout.

The two-digit year

Columns were scarce. A card had 80, and every field competed for them: number, dates, codes, amounts, names. Writing the year as two digits saved two columns on every date on every card. The United States Senate's Special Committee on the Year 2000 Technology Problem, in its report *Investigating the Impact of the Year 2000 Problem*, explains the habit in these terms: to save memory, programmers represented four-digit years with two digits, the 19 was implied, much as personal checks once had 19 preprinted on the dateline; programs and data were entered on 80-column cards, each column one character, and "the cumbersome nature of punched cards encouraged using as few of them as possible." The committee was writing about computers, and the habit reached computers through cards. It was not an error when it was made. It was a choice that assumed a reader who knew which century it was. That reader is the one who leaves first.

FIGURE 4.1

The 1931 card decoded: the layout is the schema, and it has no slash and no century



Policy 1898/0417 on the tabulating department's master card (illustrative layout): fixed columns carry the meaning, the numeric key field has no slash, the year loses its century, and the remarks field holds a numeric code, 9 for perpetual, that a later program will call unused.

Author's diagram; illustrative card layout for the book's running example. Card format: IBM, The punched card (80 columns, introduced 1928).

THE HOP · 1 · 1931 · POLICY REGISTER TO IBM 80-COLUMN CARD, ONE CARD PER POLICY

Record	Policy 1898/0417 (perpetual fire policy), hop 1, 1931
From	Policy register
To	IBM 80-column card, one card per policy
Mechanism and schedule	Keypunch operator transcribes the register; cards sorted and tabulated monthly
Format change; what was lost or assumed	IDENTITY TIME Holes in 80 columns; numbers only in most fields; a fixed card layout is the schema. **Key 1898/0417 cannot be punched with a slash: renumbered 980417; the year shrinks to two digits to save columns**
Accountable owner	Head of the tabulating department
Who can change it	Keypunch operators; the card layout set by the tabulating supervisor
Classification; retention or erasure duty	Commercial record; master deck kept in steel cabinets
Read by (people, apps, models, agents)	Sorter and tabulator; accounting clerks reading printed reports
Cost per month; basis	Card stock, machine rental from the vendor, two operators (illustrative)

In 1931 the insurer's new tabulating department converted its policy registers to cards, one card per policy, punched by keypunch operators from the ink registers and sorted and tabulated monthly. The register key 1898/0417 could not be punched as written: the six-column numeric key field had no place for a slash, and the year was cut to two digits to save columns. The policy became 980417. From then on new policies were numbered in a separate seven-digit series, so 980417 stayed unique until the merger of 1999. Its issue date became month, day and a two-digit year: 031498. The register's remarks column, where the clerk had written PERPETUAL, became a one-digit status code, 7, defined in the tabulating department's typed code book; the remarks columns of the card carried a numeric remark code, and code 9 in column 61 marked a perpetual policy a second time. The card layout, set by the tabulating supervisor, was now the schema, and the head of the tabulating department owned the deck. Two losses entered the Ledger that year: IDENTITY, because the key was renumbered, and TIME, because the century was no longer on the record. In 1931 nobody at the insurer was likely to misread 98; every reader knew the company was founded in 1898. In 1999 a program would read it as 1998.

ERA CARD · CARDS, 1890 TO THE 1950S

Reader. The sorter and the tabulator, wired to a layout; and accounting clerks who read the printed reports, not the cards.

Security. Physical: locked cabinets, counted decks and control totals on each run. Any keypunch operator could create a card; the check was a batch total, not a person.

Cost. Card stock, machines rented from the vendor, keypunch operators and the supervisor; columns were the scarce resource, and design saved them.

AI. A model reading a card image or its digits sees 980417 and 98 with no layout sheet. Without the schema it will guess the field and the century, and the guess will sound sure.

Monday rule: for your oldest system still in service, find the layout of its core record and check whether any field drops a century, a sign, a decimal point or a key separator; record each one as a known loss before someone feeds that field to an assistant.

CHAPTER FIVE

Tape, Disk and the First Databases



On 31 March 1951, by the Census Bureau's own history, the Bureau signed for UNIVAC I (other histories call that day the machine's acceptance), and on 14 June Census officials attended its dedication at the Eckert-Mauchly laboratory in Philadelphia. The Bureau's history says it was soon used to tabulate part of the 1950 population census and all of the 1954 economic census. Sixty years after Hollerith, the census again moved its records to a new reader: this time a stored program that read magnetic tape. Over the next twenty years records moved from cards to tape, from tape to disk, and from files owned by one program to databases shared by many. Each move needed a conversion, and each conversion was a hop. This chapter covers that era and the decision it changes: **a conversion run is a transformation with an owner and a loss, and it gets a Ledger row like any other**.

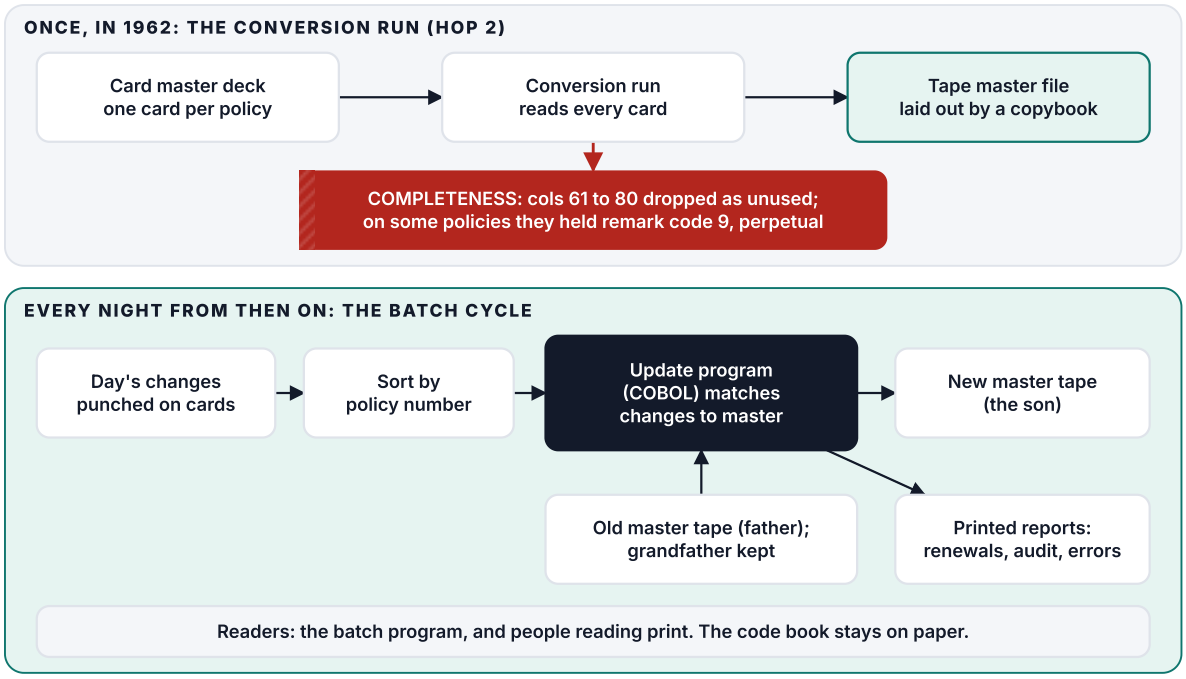
Tape, the batch cycle, then disk

UNIVAC I read and wrote data on the UNISERVO tape drive, which the Computer History Museum lists among the events of 1951; its tape was metal, not plastic. Tape held far more than a card cabinet and read far faster, but it kept the card's habit: it was read from start to end. To change one policy you read the whole master file and wrote a new one.

That shape is the **batch cycle**. During the day, changes were punched on cards. At night they were sorted into the same order as the master file, by policy number. An update program read the old master tape and the sorted changes side by side, applied each change to the matching record, and wrote a new master tape and printed reports: renewals due, audit listings, rejected changes. The old tape was kept, and so was the one before it, so a failed run could be redone from the grandfather, father and son generations. The pattern was robust and easy to audit. It also meant the record had two kinds of reader: programs, at night, and people holding printouts.

FIGURE 5.1

The batch cycle: one conversion run drops a field, then every night rewrites the master



In 1962 the insurer's conversion run copied the card deck to tape without the remarks columns; from then on a nightly COBOL update read the old master and the day's changes and wrote a new master and printed reports.

Author's diagram based on the book's running example; a generic sequential master-file update, illustrative.

In 1956 IBM introduced the 305 RAMAC, which IBM's history calls the first computer to use a random-access disk drive. The Computer History Museum describes its disk unit as 50 magnetically coated metal platters storing about 5 million characters, giving real-time random access that tape and punched cards could not. IBM's own records are not consistent about the day in September 1956 on which the system was announced, so this book gives the year. The point is not the capacity. It is that, for the first time in this story, a program could go to one policy without reading every policy before it. That made a new reader possible: a clerk at a terminal asking about one record now, not a report about all records tomorrow.

COBOL and the copybook

Programs needed a language that business people could read and that would run on more than one maker's machine. The Computer History Museum records that on 28 May 1959 a committee of researchers from computer manufacturers and the Pentagon formed to

develop COBOL, the Common Business Oriented Language; its initial specification was published by the Department of Defense in April 1960. COBOL describes records in a **data division**: level numbers, field names and a picture clause, `PIC`, that gives each field's digits, sign and implied decimal point.

A record layout used by many programs was kept once, in a library member called a **copybook**, and pulled into each program by the COPY statement. IBM's Enterprise COBOL reference describes COPY as bringing prewritten library text into a program at compile time. The copybook is the card layout sheet made executable. It is also the moment the schema became code: a field that a programmer leaves out of the copybook is invisible to every program that copies it.

Zoned and packed decimal

Business money on mainframes is usually held as decimal, not binary floating point. On the insurer's first computer of 1962 it was held as decimal digits in a six-bit BCD character code; from IBM's System/360, announced on 7 April 1964, it comes in two forms, and IBM's *System/360 Principles of Operation* defines both. In the **zoned** format, each digit takes one 8-bit byte: the low four bits hold the digit and the high four bits, the zone, make it a printable character; in the rightmost byte the zone position carries the sign. The manual says the zoned format is required for character-set sensitive I/O devices and is not used in decimal arithmetic. In the **packed** format, two decimal digits share one byte, four bits each, and the rightmost byte holds the last digit and the sign. Arithmetic is done on packed data, and the PACK and UNPACK instructions convert between the two.

Two common shortcuts are wrong. Packed decimal is binary-coded decimal: each four-bit group holds one digit and the last group holds the sign. It is a numeric format, not text, so "EBCDIC packed decimal" is a misnomer. Zoned decimal is the form that belongs to the character set: in EBCDIC, the manual's preferred codes are a zone of 1111 and signs of 1100 for plus and 1101 for minus.

TYPE CONTRACT: THE DEPOSIT OF POLICY 1898/0417 IN THE IMS DEPOSIT SEGMENT, FROM 1974

Type, precision and rounding. a signed fixed-point decimal of seven digits, two after the point, declared `PIC S9(5)V99`; the point is not stored, and the V in the copybook says where it is; no rounding at this hop. Value: 412.50 dollars, the deposit held since 1898.

Encoding and serialization. the USAGE clause picks the bytes, and a field is one or the other. DISPLAY, the default, gives the zoned form, 7 bytes in EBCDIC: F0 F0 F4 F1 F2 F5 C0, each digit in one byte with zone F, the sign C (plus) in the zone of

the last byte. COMP-3 (PACKED-DECIMAL) gives the packed form, 4 bytes: 00 41 25 0C, two digits per byte, sign C in the last half-byte. The 1962 tape held the same seven digits in six-bit BCD, a code that predates EBCDIC.

Unit or currency. US dollars; no currency code in the field.

Time basis. the amount at the date of deposit; no date in the field.

Identity scope. the deposit of one policy, 1898/0417, held in that policy's deposit segment.

Absence. none defined. Zero means "no deposit"; a policy without a deposit has 0000000, which a later reader may confuse with "unknown".

Authority. the copybook comments and the policy department hold the currency and the date; nothing in the record does.

Chapter Eleven follows the same 412.50 through every later representation, including binary floating point, where 412.50 happens to be exact and a cent either side is not.

The first databases: IDS, IMS and CODASYL

A file belonged to the program that wrote it. A **database**, in the sense this book uses, is a store of records with its own description, shared by many programs through a manager that controls access. Three systems set the shape of that idea.

At General Electric in the early 1960s, Charles Bachman built the Integrated Data Store, IDS. The Computer History Museum, which made Bachman a Fellow, calls IDS "the first disk-based database management system used in every day production", and credits him with the network data model and the data structure diagrams known as Bachman diagrams. He received the ACM Turing Award in 1973.

IBM's history tells the IMS story through the space program. In IBM's account, in 1963 IBM's Uri Berman teamed with Peter Nordyke of North American Aviation, later Rockwell, on a parts tracker for the lunar missions, and a team from IBM, Rockwell and Caterpillar built the original release; IBM says the first version shipped in 1967. The system, then called ICS/DL/I, broadcast its first "READY" message on 14 August 1968 at Rockwell's Space Division in Downey, California. For the commercial market IBM renamed the technology Information Management System and, in IBM's words, "in 1968 announced its release on mainframes". The commercial product had two parts: a database manager built on a **hierarchical** model, records as a tree of segments under a root, and a transaction manager for high volumes of short requests.

Bachman's network model became a standard proposal. The Data Base Task Group of CODASYL, the Conference on Data Systems Languages, which had produced COBOL, published its report in April 1971, specifying a data description language and a data manipulation language for network databases. In both IMS and the CODASYL model, a program navigated: it found a record, then followed a path to the next one. The meaning of the data was split between the database description, which named the records and the paths, and the programs, which knew why each path mattered. Codes inside the records, such as a one-digit status, were usually described nowhere the database could see. Chapter Six tells how the relational model set out to separate the logical view from the access path.

What the hierarchy assumed

A hierarchical database fits an insurer well. A policy is a root; under it sit the insured building, the deposit, the endorsements and the claims. A program asks for policy 980417 and walks down to its deposit segment. The tree is the business's own picture of the record, and it is fast for the questions the tree was built for.

It is slow, or impossible, for other questions. "All deposits above 400 dollars on brick warehouses" means walking every tree, unless someone had built an index for that question. The usual answer was a new program, a new path, or a nightly extract to a file sorted the other way. Every one of those extracts was a copy with its own layout and its own reader, and most of them never appeared on any diagram. The hop count of a record grew in this era without anyone deciding to grow it.

The hierarchy also fixed one assumption into the structure: that in the physical tree each child has one parent. A building insured under two policies, or a deposit moved when ownership changed, had to be copied or modelled with a workaround. Chapter Six shows the relational answer, and Chapter Eleven shows what the copies did to identity.

> **WHAT HASN'T CHANGED**
>
> IMS is still a product. IBM's history of it says that companies around the world continue to rely on it for everyday business needs. In this book's thread, when the warehouse of 2021 and the agent of 2025 read policy data, the layout they inherit still carries field choices made in the copybook of 1962, the dropped remarks field among them.

THE HOP · 2 · 1962 · CARD MASTER DECK TO TAPE MASTER FILE ON THE INSURER'S FIRST COMPUTER

Record	Policy 1898/0417 (perpetual fire policy), hop 2, 1962
From	Card master deck
To	Tape master file on the insurer's first computer
Mechanism and schedule	One conversion run reads every card; COBOL batch programs then update the tape each night
Format change; what was lost or assumed	COMPLETENESS Fixed-length records described by a COBOL copybook; six-bit BCD character code on tape; numbers as decimal digits. **The remarks field (card columns 61 to 80) is dropped as 'unused'; it held remark code 9 in column 61, which marked perpetual policies**
Accountable owner	Data processing manager
Who can change it	Programmers who change the copybook; the night operator
Classification; retention or erasure duty	Commercial record; tape generations kept (grandfather, father, son)
Read by (people, apps, models, agents)	Batch COBOL programs; printed renewal and audit reports
Cost per month; basis	Computer lease, tape reels, programmers and operators (illustrative)

In 1962 the insurer installed its first computer. A one-off conversion run read every card in the master deck and wrote the tape master file, laid out by a COBOL copybook, its characters in the machine's six-bit BCD code and its amounts as decimal digits. The programmer who wrote the copybook left out card columns 61 to 80, the remarks field, as unused: no report printed it, and no program had ever read it. For some policies, including 980417, it held remark code 9 in column 61, the card's second mark of a perpetual policy. Status 7 survived the run, so the policy was still marked; from that night the digit was its only mark, and what the digit meant was on paper. From that night on, COBOL batch programs updated the tape and printed renewal and audit reports, and the data processing manager owned the file. The loss is COMPLETENESS: a field dropped, quietly, because the reader that decided what was "used" was the set of existing programs, not the people who had written the cards.

THE HOP · 3 · 1974 · TAPE MASTER FILE TO IMS HIERARCHICAL DATABASE ON MVS

Record	Policy 1898/0417 (perpetual fire policy), hop 3, 1974
From	Tape master file
To	IMS hierarchical database on MVS
Mechanism and schedule	Conversion programs load policy, building and deposit segments into an IMS hierarchy
Format change; what was lost or assumed	MEANING Policy root segment with child segments; status as a one-digit code. **Status codes are kept as digits; the code book that says what 7 means stays on paper in the policy department**
Accountable owner	Data processing manager; the database administrator (a new role)
Who can change it	DBA and application programmers; online clerks through screens
Classification; retention or erasure duty	Commercial record
Read by (people, apps, models, agents)	Online clerks on terminals; batch reports
Cost per month; basis	Mainframe, IMS licence, DBA and programmers (illustrative)

In 1974 the insurer moved its policies from tape to an IMS hierarchical database on MVS. Conversion programs loaded each policy as a root segment, with child segments for the building and the deposit. Online clerks could now see one policy on a screen, and a new role, the database administrator, shared ownership with the data processing manager. The status of policy 980417 came across as it had been since the 1931 card, a one-digit code: 7, now in EBCDIC with the amounts in zoned and packed decimal. The code book that said 7 meant "perpetual, deposit held, no premium due" stayed on paper in the policy department, where the clerks who used the screens could ask about it. The loss is MEANING. It cost little in 1974, while the clerks and the code book sat on the same floor. It sets up the wrong answer of 2026, which the warehouse load of 2021 completes, three hops later.

ERA CARD · TAPE, DISK AND THE FIRST DATABASES, 1951 TO THE 1970S

Reader. The batch program at night and the person with the printout; then, with disk and IMS, the clerk at a terminal asking about one record.

Security. The machine room door, tape library logs and job control; then the database administrator, a new role with power over every record, and the programs that could write.

Cost. Leased computers, tape reels, disk, licences, and above all programmers and operators; every conversion was a project, priced in people.

AI. A model reading a copybook, a dump or an unloaded file sees PIC clauses, packed bytes and one-digit codes. It can decode the bytes exactly. It cannot know what a code means unless the code table travels with the data.

Monday rule: list every conversion your core records have been through, and for each one ask the team for the field list before and after; any field that disappeared, or any code whose table was not carried, goes into the Ledger this week.

CHAPTER SIX

The Relational Bet



In June 1970 an IBM researcher in San Jose, Edgar F. Codd, opened a paper in *Communications of the ACM* with a sentence that reads today like a requirement for every data platform: "Future users of large data banks must be protected from having to know how the data is organized in the machine." Twenty-nine years later, in 1999, the insurer of this book's thread loaded policy 1898/0417 into a relational database for the first time. The same project merged two insurers' policies and applied a rule for the year 2000. The program no longer needed to know where the policy was stored. The project also keyed the paper code book into a table, so that for the first time the database could say what status 7 meant. Nothing obliged the next copy of the data to carry that table. This chapter covers the relational bet: what Codd separated, what the products and the standard made of it, the transaction and its log, and why a migration is a hop. The decision it changes: **every migration and merger gets a Ledger row and a reconciliation before cutover, not after**.

Codd's bet: the program should not know the path

Codd's paper, "A Relational Model of Data for Large Shared Data Banks" (CACM volume 13, number 6, June 1970, pages 377 to 387), names its problem in the first section. It calls it **data independence**: "the independence of application programs and terminal activities from growth in data types and changes in data representation". He argues that the systems of the day, which gave users "tree-structured files or slightly more general network models of the data", tied programs to the way data was stored.

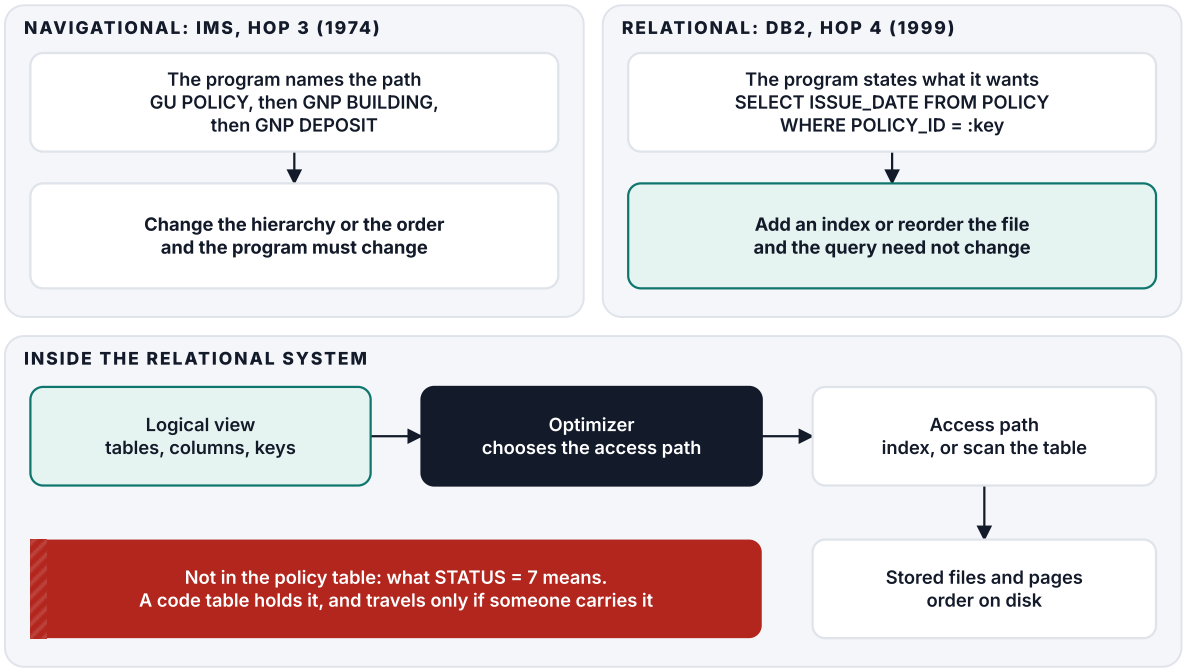
He lists three dependencies "which still need to be removed": **ordering dependence** (a program assumes records arrive in the order they are stored), **indexing dependence** (a program relies on an index that someone may later drop), and **access path dependence** (a program walks a fixed route through a hierarchy or network to reach a record). Each one means that a change made for performance or growth breaks a program that was correct the day before.

The insurer's IMS database of 1974 (Chapter Five, hop 3) shows the third dependency plainly. A program that wanted the deposit on a policy asked IMS for the policy root segment and then stepped down to its child segments in hierarchic order, with calls such as GU (get unique) for the root and GNP (get next within parent) for its dependents, which IBM's IMS documentation still describes. The route was in the code. Reorganize the hierarchy and the program had to change.

Codd's answer was to describe data "with its natural structure only", as relations: tables of rows, with columns that have names and keys that identify rows by value. In the systems built on the model, a program states what it wants, and the database system works out how to get it. The **logical view** (tables, columns, keys) is separated from the **access path** (which index, which scan, which order on disk). That separation is the bet, and Figure 6.1 draws it.

FIGURE 6.1

Data independence: the program states what it wants, the system chooses the path



In a navigational database the program carries the access path; in a relational one the optimizer chooses it. Neither schema says what a code value means unless someone models a code table and carries it.

Author's diagram based on Codd, CACM 13(6), June 1970, section 1.2, and IBM IMS application programming documentation.

Be precise about what was separated. Data independence protects a program from changes in storage and access. It does not tell a reader what a value means. In a relational table, STATUS is a column of small integers, and 7 is a valid value. The model has room for a code table that says what 7 means, and a foreign key that makes every status point to a row in it, but someone has to create that table, fill it and carry it on every later hop. At the insurer the status digit had sat in a fixed card column since 1931, and the code book that said what each digit meant had been a typed paper document since then; in 1999 the conversion team keyed it into a DB2 table named STATUS_CODES. The relational bet removed the path from the program. It did not move meaning into the schema by itself: people did, and only for the copy they built.

The standard moved slowly in that direction. The first ISO edition of SQL appeared in 1987. Two years later ISO/IEC 9075:1989 replaced it under the title "Database Language SQL with integrity enhancement". Integrity means the rules a database enforces on what a row may contain. A code table with a foreign key is such a rule. It makes the database refuse a status that is not in the code table, and it gives every reader of that database one place to look the meaning up. It costs a table, a constraint and an owner who keeps the table current. A team that believes the application "knows" what the codes mean is tempted to skip it. The application was one reader. It was never the last.

> *Data independence took the access path out of the program. It did not put the meaning of a code into the database.*

From a paper to products and a standard

Codd's paper said that "implementations of systems to support the relational model are not discussed". Others built them. IBM's own history dates the start of System R, a research project to test the model at industrial scale in San Jose, to 1973. In May 1974 Donald Chamberlin and Raymond Boyce presented "SEQUEL: A Structured English Query Language" at an ACM workshop on data description, access and control: a query language for a relational database, meant for both programmers and occasional users. Oracle's SQL reference records that SEQUEL "later became SQL".

At the University of California, Berkeley, Michael Stonebraker, Eugene Wong and colleagues built Ingres. Their paper "The Design and Implementation of INGRES" (ACM Transactions on Database Systems, volume 1, number 3, September 1976) describes it as of March 1976: a relational system written mostly in C, running on UNIX on PDP-11 computers, with its own query language. Two lines of work, one at IBM and one at a university, proved that the model could run on real machines.

Products followed. The dates below are the ones each source gives; where a "first" appears, it is the vendor's own description.

Year	Event	Source and wording
1970	Codd's relational model published	CACM 13(6), June 1970
1973	System R project starts at IBM San Jose	IBM history
1974	SEQUEL presented (Chamberlin and Boyce)	ACM SIGFIDET workshop, May 1974
1976	Ingres described as running on UNIX	ACM TODS 1(3), September 1976
1979	Oracle Version 2 released by Relational Software, Inc.	Oracle's SQL reference: Relational Software "introduced the first commercially available implementation of SQL"
Early 1980s	SQL/DS from IBM's Endicott laboratory, for VSE and then VM	Harris and Nicol, "SQL/DS: IBM's First RDBMS" (IEEE Annals, 2013)
1983	DB2 announced for MVS (7 June); Version 1.1 released 1985	IBM: "announced in 1983", Version 1.1 "launched on MVS in 1985"
1986	ANSI X3.135-1986, Database Language SQL, approved 16 October	ANSI standard, adopted by the US government as FIPS PUB 127
1987	ISO 9075:1987, the international SQL standard	ISO catalogue, first edition June 1987

Three points from that table matter for a CTO. First, the relational database reached IBM's mainframes as products in the early and mid-1980s, SQL/DS on VSE and VM and then DB2 on MVS, more than ten years after the paper. Companies did not move their master files at once; many, like the insurer, kept IMS for decades and moved only when a larger project, such as a merger, paid for it. Second, the standard arrived after the products, so each product kept dialects of its own, and SQL written for one engine often needs changes for another. Third, despite the dialects, SQL gave queries one family of language that people

could read. An analyst, an auditor or an actuary with access, a query tool and some training could ask the database a question that no program had been written for. The query, not only the program, became a reader of the record.

The transaction and its log

The relational model said what the data looks like. It did not say what happens when two people change it at once, or when the machine stops in the middle of a change. That answer came from transaction processing.

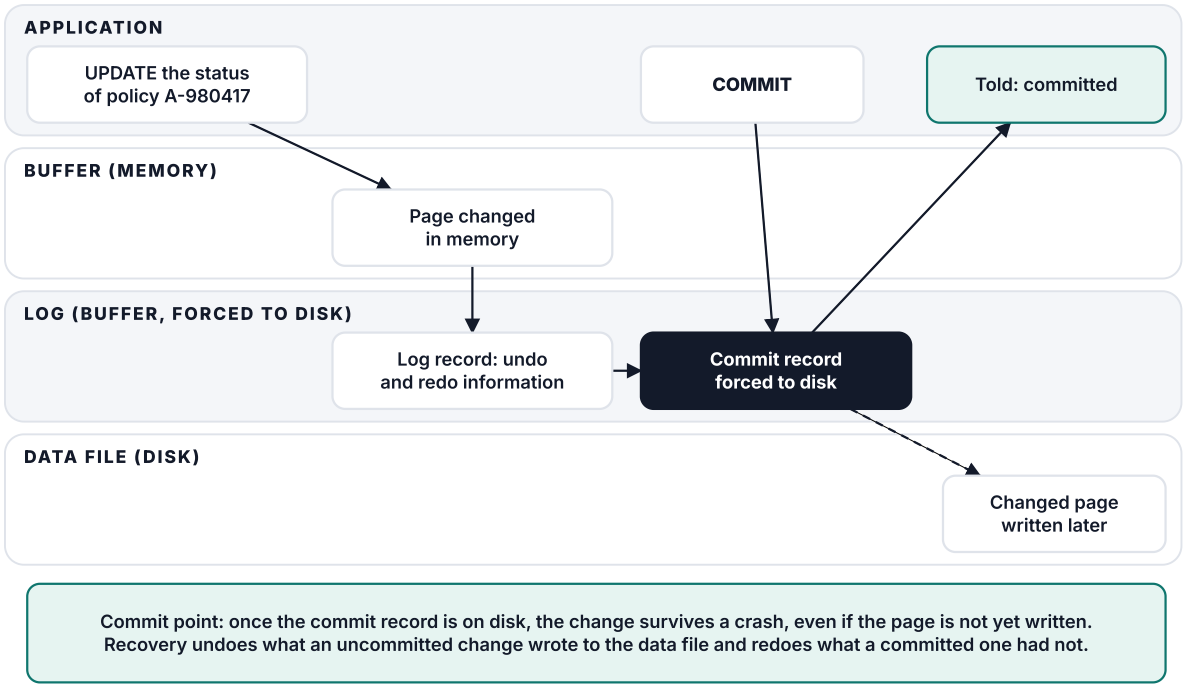
In "The Transaction Concept: Virtues and Limitations", presented at the Seventh International Conference on Very Large Data Bases in September 1981, Jim Gray of Tandem Computers starts from contract law: parties negotiate and then make a deal binding. He defines a transaction as "a transformation of state which has the properties of atomicity (all or nothing), durability (effects survive failures) and consistency (a correct transformation)". Each transaction has exactly one of two outcomes: committed or aborted.

In December 1983 Theo Haerder and Andreas Reuter published "Principles of Transaction-Oriented Database Recovery" in ACM Computing Surveys (volume 15, number 4). They add isolation to Gray's three properties and name the set: "These four properties, atomicity, consistency, isolation, and durability (ACID), describe the major highlights of the transaction paradigm". They also consider whether a system supports transactions "to be the ACID test of the system's quality".

The same paper states the two rules for writing the log, where the database records the information it needs to undo or redo each change, ahead of writing the changed data. Undo information "must be written to the log file before the corresponding updates are propagated to the materialized database", which "has come to be known as the 'write ahead log' (WAL) principle", citing Gray. Redo information must be on the log "before EOT is acknowledged to the transaction program", that is, before the program is told the transaction ended. Figure 6.2 draws one update to policy A-980417 (its key after the 1999 merger, described below) under those rules.

FIGURE 6.2

A transaction and its log: the commit record, not the data file, holds the promise



Undo information is logged before the changed page is written, and the commit is acknowledged only once the commit record is on disk, so recovery can undo or redo from the log.

Author's diagram based on Gray, "The Transaction Concept" (VLDB 1981) and Haerder and Reuter, ACM Computing Surveys 15(4), 1983.

Read the figure as a lineage diagram, because it is one. The log is the database's own ordered record of what changed, written before the change counts; it names the transaction, though not always the person or the reason. For the insurer's DB2 tables it held every change to a policy row in the order written, including work later rolled back, with commit records marking which units of work counted, for as long as the log was kept. Two later chapters build on it: change data capture reads this log to publish policy events (Chapter Nine, hop 7), and the Proof chapter asks what a log can and cannot prove (Chapter Fourteen). The log keeps the history of a value. It does not keep the reason for it, unless a person or a program writes the reason down.

WHAT HASN'T CHANGED

In a database configured for durable commits, the two log rules that Haerder and Reuter restated in 1983, crediting Jim Gray, still decide when a change is safe: undo information reaches the log before the changed page reaches the database, and the commit record reaches durable storage before the program is told the change is done. A cloud database may keep its log in another place and in more copies; where it follows these rules, the log still comes first.

Migrations and mergers are hops

The relational era made one kind of hop routine: the **conversion**. A conversion reads every record from an old system, transforms it and loads it into a new schema. Mergers make conversions larger, because two companies' records, built by different people with different codes and keys, must fit one schema. The insurer's 1999 project did both at once: IMS to DB2, two insurers to one policy system, and a fix for the year 2000.

Three things tend to break in such a project, and the Ledger has a loss code for each.

- **Keys collide (IDENTITY).** The two insurers' policy numbers overlapped, so one number could name two different policies. The conversion team added a prefix: A for the old insurer's policies, B for the other company's. Policy 1898/0417, already renumbered once in 1931, became A-980417. Every report, letter and extract that quoted the old number now pointed at a key that no longer existed as written.
- **Codes keep their digits and lose their meaning (MEANING).** In any merger, two companies' status codes need not mean the same thing. When both are loaded into one STATUS column, a 7 from one company and a 7 from the other look identical. At the insurer the conversion team keyed the paper code book into STATUS_CODES, which gave the merged company one place to look the meaning up. The table lived beside the policy table, not inside it, so it would travel only if a later project chose to carry it.
- **Time is reinterpreted (TIME).** A conversion program decides how to read every date it moves. In 1999 that decision had a name: the century window.

A conversion has one control that every team can afford: **reconciliation**, the comparison of what left the old system with what arrived in the new one. Count the rows by type and status on both sides. Sum the money columns, such as deposits, on both sides. Count the converted dates by decade and compare them with an independent source, such as the register or the number of policies the company actually wrote in each decade; the source file alone cannot help, because its years had two digits. The 1990s would have shown a block of policies too many. List every key that was changed and keep the old key beside the new one. None of

these checks needs new technology; each needs someone to decide, before cutover, what must match and who signs when it does not. Reconciliation is older than computers (Chapter Fourteen traces it to double entry), and conversions are where it pays most.

TYPE CONTRACT: POLICY ROW AFTER HOP 4

Type, precision and rounding. key: character string; issue date: SQL DATE with a four-digit year; status: small integer; deposit: decimal with two places.

Encoding and serialization. DB2's own column formats; the conversion program, not the reader, turned the IMS fields into them.

Unit or currency. deposit in dollars; key and status have none.

Time basis. issue date: for two-digit sources the century comes from the conversion's window rule, not from the source.

Identity scope. key: prefix plus six digits (A-980417); the prefix means "source company" and is not part of the original number.

Absence. no issue date means "not converted", not "unknown".

Authority. status meaning in the lookup table STATUS_CODES, keyed in from the paper code book during this conversion (7 = perpetual, deposit held, no premium due).

The characteristic reader in this era was the query, and the person or report that ran it. A SQL report that counted policies by issue year, or selected those with no premium paid in the last decade, read the record exactly as converted. It did not hesitate, and it could not ask. Unlike the batch program before it, a query was cheap to write, so many more people asked the record many more questions. That is the relational era's gift and its exposure: more readers, each trusting the schema.

Y2K: the window that read 98 as 1998

The year 2000 problem has one home in this book, here. The US General Accounting Office put it plainly in September 1997, in *Year 2000 Computing Crisis: An Assessment Guide* (GAO/AIMD-10.1.14): "For the past several decades, systems have typically used two digits to represent the year, such as "97" representing 1997, in order to conserve on electronic data storage and reduce operating costs. With this two-digit format, however, the Year 2000 is indistinguishable from 1900". Chapter Four showed where the two-digit year came from: card columns were scarce.

Teams had two broad choices. They could expand every stored year to four digits, which meant changing files, records and every program that read them. Or they could keep two-digit years and add a rule that decides the century when a program reads them. IBM's COBOL for AIX documentation defines that rule: "A century window is a 100-year interval, such as 1950-2049, within which any two-digit year is unique." With such a window, 49 is read as 2049 and 50 as 1950. Windowing was faster and cheaper, which is why many teams chose it. The cost was large either way: GAO testified in June 1999 that estimated federal Year 2000 costs had risen "from $2.3 billion in February 1997" to "$8.7 billion".

The insurer's 1999 conversion used a window of that shape: 00 to 49 means 20xx, 50 to 99 means 19xx. For most policies that was right, because most policies in force had been written after 1950. For a perpetual policy written in 1898, it was wrong. The two-digit year 98 in the issue-date field was read as 1998, and it matched the 98 at the head of the key, so nobody doubted it. The new DATE column stored 1998 with four confident digits. The window solved the century problem for records of the twentieth century. It created one for the records it was never designed for, and nothing on the row said that a rule, not a source, had supplied the century.

That is the general lesson of Y2K for lineage. The fix was a reader's assumption applied at a hop. Once the four-digit year was stored, the assumption disappeared from view. A later reader, human or model, had no way to tell a year that was written from a year that was inferred, unless the conversion recorded which rule it applied and to which rows.

THE HOP · 4 · 1999 · IMS DATABASE TO DB2 FOR OS/390 ON THE MAINFRAME (NEW POLICY SYSTEM AFTER THE MERGER)

Record	Policy 1898/0417 (perpetual fire policy), hop 4, 1999
From	IMS database
To	DB2 for OS/390 on the mainframe (new policy system after the merger)
Mechanism and schedule	Merger conversion: both insurers' policies loaded into one DB2 schema; Y2K date windowing applied in the same project
Format change; what was lost or assumed	TIME IDENTITY Relational tables; issue date as DATE; key widened with a prefix; the paper status code book is keyed into a DB2 lookup table (STATUS_CODES) for the first time. **98 is read as 1998 by the date window; the merged insurer's numbers collide, so keys get prefixes (A-980417)**
Accountable owner	Head of policy administration
Who can change it	DBAs, the conversion team, batch IDs
Classification; retention or erasure duty	Commercial record; state record retention rules apply
Read by (people, apps, models, agents)	Policy system screens; SQL reports; actuaries' extracts
Cost per month; basis	Conversion project, mainframe, DB2 licence, staff (illustrative)

In 1999 policy 1898/0417 crossed hop 4. The merger conversion loaded both insurers' policies from IMS into one schema in DB2 for OS/390, and the Y2K date window ran in the same project. The policy's key gained a prefix to avoid a collision with the other company's numbers and became A-980417. Its issue year, two digits since the 1931 card, was read through the window as 1998. Status 7 arrived as a digit, as it had in IMS, and for the first time its meaning sat in the same database, in the STATUS_CODES table keyed in from the paper code book. The head of policy administration owned the new tables; DBAs, the conversion team and batch IDs could change them; policy screens, SQL reports and actuaries' extracts read them. For twelve years, from 1999 to 2011, the policy appeared to have been issued in 1998. Chapter Seven tells how a clerk's spreadsheet put the year back for some 3,400 policies (an illustrative count), this one among them, and got a few hundred of them wrong.

ERA CARD

Reader. The query: SQL written by programmers, report writers and analysts, reading the logical view and trusting the schema.

Security. The DBA role and SQL privileges decide who can read and change a table; batch IDs and conversion accounts often hold more rights than any person.

Cost. Licences, mainframe capacity and conversions; a large line is people (DBAs, conversion teams) and the rework when a conversion's assumptions surface years later.

AI. A model with a SQL tool reads the logical view as it stands; a code whose code table was left behind, or a year supplied by a window, reaches the model as plain fact.

Monday rule: for the next migration or merger on your roadmap, require one Ledger row per hop, covering every table moved and every code table it depends on, with each rule that supplies a value (date windows, key prefixes, code mappings) listed by name and the rows it touched, before cutover is approved.

CHAPTER SEVEN

Data on Every Desk



In 2004 a clerk at one of the insurer's branches noticed policies whose issue dates looked wrong: many dated from 2000 to 2049, the date window's reading of policies written from 1900 to 1949. Checking them against the paper register, she found a few older policies, from the 1890s, that the window had made look recent. She ran an export of the policies with suspicious dates into an Excel workbook, saved it on the branch shared drive, and began typing the years she believed were right. Nobody asked her to and nobody stopped her. The workbook had no owner in any register, no retention rule and no link back to the system it came from. Seven years later it flowed back into that system. This chapter covers the era when data reached every desk: the file, the spreadsheet, the desktop database, the CSV and the shared drive, and the reader they created, the person at the desk. The decision it changes: **any copy that can flow back into a system of record gets a named owner and a controlled return path, or it does not flow back**.

The command line and the file

The desk era began with a prompt and a file name. Microsoft's *MS-DOS Encyclopedia* (1988) traces the line: Digital Research released CP/M, its Control Program/Monitor, for microcomputers in 1976. In 1977 Marc McDonald designed the file allocation table, the FAT, for Microsoft's Stand-alone Disk BASIC. At Seattle Computer Products, Tim Paterson wrote the first version of 86-DOS in 1980; Microsoft acquired it, and MS-DOS made its debut in August 1981 on the IBM Personal Computer. How a file system lays out bits is the subject of *From Bit to Pixel*, Book 1 of *Every Pixel Matters*; this chapter only needs what the file meant for records.

A file on a desk machine is a named container. A file system of that era, such as FAT, records its name, its size, a few attributes and a date or two. It does not record who is accountable for the content, what the columns mean, or which system the data came from. Before the

desk, a record lived in a system that a department ran, with a manager who answered for it. On the desk, anyone with the machine could make one, copy it and give it a new name. That is the root of the era's power and of the Ledger's OWNERSHIP code.

The spreadsheet, the desktop database and the CSV

The Computer History Museum calls VisiCalc, published in 1979 for the Apple II, "the first spreadsheet for personal computers", and says many buyers bought the machine to run it. Lotus 1-2-3, which the museum dates to January 1983, combined a spreadsheet with charts and simple database functions on the IBM PC. Microsoft's own history dates Excel for the Macintosh to 30 September 1985 and Excel for Windows to 1987. In November 1992, at COMDEX in Las Vegas, Microsoft announced the availability of Access, a database for Windows on the desktop.

The spreadsheet changed who could build a system. A cell holds a value, a formula or both, and a format that decides how the value is shown. A sheet can be a report, a model, a list of records or all three. A workbook on a shared drive in 2004 had no schema that the program enforced, no transaction and no log of who changed which cell, unless someone added them. Later cloud versions add version history, and tables and validation rules can add some structure; they still add no database schema and no transactions. Access brought tables, keys and queries to the same desk, which made the desktop copy look even more like a real system. Microsoft's history lists the Open Database Connectivity (ODBC) Software Development Kit 1.0 among its 1992 releases: a common interface through which desktop tools could query server databases. Once a driver, a data source and a login were set up, an extract from the system of record into a spreadsheet or a desktop database could be a menu choice, not a request to the data processing department. The department that owned the record no longer saw most of the copies made from it.

Between these tools sat the plain text file of comma-separated values. CSV was used for decades before anyone wrote it down. RFC 4180 (October 2005) describes a common format, but as an Informational document that "does not specify an Internet standard of any kind". It also says why it was needed: "there is no formal specification in existence, which allows for a wide variety of interpretations of CSV files." A CSV carries no types, no units and no time zone. Every reader decides for itself whether 04/05/1898 is April or May, and whether 0417 keeps its leading zero.

TYPE CONTRACT: A DATE IN A DESK COPY

Type, precision and rounding. a calendar date (an issue date), with no time of day.

Encoding and serialization. in a spreadsheet, either a serial number displayed as a date or a text string typed by hand, which look the same on screen; in a CSV, always text.

Unit or currency. a calendar day.

Time basis. the century is taken from whatever the person typed or the program guessed.

Identity scope. the row it sits on, matched to a policy only by whatever key column the copy carried.

Absence. an empty cell can mean "not checked", "not known" or "deleted by mistake".

Authority. none recorded: the copy has no owner. Ask which encoding each column holds before you load it anywhere.

The spreadsheet's own date rules matter for a record from 1898. Microsoft's documentation says that in Excel's 1900 date system, the default in Excel for Windows, "the first day that is supported is January 1, 1900"; a date is stored as a serial number of days from that start. In that default date system, a date in 1898 cannot be stored as an Excel date. Typed in full, it stays text that looks like one. Microsoft also documents that Excel "incorrectly assumes that the year 1900 is a leap year", a choice made so that worksheets stayed compatible with Lotus 1-2-3. And when a person types a two-digit year, Excel applies a window of its own, set in Windows: by default "00 through 29 is interpreted as the years 2000 through 2029" and "30 through 99 is interpreted as the years 1930 through 1999". A clerk who typed a date such as 3/14/98 got 14 March 1998 again, stored as a date. A clerk who typed 3/14/1898 got text. A clerk who typed only the year 1898 got a plain number, which a cell formatted as a date shows as 12 March 1905. In a column of hand-typed years the three were easy to mistake for one another.

The copy as default

Networks moved the file from the desk to a shared drive: a folder on a server that a branch or a department could all read. Document management and cloud storage moved it again: Microsoft's SharePoint and OneDrive, and Google Drive, which Google announced on 24 April 2012 as a place to create, share and keep files. Each step made sharing easier, and a link began to replace the attachment. Microsoft's page on shareable links, dated 6 November 2026, says that in OneDrive and SharePoint "each item has one hero link", that its usual starting scope is that only people added to the file have access, and that narrowing the scope

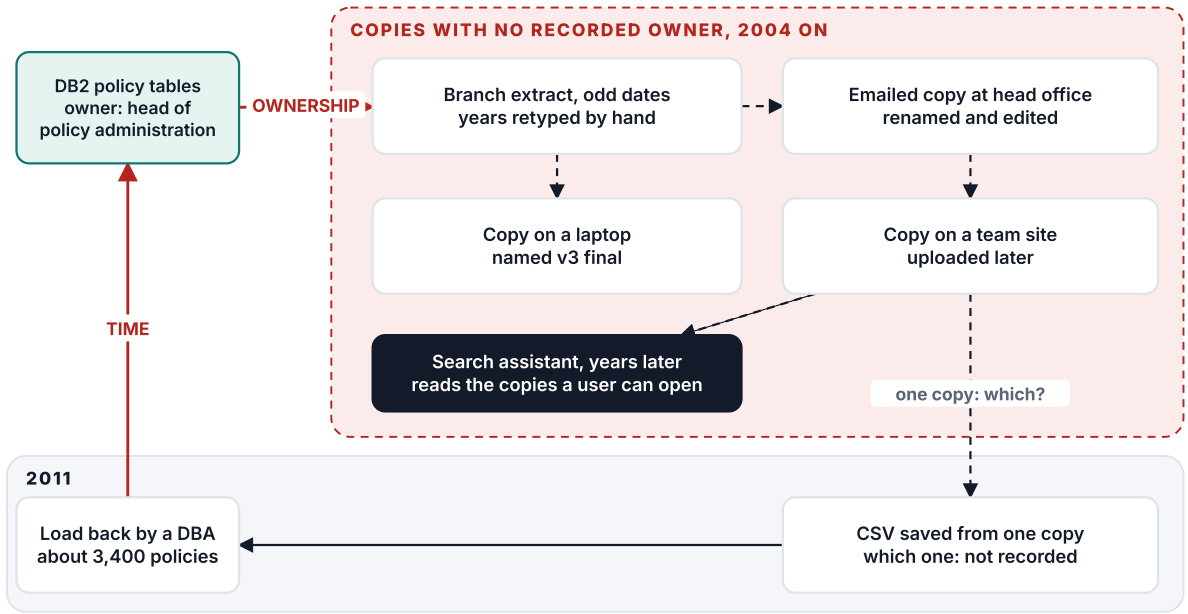
"applies to links already sent". A link controls who can open one file; it does nothing about the copies already downloaded, attached or exported, which is why, on the desk, sharing still mostly meant copying.

The hotel and retail group ran on one such file for years. Each season, the reservations manager at head office (later the revenue team) built a rate sheet: a workbook with a tab per hotel and, on each tab, one row per room type and date range with its rate code and rate. It went out by email to about 60 hotels each season until its replacement in 2022, and some hotels kept emailing leftover copies after that. A general manager changed two rates for a local festival and saved the file as "rates final GM". Another hotel copied last year's sheet and edited it. In March 2022 the rates were imported into the new revenue system from one of these copies. Which one, and whose edits it carried, was not recorded anywhere a later reader could find. When a guest disputed a price, the group had to reconstruct the answer from inboxes, and the old copies are still there.

Figure 7.1 draws the same pattern for the insurer's 2004 workbook: one export, copies with no owner, a search assistant that reads them all, and one copy that flows back.

FIGURE 7.1

One spreadsheet's copies across a company, and the one that flowed back



The 2004 extract came from tables with an owner and became copies with none; in 2011 one copy, with no record of which, was loaded back.

Author's diagram of the book's illustrative thread (hops 5 and 5b); the copy chain beyond the branch extract is illustrative.

Two features make the desk copy different from earlier hops. It branches: one extract becomes many files, each edited by someone. And it can flow back: a corrected copy is loaded into the system it came from, carrying edits that no system checked. The Ledger handles both by giving each copy that matters its own row, and by treating the return load as a hop of its own.

When the spreadsheet is the system

Spreadsheet errors are common, and most never reach a public report. Two did, and both reports name the spreadsheet as a mechanism, not a side note.

SOURCED FAILURE

JPMorgan Chase, 2012. The *Report of JPMorgan Chase & Co. Management Task Force Regarding 2012 CIO Losses* (16 January 2013) reviews losses in the Chief Investment Office's Synthetic Credit Portfolio, which the firm put at approximately 5.8 billion dollars year to date by 30 June 2012 (page 7). Its appendix on the new value-at-risk model says the model "operated through a series of Excel spreadsheets, which had to be completed manually, by a process of copying and pasting data from one spreadsheet to another" (page 124). It reports that "a spreadsheet error caused the VaR for April 10 to fail to reflect the day's $400 million loss" (page 127), and that, in calculating changes in hazard rates and correlations, "the spreadsheet divided by their sum instead of their average, as the modeler had intended", which "likely had the effect of muting volatility by a factor of two and of lowering the VaR" (page 128). The firm's Model Review Group had noted that the computation was done "on spreadsheets using a manual process" and was "error prone" (page 105).

Reinhart and Rogoff, 2010 (error found in 2013). In PERI Working Paper 322 (April 2013), Thomas Herndon, Michael Ash and Robert Pollin replicated a widely cited study of public debt and growth using the authors' own working spreadsheet. They found "a coding error in the RR working spreadsheet" that "entirely excludes five countries, Australia, Austria, Belgium, Canada, and Denmark, from the analysis". Combined with selective exclusion of available years and an unusual weighting, the corrected average growth for countries with public debt over 90 percent of GDP was 2.2 percent, "not -0.1 percent" as published. The paper attributes a 0.3 percentage point error in that category to the spreadsheet error, compounded with the others.

Neither case is a story about a careless person. Both are stories about a spreadsheet that carried decisions without the controls of a system of record: in the reports' own accounts, manual copying, a formula nobody checked, and a range that left rows out. The Task Force

report's remedies include "a review of all manual spreadsheets and the implementation of enhanced controls for key spreadsheets" (page 108). That is close to what the Ledger asks for: a list of the copies that matter, each with an owner and a check.

The assistant that reads the copies

For years the copies on shared drives had a natural protection: obscurity. Nobody could find the 2004 workbook unless they knew where it was. Enterprise search, and now AI assistants built on it, remove that protection.

Microsoft's documentation on data, privacy and security for Microsoft 365 Copilot, read in November 2026, states the design plainly: it "only surfaces organizational data to which individual users have at least view permissions", and it asks customers to use the permission models in services such as SharePoint "to help ensure the right users or groups have the right access to the right content". The assistant honors permissions. It does not fix them. A folder shared with "everyone in the company" in 2009 is, to the assistant, a folder every employee may read today, and every copy in it is a candidate source for an answer.

Two risks follow. The first is **oversharing**: personal and commercial data in old copies becomes easy to retrieve for people who always had access on paper but never in practice. The second is **authority**: an assistant that finds three versions of the rate sheet, or the clerk's workbook beside the policy system's report, has no built-in way to know which is the record. Nothing in that permission design tells it which copy is the record. Chapter Ten covers retrieval in depth; the desk era's contribution is the pile of ownerless copies that retrieval now reads.

> **OUR VIEW, AS OF NOVEMBER 2026**
>
> Before a company turns on an assistant across its file shares, the cheapest control is not a new AI product. It is a review of who can read each share, and a short list of the copies that act as systems of record, each with a named owner. The assistant will read whatever that review leaves behind.

THE HOP · 5 · 2004 · DB2 POLICY TABLES TO EXCEL WORKBOOK ON A SHARED DRIVE (BRANCH)

Record	Policy 1898/0417 (perpetual fire policy), hop 5, 2004
From	DB2 policy tables
To	Excel workbook on a shared drive (branch)
Mechanism and schedule	A clerk exports policies with odd dates to check them, and types corrected years by hand
Format change; what was lost or assumed	OWNERSHIP Spreadsheet; dates typed as text and as Excel dates. **A copy with no accountable owner; the workbook is copied and emailed**
Accountable owner	None recorded
Who can change it	Anyone with access to the shared drive
Classification; retention or erasure duty	Contains personal and commercial data; no retention rule applied
Read by (people, apps, models, agents)	Clerks; later an assistant that can search the shared drive
Cost per month; basis	Near zero in storage; real cost in rework later (illustrative)

THE HOP · 5B · 2011 · EXCEL WORKBOOK TO DB2 POLICY TABLES (CORRECTION LOAD)

Record	Policy 1898/0417 (perpetual fire policy), hop 5b, 2011
From	Excel workbook
To	DB2 policy tables (correction load)
Mechanism and schedule	The workbook is loaded back as a one-off correction
Format change; what was lost or assumed	TIME CSV export of the workbook, loaded by a utility. **Hand-typed dates overwrite the windowed ones for some 3,400 policies (illustrative count): this policy is fixed to 1898, a few hundred others are made wrong**
Accountable owner	Head of policy administration (signed the change)
Who can change it	One DBA running the load
Classification; retention or erasure duty	Commercial record
Read by (people, apps, models, agents)	Policy system
Cost per month; basis	One day of a DBA; years of later reconciliation (illustrative)

In 2004 policy A-980417 left the policy system at hop 5. The branch clerk exported the DB2 rows with odd dates into an Excel workbook on the shared drive and typed corrected years by hand, some as text and some as Excel dates. The workbook was copied and emailed. No owner was recorded; anyone with access to the shared drive could change it, and no retention rule applied, although it held personal and commercial data. In 2011, at hop 5b, a CSV export of the workbook was loaded back into the DB2 policy tables as a one-off correction, signed by the head of policy administration and run by one DBA. For this policy the load did what the clerk intended: the issue date, typed as text in the workbook and parsed by the load utility, went back to 1898, ending twelve years in which the policy appeared to date from 1998. For some 3,400 policies (an illustrative count), this one among them, the hand-typed dates overwrote the windowed ones, and a few hundred of those hand corrections were wrong: where a year had been typed with two digits, Excel's own window had already chosen a century. It took one day of a DBA's time and years of later reconciliation. The workbook still sits on the shared drive, where a search assistant can read it.

ERA CARD

Reader. The person at the desk: a clerk, an analyst or a manager reading and editing a copy, often without knowing where it came from.

Security. Folder and file permissions, set once and rarely reviewed; email as the main way data moves; no record of who changed which cell.

Cost. Storage near zero; the real cost is people's time to reconcile copies, and the rework when a copy flows back.

AI. An assistant reads every copy the user is allowed to read and cannot tell the record from its stale or hand-edited copies.

Monday rule: list every spreadsheet or file that is loaded into a system of record, or that a finance or pricing decision depends on, and give each a named owner, a Ledger row and a review of who can read it before your assistant indexes it.

CHAPTER EIGHT

Warehouse to Lakehouse



In 1988 the IBM Systems Journal printed a paper by Barry Devlin and Paul Murphy of IBM Europe, "An Architecture for a Business and Information System". Devlin later wrote that he and his colleagues had defined the architecture in 1986 for IBM's internal use. Its idea outlived the products of its time: copy the records out of the systems that run the business into a separate store built for questions, and let a new reader, the analyst, ask those questions without touching the systems that write. Thirty-three years later, in 2021, the insurer did the same thing with policy 1898/0417 and moved it into a cloud data warehouse. The columns arrived. The meaning of one of them did not. The decision this chapter changes is small and easy to skip in a migration: **a code column does not move without its code table**.

A second store, built for a new reader

In an operational system the readers of a policy record were programs, clerks at screens and, where a relational database had arrived, queries. A clerk read one record at a time. A manager who wanted a total usually waited for a batch report or a query, written by a specialist and run against the same system that served the clerks, often at night so as not to slow them down.

The warehouse changed the reader. Instead of asking the operational database, which was tuned for many small updates, a company copied its records into a store tuned for large reads, kept history in it, and gave analysts tools to query it. W. H. Inmon's *Building the Data Warehouse* (QED, Boston, 1992) made the case for one integrated enterprise warehouse fed from all sources. Ralph Kimball's *The Data Warehouse Toolkit* (Wiley, 1996), subtitled "practical techniques for building dimensional data warehouses", made the case for building it as dimensional models: **fact tables** that hold measurements (a premium, a claim payment) and **dimension tables** that hold the descriptions a reader filters by (the policy, the branch, the date). The two schools argued for years about the order of the work. They agreed on the move itself: data is copied, cleaned and modeled before analysts read it.

In 1993 a white paper by E. F. Codd, S. B. Codd and C. T. Salley, "Providing OLAP to User-Analysts: An IT Mandate", named the reader's style of work. In its words it "defines a category of database processing: Online Analytical Processing (abbreviated OLAP)". OLAP tools showed data as cubes: totals by product, by region, by month, sliced and drilled by the analyst once someone had built the cube.

This is where the hop lives. Every warehouse is filled by a job usually called **ETL** (extract, transform, load). The job decides which columns travel, how they are renamed and what they become. In a dimensional design, a status code should arrive as a dimension: the code, its label, its rule. If the modeler did not know the code mattered, it arrives as a bare number in a fact table. The warehouse writes its schema before data arrives, which the Lakehouse paper (Armbrust and colleagues, CIDR 2021) calls **schema on write**: the data model was "optimized for downstream BI consumption". Optimized for the questions someone expected, that is. A meaning nobody expected has no column.

The cloud took the box away, and the bill came with it

For two decades a warehouse was a machine the company bought. The Lakehouse paper describes that first generation: systems that "typically coupled compute and storage into an on-premises appliance", which "forced enterprises to provision and pay for the peak of user load and data under management". The cost of a question was hidden inside a capital budget approved every few years.

Three services moved the warehouse into rented capacity:

- **Amazon Redshift**: a limited preview announced on 28 November 2012; available to all customers on 14 February 2013, priced by AWS at "under $1,000 per terabyte per year" on reserved pricing.
- **Google BigQuery**: made "publicly available" on 1 May 2012, with up to 100 GB of queries a month free and the promise that users "pay only for what you use".
- **Snowflake**: introduced in October 2014 and generally available on 23 June 2015, by the company's announcement.

Two things changed for a CTO. First, the bill became an operating cost, and in some services storage and compute were billed apart: BigQuery charged for the data its queries processed, and Snowflake billed stored data and computing separately. Redshift at launch priced provisioned capacity, with storage and compute together, and added nodes that scale and bill them independently (RA3) on 3 December 2019. Where storage is billed apart, a team can keep years of history cheaply and pay for computing only while queries run. Second, the cost of questions became visible and variable, though the unit of billing differs by service: a dashboard that refreshes every five minutes against a large table now has a line on a monthly

invoice. The bill moved from the data center budget to the data platform, and from the hardware buyer to whoever writes the query. Chapter Thirteen prices store, move, query, keep and delete in detail.

The hop did not change. Data still left the policy system through a job, and the job still chose what to carry.

Files become tables: Parquet, table formats and the lakehouse

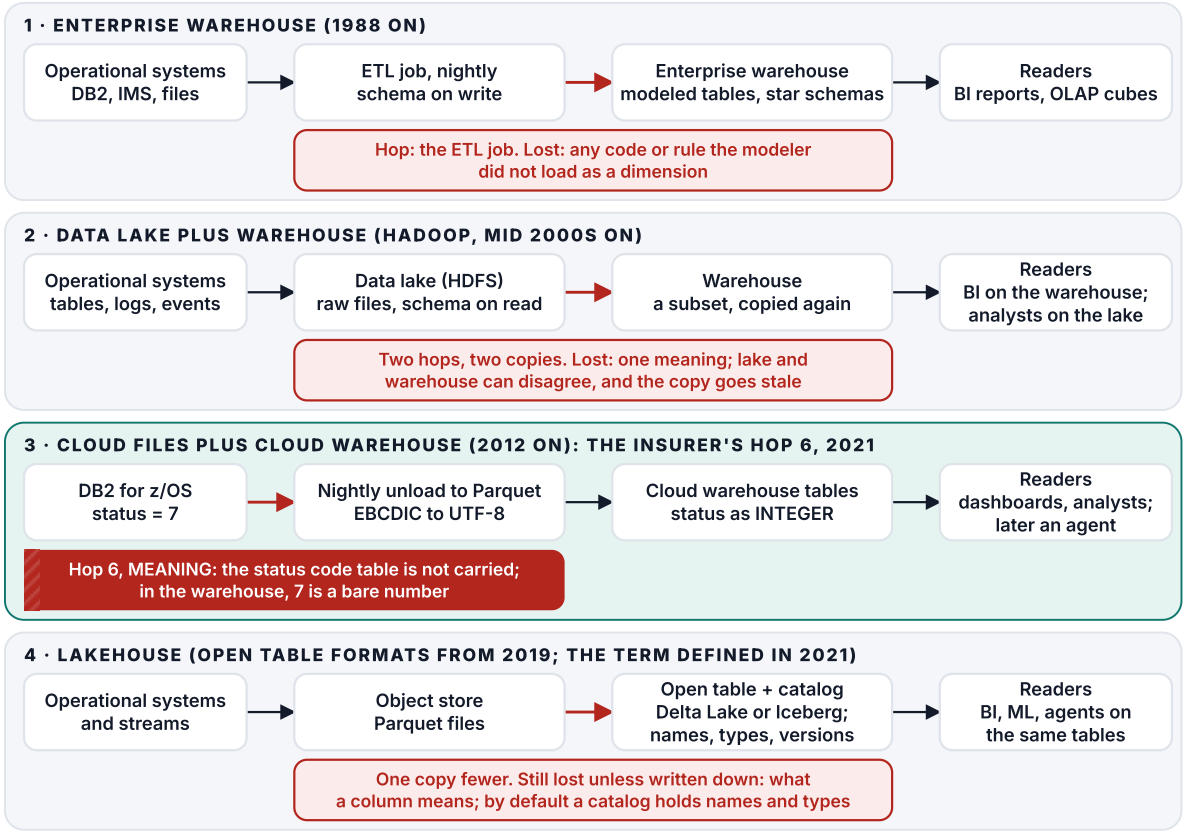
Beside the warehouse, a second pattern grew from the Hadoop projects of the mid 2000s (Chapter Nine): the **data lake**, cheap storage that holds files in open formats and lets each reader decide how to read them. The Lakehouse paper calls it "a schema-on-read architecture" that gave agility but "punted the problem of data quality and governance downstream". Many companies ran both: raw files in the lake and a subset copied again into a warehouse, which the paper calls the two-tier model. Its cost is concrete: "users pay double the storage cost for data copied to a warehouse". The paper also names the second cost, time: data in the warehouse "is stale compared to that of the data lake, with new data frequently taking days to load". Every extra copy is an extra hop, with its own job, its own schedule and its own chance to drop a column.

Open file formats made the lake readable by many engines. On 30 July 2013 Twitter announced Parquet 1.0, "the result of a collaboration between Twitter and Cloudera", an "open-source columnar storage format" for Hadoop. A **columnar** file stores each column together, so a query that reads three columns of a hundred reads only those three. Parquet also carries types: its `DECIMAL` annotation stores an exact number as an unscaled integer plus a scale, and the precision and scale sit in the file's schema, so an amount such as 412.49 stays exact rather than becoming a binary fraction. Parquet became an Apache top-level project in April 2015.

A folder of files is not yet a table: two writers can collide, and a reader can see half a change. **Table formats** add a metadata layer that records which files make up the table at each version. Databricks open sourced Delta Lake on 24 April 2019, saying users could access earlier versions of their data "for audits, rollbacks" or to reproduce experiments. Iceberg, which started at Netflix, entered the Apache Incubator on 16 November 2018 and describes itself as "a table format for large, slow-moving tabular data". On top of these layers, Armbrust and colleagues defined the **lakehouse** in 2021 as "a data management system based on low-cost and directly-accessible storage that also provides traditional analytical DBMS management and performance features such as ACID transactions, data versioning, auditing, indexing, caching, and query optimization".

FIGURE 8.1

Warehouse to lakehouse in four architectures: each changed where the copy lives and who pays; none carries meaning by design



Four architectures from 1988 to today, with the hop where meaning can be lost in each; the third row is the insurer's hop 6 in 2021.

Author's diagram based on Armbrust et al., "Lakehouse", CIDR 2021 (Figure 1); Devlin and Murphy, IBM Systems Journal, 1988; hop 6 from the thread (illustrative).

Figure 8.1 puts the four architectures side by side. Each changed where the copy lives and who pays for it; the two-tier design added a copy, and the lakehouse removed one. None of them, by design, carries what a value means. That job belongs to the catalog, and to the people who fill it. How a company arranges its layers and the trust each one earns is the subject of the Medallion book *From Bronze to Gold*; this chapter stays with the hop.

The catalog and schema evolution

A **catalog** is the register of the platform: which tables exist, where their files live, what their columns are called and typed, who may read them. The Iceberg specification says a table's location is "intended to be managed and supplied by a catalog". Commercial catalogs add

owners, tags, descriptions and lineage graphs. What a catalog holds by default is the schema: names and types. What a column *means* lives there only if someone writes it down.

Schemas also change, and table formats now make that a managed event. The Iceberg specification states that "schema evolution supports safe column add, drop, reorder and rename", and that columns in data files "are selected by field id", so a renamed column keeps its identity: "renaming an existing field must change the name, but not the field ID". That is a real gain over the copybook era of Chapter Five, where a moved field could silently shift every value after it. It is a gain in identity, not in meaning. Iceberg will keep the `status` column as the same field, say field id 12, through any rename. It will not tell a reader that the value 7 in that column means "perpetual, deposit held, no premium due".

TYPE CONTRACT: STATUS IN THE WAREHOUSE POLICY TABLE (2021)

Type, precision and rounding. an integer column holding a code; no arithmetic applies.

Encoding and serialization. integer column, unloaded from a one-digit code in DB2.

Unit or currency. none; it is a code.

Time basis. status as of the last nightly load, not as of the question.

Identity scope. one row per policy in the warehouse policy table.

Absence. not defined.

Authority. the code table was not carried.

The contract a pipeline test checks (integer, not null, in range) passes; the contract a reader needs (what 7 means) is missing.

This is the point where a common objection to lineage deserves an answer. A team can put a contract at the last hop: a test that the column is an integer, never null, between 0 and 9. That test is cheap and useful, and in 2021 it would have passed. A contract checks shape. The STATUS_CODES table was a fact about meaning, kept in DB2 beside the policy table, and no test of the warehouse column could notice that it never arrived. Only a record of the hop, with its loss written in the row, shows the gap.

The fix is plain work, not a new product. For each code column a team names in its catalog five things: the code table (as a table, joined or linked, not a PDF), the owner who may change a code's meaning, the date each meaning took effect, what a value outside the table means, and which readers depend on the column. Most catalogs can hold all five as a linked

table, a description and tags. The scarce part is noticing, before the move, that the source keeps a code table beside the column, and putting that table on the list. Where a code book exists only on paper or in someone's head, the scarce part is the hour it takes to write it down before that person retires.

> **WHAT HASN'T CHANGED**
>
> Devlin's architecture of 1986 and the lakehouse of 2021 make the same move: records are copied from the systems that write them to a store built for readers who ask new questions. The appliance, the cluster and the object store each changed where the copy lives and who pays for it. None changed the fact that the copy is a hop, and that the job making it decides what travels.

The hop: policy 1898/0417 reaches the cloud

By 2021 the same DB2 policy tables ran on z/OS, after upgrades of the operating system and of DB2 that did not change the policy rows, so they have no Ledger row. In 2021 the insurer's head of data platform copied the policy domain from DB2 for z/OS into a cloud data warehouse. A nightly job unloads the DB2 tables to Parquet files, transfers them and loads them into warehouse tables. EBCDIC text became UTF-8; packed decimal amounts became `DECIMAL(18,2)`, so the 412.50 dollar deposit survived exactly. The status became an integer column. The STATUS_CODES lookup table, keyed in from the paper code book in 1999, was not on the list of tables to move, so in the warehouse 7 is a bare number. Finance and actuarial dashboards read the table, and so do data scientists; a premium report has no use for a policy that pays no premium, so no dashboard showed the gap. The pipeline service account, three data engineers and the warehouse load role can change the data, and the platform costs about 2,800 dollars a month to store, move and query this domain (illustrative). From 2025 an agent would read the same bare number (Chapter Ten).

THE HOP · 6 · 2021 · DB2 FOR Z/OS TO CLOUD DATA WAREHOUSE TABLES (LOADED VIA PARQUET FILES)

Record	Policy 1898/0417 (perpetual fire policy), hop 6, 2021
From	DB2 for z/OS
To	Cloud data warehouse tables (loaded via Parquet files)
Mechanism and schedule	Nightly unload job, file transfer and load into warehouse tables
Format change; what was lost or assumed	MEANING EBCDIC and packed decimal converted to UTF-8 text and DECIMAL(18,2); status as an integer column. **The STATUS_CODES lookup table is not carried: in the warehouse 7 is a bare number**
Accountable owner	Head of data platform
Who can change it	Pipeline service account; three data engineers; the warehouse load role
Classification; retention or erasure duty	Commercial record; the policy domain also holds personal data of individual policyholders under US federal (GLBA) and state insurance privacy rules; retention duties
Read by (people, apps, models, agents)	Finance and actuarial dashboards; data scientists
Cost per month; basis	About 2,800 dollars a month to store, move and query this domain (illustrative)

ERA CARD

Reader. The analyst with a query tool, then the platform itself: dashboards, notebooks and, later, agents, all reading copies made for questions someone expected.

Security. Every load is a new copy of personal data with its own roles; the service account that loads the warehouse can change every row it carries, and is often the least reviewed identity in the company.

Cost. The bill moved from hardware to the platform: storage, compute and transfer are metered, and the copy between lake and warehouse is paid twice.

AI. Warehouse tables are what agents query by tool; whatever meaning the load left behind, the agent receives as a bare value and fills in.

Monday rule: list the code columns that your dashboards and agents read most, and for each one check that its code table is in the catalog, linked to the column and owned by a named person.

CHAPTER NINE

Meaning Moves to the Reader: Web, JSON, NoSQL, Logs and Streams



On 10 February 1998 the World Wide Web Consortium (W3C) published Extensible Markup Language (XML) 1.0 as a Recommendation. A record could now carry its own field names inside it, in plain text, readable by any program with an XML parser, on any machine. It is tempting to read that as the end of the meaning problem: if every value arrives with a label, every reader knows what it is. It moved the problem instead. A label is a name, not a definition, and in the systems that followed, the web, JSON, NoSQL stores, logs and streams, the job of deciding what a record means passed from the database that stored it to each program that read it. The decision this chapter changes: **for every record that crosses a system boundary, name who holds the schema and which clock orders it**.

Records that describe themselves: XML, JSON and Protocol Buffers

The relational database of Chapter Six kept the schema in one place, the catalog of the database, and every reader went through it. Records sent between companies or between programs left that catalog behind. XML put the names back into the record: `<status>7</status>` tells a reader the field is called status. It does not tell the reader what 7 means.

JSON (JavaScript Object Notation) did the same with less text. RFC 8259 (IETF, December 2017) defines it as "a lightweight, text-based, language-independent data interchange format", and states that it is also described in ECMA-404. Three details of the standard matter to anyone who moves records through JSON:

- **An object is unordered.** The standard says "an object is an unordered collection of zero or more name/value pairs", so a reader that depends on field order is depending on luck.

- **Numbers have no declared precision.** RFC 8259 "allows implementations to set limits on the range and precision of numbers accepted", and notes that good interoperability comes from expecting no more than IEEE 754 binary64 (double precision). Integers are safe only within plus or minus 2^{53} minus 1. An amount such as 412.49 sent as a JSON number is read by most parsers as the nearest binary fraction, close to it but not equal; sent as a string, it is exact but untyped. The deposit, 412.50, happens to be exact in binary. Chapter Eleven shows why that luck misleads and follows the value through every form.
- **There is no date type.** The grammar has objects, arrays, numbers, strings and three literals (true, false, null). A date is a string, and its format, time zone and meaning are whatever writer and reader agree.

Google went the other way. On 7 July 2008 it made **Protocol Buffers** available to the open source community. In Kenton Varda's announcement, they "allow you to define simple data structures in a special definition language", which programs compile into code for reading and writing a compact binary form. XML could already point to a schema of its own (a DTD); Protocol Buffers made a separate schema the normal case. The schema is back, but it lives in a separate file, outside the record, and every reader must hold the right version of it.

The same choice sits under every web **API** (application programming interface): a service exposes records as responses, usually JSON, and the reader gets the representation the API designer chose, not the stored row. When the insurer later gives its service agent a tool, the agent will read policies through exactly such a response. Whatever the designer left out, the agent never sees.

TYPE CONTRACT: A POLICY EVENT IN JSON (2023)

Type, precision and rounding. one change to one policy; amounts as strings with two decimals, not JSON numbers, so no binary rounding.

Encoding and serialization. UTF-8 JSON text.

Unit or currency. dollars, stated in a field, not assumed.

Time basis. two timestamps in UTC: `event_time`, when the change committed in the policy system, and `processing_time`, when the connector observed and published it; plus `log_position`, the change's place in the DB2 log, which sets the order to apply changes. A business date, such as the day an endorsement takes effect, is a field of the record, not an ordering clock.

Identity scope. the policy key inside the payload; the topic itself is not keyed by policy.

Absence. a missing field means "not sent", never "zero" or "none".

Authority. the policy system of record; the event is a copy of a committed change, not a decision of its own.

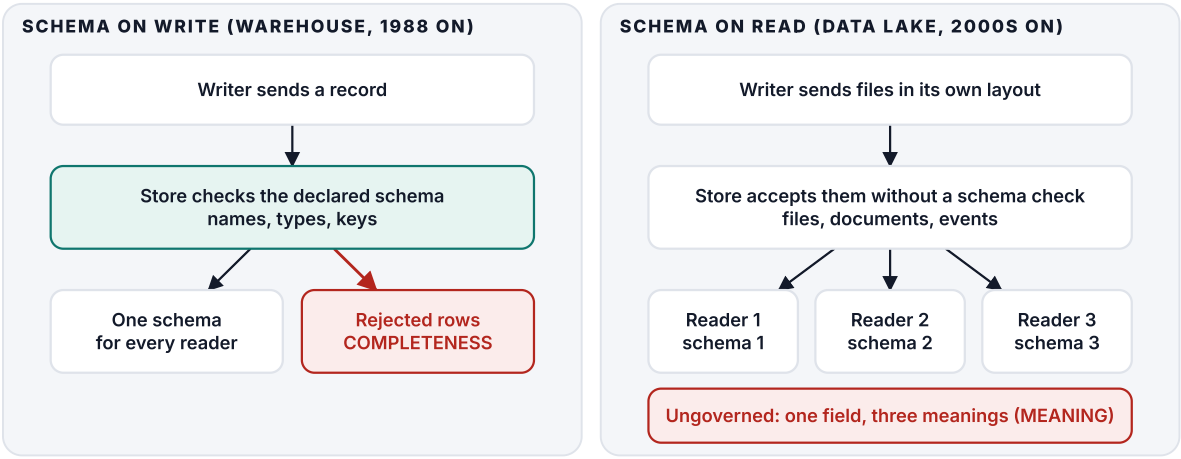
Schema on read: GFS, MapReduce, Hadoop, Bigtable and Dynamo

The second move came from companies whose data no longer fit a single database. Google described its file system (GFS) at the ACM Symposium on Operating Systems Principles (SOSP) in October 2003, and MapReduce, "a programming model and an associated implementation for processing and generating large data sets", at the OSDI symposium in 2004. The Apache Hadoop project built open source software on both ideas (HDFS and Hadoop MapReduce), not Google's own code; its release 0.1.0 is dated April 2006 in the Apache archive.

These systems stored files and let programs decide how to read them. The Lakehouse paper of 2021 calls the resulting data lake "a schema-on-read architecture". **Schema on write** means the store checks every record against a declared schema when it arrives and rejects what does not fit. **Schema on read** means the store accepts records without checking them against a declared schema, and each reader applies its own schema when it reads. The first loses rows that do not fit; the second keeps them and, where no one governs the meaning, multiplies it: ten readers can hold ten interpretations of the same field.

FIGURE 9.1

Schema on write checks shape once, at the door; schema on read lets each reader decide



Schema on write rejects what does not fit the declared schema; schema on read accepts records without checking them and moves the schema into each reader's code.

Author's diagram based on Armbrust et al., "Lakehouse", CIDR 2021 (schema-on-write warehouses, schema-on-read data lakes).

Databases followed. Bigtable, presented at OSDI in 2006 by Fay Chang and colleagues at Google, offered a "simple data model" that gives clients "dynamic control over data layout and format". Amazon's Dynamo, presented at SOSP in October 2007 by Giuseppe DeCandia and colleagues, was built for an "always-on" experience; to reach that availability, the paper says, "Dynamo sacrifices consistency under certain failure scenarios" and "makes extensive use of object versioning and application-assisted conflict resolution". When two versions of a record disagree, the application, the reader, decides which one is true. That is the chapter's title in one sentence of a paper.

None of this was a mistake. These designs bought scale, availability and speed of change, and many companies still need all three. The cost is that meaning, which a relational catalog once held in one place, now lives in the code of every reader, written by different teams at different times.

The log becomes the record

The third move turned the database's private diary into a public one. Chapter Six described the write-ahead log, where a database writes each change before it applies it. Engineers at LinkedIn built a system that made a log the main road for data between systems. Jay Kreps,

Neha Narkhede and Jun Rao presented "Kafka: a Distributed Messaging System for Log Processing" at the NetDB workshop in June 2011, and reported that it was "processing hundreds of gigabytes of new data each day" in production.

In December 2013 Kreps explained the idea in an essay on LinkedIn's engineering blog. A log, he wrote, "is an append-only, totally-ordered sequence of records ordered by time", and logs "record what happened and when". He described the duality between a table and its log: "if you have a table taking updates, you can record these changes and publish a 'changelog' of all the updates to the state of the table". A Kafka topic keeps that total order within each partition, not across the whole topic, so a design that needs one policy's changes in order sends them all to the same partition, usually by using the policy key. Publishing a table's changes this way is **change data capture** (CDC): a connector reads the database's own log and publishes each committed change as an event, within seconds, for any number of readers to subscribe to.

A CDC event typically carries the row as it was after the change, often the row before it, the kind of operation (insert, update, delete) and, depending on the connector, the time of the commit. It does not carry why. An update that sets a policy's owner looks the same whether it was a sale of the building, an endorsement or a clerk fixing a typo. Business intent, like a code's meaning, stays in the source system and in the heads of the people who use it, unless someone adds it to the event.

For a CTO, CDC changes the shape of lineage. A nightly unload (Chapter Eight) is one hop a day. A stream is one hop per change, read by consumers the source team may not know: a dashboard, a fraud model, a search index, an agent's tool. Each consumer decides how to order the events, how to handle duplicates and what a missing field means.

The hotel and retail group's booking events show how quickly that goes wrong, even for records seconds old. A guest books a room on a hotel's website; the booking engine publishes a booking event to the group's stream, and the warehouse, the CRM and the pricing team read it. The booking engine writes the check-in time in the hotel's local time with no offset; a consumer that assumes UTC and takes the date from it puts some late-evening check-ins on the wrong day. No field was dropped. The loss is **TIME**, and it sits in hop 4 of the second Ledger in Chapter Fifteen.

Two clocks: event time and processing time

Every record in a stream has two times, whether or not both are written into it, and confusing them is a loss this era made routine. Tyler Akidau and colleagues at Google defined them in "The Dataflow Model" (Proceedings of the VLDB Endowment, volume 8, number 12, 2015):

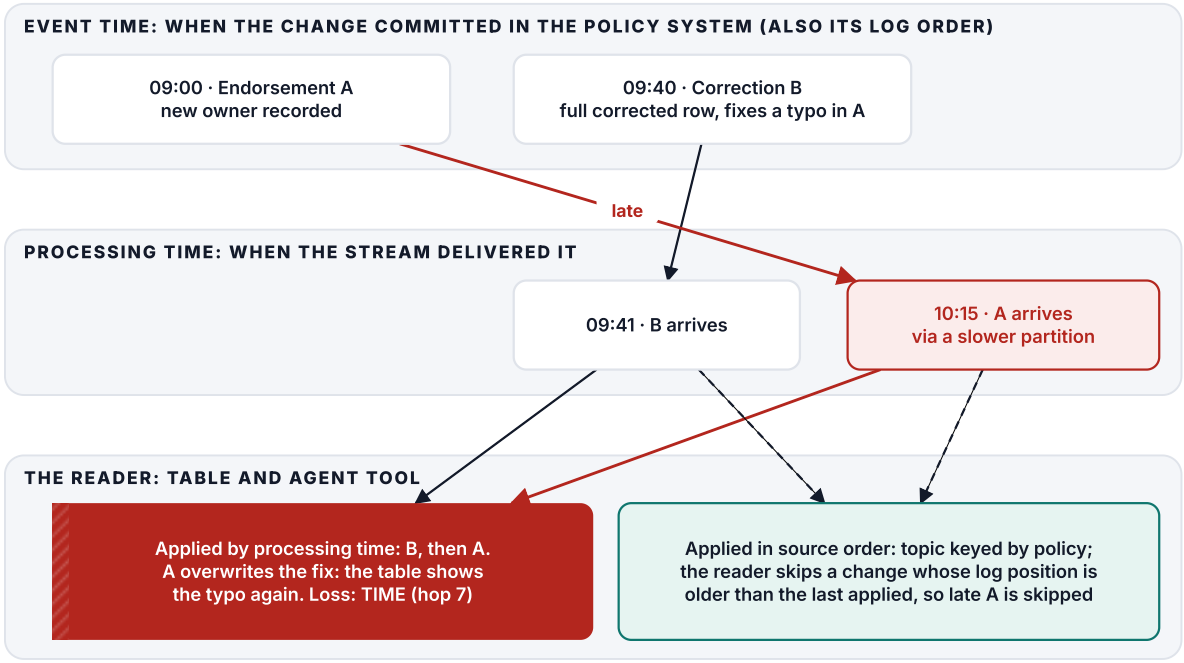
- **Event time** is "the time at which the event itself actually occurred".
- **Processing time** is "the time at which an event is observed at any given point during processing within the pipeline".

The paper adds the line every stream design should start from: "Event time for a given event essentially never changes, but processing time changes constantly for each event as it flows through the pipeline." Networks delay, connectors restart, partitions lag, and so events arrive out of order. A consumer that orders by processing time sees the world in the order the pipeline happened to deliver it, not in the order it happened. The paper's answer is a model of windows, triggers and, where needed, retractions; the business answer is simpler to state: decide which clock each reader must use, and write it into the contract. Carrying an event-time field is not enough on its own; the reader must also act on the order. For change data capture the safe order is the source's own: Kafka keeps order only within a partition, so the topic is keyed by policy and all of one policy's changes travel through one partition in the order the connector wrote them; and the reader keeps, for each policy, the log position of the last change it applied and skips any change with an older one, which also covers a retry that delivers a change twice. For one policy row, log order and commit time agree, because the database lets only one transaction change the row at a time; timestamps can tie, log positions do not.

The paper is also honest about completeness. Stream systems track a **watermark**, which it describes as "a lower bound (often heuristically established) on event times that have been processed", and it warns that "for most real-world distributed data sets, the system lacks sufficient knowledge to establish a 100% correct watermark": a phone taken offline may upload its events days later. A reader that closes the books for a day at midnight is betting that nothing from that day is still on its way. For money, the bet should be written down, with a rule for what happens when a late event arrives after the books are closed.

FIGURE 9.2

Event time and processing time: applied in delivery order, a late change overwrites the correction



Two endorsements on one policy (illustrative times), sent through different partitions of an unkeyed topic: applied in processing-time order, the late original overwrites its own correction; a reader that applies the source's log order keeps the correction.

Author's diagram based on Akidau et al., "The Dataflow Model", PVLDB 8(12), 2015 (definitions of event time and processing time); hop 7 from the thread (illustrative).

WHAT HASN'T CHANGED

The insurer's register of 1898 was a log in Kreps's sense: lines written in order, one per policy, with corrections initialled by the secretary (hop 0). A clerk reading it knew that line order was writing order, not the order in which fires or sales happened. The stream has the same two clocks as the bound register; it runs them in seconds instead of days.

The hop: policy events join the stream

In 2023 the insurer added change data capture to its policy system. The policy tables were altered to DATA CAPTURE CHANGES, which IBM documents as the setting that makes DB2 write full before images to the log; a connector reads that log and publishes each policy change as a JSON event within seconds, with an event time and a processing time, into the stream platform; the warehouse of Chapter Eight now receives changes through the day

instead of once a night. Real-time dashboards read the stream, and from 2025 so do the service agent's tools, through the warehouse. The head of data platform owns it; the stream platform team and the connector configuration can change it; its share of the platform costs about 600 dollars a month (illustrative). Two settings were chosen in a hurry and never reviewed. The connector was configured to publish without a message key, so the topic is not keyed by policy, so two changes to one policy can travel through different partitions at different speeds. And the warehouse applies endorsements, the changes made to a policy after it is issued, in processing-time order, not event-time order. When a correction travels faster than the change it corrects, the correction can arrive first and the original change then overwrites it, as Figure 9.2 shows. Policy 1898/0417 passes through this hop like every other policy; the TIME loss in its row is a property of the hop, which any endorsement to this policy would meet. Its status, still a bare 7 since hop 6, is now delivered within seconds, to more readers, with no more meaning than before.

THE HOP · 7 · 2023 · POLICY SYSTEM EVENTS TO EVENT STREAM (CHANGE DATA CAPTURE) INTO THE WAREHOUSE

Record	Policy 1898/0417 (perpetual fire policy), hop 7, 2023
From	Policy system events
To	Event stream (change data capture) into the warehouse
Mechanism and schedule	CDC reads the DB2 log and publishes policy change events within seconds; the topic is not keyed by policy
Format change; what was lost or assumed	TIME JSON events with an event time and a processing time. **Endorsements are ordered by processing time, not event time: when an original change arrives late, the correction that follows it can appear first**
Accountable owner	Head of data platform
Who can change it	Stream platform team; connector configuration
Classification; retention or erasure duty	Commercial record; the policy domain also holds personal data of individual policyholders under US federal (GLBA) and state insurance privacy rules; retention duties
Read by (people, apps, models, agents)	Real-time dashboards; from 2025 the service agent's tools, through the warehouse
Cost per month; basis	Stream platform share, about 600 dollars a month (illustrative)

ERA CARD

Reader. Every program that consumes a feed: services through APIs, analytics on the lake, stream consumers; each holds its own schema and its own clock.

Security. A stream topic is a copy that many teams can subscribe to; access is granted per topic, often broadly, and a consumer the source team does not know can read personal data within seconds.

Cost. Storage got cheap and the bill moved to traffic and people: every reader that re-implements a schema is a team's time, and every stream is a platform that runs all day.

AI. Agents read through APIs and streams; they receive a field name, a value and an order, and if the order is processing time, they reason over events that never happened in that sequence.

Monday rule: for each stream that feeds a decision, write down who holds its schema and which clock its consumers order by, then send one late event through it and see what the reader does.

CHAPTER TEN

The Model as Reader



On 25 November 2024 Anthropic introduced the Model Context Protocol (MCP), "a new standard for connecting AI assistants to the systems where data lives". It was one of several ways, then and since, to let a language model call a tool: run a query, fetch a document, write a note. A year later the insurer's service agent could read policy rows through a SQL tool and scanned documents through a search index. In February 2026 it read policy 1898/0417, found no premium since 1898 and a status of 7, and told the building's owner that the policy had lapsed. Every earlier reader in this book either executed a fixed rule or was a person who could ask a colleague. The model does neither: it **interprets instead of executing**, and where meaning is missing it supplies a plausible one. The decision this chapter changes: **every path by which a model reads or writes a record gets a Ledger row, an owner and a purpose**.

From the index to the vector

Search came before models. An **inverted index** lists, for each word, the documents that contain it, so a query finds its documents without reading them all. Ranking decides which come first. One ranking formula, BM25, was described by Stephen Robertson and his colleagues at City University in "Okapi at TREC-3", their paper for the third Text REtrieval Conference, run by NIST in 1994. It scores a document higher when the query's words appear in it often and are rare elsewhere. The reader of a ranked list was still a person, who opened the documents and judged them.

Words had already begun to become numbers. In 1990 Scott Deerwester, Susan Dumais and colleagues published "Indexing by Latent Semantic Analysis" in the Journal of the American Society for Information Science: a way to place documents and terms in a shared space, so that a search could match documents that share meaning without sharing words. In January 2013 Tomas Mikolov and colleagues at Google posted "Efficient Estimation of Word Representations in Vector Space" (word2vec), which learned "continuous vector representations of words" and, by their account, took "less than a day to learn high quality

word vectors from a 1.6 billion words data set". In June 2017 Ashish Vaswani and colleagues posted "Attention Is All You Need", proposing "a new simple network architecture, the Transformer, based solely on attention mechanisms".

A **vector** (an embedding) is a list of numbers that places a piece of text near other texts with similar use. Compared with another vector, it measures similarity. It does not keep definitions: nothing in the embedding of "status 7" says "perpetual, deposit held, no premium due". Chapter Eleven shows the deposit as token ids and as a vector from a named open model.

Retrieval, and the separate paths into a model

In May 2020 Patrick Lewis and colleagues at Facebook AI Research, University College London and New York University posted "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks", accepted at NeurIPS 2020. Their abstract states the problem plainly: models "store factual knowledge in their parameters", but "providing provenance for their decisions and updating their world knowledge remain open research problems". Their answer, retrieval-augmented generation (RAG), joins the model to a "non-parametric memory": in their case "a dense vector index of Wikipedia, accessed with a pre-trained neural retriever". The model reads retrieved passages before it writes.

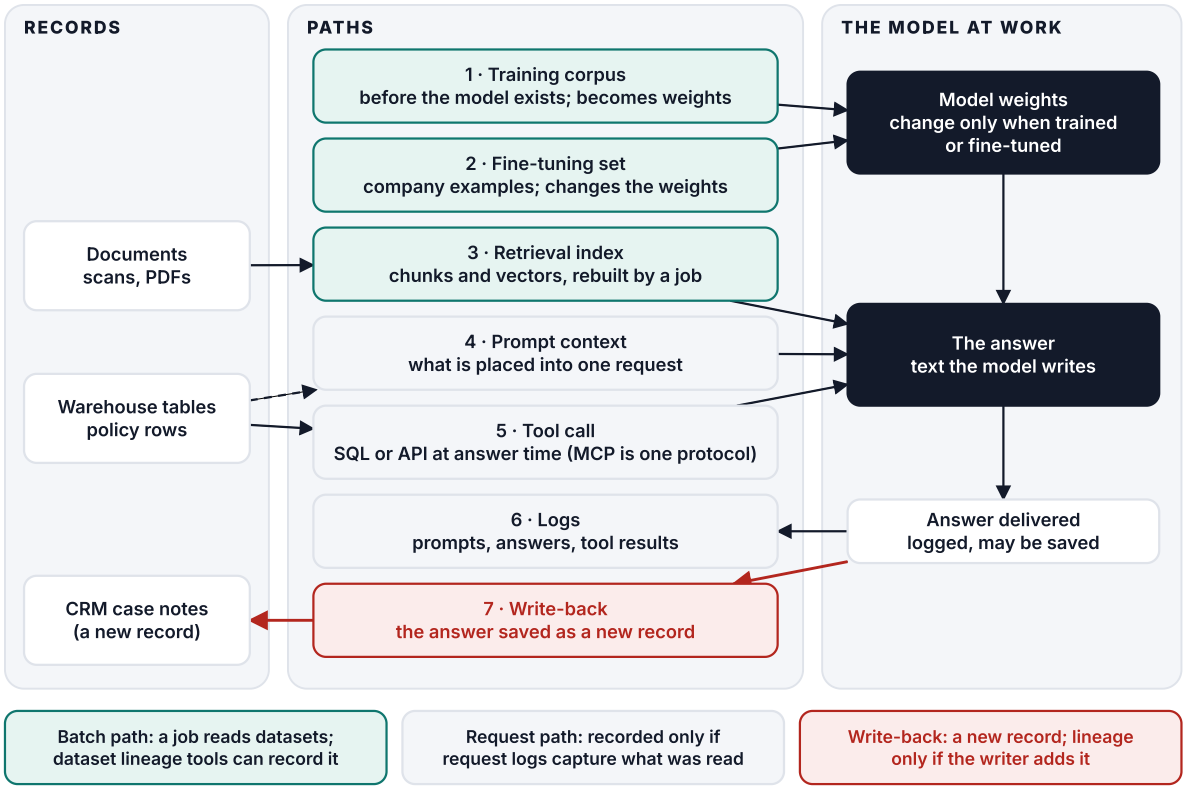
Retrieval is one path among several. Teams often talk about "the data the AI uses" as if it were one thing. It is at least seven paths, each with its own copy, owner, deletion duty and lineage:

- **Training.** The corpus a model is built from. For most companies, a vendor's choice made before purchase; for a company that trains its own model, a dataset with a lineage like any other.
- **Fine-tuning.** Further training on a company's own examples. It changes the weights, so what it teaches cannot be traced back to a row without records kept at the time.
- **Retrieval index.** Documents split into chunks, embedded and stored, rebuilt by a job. The text is usually kept with the vectors or beside them; deleting a chunk removes it from what retrieval can return once the index, its replicas and any caches are updated.
- **Prompt context.** What is placed into one request: instructions, the user's question, retrieved chunks, earlier turns.
- **Tool calls.** Queries and API calls the model makes while answering. MCP is one protocol for them; in Anthropic's words it lets developers "build secure, two-way connections between their data sources and AI-powered tools". Tool calls often read live records with the tool's own service identity rather than the customer's, unless the design passes the user's identity through.

- **Logs.** Prompts, answers and tool results, kept for support, audit or improvement under whatever retention the operator sets.
- **Write-back.** The answer saved somewhere: a case note, a ticket, a changed price. It is a new record, born from a model.

FIGURE 10.1

The AI data paths: seven ways a record reaches or leaves a model, and how each can carry lineage



Training, fine-tuning and index builds are batch jobs that dataset lineage can record; context and tool calls are per request; write-back creates a new record.

Author's diagram based on Lewis et al., NeurIPS 2020 (parametric and non-parametric memory) and Anthropic, "Introducing the Model Context Protocol", 25 November 2024; the path classes are the author's.

Figure 10.1 separates the paths by how lineage can be captured. Batch paths (training, fine-tuning, index builds) are jobs that read datasets and write datasets, and the lineage tools of Chapter Two can record them at dataset and job grain. Request paths (context and tool calls) happen thousands of times a day and are recorded only if request logging captures what was read. Write-back creates a record whose lineage exists only if the writer adds it. A Ledger for an AI system needs a row for each path in use, not one row for "the AI".

What a model keeps

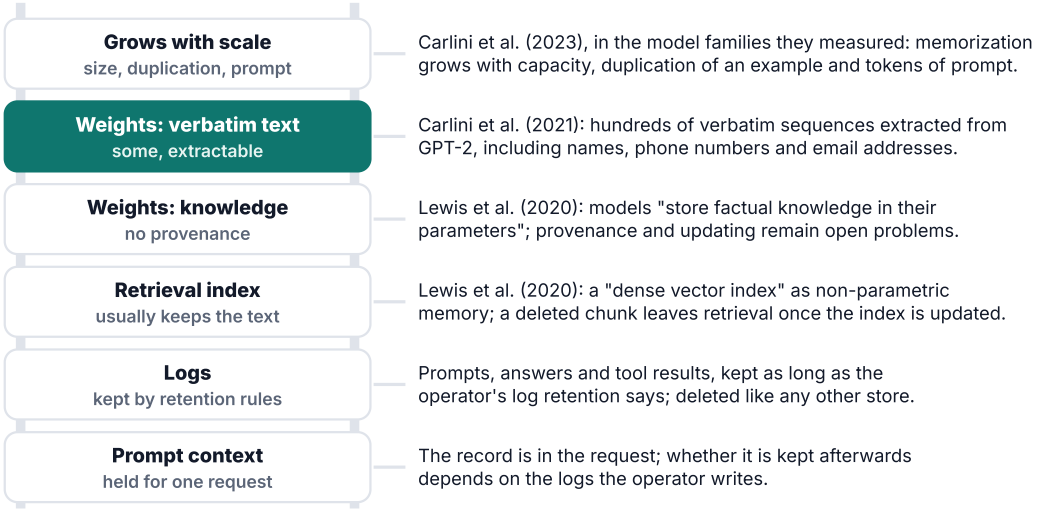
Deletion, consent and purpose questions turn on one fact: what stays inside the model after the data has gone. The claims here are limited to what the papers support.

Nicholas Carlini and colleagues, at the USENIX Security Symposium in 2021 ("Extracting Training Data from Large Language Models"), attacked GPT-2 and were "able to extract hundreds of verbatim text sequences from the model's training data". The extracted text included "(public) personally identifiable information (names, phone numbers, and email addresses), IRC conversations, code, and 128-bit UUIDs", and the attack worked "even though each of the above sequences are included in just one document in the training data". They also found that "larger models are more vulnerable than smaller models".

In "Quantifying Memorization Across Neural Language Models" (ICLR 2023), Carlini and colleagues measured three relationships: memorization "significantly grows as we increase (1) the capacity of a model, (2) the number of times an example has been duplicated, and (3) the number of tokens of context used to prompt the model". They add that the picture "becomes more complicated when generalizing these results across model families".

FIGURE 10.2

What a model keeps: from one request to the weights, only what the papers support



Bottom to top, the places a record can persist around a model, from one request to the weights, ending with what the published studies found about how memorization grows.

Sources: Lewis et al., arXiv 2005.11401 (NeurIPS 2020); Carlini et al., USENIX Security 2021; Carlini et al., ICLR 2023. Lower rungs: author's description of system design.

Two cautions follow. These results do not show that every model reproduces every record; they show that some training text can be extracted, under conditions the papers describe. And they do not apply to a retrieval index or a log, which keep text outright and are deleted like any other store. The practical rule is in the paths: a deletion duty must name each path the record took, because removing a row from the warehouse does not remove its chunk from the index, its copy in the logs or anything a fine-tuning run learned from it.

The hop: the agent reads policy 1898/0417

A batch program that met status 7 did what its code said for 7, and a clerk who did not know would ask the policy department. A model given a bare 7 and no code table, and not told to stop when it lacks a definition, does what it is built to do: it writes the most plausible continuation. "No premium since 1898" plus an unexplained status reads, plausibly, as a lapse. Nothing in the answer marks the guess. That is why missing meaning, which earlier readers tolerated or stopped at, becomes an answer in this era.

One counter is to give the model the meaning as data. In May 2012 Google introduced its Knowledge Graph, which "understands real-world entities and their relationships to one another: things, not strings", and which then held "more than 500 million objects" and "more than 3.5 billion facts". A **knowledge graph** stores facts as entities and relationships that a program can look up. An enterprise does not need half a billion objects to use the idea. A code table can serve as a very small knowledge graph: status 7, has meaning, "perpetual, deposit held, no premium due". Served to the agent as a lookup next to its SQL tool, it turns a guess into a lookup. Graphs, semantics and agent design are covered in depth later in this series.

In 2025 the insurer gave its service agent two ways to read policies. A read-only SQL tool returns warehouse rows as text to the model, and a retrieval index holds scanned documents, including the 1898 register page, as OCR text chunks and vectors. Two things were lost at this hop. The purpose of processing, answering a customer about their own policy, was not attached to the tool, so the tool would answer any question it was given (**RIGHTS**). And the scanned page that says "perpetual" is in the index but ranks below newer documents: its OCR text from 1898 handwriting is poor, and newer typed policy summaries match a question about coverage more closely, so it rarely reaches the prompt (**MEANING**). The head of customer service owns the agent; the head of data platform owns the data; the agent platform team, the prompt and tool configuration and the vector index job can change what it reads; it costs about 1,100 dollars a month in model and retrieval charges for this use (illustrative). On 12 February 2026 the building's owner, a logistics firm, asked whether the warehouse was still covered. The agent answered that the policy had lapsed for non-payment. A claims handler corrected it the next morning, Friday 13 February

2026, from the STATUS_CODES table in DB2 and the register scan, but the answer was already saved as a case note in the CRM: hop 9, told in the Prologue and closed in Chapter Fifteen.

THE HOP · 8 · 2025 · WAREHOUSE TABLES AND THE SCANNED REGISTER PAGE TO SERVICE AGENT: A SQL TOOL FOR POLICY ROWS, A RETRIEVAL INDEX FOR DOCUMENTS

Record	Policy 1898/0417 (perpetual fire policy), hop 8, 2025
From	Warehouse tables and the scanned register page
To	Service agent: a SQL tool for policy rows, a retrieval index for documents
Mechanism and schedule	The agent calls a read-only SQL tool; scanned documents are embedded into a vector index for retrieval
Format change; what was lost or assumed	RIGHTS MEANING Rows returned as text to the model; register page scan as OCR text chunks and vectors. **The purpose of processing is not attached to the tool; the scanned page that says perpetual exists but ranks below newer documents**
Accountable owner	Head of customer service (agent owner); head of data platform (data)
Who can change it	Agent platform team; prompt and tool configuration; the vector index job
Classification; retention or erasure duty	Commercial record; personal data of individual policyholders under US federal (GLBA) and state insurance privacy rules; agent access logged
Read by (people, apps, models, agents)	The service agent and, through it, customers
Cost per month; basis	Model and retrieval costs, about 1,100 dollars a month for this use (illustrative)

OUR VIEW, AS OF NOVEMBER 2026

The agent did not fail on its own. Its answer was assembled from losses at four hops: the remarks field whose code 9 was a second mark of perpetual, dropped in 1962, which left status 7, a digit whose meaning was on paper, as the only one; the code book left on paper again in 1974; the STATUS_CODES table, keyed in at last in 1999, not carried to the warehouse in 2021; and, at the agent's own hop in 2025, the scan ranked low. A model that interprets makes old losses visible to customers. That is a reason to fix the hops, not a reason to stop the agent.

ERA CARD

Reader. The model: it reads through tools, retrieval and context, and interprets instead of executing; where meaning is missing it supplies a plausible one.

Security. A tool runs with its own identity and no purpose attached, so the agent can read whatever the tool can; logs and write-backs create new copies and new records of personal data.

Cost. Charged per request and per token, on top of the platform the data already sits on; the larger cost is people checking and correcting answers.

AI. This is the era of the AI reader: lineage becomes the evidence that the record behind an answer is the record that was written, and who changed it on the way.

Monday rule: list every path by which a model in your company reads or writes a record (training, fine-tuning, index, context, tool, log, write-back), and give each path in use a Ledger row, a named owner and a stated purpose.

One record, ten hops and two corrections

Part II ends where the thread stands in 2026. The tables below are the full Ledger of policy 1898/0417: ten hops, numbered 0 to 9, plus two correction hops (5b and 9b), generated from the same data as every hop box in this book. Read down the loss column. MEANING appears four times (hops 3, 6, 8 and 9) and TIME four times (hops 1, 4, 5b and 7); IDENTITY twice; CONTEXT, COMPLETENESS, OWNERSHIP and RIGHTS once each. PRECISION does not appear: in every system of record in the thread, the deposit of 412.50 dollars was carried in exact forms, from ink to packed decimal to `DECIMAL(18,2)`. The value was kept and its meaning was not. Part III takes the forces behind these losses one at a time: meaning, control and law, cost and deletion, and proof.

THE LEDGER, PART 1 OF 2

One Record, Since 1898: the full Ledger: movement and meaning

Hop, year	From	To	Mechanism and schedule	Format change; what was lost or assumed
0 1898	Applicant and the insurer's surveyor	Policy register, line 417 (ink, bound volume)	Clerk writes the line by hand from the application and the survey report	CONTEXT Ink; fields in ruled columns: number, date, insured, property, sum insured, deposit, remarks. **The surveyor's notes on the building (brick, slate roof, distance to the next warehouse) stay in a separate survey book**
1 1931	Policy register	IBM 80-column card, one card per policy	Keypunch operator transcribes the register; cards sorted and tabulated monthly	IDENTITY TIME Holes in 80 columns; numbers only in most fields; a fixed card layout is the schema. **Key 1898/0417 cannot be punched with a slash: renumbered 980417; the year shrinks to two digits to save columns**
2 1962	Card master deck	Tape master file on the insurer's first computer	One conversion run reads every card; COBOL batch programs then update the tape each night	COMPLETENESS Fixed-length records described by a COBOL copybook; six-bit BCD character code on tape; numbers as decimal digits. **The remarks field (card columns 61 to 80) is dropped as 'unused'; it held remark code 9 in column 61, which marked perpetual policies**
3 1974	Tape master file	IMS hierarchical database on MVS	Conversion programs load policy, building and deposit segments into an IMS hierarchy	MEANING Policy root segment with child segments; status as a one-digit code. **Status codes are kept as digits; the code book that says what 7 means stays on paper in the policy department**
4 1999	IMS database	DB2 for OS/390 on the mainframe (new policy system after the merger)	Merger conversion: both insurers' policies loaded into one DB2 schema; Y2K date windowing applied in the same project	TIME IDENTITY Relational tables; issue date as DATE; key widened with a prefix; the paper status code book is keyed into a DB2 lookup table (STATUS_CODES) for the first time. **98 is read as 1998 by the date window; the merged insurer's numbers collide, so keys get prefixes (A-980417)**
5 2004	DB2 policy tables	Excel workbook on a shared drive (branch)	A clerk exports policies with odd dates to check them, and types corrected years by hand	OWNERSHIP Spreadsheet; dates typed as text and as Excel dates. **A copy with no accountable owner; the workbook is copied and emailed**
5b 2011	Excel workbook	DB2 policy tables (correction load)	The workbook is loaded back as a one-off correction	TIME CSV export of the workbook, loaded by a utility. **Hand-typed dates overwrite the windowed ones for some 3,400 policies (illustrative count): this policy is fixed to 1898, a few hundred others are made wrong**

Hop, year	From	To	Mechanism and schedule	Format change; what was lost or assumed
6 2021	DB2 for z/OS	Cloud data warehouse tables (loaded via Parquet files)	Nightly unload job, file transfer and load into warehouse tables	MEANING EBCDIC and packed decimal converted to UTF-8 text and DECIMAL(18,2); status as an integer column. **The STATUS_CODES lookup table is not carried: in the warehouse 7 is a bare number**
7 2023	Policy system events	Event stream (change data capture) into the warehouse	CDC reads the DB2 log and publishes policy change events within seconds; the topic is not keyed by policy	TIME JSON events with an event time and a processing time. **Endorsements are ordered by processing time, not event time: when an original change arrives late, the correction that follows it can appear first**
8 2025	Warehouse tables and the scanned register page	Service agent: a SQL tool for policy rows, a retrieval index for documents	The agent calls a read-only SQL tool; scanned documents are embedded into a vector index for retrieval	RIGHTS MEANING Rows returned as text to the model; register page scan as OCR text chunks and vectors. **The purpose of processing is not attached to the tool; the scanned page that says perpetual exists but ranks below newer documents**
9 2026	Service agent answer	CRM case note (a new record)	The conversation and the agent's answer are saved to the customer's case	MEANING Free text with a timestamp and the agent's tool calls. **A wrong fact (lapsed for non-payment) now has its own lineage and can be read by the next agent**
9b 2026	Claims handler, from the STATUS_CODES table in DB2 and the 1898 register page scan	CRM case note that supersedes the agent's note (the agent's note is flagged, not deleted)	Manual correction on Friday 13 February 2026, linked to the wrong note by its identifier	Free text citing its evidence: register line 417 and STATUS_CODES row 7. **No material loss: the correction carries its evidence**

THE LEDGER, PART 2 OF 2

One Record, Since 1898: the full Ledger: control, readers and cost

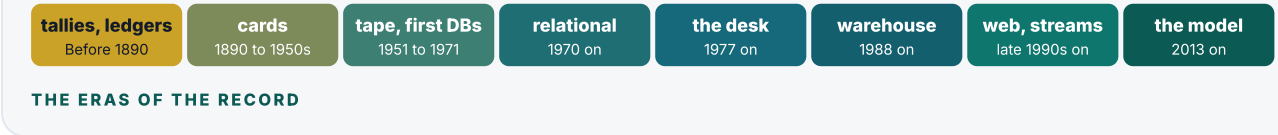
Hop	Accountable owner	Who can change it	Classification; retention or erasure	Read by	Cost per month; basis
0	Company secretary	Register clerk; corrections initialled by the secretary	Commercial record; kept for the life of the policy	Clerks, the auditor, the board	Clerk time, about 10 minutes a month (2 hours a year) for the register line and the annual audit (illustrative)
1	Head of the tabulating department	Keypunch operators; the card layout set by the tabulating supervisor	Commercial record; master deck kept in steel cabinets	Sorter and tabulator; accounting clerks reading printed reports	Card stock, machine rental from the vendor, two operators (illustrative)
2	Data processing manager	Programmers who change the copybook; the night operator	Commercial record; tape generations kept (grandfather, father, son)	Batch COBOL programs; printed renewal and audit reports	Computer lease, tape reels, programmers and operators (illustrative)

Hop	Accountable owner	Who can change it	Classification; retention or erasure	Read by	Cost per month; basis
3	Data processing manager; the database administrator (a new role)	DBA and application programmers; online clerks through screens	Commercial record	Online clerks on terminals; batch reports	Mainframe, IMS licence, DBA and programmers (illustrative)
4	Head of policy administration	DBAs, the conversion team, batch IDs	Commercial record; state record retention rules apply	Policy system screens; SQL reports; actuaries' extracts	Conversion project, mainframe, DB2 licence, staff (illustrative)
5	None recorded	Anyone with access to the shared drive	Contains personal and commercial data; no retention rule applied	Clerks; later an assistant that can search the shared drive	Near zero in storage; real cost in rework later (illustrative)
5b	Head of policy administration (signed the change)	One DBA running the load	Commercial record	Policy system	One day of a DBA; years of later reconciliation (illustrative)
6	Head of data platform	Pipeline service account; three data engineers; the warehouse load role	Commercial record; the policy domain also holds personal data of individual policyholders under US federal (GLBA) and state insurance privacy rules; retention duties	Finance and actuarial dashboards; data scientists	About 2,800 dollars a month to store, move and query this domain (illustrative)
7	Head of data platform	Stream platform team; connector configuration	Commercial record; the policy domain also holds personal data of individual policyholders under US federal (GLBA) and state insurance privacy rules; retention duties	Real-time dashboards; from 2025 the service agent's tools, through the warehouse	Stream platform share, about 600 dollars a month (illustrative)
8	Head of customer service (agent owner); head of data platform (data)	Agent platform team; prompt and tool configuration; the vector index job	Commercial record; personal data of individual policyholders under US federal (GLBA) and state insurance privacy rules; agent access logged	The service agent and, through it, customers	Model and retrieval costs, about 1,100 dollars a month for this use (illustrative)
9	Head of customer service	Agents (human and AI); CRM administrators	Personal data; complaint-handling retention	Claims handlers, the next agent, the complaints team	CRM licence share (illustrative)
9b	Head of customer service	Claims handlers; CRM administrators	Personal data; kept under the insurer's complaint-handling retention policy, so the wrong note is flagged, not deleted	Claims handlers, the complaints team, and every later agent, which now reads the flag before the note	Ten minutes of a claims handler (illustrative)

PART III · THE FORCES

CHAPTER ELEVEN

Meaning: the Same Value in Every Representation



THE ERAS OF THE RECORD

On 23 September 1999 NASA lost Mars Climate Orbiter during the maneuver meant to put it into orbit around Mars. The Mishap Investigation Board found that one ground software file reported thruster impulse in pound-seconds where the interface specification required newton-seconds. The numbers were valid in the unit they were written in. The required unit was in the interface specification, not in the file, and nothing checked one against the other. In the same autumn the insurer's conversion team was loading policy 1898/0417 into DB2 and setting a date window that would read the two-digit year 98 in its issue date as 1998. Part II followed the record from the register to the model. This part looks at the forces that act in every era, and meaning comes first: the same value, written nine ways, and what each way leaves out. The decision this chapter changes is small and concrete: **every field that crosses a hop travels with a Type Contract**, seven lines that say its type with precision and rounding, its encoding and serialization, its unit or currency, its time basis, its identity scope, what absence means, and the authority that defines it.

One value, nine forms

Take the one number in the thread that never changes: the deposit of 412.50 dollars paid in 1898. Its value is the same in 2026. Its **representation**, the physical form a reader receives, has changed at almost every hop. Figure 11.1 stacks the forms the thread gives it, from ink to a vector, plus two forms for comparison, the deposit as CSV text and as a binary float, with the bytes computed for this book.

FIGURE 11.1

One deposit, nine representations: one cannot hold every cent, and only some carry their meaning

Policy 1898/0417, the deposit: 412.50 dollars, from the register to the vector

Ink, 1898 **register line 417**	`412.50`	Dollars and 'deposit' live in the ruled column heading, not the line
Card holes, 1931 **80-column card**	`0 0 4 1 2 5 0 (columns 14 to 20)`	No point punched: the card layout says where it falls
Zoned decimal **EBCDIC, display form**	`F0 F0 F4 F1 F2 F5 C0`	One digit per byte; the last zone holds the sign (C = plus)
Packed decimal (BCD) **IMS 1974, DB2 1999**	`00 41 25 0C`	Two digits per byte; the point is in PIC S9(5)V99 COMP-3; DB2 catalog
ASCII text in a CSV **any desk extract (comparison)**	`34 31 32 2E 35 30 "412.50"`	Six characters; every reader parses them again
Binary float **IEEE 754 double (comparison)**	`412.50 - 412.49 = 0.009999999999990905`	412.5 is exact; 412.49 is not, so a one-cent check fails
Parquet DECIMAL(18,2) **warehouse, 2021**	`INT64 41250, scale 2`	Exact; scale in the file schema. STATUS_CODES did not travel
Token ids **tiktoken o200k_base**	`[60868, 25, 220, 31989, 13, 1434, 16713]`	'Deposit: 412.50 dollars' as text pieces: 412 . 50 are three tokens
Vector, 384 numbers **all-MiniLM-L6-v2**	`[0.0667, 0.0432, -0.0170, ...]`	Same sentence with 41250: cosine 0.973, a near neighbour

The deposit's digits survive every form the thread used (the 1962 tape's six-bit BCD is not shown); CSV text and the binary float are shown for comparison, and the float cannot hold every cent. What fails more often is the meaning around the digits: unit, scale, sign and the word 'deposit'.

Author's computation, 19 November 2026 (rerun the same day): Python 3.13 floats; IBM System/360 Principles of Operation (A22-6821-0), pp. 34 to 35; Apache Parquet LogicalTypes (DECIMAL); tiktoken 0.9.0, o200k_base; sentence-transformers/all-MiniLM-L6-v2, mean pooling, normalized.

Read the stack for where the meaning lives, not for the digits. In the register the clerk wrote 412.50 in a ruled column whose printed heading said what it was. On the 1931 card the seven digits 0041250 became one punch per column, with no point punched; the card layout, a sheet in the tabulating supervisor's drawer, said that the last two columns were cents. On the 1962 tape the digits were six-bit BCD characters, and the copybook did the same job. From 1974 the amount sat in packed decimal. Under IMS the decimal point existed only in the COBOL declaration `PIC S9(5)V99 COMP-3` (COMP-3 is IBM's synonym for PACKED-DECIMAL; without it the same picture is zoned), where IBM defines the symbol V as the position of an "assumed decimal point" that takes no storage; from 1999 DB2 also declared it in the column type, DECIMAL, in the database's own catalog. In a CSV extract the meaning

was a header row, if the file had one. In the 2021 warehouse the Parquet file's own schema says DECIMAL(18,2): for the first time the scale travels inside the file, beside the value, where any reader of the file can find it. And in the last two rows the meaning is something the model works out from the surrounding words.

Each form is a reader's form. The tabulator needed holes. The mainframe needed bytes it could add. The CSV served a person and a loader. The token ids serve a language model, and the vector serves a similarity search. A representation is chosen for the reader of its era, and the next era's reader inherits it without the knowledge that made it obvious. That is the hop problem of this book, seen at the level of a single field.

Decimal in the machine: zoned, packed and the copybook

IBM's System/360 Principles of Operation (form A22-6821-0, 1964) defines two decimal formats, and the difference between them is still in every mainframe file a cloud team unloads. In the **zoned** format "the low-order four bits of a byte, the numeric, are normally occupied by a decimal digit", the four high-order bits are the zone, and in the rightmost byte the sign takes the zone position. In the **packed** format "two decimal digits normally are placed adjacent in a byte", with the sign in the right half of the last byte. Packed decimal is binary-coded decimal (BCD): each digit is four bits, 0000 to 1001. Zoned decimal is the character form; the manual notes that in it "the digits are represented as part of an alphameric character set", and that a PACK instruction converts one to the other.

So "EBCDIC packed decimal" is a confusion. EBCDIC is a character code; packed decimal is not characters at all. The manual says which sign codes each code set prefers: when a mode bit selects EBCDIC, decimal results carry plus 1100, minus 1101 and zone 1111; when it selects ASCII, plus 1010, minus 1011 and zone 0101. The copybook declares the deposit as seven digits, 00412.50, so in zoned EBCDIC it is F0 F0 F4 F1 F2 F5 C0, and in packed form 00 41 25 0C: four bytes for seven digits and a sign.

Neither form holds a decimal point. Without the copybook or the catalog, a conversion program that reads 00 41 25 0C sees only 41250, and nothing in the bytes says it is wrong. Hop 6, the nightly unload from DB2 for z/OS to Parquet, worked for the deposit because the unload read the column's declared scale from the DB2 catalog and wrote a matching decimal type. The same unload carried status 7 as a bare integer, because no type in any format says what a code means. The copybook, and from 1999 the DB2 catalog, carried the scale for sixty years. The code table sat beside the data, in STATUS_CODES, and the 2021 unload left it behind.

Floats are not money

The one red row in Figure 11.1 is the binary float, the form the thread avoided. IEEE Std 754-1985, the IEEE Standard for Binary Floating-Point Arithmetic, was approved on 21 March 1985 and later superseded by the 2008 revision. Python's documentation notes that "almost all platforms map Python floats to IEEE 754 binary64 'double precision' values". A binary float stores a number as a sum of powers of two. Many decimal fractions have no exact sum, so the stored value is the nearest one the format can hold.

The classic test, run for this book in Python 3.13, prints `0.1 + 0.2 = 0.30000000000000004`. The stored value of 0.1 is exactly 0.1000000000000000055511151231257827021181583404541015625. The deposit is a trap of a different kind: 412.5 equals 825/2, so it is stored exactly, and every test written with round amounts passes. One cent away the luck ends. The stored value of 412.49 is 412.49000000000009094947017729282379150390625 and of 412.51 is 412.50999999999999090505298227071762084960937 5. A reconciliation that subtracts a hand-typed 412.49 from the deposit gets 0.009999999999990905, not 0.01, and a check written as "difference equals one cent" fails.

Floats are right for measurements, where the error is far smaller than the instrument's. They are the wrong choice for money and identifiers, where the business rule is exact. The fix is a decimal type with a declared scale: COBOL packed decimal, SQL DECIMAL, Parquet DECIMAL, or integer cents. The Parquet format specification defines DECIMAL as "unscaledValue * 10^(-scale)" and allows a 64-bit integer for a precision of up to 18 digits, so DECIMAL(18,2) stores the deposit as the integer 41250 with scale 2. The risk moves to the readers outside the warehouse: a notebook that loads the column into a float, or a JSON serializer that writes it as a number, quietly turns the exact value back into a near one.

Characters: ASCII, EBCDIC, Unicode and UTF-8

When the deposit becomes text it needs a character code. The American Standard Code for Information Interchange (ASCII) was, in the words of the National Bureau of Standards, "first issued in 1963 and then published twice again with minor revisions as X3.4-1967 and X3.4-1968"; the IEEE milestone for ASCII credits the American Standards Association's X3.2 subcommittee with the 1963 first edition. IBM's System/360 used EBCDIC, and its Principles of Operation already carried a mode bit for ASCII signs. The two codes put the same digits at different byte values: "412.50" is 34 31 32 2E 35 30 in ASCII and F4 F1 F2 4B F5 F0 in EBCDIC code page 037. A file of digits moved between them without conversion is not slightly wrong; it is obviously wrong, which is the safer failure. A conversion table that is wrong for only a few characters, such as accented letters, is the dangerous case.

The Unicode Standard, Version 1.0, Volume 1, was first printed in October 1991, with Volume 2 in June 1992. UTF-8, the byte encoding most data now travels in, was, in the words of RFC 3629, "devised in September 1992 by Ken Thompson, guided by design criteria specified by Rob Pike", for the Plan 9 operating system. It was first published as RFC 2044 in October 1996, replaced by RFC 2279 in January 1998 and revised as RFC 3629 in November 2003, which restricted the range to U+0000 to U+10FFFF. UTF-8 encodes the ASCII characters as the same single bytes, which is why a CSV of digits looks identical in both and why a name with an accent is where an encoding mistake first shows.

Hop 6 converted EBCDIC and packed decimal to UTF-8 text and DECIMAL(18,2). For the deposit that was lossless. For the insured's name, written in 1898 and retyped onto cards, tape and screens, it depended on whether each conversion table knew every character. The Type Contract names the encoding so that the next team does not have to guess it from the bytes.

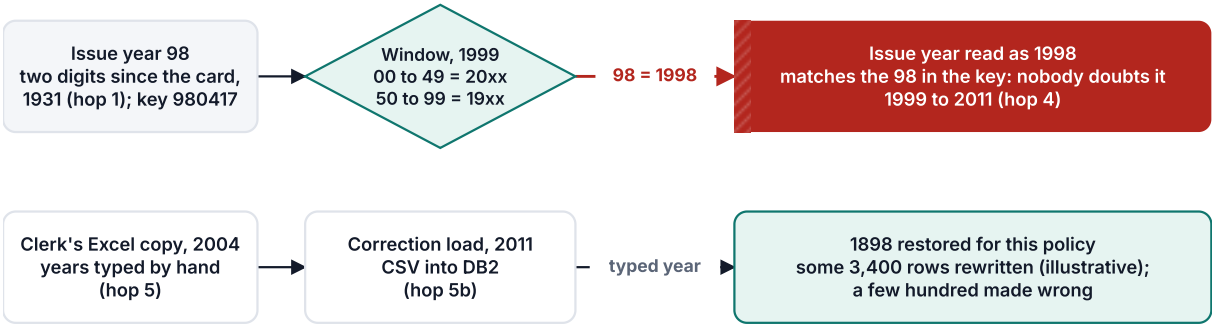
Time and units: zones, windows and the wrong field

A date is a value with a hidden basis. The health system's laboratory results show it daily. A specimen collected at 23:40 in a clinic on Eastern time, with the result stored as a local time without an offset, is treated as 23:40 UTC, four or five hours early, by a warehouse that assumes UTC, and a 24-hour rule for repeat testing starts from the wrong moment. HL7's FHIR standard, Release 5, closes the gap at the format level: for the dateTime type, "If hours and minutes are specified, a timezone offset SHALL be populated", and an instant "SHALL include a timezone offset". A FHIR date "SHALL" have no offset, so a date of birth is not shifted by a time zone. Older feeds and spreadsheet copies carry no such rule, and the time basis has to be written into the contract by hand.

The thread's own time loss is the key. Chapter Six tells the Y2K story; here one sentence is enough: from 1999 the insurer's date window read 00 to 49 as 20xx and 50 to 99 as 19xx. The key 980417 was born in 1931 by dropping the century to save card columns, and the issue-date field shared the habit. Figure 11.2 shows what the window did to a policy written in a century it did not expect, and how a hand-typed year in a desk copy restored the truth twelve years later while making a few hundred other rows wrong.

FIGURE 11.2

The Y2K window read the issue year 98 as 1998, and a hand-typed year undid it twelve years later



The window was a rule about the century, applied to a date field written in a century the rule did not expect; the key's matching 98 made the wrong year look right.

Author's diagram based on the book's thread (policy 1898/0417, hops 1, 4, 5 and 5b); the window rule is the one the insurer chose, as told in Chapter Six.

A key that holds a year is two values in one field. The register number 1898/0417 meant "line 417 of the 1898 register"; to the keypunch operator, 980417 still meant line 417 of the 1898 register. The issue-date field kept the same two digits, and to the date window of 1999 they were a year in need of a century. The general rule is old and still broken weekly: a key identifies, it does not describe. When a key must keep its old digits, the contract says which digits are a date and in which century.

Units, ranges and fields fail the same way, and the official reports are exact about it.

SOURCED FAILURE: ARIANE 501, 4 JUNE 1996

The Inquiry Board chaired by Jacques-Louis Lions reported on 19 July 1996. Its chain of technical events (section 2.1) is a representation failure end to end. In the inertial reference system, an alignment function that "serves no purpose" after lift-off was set to keep running for about 40 seconds of flight, a sequence "based on a requirement of Ariane 4". A "data conversion from 64-bit floating point to 16-bit signed integer value" received a horizontal bias value (BH) "greater than what could be represented by a 16-bit signed integer", because Ariane 5's early trajectory gave "considerably higher horizontal velocity values". The result was an Operand Error; the conversion was not protected, "although other conversions of comparable variables in the same place in the code were protected". The backup unit had failed the same way one 72-millisecond cycle earlier. The active unit then sent "a diagnostic bit pattern" that the on-board computer "interpreted as flight data", commanded full nozzle deflection, and the launcher broke up. The Board's conclusion (page 13) dates the loss of guidance at "37 seconds after start of the main engine ignition sequence (30 seconds after lift-off)". Two readers, two meanings: a value too large for its type, then a status word read as data.

SOURCED FAILURE: MARS CLIMATE ORBITER, 23 SEPTEMBER 1999

The Mishap Investigation Board's Phase I report (10 November 1999) states the root cause (section 4) as "the failure to use metric units in the coding of a ground software file, 'Small Forces,' used in trajectory models". The Software Interface Specification required thruster impulse in newton-seconds; "Instead, the data was reported in English units of pound-seconds (lbf-s)". Section 2 records that on 29 September the team found the small-forces velocity changes were low by a factor of 4.45, "(1 pound force=4.45 Newtons)". The interface had a specification. The file did not carry its unit, and no reader checked it against the specification.

SOURCED FAILURE: AN ENTERPRISE VERSION, 2 MAY 2022

The UK Financial Conduct Authority's Final Notice to Citigroup Global Markets Limited (17 May 2024) describes the same failure on a trading desk. A trader "intended to sell a basket of equities to the value US$58m" but "erroneously loaded a basket with a notional size of US$444bn comprising 349 stocks" (paragraph 2.4). The reason (paragraph 2.5): "The trader had entered the value of the basket of equities in the wrong field, the unit quantity field rather than the notional value field." In the order system either field could be filled in to set the size of a basket (paragraph 4.10). Hard blocks stopped part of the basket; US$1.4bn of sell orders were executed before the trader cancelled the rest at 09:10 (paragraphs 2.6 and 4.6), and the incident "resulted in a US$48m loss" for the firm (paragraph 4.7). The FCA fined the firm GBP 27,766,200 for failures in its trading systems and controls. A number entered once, with the right digits, in a field whose unit was different.

The Type Contract, and the model as the last reader

In each failure above the meaning existed somewhere: in a specification, an interface document, a field label. What failed was that the meaning did not travel with the value to the reader who needed it. The Type Contract is the smallest fix: seven lines per field, written once, carried at every hop as metadata next to the data, and checked by the reader on arrival.

TYPE CONTRACT: THREE FIELDS OF POLICY 1898/0417

Line	Deposit	Status code	Issue date
Type, precision and rounding	decimal, two places; never a binary float	one-digit integer code, 0 to 9; no arithmetic	calendar date, four-digit year
Encoding and serialization	DECIMAL(18,2) in the warehouse; packed decimal, S9(5)V99 COMP-3, at source	one-digit integer	ISO 8601 date
Unit or currency	US dollars	none	calendar day
Time basis	amount as at 14 March 1898; not indexed	state as at the last change event, by event time	local date at the issuing office, no offset
Identity scope	the single deposit paid at issue on policy 1898/0417, held by the insurer, refundable on cancellation; not a premium	the current state of policy 1898/0417	the issue of policy 1898/0417
Absence	empty means not recorded, never zero	unknown; a reader must not infer lapse	not known; never derived from the key
Authority	the policy record as issued	the code book (on paper from 1931; the STATUS_CODES table in DB2 from 1999), attached by reference and version: 7 = perpetual, deposit held, no premium due	the policy record as issued

Now the last two rows of Figure 11.1. A language model is not handed bytes or column types. It reads **tokens**, pieces of text from a fixed vocabulary. OpenAI's open-source tiktoken library describes itself as "a fast BPE tokeniser for use with OpenAI's models"; with its

o200k_base encoding, the sentence "Deposit: 412.50 dollars" becomes seven ids, and the amount is three of them: "412", "." and "50". The cl100k_base encoding splits it the same way but gives different ids to the digits (17574 and 1135; the period is 13 in both): a token id has meaning only inside the tokenizer that made it. The model sees no DECIMAL, no scale and no unit; it sees text that looks like money.

A retrieval index goes one step further and turns the text into a **vector**. The open model all-MiniLM-L6-v2, whose model card says it "maps sentences & paragraphs to a 384 dimensional dense vector space", gives the deposit sentence a vector that begins 0.0667, 0.0432, -0.0170. We compared it with near variants by cosine similarity, where 1 means the same direction. "Deposit: 41250 dollars", a value one hundred times larger, scored 0.973. "Deposit: 412.05 dollars" scored 0.965. "Premium: 412.50 dollars", the same number with a different word, scored 0.469. The vector holds the meaning of the words far more than the value of the number, which is what a similarity search wants and the opposite of what an amount needs.

That is the planted error of this book in miniature. The service agent in 2026 got a row with status 7 and no premium since 1898, from a warehouse where the code table had not arrived. It had no contract to read, so it read the words it had, and in ordinary usage "no premium" sits close to "lapsed for non-payment". A Type Contract on the status column, passed to the tool with the row, gives the model the one sentence it was missing.

> **OUR VIEW, AS OF NOVEMBER 2026**
>
> Models will get better at noticing that a code needs a table. They will not get better at knowing a table they were never given. Treat the contract as part of the data an agent receives, not as documentation a person might read.

ERA CARD

Reader. Every era's reader, from the clerk with a column heading to the model with a token stream; each needed the meaning in a different place.

Security. A representation error is also a control gap: a field that accepts a quantity or a value, or a key that hides a date, lets a correct-looking entry do damage that no access control stops.

Cost. Contracts are cheap to write at the hop and expensive to reconstruct later; Ariane 501, Mars Climate Orbiter and a US$48m trading loss were all paid at the reader's end.

AI. Tokens and vectors carry the words around a value better than the value itself; an agent needs the contract with the row, or it supplies a plausible meaning.

Monday rule: pick the five fields your agents and dashboards read most, write a seven-line Type Contract for each (type with precision and rounding, encoding and serialization, unit or currency, time basis, identity scope, absence, authority), and make the tool return it with the data.

CHAPTER TWELVE

Control and Law: Who Could Change It, Who Could Read It



THE ERAS OF THE RECORD

On 7 July 2023 Royal Commissioner Catherine Holmes signed the report of the Royal Commission into the Robodebt Scheme. It describes a government program that compared two sets of records, spread one of them evenly across fortnights it did not describe, and raised debts against people on the result. Its verdict, on page xxix of the first volume: "Robodebt was a crude and cruel mechanism, neither fair nor legal". The scheme did not rest on one clerk changing one record. It rested on a rule applied to hundreds of thousands of them. Chapter Eleven asked what a value means. This chapter asks who could change it and who could read it, from the clerk with the key to the register room to a service account and an agent, and which laws apply to each running company. The decision it changes: **every reader and writer of an important record, human or machine, has a named owner and a written purpose**, and the access review lists the machines.

From the keeper's key to the agent

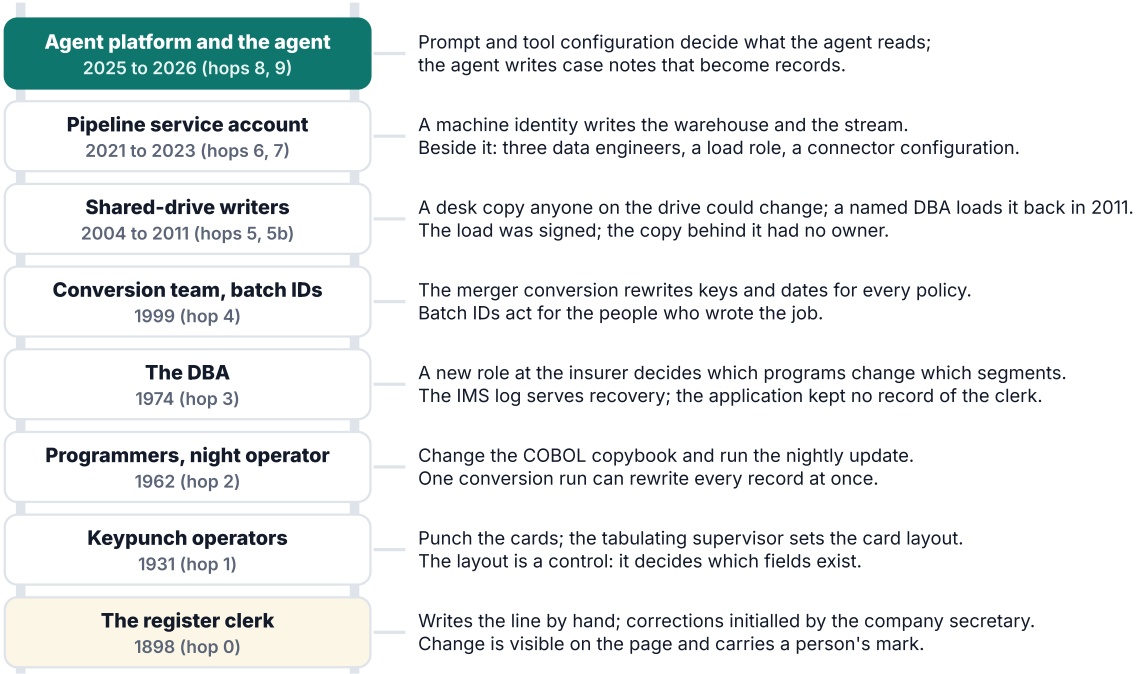
In 1898 the answer to "who can change policy 1898/0417?" was a person you could name. The register clerk wrote the line; the company secretary initialled corrections; the bound volume sat in one room. A change left ink on the page, and the initials showed who had approved it. That is control by custody and by signature.

In the thread, the power to change the record then spread from named custodians to roles, batch identities, service accounts and configuration. Attribution did not vanish at every hop (the 2011 load was signed), but it stopped living in one place. Figure 12.1 follows it up the thread's own column "who can change it". The keypunch operators changed what they punched, but the tabulating supervisor's card layout decided which fields could exist at all. From 1962 a programmer who changed the COBOL copybook changed the layout every program expected, and one nightly conversion or update run could then rewrite every record on the tape. With the 1974 IMS database the insurer created a new role, the database

administrator (DBA), who with the programmers decided which programs could change which segments of the hierarchy. The clerk at a screen changed a segment without leaving a mark on it; the change reached the IMS log, which the insurer kept for recovery, and its application kept no record of which clerk had made it. In 1999 the merger conversion rewrote keys and dates for every policy under batch IDs that acted for the people who wrote the job.

FIGURE 12.1

Who could change policy 1898/0417: from one named clerk to roles, service accounts and an agent



Across the eras, the power to change the record spread from named custodians to roles, batch identities, service accounts and configuration; attribution did not vanish, but it stopped living in one place, and the access review has to follow it.

Author's diagram based on the book's running example (Ledger column 'who can change it', hops 0 to 9 plus 5b).

The upper rungs are where most teams now lose track. The 2004 desk copy had no owner and anyone on the shared drive could change it; in 2011 one DBA loaded it back under a change the head of policy administration signed, so a signed change carried an unowned copy into the system of record. From 2021 a pipeline service account writes the warehouse, and from 2023 a connector configuration decides what the stream sends. In 2025 the service agent received a read-only SQL tool and a retrieval index, and in 2026 its answer became a

case note, a record written by a machine. The question "who could change it?" now has three kinds of answer: people with roles, machine identities with grants, and configurations that the insurer's access review did not list.

Reading changed in the same way. The register's readers were clerks, the auditor and the board. Among the 2026 readers is a model called by a customer, through a tool whose purpose nobody wrote down. That is hop 8's RIGHTS loss: the data reached a new reader without the reason it was collected.

Where the privacy principles came from

Confidentiality rules are older than computers, but the modern principles on who may read a personal record began as answers to computers that held personal records. In July 1973 the US Department of Health, Education and Welfare published *Records, Computers and the Rights of Citizens*, the report of the Secretary's Advisory Committee on Automated Personal Data Systems, chaired by Willis H. Ware of the RAND Corporation. Its five safeguards are still the shortest statement of the field:

- "There must be no personal data record-keeping systems whose very existence is secret."
- "There must be a way for an individual to find out what information about him is in a record and how it is used."
- "There must be a way for an individual to prevent information about him that was obtained for one purpose from being used or made available for other purposes without his consent."
- "There must be a way for an individual to correct or amend a record of identifiable information about him."
- Any organization holding such records "must assure the reliability of the data for their intended use and must take precautions to prevent misuse of the data."

On 23 September 1980 the Council of the OECD adopted its Guidelines Governing the Protection of Privacy and Transborder Flows of Personal Data, revised on 11 July 2013. They set eight principles: collection limitation, data quality, purpose specification, use limitation, security safeguards, openness, individual participation and accountability. Read them against the Ledger. The Ledger has no purpose column: for personal data the purpose and legal basis are written in column 8, beside the classification, and the RIGHTS code marks a hop where they did not travel. That is where purpose specification and use limitation land. Data quality is the proof question of Chapter Fourteen. Accountability is the owner column. The third HEW safeguard, one purpose not silently reused for another, is put at risk when a retrieval index embeds every scanned document for every question without recording the purpose each was collected for.

The laws, by the company they reach

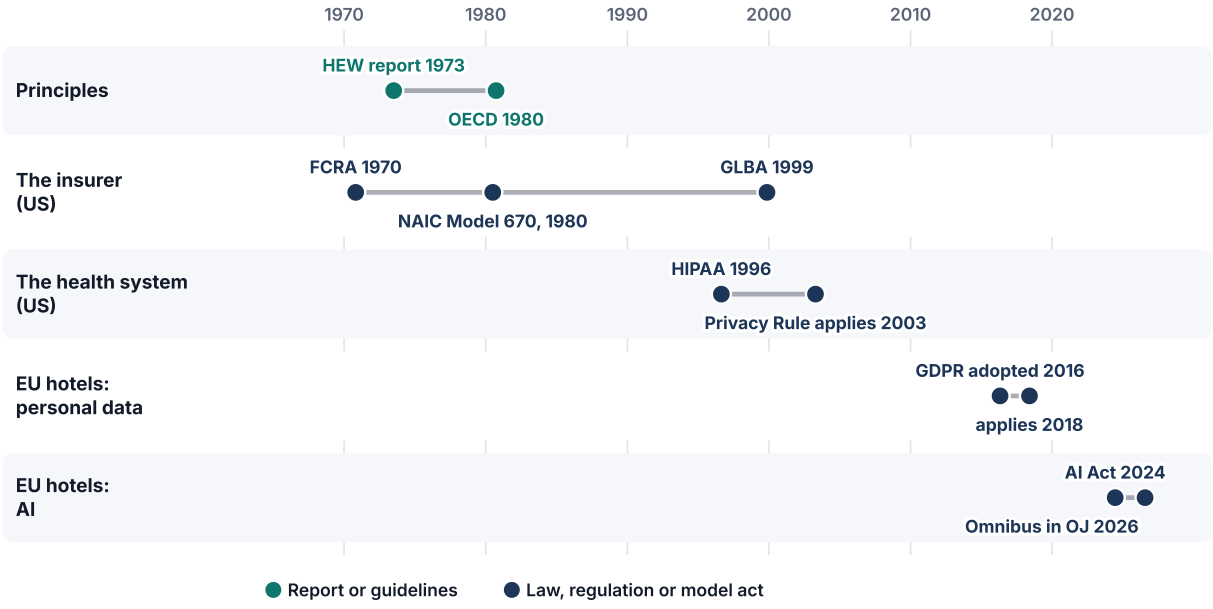
No company is bound by "privacy law" in general. It is bound by specific statutes, for specific records, in specific places. Table 12.1 lists the principles, statutes and model rules that may apply to the three running companies, and Figure 12.2 puts them on one time axis. Whether each one applies depends on the record, the activity and the company's role.

Company	Law	Date	What it asks of the record
The insurer	Fair Credit Reporting Act (FCRA), 15 U.S.C. 1681 et seq.	26 Oct 1970	Rules for consumer reports and their users, such as an insurer that underwrites with them
The insurer	NAIC Insurance Information and Privacy Protection Model Act (Model 670)	Adopted 1980; revised 1981; amended 1982	A model for state legislatures; it binds only where a state enacts it
The insurer	Gramm-Leach-Bliley Act (GLBA), Pub. L. 106-102, Title V	12 Nov 1999	Privacy notices; no disclosure of nonpublic personal information to nonaffiliated third parties without notice and a chance to opt out (section 502), subject to the exceptions in section 502(e); enforced for insurers by the state insurance authority (section 505(a)(6))
The insurer	State record retention rules	Varies by state	How long policy and claim records are kept; set state by state
The health system	HIPAA, Pub. L. 104-191, and its Privacy Rule (45 CFR Parts 160 and 164)	21 Aug 1996; compliance required for most entities from 14 Apr 2003	Uses and disclosures of protected health information; policies and documentation kept six years (45 CFR 164.530(j))
The hotel group's EU hotels	GDPR, Regulation (EU) 2016/679	27 Apr 2016; applies from 25 May 2018 (Art. 99)	Purpose limitation and storage limitation (Art. 5(1)(b), (e)); right to erasure (Art. 17)

Company	Law	Date	What it asks of the record
The hotel group's EU hotels	AI Act, Regulation (EU) 2024/1689, as amended by Regulation (EU) 2026/1744	13 Jun 2024; amended 8 Jul 2026	Data governance for high-risk systems (Art. 10); duties of general-purpose model providers (Art. 53); which apply depends on the system's class and the group's role (provider or deployer)

FIGURE 12.2

The laws that may apply to each company arrived over half a century, and the newest was amended in 2026



The insurer's FCRA of 1970 came before the HEW and OECD principles; the laws after them were mostly written for one sector or one region, so each company carries its own set.

Sources: HEW (1973); OECD/LEGAL/0188; 15 U.S.C. 1681; Pub. L. 106-102; NAIC Model 670; Pub. L. 104-191 and 45 CFR 164.534 (compliance date); Regulations (EU) 2016/679, 2024/1689 and 2026/1744 (EUR-Lex). NAIC model placed at its year of adoption; the 2026 amendment at its Official Journal date, 24 July 2026 (adopted 8 July).

The insurer. The FCRA was enacted on 26 October 1970 as title VI of the Consumer Credit Protection Act. The GLBA, signed on 12 November 1999, made financial institutions, a term that covers insurers, give privacy notices and, outside the exceptions the act lists, let consumers opt out before nonpublic personal information goes to nonaffiliated third parties. Section 505 leaves enforcement for "any person engaged in providing insurance" to "the applicable State insurance authority", which is why the National Association of Insurance Commissioners (NAIC) models matter: Model 670, adopted in 1980, and Model 672, the privacy regulation adopted in 2000 to carry the GLBA into state insurance law. For policy 1898/0417 the hard question is not consent. It is retention: a perpetual policy is never finished, so a rule written as "keep for N years after expiry" never starts its clock.

The health system. HIPAA was approved on 21 August 1996; most covered entities had to comply with its Privacy Rule from 14 April 2003 (45 CFR 164.534). The rule makes the health system keep its privacy policies and documentation "for six years from the date of its creation or the date when it last was in effect, whichever is later" (45 CFR 164.530(j)(2)). HHS states that the Privacy Rule does not set a retention period for medical records themselves; state law does. A lab result therefore carries at least three clocks: federal rules for the privacy paperwork and for the laboratory's own test reports, which it must keep at least two years, pathology reports ten (42 CFR 493.1105), and state law for the medical record.

The hotel group's EU hotels. The GDPR, Regulation (EU) 2016/679 of 27 April 2016, "shall apply from 25 May 2018" (Article 99(2)). Article 5(1)(b) requires personal data to be "collected for specified, explicit and legitimate purposes and not further processed in a manner that is incompatible with those purposes"; Article 5(1)(e) requires that they be "kept in a form which permits identification of data subjects for no longer than is necessary". Where the group relies on consent, it "shall be able to demonstrate" it (Article 7(1)); a consent box that lives only in a web session log, hop 1 of the second Ledger in Chapter Fifteen, is then a GDPR problem before it is a data problem.

The AI Act, Regulation (EU) 2024/1689 of 13 June 2024, entered into force on 1 August 2024. Its Article 10 governs the training, validation and testing data of high-risk AI systems. Article 10(2) requires "data governance and management practices appropriate for the intended purpose", including "data collection processes and the origin of data, and in the case of personal data, the original purpose of the data collection". Article 10(3) requires data sets that are "relevant, sufficiently representative, and to the best extent possible, free of errors and complete in view of the intended purpose". That duty overlaps the Ledger: origin is its From column, and the original purpose goes in column 8, though Article 10 does not ask for a Ledger as such. Article 53(1)(d) requires providers of general-purpose AI models to publish "a sufficiently detailed summary about the content used for training", on a template from the AI Office.

Regulation (EU) 2026/1744 of 8 July 2026, the "Digital Omnibus on AI", published in the Official Journal on 24 July 2026 and in force from 27 July 2026, moved the dates. Under the amended Article 113, Chapter III, Sections 1, 2 and 3 (except Article 6(5)) apply from "2 December 2027 as regards AI systems classified as high-risk pursuant to Article 6(2) and Annex III" and from "2 August 2028 as regards AI systems classified as high-risk pursuant to Article 6(1) and Annex I". It deleted Article 10(5) and moved the processing of special categories of personal data for bias detection into a new Article 4a. Whether the group's pricing agent is a high-risk system is a question for its lawyers. Article 10 covers training, validation and testing data, not every row an agent reads at run time, but its duties are worth meeting either way, and a filled Ledger is documentation that can support them; whether it suffices is for counsel.

Records that harmed people

Access control asks who may change a record. The two cases below ask a harder question: what happens when an institution changes many records by rule, and treats the result as true. Each is told only from its official report.

SOURCED FAILURE: ROBODEBT, AUSTRALIA, FROM 2015

The Royal Commission's report (7 July 2023) explains the mechanism in its executive summary (volume 1, page xxiv). Income that employers had reported to the tax office for a period was compared with what welfare recipients had declared fortnight by fortnight: "the amount declared by the employer would be divided up evenly, and allocated into the number of fortnights included in the period being reviewed", a process the report says was called income "averaging", "smoothing" or "apportioning". Where the averaged amount pointed to an overpayment, a debt was raised. The scheme "began with some pilots in 2015, was rolled out with its online platform in September 2016 and continued in various iterations for four years" (page xxiv). The report records "approximately 794,000 debts" raised across "approximately 526,000 recipients", and that the department expected to write off "a total of $1.751 billion in debts, including the refund of debt payments received of $746 million" (volume 2, page 402); the Federal Court approved a class action settlement of $112 million, inclusive of legal costs (page 401). On 23 June 2026 the Court set those 2021 orders aside and approved a new settlement with a compensation sum of $475 million (*Prygodicz v Commonwealth of Australia (Robodebt settlement)* [2026] FCA 1029). In Ledger terms the losses were TIME and MEANING: a figure for a period was given a fortnightly meaning it never had, and the derived record was believed over the people it described.

SOURCED FAILURE: THE DUTCH CHILDCARE BENEFITS AFFAIR

The parliamentary questioning committee on childcare benefits (parlementaire ondervragingscommissie) of the Dutch House of Representatives delivered its report, *Ongekend onrecht* ("Unprecedented injustice"), on 17 December 2020 (Kamerstuk 35 510, nr. 2). Its first finding (page 7 of the House's PDF, whose pages are cited here) is that in administering the benefit "fundamental principles of the rule of law were violated", and that the blame falls not only on the administration but also on the legislature and the courts. Where any shortcoming was found, "such as an unpaid or partly paid own contribution or a missing signature", the benefit for the whole year was reclaimed; the committee notes that this "all-or-nothing" approach "does not follow directly from the law" (pages 20 and 22). A label of "intent or gross negligence" meant no personal payment arrangement and, in principle, no end date for collection (pages 24 to 25), and it branded people as deliberate fraudsters "also at other agencies". The report records that State Secretary Van Huffelen told parliament in March 2020 that the label had been given to 25,000 to 35,000 people between 2012 and 2019, and months later that in a sample 94 percent could be considered unjustified by current standards (page 25). A separate investigation by the Dutch Data Protection Authority (report dated 16 July 2020, published 17 July 2020) found that the benefits administration processed applicants' nationality unlawfully, including as an indicator in a risk classification model "at least from March 2016 to October 2018", and that this use was discriminatory (chapter 4). The committee's quotations are the author's translations from the Dutch.

Both cases have a Ledger shape. A record was derived (an average, a label), the derivation lost what made it fair (the fortnights, the reason), and the derived record became the one other readers trusted. Hop 9 of the thread, the wrong answer saved as a case note, is the same shape at the scale of one policy.

The census: who may read a record in aggregate

Census records show that the rules on reading can change under pressure. The wartime cases are documented by historians, not by an official report. In a study in *Social Research* (2001), William Seltzer and Margo Anderson describe how, by early 1941, the head of the Dutch population register adapted the "cradle to grave" registration system "to create special registration systems covering the Jewish and Gypsy populations", and that these systems "played an important role in the apprehension of Dutch Jews and Gypsies" (page 486). For the United States they document that in 1942 the Census Bureau provided tabulations and block-level maps used in the removal of Japanese Americans (page 492). In a later peer-reviewed article (*Journal of Privacy and Confidentiality*, 2009), Anderson and Seltzer show that section 1402 of the Second War Powers Act of 27 March 1942 (56 Stat. 186, on the availability of reports to other government agencies) "explicitly set aside the confidentiality

provisions of the Census Act" for data needed in the war, from the spring of 1942 to the end of March 1947 (pages 9 and 17). The lesson for a CTO is not about wartime. A legal basis for reading can be changed by a later law, and the data already collected is what the new rule reads.

The Bureau's answer for the 2020 Census is a reading control built into the published numbers. Before the 2020 Census it ran an experiment on its own 2010 tables: it reconstructed records for "144 million people or 46 percent of the U.S. population" with block, sex, age, race and ethnicity identical to the census responses, and could correctly re-identify "about 52 million people or 17 percent" of the 2010 population (*Disclosure Avoidance for the 2020 Census: An Introduction*, 2021, chapter 1). For 2020 it adopted differential privacy, a method that adds calibrated noise and sets a stated bound, the privacy-loss budget, on what the published tables can reveal about any one person, through its TopDown Algorithm. On 9 June 2021 it set the privacy-loss budget for the redistricting data at "ε=19.61, which includes ε=17.14 for the persons file and ε=2.47 for the housing unit data", and it released those data on 12 August 2021. The point is not the mathematics. "Who could read it" applies to totals as well as rows: enough accurate aggregates can rebuild many of the records they came from, and an agent that can run many aggregate queries is that kind of reader.

WHAT HASN'T CHANGED

The HEW committee wrote in July 1973 that information "obtained for one purpose" must not be used for other purposes without consent. Fifty-one years later, Article 10(2) of the EU AI Act asks a high-risk system's provider to record, for personal data, "the original purpose of the data collection". The two duties differ in who owes them and why, and share one idea: the purpose has to travel with the record.

ERA CARD

Reader. Every era's reader, from the auditor with the bound register to a model reached through a tool; the law follows the reader and the purpose, not only the storage.

Security. The power to change the record moved from named people to roles, service accounts and configurations; access reviews that list only people miss most of it.

Cost. An access review and a written purpose are small next to the harms in this chapter: a government scheme that ended in a Royal Commission, a parliamentary committee that found the rule of law breached.

AI. What an agent reads is the company's own use or disclosure under every law here: give each agent a recorded purpose, a named owner and only the columns that purpose needs.

Monday rule: list every service account, connector and agent that can read or write your five most important datasets, and give each one a named owner and a written purpose before the next access review.

CHAPTER THIRTEEN

Cost and Death: Store, Move, Query, Keep, Delete



THE ERAS OF THE RECORD

At about five o'clock on the afternoon of 10 January 1921, smoke was noticed in the Commerce Building in Washington. The population schedules of the 1890 census, the forms Hollerith's machines had counted, were stacked on pine shelves in an unlocked basement file room, outside the vault. The Census Director estimated that 25 percent were destroyed and half of the rest damaged by water, smoke and fire, according to the National Archives' own account. For twelve years the damaged schedules waited in storage, while historians and genealogical societies protested against any plan to destroy them. On 21 February 1933 Congress authorized the destruction of the papers, one day after the cornerstone of the National Archives building was laid, and they were destroyed in 1934 or 1935; the reports disagree. Today fragments with no fewer than 6,160 names survive from a count of the whole nation. The fire damaged the record. A decision, written into a list of useless papers, ended it. This chapter prices all five things every record costs and spends most of its pages on the last two, keeping and deleting. The decision it changes is simple to state: **deletion is a hop, and it gets a Ledger row like any other**.

Five verbs on the bill, and one line that is not

The fourth of the six questions asks what a record costs, and the answer has five parts. To **store** is to hold the bytes. To **move** is to copy them across a network or a hop. To **query** is to read them, which on a cloud platform is often billed by the data scanned or the compute used. To **keep** is to store them for as long as a rule or a reader needs, which is storage multiplied by time, plus the work of proving you still have them. To **delete** is to end every copy and show that you did.

The price of the first two is far lower in 2026 than in 1956 or 2006, and it is tempting to draw that as one smooth curve. The table below refuses to. Each row is a dated price from a primary source, and no two rows are the same product.

Date	What was priced	Price as published	Source
June 1956 (first shipped)	IBM Model 350 disk unit, part of the 305 RAMAC; the equivalent of 3.75 megabytes on 50 disks	Leased for 750 dollars a month (about 200 dollars per megabyte per month)	Computer History Museum, The Storage Engine
14 March 2006	Amazon S3 at launch: storage and transfer	0.15 dollars per GB of storage per month; 0.20 dollars per GB transferred	Amazon press release, 14 March 2006
21 August 2012	Amazon Glacier at launch: archive storage, slow to read back	"as little as $0.01/GB/month"	Amazon press release, 21 August 2012
19 November 2026	S3 Standard storage, US East (N. Virginia), first 50 TB a month	0.023 dollars per GB-month (the class the price list names "Amazon Glacier" 0.0036; Intelligent-Tiering's Deep Archive Access tier 0.00099)	AWS Price List API, publication of 19 November 2026
19 November 2026	Data transfer out to the internet from the same region, first 10 TB a month beyond the free tier	0.09 dollars per GB	AWS Price List API, same date

Read the table for what it does show. A leased disk in 1956 and a cloud object store in 2026 are different services: one includes the machine room and its engineers, the other includes durability across sites and an API. The 2006 transfer price was one number; in 2026 moving data out costs about four times as much per gigabyte as storing it for a month. The archive rows are cheap to keep and slow or costly to read back. None of the rows is adjusted for inflation, and none of them prices a query. That is why the book draws no curve. A curve would hide the fact that the bill's shape changes with the reader: a warehouse that is cheap to store and expensive to scan, an index that is cheap to query and expensive to rebuild.

The thread gives the shape for one domain. At hop 6 the insurer's warehouse costs about 2,800 dollars a month to store, move and query the policy domain; the stream at hop 7 adds about 600, and the agent's model and retrieval at hop 8 about 1,100 (all illustrative). Each row in the Ledger also carries a line that no price list shows: the clerk's two hours a year on the register in 1898, the two keypunch operators in 1931, the DBA's day for the 2011 correction load, and the years of reconciliation that followed it.

> **OUR VIEW, AS OF NOVEMBER 2026**
>
> In most data programs we have seen, people are the largest line: the engineers who build and repair hops, the analysts who reconcile, the owners who answer questions. That is a principle, not a measured split, and you should test it on your own five datasets. Put a person-days estimate in column 10 of every Ledger row next to the platform cost. If the people line is missing, every saving on storage will look larger than it is.

Keeping: retention, and the hold that stops the clock

A **retention rule** says how long a class of record must be kept and when it must be ended. It is the "keep" and the "delete" verbs written down together. Retention comes from law (the state record rules that bind the insurer, the health records rules that bind the hospitals), from contracts, and from the business's own need. Chapter Twelve lists the laws by company. This chapter is about what happens to the clock.

The clearest statement of the clock's limits is a set of opinions by Judge Scheindlin of the US District Court for the Southern District of New York, in an employment case, *Zubulake v. UBS Warburg*. Laura Zubulake, a director and senior salesperson on its US Asian equities sales desk, sued her former employer. The e-mails she needed were, in part, no longer on UBS's servers and existed only on backup tapes; in the first opinion (2003) the court ordered UBS to pay to restore a sample of them. When tapes were restored, some were missing and others held e-mails absent from the active files.

In the fourth opinion, of 22 October 2003, the court set out the rule that records managers still quote: "Once a party reasonably anticipates litigation, it must suspend its routine document retention/destruction policy and put in place a 'litigation hold' to ensure the preservation of relevant documents." As a general rule, the court said, the hold "does not apply to inaccessible backup tapes (e.g., those typically maintained solely for the purpose of disaster recovery)", which may keep being recycled on schedule; but if a company can identify where the documents of the "key players" sit on backup tapes, those tapes should be

kept when the information is not available elsewhere (220 F.R.D. 212, 218). UBS's own policy kept monthly backup tapes for three years, and the court found the duty to preserve arose as early as April 2001.

The fifth opinion (2004) is about whether a hold works. UBS's lawyers had issued one in August 2001 and repeated it. Employees deleted relevant e-mails anyway; at least 45 were later recovered from tapes or colleagues' files, and recovered e-mails reached Zubulake 22 months after she asked for them (229 F.R.D. 422, 429, 436). The court required counsel to issue the hold at the outset, re-issue it "periodically", speak directly to the key players, and make sure backup tapes the party must keep are "identified and stored in a safe place" (at 433 to 434). It ordered that the jury be given an adverse inference instruction for e-mails deleted after August 2001 (at 437): the jury could infer the missing evidence would have been unfavorable to UBS. That standard is dated: since 1 December 2015, Federal Rule of Civil Procedure 37(e)(2) allows that instruction for lost electronic information only on a finding that a party "acted with the intent to deprive" another of its use.

Three lessons carry beyond US civil procedure. A retention schedule is not a deletion guarantee: a hold can stop it on any day. A hold is not a memo: it is a state on specific records, in specific stores, that someone must check. And the stores that matter include the ones nobody thinks of as records, like backup tapes, if they are the only place a copy survives.

Deleting: backups, immutable logs and the arithmetic of copies

Deleting a row from the primary database ends one copy. The thread shows how many others a record of any age collects. Policy 1898/0417 has a primary row in DB2, a nightly copy in the warehouse (hop 6), events in a change stream (hop 7), chunks and vectors of its scanned register page in a retrieval index (hop 8), a case note in the CRM (hop 9), and, through the 2004 branch at hop 5, an Excel workbook on a shared drive that the thread records as having no owner and "no retention rule applied". This policy is perpetual and will not be deleted. The insured's personal details in those copies are another matter: where a law or contract gives a right to deletion, they are what a deletion request reaches.

Backups make the arithmetic plain. Take an illustrative schedule: a daily backup kept 35 days and a monthly backup kept 7 years. Delete a customer's row today. The daily copies that hold it expire over the next 35 days. Up to 84 monthly copies made while the row existed still hold it, and the newest of them expires seven years from now. So the honest answer to "when is this person's data gone?" is "from the live system today, from daily backups in five weeks, from monthly backups in up to seven years", unless the schedule changes or the

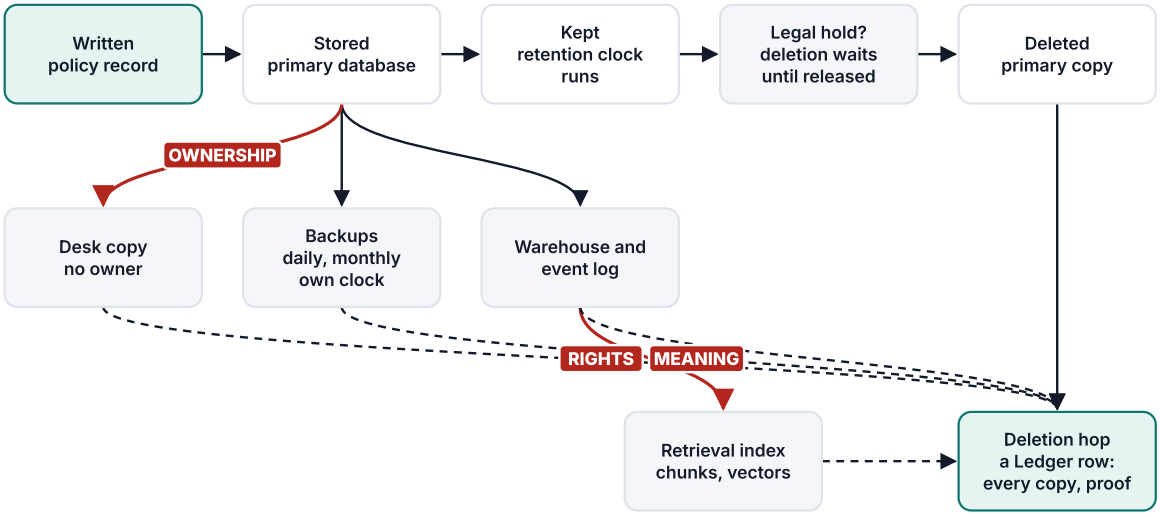
restore process is built to re-apply deletions after a restore. A team that cannot say which of those it does may have deleted the live row, but cannot yet say when every recoverable copy will be gone.

Some stores are built so that nothing can be deleted early. Amazon describes S3 Object Lock as a "write-once-read-many (WORM) model". In its compliance mode, a protected object version "can't be overwritten or deleted by any user, including the root user", until its retention date; a legal hold "has no expiration date" and lasts until removed. That can be the right tool for an audit log or a regulatory archive, and the wrong place to put personal data that may have to be erased in year two of a seven-year lock. The conflict is not solved by a better delete command. It is solved before the write, by deciding what goes into an immutable store, or by crypto-shredding: encrypting each person's data with its own key, so that destroying the key, and every backup of it, leaves the locked copies unreadable, provided no plaintext copy survives elsewhere. Whether a regulator accepts that as erasure is a question to settle before the design, not after. Whichever you choose, write it into the hop's row.

Figure 13.1 draws the lifecycle with deletion as a hop. The primary copy ends on the top row. The deletion is finished only when the bottom-right box can be filled in: every copy named, its own clock stated, and evidence that each ended or why it may not.

FIGURE 13.1

Deletion is a hop: the record dies in the primary store, and every copy needs its own end



Red chips: loss codes (Chapter 2). Dashed arrows: copies the deletion hop must also reach.

A record's lifecycle from writing to deletion; a legal hold suspends deletion until it is released, and the deletion hop is finished only when every copy (backups, warehouse and log, retrieval index, desk copy) is named and its end is proven.

Author's diagram; the legal hold step follows Zubulake v. UBS Warburg LLC, 220 F.R.D. 212, 218 (S.D.N.Y. 2003); copies follow the book's running example.

A deletion row uses the same columns as any hop. **From**: the record and every copy the Ledger already lists. **To**: nothing, or an anonymized remainder. **Mechanism**: the delete job, the backup expiry, the key destruction, the index rebuild. **What was lost**: everything, on purpose; write COMPLETENESS and name the duty that required it. **Owner**: the person who signs that the record is gone. **Who can change it**: who can place or lift a hold. **Read by**: nobody, which the access logs should show from that date. For the 2004 workbook, the row is short and overdue: no owner, no rule, and a copy that an assistant searching the shared drive can still read.

When the reader is gone: format death

A record can also die with every byte intact, when no reader is left that understands its form. The Library of Congress assesses digital formats against seven sustainability factors: disclosure, adoption, transparency, self-documentation, external dependencies, impact of

patents, and technical protection mechanisms. Its definition of external dependencies is the one a data architect will recognize: the degree to which a format depends on particular hardware, operating system or software.

The thread carries several of these risks. The 1962 tape generations needed a drive, a label format and the COBOL copybook to be read. From 1974 the amounts sat in packed decimal, and without the copybook packed decimal is a run of half-bytes with no decimal point (Chapter Eleven). The IMS database of 1974 kept status codes whose meaning sat in a paper code book until 1999, when it became the DB2 table STATUS_CODES, which the 2021 warehouse never received. The 2004 workbook mixes dates typed as text with spreadsheet dates, so what a reader sees depends on the cell type and the program. A Parquet file in 2021 is open and self-describing for its types, and still says nothing about what status 7 means. Self-documentation is the factor most often missed: a format can be perfectly readable and still not tell the next reader what the values mean.

The 1890 schedules show the other way records die, by neglect and then decision. They survived the fire in part. They were then kept in storage for twelve years, listed among papers judged not worth keeping despite the protests, and destroyed under an authority that had no column for the principal person-level record of a national count. Chapter Three told how the Exchequer's retired tally sticks burned down Parliament in 1834; the 1890 schedules are the opposite failure. One retired record had no safe method of ending and was ended badly. The other had a stated end that nobody questioned. Both needed an owner and a row.

> **WHAT HASN'T CHANGED**
>
> The 1890 schedules were counted by machine and kept on paper; the counts were published, and the forms held the only person-level detail. Many warehouses do the same today: the aggregates are published and kept, and the detail that would let anyone check them sits in a store with the shortest retention and the least ownership.

The cost of a broken hop

The most expensive line in this chapter is not on any price list. It is rework: the cost of finding, explaining and repairing what a hop broke, paid years later by people who did not make the hop. The thread's clearest case is hop 5b. In 2011 one DBA spent one day loading a clerk's hand-corrected workbook back into DB2. It fixed this policy's issue date to 1898. It also overwrote the windowed dates on some 3,400 policies with hand-typed ones, and a few hundred of those were wrong (illustrative count). The Ledger row prices the load at "one day

of a DBA" and the consequence at "years of later reconciliation". The asymmetry is the lesson: a hop is cheap to run and expensive to undo, because undoing it means finding every row it touched and every reader who has read them since.

SOURCED FAILURE: TSB'S MIGRATION, APRIL 2018

Over the weekend of 20 to 22 April 2018, TSB moved its c.5 million customers from the Lloyds Banking Group platform to a new platform designed, built and operated by SABIS, a sister company in the Sabadell group (Slaughter and May independent review, paragraphs 1.1 and 2.1; the report is dated 31 October 2019 and was published by TSB's Board on 19 November 2019). The review's first finding is the one data teams should read twice: "the migration of customer records proceeded as planned" (paragraph 2.2), yet the internet and mobile channels were "unstable and almost unusable", customers waited about 90 minutes on the phone on the first day, and by 2 May the bank had 33,101 complaints, more than ten times usual levels (paragraph 2.3). Fraud attacks peaked on 15 May at about 70 times usual levels (paragraph 2.4). The review concluded that the platform was not ready to support the full customer base and SABIS was not ready to operate it (paragraph 2.7). The programme had involved over 70 third-party suppliers and over 1,400 people (paragraph 2.12).

Some of the damage was meaning. Over an undisclosed period, some customers who had closed or switched their accounts were given an incorrect reason code and recorded as deceased, and some received letters from organisations they paid by direct debit addressed to their personal representative; from 2 to 5 August 2018 a very small number of customers linked by false "also known as" relationships could see each other's accounts (paragraph 19.19(F) and (G)). In Ledger terms those are MEANING and IDENTITY losses at a hop whose rows arrived complete.

TSB is not a story about a bad data load; the review says the records moved. It is a story about a hop judged by its row counts while its readers (the channels, the people in contact centres, the reason codes) were not ready. The thread has the same shape at hop 2, where every card reached the tape and the remarks field did not. A hop that keeps every row and loses a meaning passes every completeness check, and its cost arrives later, at a reader, as rework.

ERA CARD

Reader. Every era's reader also decided the record's end: the clerk who kept the register, the operator who recycled tapes, the platform that expires backups, and now the retrieval index that keeps chunks until someone rebuilds it.

Security. Who can delete is as sensitive as who can change: a hold that anyone can lift, or a WORM store that no one can clear, are both control decisions to record in the row.

Cost. Storage is far cheaper per gigabyte than in 1956, though the rows are different products; moving, querying, keeping proof and repairing broken hops did not follow one curve, and people sit outside every price list.

AI. An agent reads whatever copies survive: a desk copy past its end, a backup restored for a test, an index built before an erasure. Deletion that does not reach the index is not deletion for the model.

Monday rule: pick one dataset with personal data, list every copy of it (backups, warehouse, logs, indexes, desk copies), and write the date each copy of a record deleted today would really be gone.

CHAPTER FOURTEEN

Proof: What Shows a Record Is True and Current



THE ERAS OF THE RECORD

On 16 December 2019 Mr Justice Fraser handed down his judgment on the "Horizon Issues" in the group litigation brought by about 555 claimants, most of them subpostmasters, against the Post Office. For nineteen years the Post Office had held branch managers liable for shortfalls that its Horizon system showed, and had told courts the system was robust. The judgment found that Fujitsu, which ran Horizon, could insert, edit or delete transaction data in branch accounts remotely, without the subpostmaster's knowledge or consent, and that an injected transaction would look as though the subpostmaster had done it. The accounts looked like the branch's own work, and they were believed over the people who said they were wrong. This chapter asks the sixth question, what proves a record is true and current, and separates it from the question the rest of the book has answered, where a record came from. The decision it changes: **lineage is evidence of the path, not of the truth; every dataset that drives a decision needs a second, independent proof.**

The oldest proof: two records that must agree

One of the oldest proofs was not a test. It was a second record. Chapter Three showed the split tally, where each party held half of a notched stick and the halves had to match, and double entry, where every transaction is recorded with equal debits and credits, so that the books must balance. Neither proves that a sale happened. Both create a comparison: the halves of one stick must match, the debits must equal the credits, and many errors by a single clerk show as a difference someone must explain. Balanced books can still hide an omission, a misposting or a fraud entered on both sides.

Reconciliation is that idea at scale: compare a record with an independent record of the same facts, and investigate every difference. A bank reconciles its ledger with the payment network's settlement file. The insurer reconciles the deposit total in its policy system with

the deposit liability in its general ledger. The word that matters is independent. A warehouse table compared with the job that loaded it proves only that the load ran.

Horizon had reconciliation, and the judgment describes it precisely. Comparing Horizon's transactions with those recorded by the Post Office's clients was "largely automated", but detected discrepancies were handled by "manual corrective fixes" and by Transaction Corrections sent to the subpostmaster (paragraph 985). The experts agreed that Horizon's automatic integrity mechanisms "have been shown to have failed in the past" (paragraph 988). A reconciliation that ends by asking one party to pay the difference is not a proof. It is a collection process.

Audit is the other old proof: a person independent of the record checks it against the world. The 1898 register was read by the auditor every year; the thread's hop 0 lists "the auditor" among its readers. An audit samples records and looks at the building, the cash, the signed form. It is slow and expensive, and with a second independent record it is one of the few proofs that can catch a record that was false when it was written.

Quality, named: the dimensions and the standard

Data quality has its own international standard series. ISO 8000-1:2022, *Data quality, Part 1: Overview*, published in April 2022 and confirmed in 2024, states the scope of the series and establishes "the principles of information and data quality". The working vocabulary most teams use is shorter. The UK Government Data Quality Framework (3 December 2020) recommends "six core data quality dimensions, as defined by DAMA UK", the UK chapter of the data management association DAMA. The table gives each with the framework's definition and the place the thread tests it.

Dimension	Definition (Government Data Quality Framework)	Where the thread tests it
Accuracy	"the degree to which data matches reality"	The 2026 answer "lapsed for non-payment" (hop 9): every field it used was accurate; the answer was not.
Completeness	"the degree to which records are present", with "all essential values in a record" populated	Hop 2 kept every record and dropped the remarks field, whose code 9 marked perpetual policies: a row count passes, an essential value is gone.
Consistency	"the degree to which values in a data set do not contradict other values representing the same entity"	Key 980417 in old reports and A-980417 after the 1999 merger (hop 4), with no mapping on the old reports.
Timeliness	"the degree to which the data is an accurate reflection of the period that they represent, and that the data and its values are up to date"	Endorsements ordered by processing time, not event time (hop 7).
Validity	"the degree to which the data is in the range and format expected"	Status 7 is a valid status code in the warehouse (hop 6).
Uniqueness	"the degree to which there is no duplication in records"	Two insurers' numbers collided in 1999, so keys gained prefixes (hop 4).

Two rows deserve a second look. Accuracy is the only dimension defined against reality, and it is the one no pipeline can measure by itself: it needs an audit, a customer, or a second record. Validity, the easiest dimension to automate, passed at the exact place the thread failed.

Tests, contracts and lineage at the column

Modern data teams turn dimensions into code. A **test** asserts something about a table after a load: no nulls in the key, values in an allowed set, row counts within a band of yesterday's. A **data contract**, as this book uses the term, is an agreement between the producer of a dataset and its consumers, checked by machine where it can be: the columns, their types, the

allowed values, the freshness, and who to call when it breaks. Contracts are often written at the **last hop**, where a dataset enters the warehouse or is handed to a consumer, because that is where a team controls the code.

Column-level lineage records, for each output column, the input columns it came from and how. The OpenLineage column lineage facet, for example, lists the input fields for each output column and classifies each relationship as DIRECT, when the output value is derived from the input (with subtypes IDENTITY, TRANSFORMATION and AGGREGATION), or INDIRECT, when the input only influences it (JOIN, FILTER, SORT, GROUP_BY, WINDOW, CONDITIONAL). It can also flag that a value was masked. With it, a team can answer "which source columns feed the status the agent read?" by query instead of by meeting.

Both are good tools, and they answer different questions. A test or contract says that this column, here, now, has the expected shape. Lineage says where its values came from and through which steps. Neither says that the values are true.

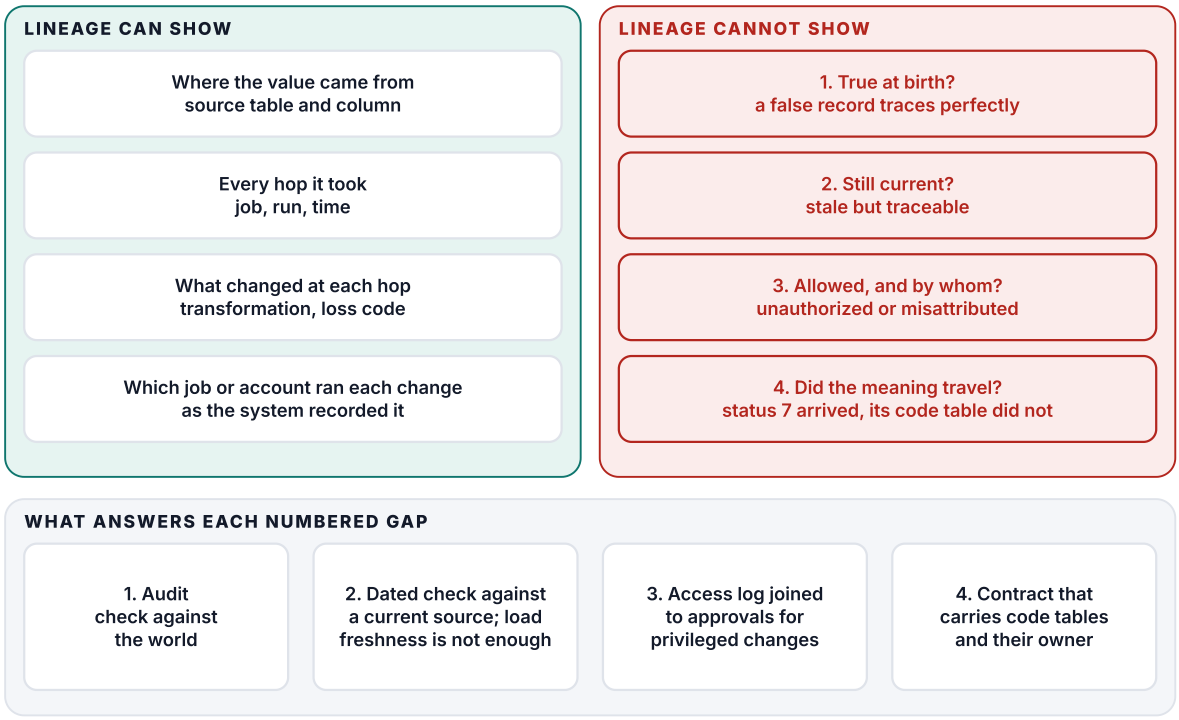
What lineage cannot prove

The book has built a Ledger whose rows show where a record went, what changed and who could change it. It is tempting to read a complete Ledger as proof. It is not, for four reasons. The first three are below; the fourth follows Figure 14.1, which maps all four.

- **False at birth.** A record that was wrong when written traces perfectly. If the 1898 surveyor had misread the roof as slate, every later hop would carry the error faithfully, and the lineage graph would be complete and green. Only an audit against the world, or a second independent record, catches it.
- **Stale but traceable.** A record can be correct when written and wrong now. The Ledger shows that status 7 came unchanged from the IMS database in 1974. It does not show whether the insured building on the river front still stands, which is the condition the perpetual policy depends on. Currency needs a freshness rule tied to the world, not to the last load.
- **Traceable but unauthorized, or misattributed.** Lineage can record which job or account changed a value, if the system captures it. It does not record whether that change was allowed, and it holds the identity the system reported, not the hand behind it. A change made through an over-wide role appears in the graph exactly like a change made through an approved release. Authority needs an access log joined to an approval record.

FIGURE 14.1

What lineage proves, what it does not, and what answers the gap



Lineage shows the path a value took and which job or account the system recorded for each change; it cannot show that the value was true when written, is still current, was changed with authority by the hand it names, or kept its meaning, so each numbered gap needs the control with the same number.

Author's diagram; column-level lineage as defined by the OpenLineage column lineage dataset facet (openlineage.io).

A fourth gap sits between lineage and contracts: meaning. Lineage at the column can show that the warehouse's status came, unchanged, from DB2's status, which came from IMS. That is DIRECT, IDENTITY all the way back. It is true and useless, because the value that travelled intact is a digit whose meaning stayed behind in DB2's STATUS_CODES table when the warehouse was loaded in 2021.

The strongest objection: lineage programs fail, contracts are cheaper

The best argument against this book deserves a full statement. Enterprise metadata repositories in the 1990s, and data catalogs since, have often become shelfware: expensive to fill, out of date within a year, read by nobody. A data contract at the last hop is cheaper. It is written once by the team that owns the warehouse, runs on every load, and fails loudly. Why trace history when you can check the present?

Concede the cost. A lineage program that tries to capture every hop of every dataset fails for the reasons its critics give, which is why Chapter Two scoped the Ledger to five datasets filled by hand, with a stated cost and a list of cases where it is not worth it. Concede the value of contracts too: every dataset that feeds an agent should have one.

Then write the contract the insurer's data team could have written in 2021 for the warehouse's policy table (illustrative), and run the thread through it. `policy_status`: INTEGER, NOT NULL, allowed values 1 to 9. `issue_date`: DATE, not in the future. `last_premium_date`: DATE or NULL. Freshness: loaded by 06:00. Row count: within 1 percent of the DB2 unload. On the morning of 12 February 2026, policy 1898/0417 passed every rule. Status 7 was in range. The issue date was 1898, valid since the 2011 correction load. The null premium date was allowed, because perpetual policies have none. The load was on time and complete.

The contract could not fail, because nothing in the warehouse was wrong. What was wrong was absent: the code table that says 7 means "perpetual, deposit held, no premium due" had not been carried at hop 6. It had lived on paper until the 1999 conversion at hop 4 keyed it into DB2 as STATUS_CODES, and the 2021 unload left it there. A last-hop contract checks what arrives. It cannot check for what never arrived, unless someone who knows the history writes that rule, and the Ledger is one place that history is written down. Hop 6's row records MEANING as the loss, and hop 4's row says where the code table lives. Those two cells would have told the team to ship the code table with the column, and the agent's tool to return it.

THE HOP · 6 · 2021 · DB2 FOR Z/OS TO CLOUD DATA WAREHOUSE TABLES (LOADED VIA PARQUET FILES)

Record	Policy 1898/0417 (perpetual fire policy), hop 6, 2021
From	DB2 for z/OS
To	Cloud data warehouse tables (loaded via Parquet files)
Mechanism and schedule	Nightly unload job, file transfer and load into warehouse tables
Format change; what was lost or assumed	MEANING EBCDIC and packed decimal converted to UTF-8 text and DECIMAL(18,2); status as an integer column. **The STATUS_CODES lookup table is not carried: in the warehouse 7 is a bare number**
Accountable owner	Head of data platform
Who can change it	Pipeline service account; three data engineers; the warehouse load role
Classification; retention or erasure duty	Commercial record; the policy domain also holds personal data of individual policyholders under US federal (GLBA) and state insurance privacy rules; retention duties
Read by (people, apps, models, agents)	Finance and actuarial dashboards; data scientists
Cost per month; basis	About 2,800 dollars a month to store, move and query this domain (illustrative)

> *A contract at the last hop checks what arrived. Only the history tells you what should have arrived and did not.*

So the answer is not lineage or contracts. It is a small Ledger that tells you which contracts to write, and contracts that enforce what the Ledger found. And for the questions neither can answer, a second, independent proof.

The record believed over people: Horizon

The Post Office case is the clearest public record of what happens when a system's data is treated as proof and nobody asks who could change it. The facts below come only from the High Court's Judgment (No 6) of 16 December 2019 and the first volume of the statutory Inquiry's report, cited by paragraph.

SOURCED FAILURE: BATES V POST OFFICE LTD (NO 6), [2019] EWHC 3408 (QB)

The Post Office introduced Horizon across its branches in 2000 and moved to an online version, HNG-X, in 2010; it maintained that the systems were robust and extremely unlikely to cause shortfalls (paragraph 1). The court found Legacy Horizon "was not remotely robust", HNG-X's robustness "questionable", and a material risk that shortfalls in branch accounts were caused by Horizon from 2000 to 2017 (paragraphs 975 to 978). It described the Post Office's stance as "the 21st century equivalent of maintaining that the earth is flat" (paragraph 929).

Who could read it. Both the Post Office and Fujitsu could read branch transaction data remotely; Fujitsu could access all of it (paragraph 991). The only record of which keys were pressed in a branch was in audit data that the Post Office had to request from Fujitsu, which charged for it (paragraphs 994 to 995). Subpostmasters could not, from the reports available to them, identify shortfalls or their causes themselves (paragraph 1000), and there was no button on Horizon to record a dispute (paragraph 1018).

Who could change it. Fujitsu could insert, inject, edit or delete transaction data, implement fixes that affected it, and rebuild branch data, without the subpostmaster's knowledge or consent: the court answered "yes" to all nine permutations, through support staff with the APPSUP role, privileged users and database administrators (paragraph 1003). Fujitsu's witnesses accepted that an injected transaction would look as though the subpostmaster had done it (paragraph 1004). The Post Office could inject transactions only through its Global User role, by someone physically present in the branch (paragraph 1005). A Balancing Transaction Tool also let Fujitsu insert transactions into branch accounts (paragraph 1006).

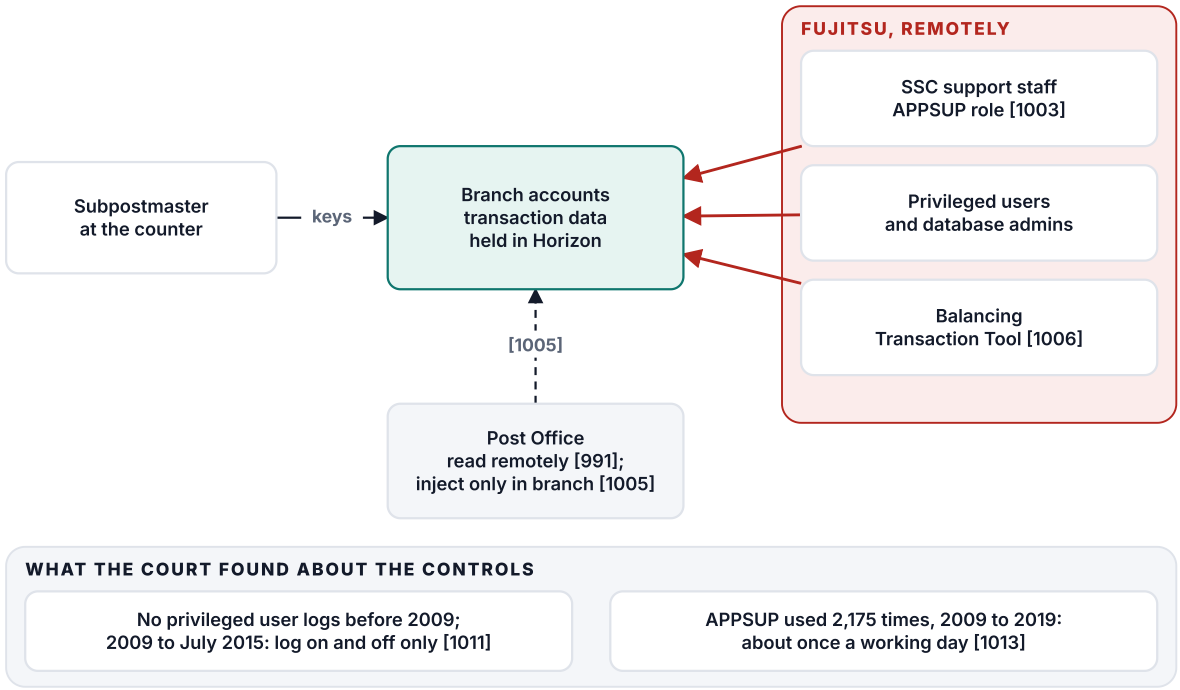
What the logs showed. Permission controls existed but the roles were very wide and, in the court's judgment, not controlled. Privileged user logs did not exist before 2009, and from 2009 to July 2015 recorded only logging on and off (paragraphs 1008 and 1011). The APPSUP role was used 2,175 times between 2009 and 2019, which the Post Office's own expert equated to about one person using it each working day (paragraph 1013). The experts agreed that anyone with APPSUP could "pretty much do whatever they wanted", and the court found instances when such use did affect branch accounts (paragraph 1016).

What it cost people. The Inquiry's first volume (8 July 2025) records that prosecutions relied on Horizon data, presented as wholly reliable, and that "many hundreds of people have been convicted, wrongly" and "many thousands" held responsible for losses that were illusory (paragraphs 1.7 to 1.9).

Figure 14.2 draws the court's answer to the question this book asks of every record: who could change it.

FIGURE 14.2

Horizon: who could change a branch record, as found by the High Court in 2019



The subpostmaster keyed the branch's transactions, but Fujitsu roles could insert, edit or delete branch transaction data remotely without the subpostmaster's knowledge or consent, under controls the court found inadequate; red arrows are Fujitsu's remote routes, the dashed arrow the Post Office's route in the branch, and bracketed numbers are paragraphs of the judgment.

Source: Bates and Others v Post Office Ltd (No 6) "Horizon Issues" [2019] EWHC 3408 (QB), 16 December 2019, paragraphs 991, 1003 to 1006, 1011 and 1013.

Read Horizon against Figure 14.1. The branch accounts looked like the branch's own work, and that was the problem: an injected transaction "would look as though the SPM had done it" (paragraph 1004). The record did not show a known path with one approval missing. It could attribute a change to the wrong hand, and the logs that could have told the two apart did not exist before 2009 and then recorded only log-ons (paragraph 1011); the court found the powerful roles "effectively unaudited" (paragraph 1008). Lineage is only as true as the identity the system records. The record was then believed over the people who kept the branch, because it was the record. The Ledger's seventh column, who can change it, is in this book because of cases like this one. For Horizon the true answer to that column was a support team in another company with a role that could do almost anything, used about once a working day, and logs that did not record what it did.

The insurer is a long way from a criminal court, and the thread's error was fixed the next morning. The pattern is the same, and smaller. On 12 February 2026 the service agent read a record and told a customer something false with the confidence of a system. The claims handler who corrected it read row 7 of STATUS_CODES in DB2 and the 1898 register scan. The next agent, reading the case note at hop 9, has neither the handler's knowledge nor the code table, unless the correction carries its proof with it.

ERA CARD

Reader. The auditor and the reconciling clerk were the first readers whose job was proof; the model is a reader that states conclusions without one, so the proof has to travel with the data it reads.

Security. Who could change a record, remotely and without a trace, is a proof question as much as an access question: Horizon's wide roles and empty logs left no way to show a branch's figures were the branch's own.

Cost. Contracts are cheap to run; audits and independent reconciliations are not, so spend them on the few datasets where a wrong record harms a person or moves money.

AI. An agent's answer is a new record (hop 9); attach to it the rows it read, their code tables and their freshness, so the next reader can check the answer instead of believing it.

Monday rule: for each dataset an agent reads, name one proof that does not come from the pipeline itself (an audit sample, a reconciliation with an outside count, or an access log joined to approvals) and the date it last ran.

CHAPTER FIFTEEN

Filling the Ledger in 90 Days



On Monday 16 February 2026 the insurer's chief data officer read two case notes on the same policy. The first, written by the service agent on 12 February, told the owner of a brick warehouse that policy 1898/0417 had lapsed for non-payment. The second, written by a claims handler the next morning, said the policy was perpetual and in force. Both notes were records now. Both could be read by the next agent. The chief data officer had a choice: fix one answer, or find out how many other answers rested on a code that never travelled. This chapter is the second choice, made small enough to finish: **five datasets, one Ledger each, in 90 days**, with a stated cost and a named owner for every row.

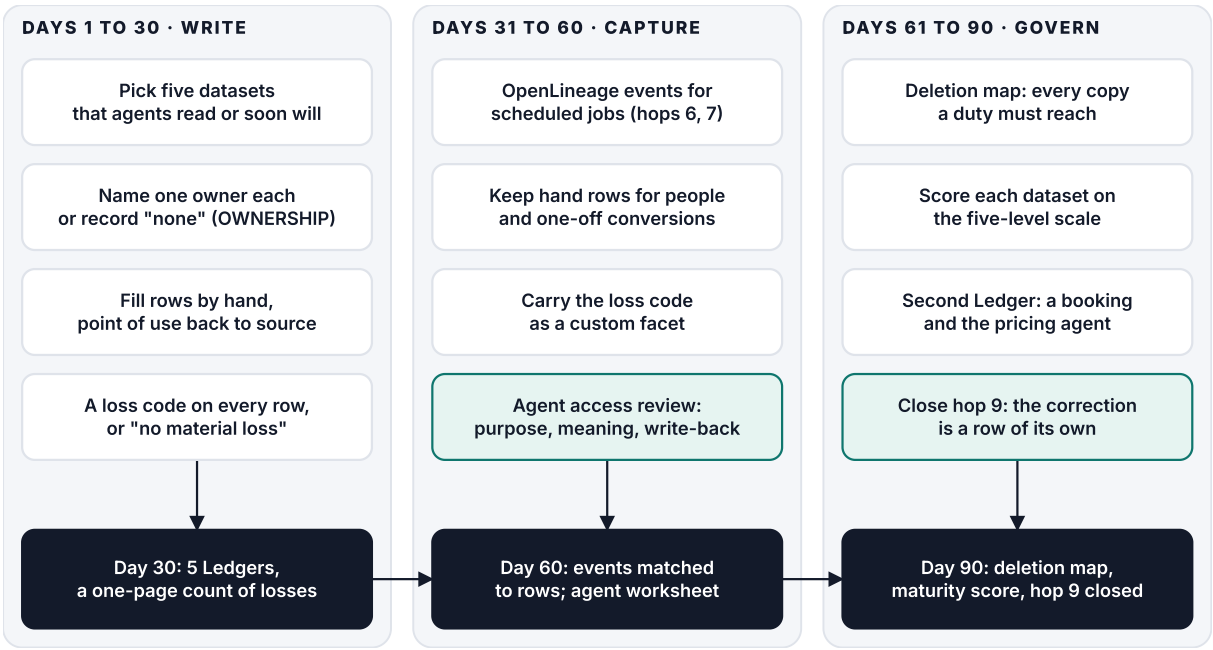
It was a hard week for the insurer's data. The same Monday, an investigation began into how the agent had found its answer, the subject of *The Model as Reader*; on Thursday 19 February the board saw two different loss ratios for the same quarter, the subject of *From Warehouse to Lakehouse*.

The plan in one page

The plan has three blocks of thirty days. Each block ends with something a board member could open and read. Days 1 to 30 produce Ledger rows written by hand. Days 31 to 60 produce automated capture where it fits, and an agent access review. Days 61 to 90 produce a deletion map, a maturity score, a second Ledger for a record only seconds old, and the closing of the thread's last hop. Figure 15.1 shows the plan.

FIGURE 15.1

The 90-day plan: rows by hand first, automation second, deletion and agents last



Each block of thirty days ends with something an executive can open and read; the work is scoped to the Ledger, not to platform design.

Author's diagram based on the plan in this chapter.

The plan is scoped to the Ledger: where each record goes, what it loses, who owns it, who reads it. How to design the stages a record passes through on a data platform (raw, cleaned, trusted) is the subject of the Medallion book *From Bronze to Gold*, and this chapter does not repeat it.

The plan also has a cost, and Chapter Two promised to state it. The illustrative budget for a company of the insurer's size is one lead (half time for 90 days), one engineer from the data platform team (for the row sessions in days 11 to 25, then a quarter of their time from day 31), and two hours a month from each dataset owner. Count it in the same people line as Chapter Thirteen. If you cannot find that time, the "not worth it when" list in Chapter Two applies to you, and you should stop at day 30 with the rows you have.

Days 1 to 30: five datasets, five owners, rows by hand

Pick five datasets. Not the five largest. Pick the five whose wrong answer would cost the most, and where an assistant or an agent already reads them or soon will. A useful test is the six questions from the Reader's Brief: if nobody in the room can answer "who can change it" or "who reads it now" for a dataset, it is a candidate. For the insurer the five were the policy

master table, the policy events stream, the customer case notes, the scanned document index and the claims reserve table. The hotel and retail group picked bookings, rates, loyalty profiles, the guest consent log and point-of-sale baskets. The health system picked lab results, because of the time zones Chapter Eleven describes.

Name an owner for each. An owner is a person with a job title, who can say yes or no to a change and who answers when the dataset is wrong. A team name is not an owner. A platform is not an owner. The insurer found two of its five had no owner at all; one of them was the case notes, which every service agent wrote and nobody had been named to own until this review named the head of customer service. Write the owner's name in the Ledger before any other column. If no one will accept it, record "none recorded" and the loss code OWNERSHIP. That is a finding, not a gap in the work.

Fill rows by hand, one hop at a time. Start at the point of use and walk back, as the Prologue did: the agent's answer, the tool it called, the table behind the tool, the job that loaded the table, the system that fed the job. One row per hop. Use the blank Ledger in Appendix A. Each row takes about 30 minutes once the people are in the room, which is why Chapter Two's exercise was timed. The loss code goes in column 5 with a sentence that says what was lost or assumed. Use the eight codes exactly; if none fits, say so in words. A row with no loss is a valid row.

Day	Output	Who	Done when
1 to 5	Five datasets chosen with a one-line reason each	CDO and CTO	Reasons name a decision or a reader, not a size
6 to 10	An owner named for each, or "none recorded"	Lead	Each owner has seen the row with their name
11 to 25	Rows by hand, from the point of use back to the source	Lead, owners, one engineer per session	Every row has a loss code or the words "no material loss"
26 to 30	A one-page summary: hops, losses by code, owners missing	Lead	Read by the executive who sponsors the work

The summary at day 30 is a count. The insurer's read: 31 rows across five datasets, 9 with MEANING, 6 with TIME, 5 with OWNERSHIP, 3 with RIGHTS, and two datasets where the trail stopped at a file nobody could explain (illustrative counts). The status code table that did not travel at hop 6 appeared twice, once for policies and once for claims.

Days 31 to 60: automate where it fits, and review the agents

Automate capture where OpenLineage fits. OpenLineage describes itself as "an open framework for data lineage collection and analysis", and it is a graduated project of the LF AI & Data Foundation. Its model has three core entities: a **job** (a process), a **run** (one execution of it) and a **dataset** (what it reads and writes). A run event carries an event type (START, RUNNING, COMPLETE, ABORT, FAIL or OTHER), an event time, the run, the job and the producer that emitted it, and can list the input and output datasets. Extra detail travels as **facets**; the column-level lineage facet records which input columns feed each output column and whether a value passed through unchanged (IDENTITY), was computed (TRANSFORMATION) or was combined across rows (AGGREGATION). The project lists integrations for Apache Airflow, Apache Spark, Apache Flink, dbt, Trino and others.

That fit is good for scheduled jobs on a modern platform: hop 6 (the nightly unload into the warehouse) and hop 7 (the stream into the warehouse) can emit events every run. The fit is poor for three kinds of hop. The first is a person: nobody emits a run event when a clerk types corrected dates into a workbook (hop 5). The second is the old system: the 1974 conversion at hop 3 ran once and left no events. The third is meaning: OpenLineage has no field for "what was lost". Appendix A maps every Ledger column to the specification and shows where a custom facet is needed. So the rule for days 31 to 60 is simple: **automate the rows that run on a schedule, keep the hand rows for the rest, and carry the loss code as a custom facet** so it is not dropped by the very tool meant to record it.

Review what the agents can read. Chapter Ten separated the AI data paths; this is where you check them, one agent at a time. The worksheet below has one row per tool the agent can call or index it can search. Fill it with the agent's owner and the owner of each dataset in the same room.

Question	The insurer's service agent (row for its SQL tool)	What a good answer looks like
Which agent, which owner?	Service agent; head of customer service	A named person who can switch the agent off
Which tool or index, and which datasets behind it?	Read-only SQL tool over the policy tables	Each dataset has its own Ledger row
Purpose recorded for this use?	No (loss code RIGHTS at hop 8)	A stated purpose and legal basis that travel with the tool
Does the meaning travel? Code tables, units, time basis	No: status 7 arrives as a bare number	Every coded column joined to its code table, or refused
Can the agent write?	Yes, indirectly: its answer is saved to the CRM	Every write-back is a Ledger row with an owner
Which rows can it return that it should not?	Other customers' policies on the same street	Rows filtered to the person the agent is serving
Is each call logged with the rows returned?	Calls yes, rows no	Calls and row keys logged, kept for the complaint period
Last review, next review	None; 30 days	A date in the calendar of the owner, not the platform team

Two answers in that worksheet matter more than the rest. "Does the meaning travel?" is the planted error of this book in one question. "Can the agent write?" is where a reader becomes a writer, and the next section shows why that changes the Ledger.

Days 61 to 90: deletion, maturity, and a record seconds old

Map the deletion duties. A Ledger shows every place a record went, so it is also the list of every place a deletion has to reach. For each dataset, take the rows and add three facts: the duty (a retention period, an erasure right, a legal hold), whether the copy at that hop honours it, and how you would prove it. The hotel group's EU hotels are bound by the General Data Protection Regulation (GDPR). Its Article 17 gives a guest "the right to obtain from the controller the erasure of personal data" on the grounds it lists, and Article 19 requires the controller to "communicate any rectification or erasure of personal data" to each recipient it was disclosed to, unless that "proves impossible or involves disproportionate effort". Without a list of where the data went, such as a Ledger, a company may not be able to name them all. Chapter Thirteen gave the harder cases: backups, immutable logs and the vector index, where a deleted record can still be found by its meaning.

Score the maturity. The scale below is the author's, not a standard. It exists so that a team can say where it stands in one word and agree what the next level costs. Each level includes the one below it.

Level	Name	What is true at this level	Evidence you could show
1	Unknown	Lineage lives in people's memory and in job code	None; the answer to "where did this come from" is a name
2	Written	The chosen datasets have hand-filled rows, owners and loss codes	The Ledger for five datasets, dated
3	Captured	Scheduled hops emit events every run; hand rows cover the rest; loss codes travel as a custom facet	Run events for the last 30 days, matched to Ledger rows
4	Governed	A change to a hop needs its owner's approval; agent access and deletion duties are reviewed against the rows	Change tickets and review minutes that cite row numbers
5	Evidential	Any answer an agent gives can be traced to the rows it used, and every write-back and correction is a row	For a sample answer, the chain of rows back to the source, produced on request

The insurer scored itself at level 1 on day 1 and level 2 on day 30. On day 90 it was at level 3 for two datasets and level 2 for three. Level 5 for one dataset is a fair two-year goal; level 5 for every dataset is not a goal, because Chapter Two's "not worth it when" list still applies.

Fill the second Ledger. The thread of this book is 128 years long, and a reader could conclude that the Ledger is for old records. It is not. The hotel and retail group's booking below, in short form (four of the ten columns), is seconds old when it reaches the pricing agent. The guest books a room on the website at 21:14; within the minute, the booking has passed through a rented booking engine, a payment provider, the CRM, the event stream, the warehouse and the pricing agent, and the agent has changed a rate.

A booking made at 21:14: seven hops in under a minute

Hop	From	To	What was lost or assumed
1	Guest on the hotel website	Booking engine (rented)	RIGHTS **The marketing consent box lives only in the web session log**
2	Booking engine	Payment provider	**Nothing material; card data tokenized**
3	Booking engine	CRM	IDENTITY **Guest matched to an existing profile by email; a family shares one address**
4	Booking engine	Event stream	TIME **Hotel local time vs UTC on the check-in date**
5	Event stream	Warehouse	MEANING **Rate code BAR-FLEX not mapped to its cancellation rule**
6	Warehouse	Pricing agent	MEANING **Agent reads cancellations as lost demand**
7	Pricing agent	Rate in the booking engine (write-back)	OWNERSHIP **A price change made by an agent: who owns it?**

Read hop 7 twice. The pricing agent is not only a reader. It wrote a new rate into the booking engine, and that rate will be shown to the next guest. It read cancellations of a flexible rate as lost demand (hop 6): guests it took to have been turned away, a sign that demand exceeded the rooms on sale. The rate code had arrived without its cancellation rule (hop 5), so the agent could not tell a free cancellation from a lost sale, and it raised the price on the strength of a meaning that never travelled. That is the same kind of loss as status 7, in under a minute instead of a century. The row has the loss code OWNERSHIP because the rate it wrote was a record with no accountable owner: nobody had decided who owns a price an agent sets. The fix was a sentence in the Ledger: the revenue manager owns every rate, whoever writes it, and the agent's write-back is a hop with its own row, its own log and a limit on how far it can move a price without a person.

Closing hop 9: the correction gets its own lineage

The thread ends where the Prologue began. On 12 February 2026 the service agent told the building's owner that policy 1898/0417 had lapsed for non-payment. Its SQL tool returned status 7 and an empty premium date, which it read as no premium since 1898; with no code table, it supplied a plausible meaning. The conversation and the answer were saved to the customer's case. That save is hop 9: a wrong fact became a record, with its own lineage, readable by the next agent.

THE HOP · 9 · 2026 · SERVICE AGENT ANSWER TO CRM CASE NOTE (A NEW RECORD)

Record	Policy 1898/0417 (perpetual fire policy), hop 9, 2026
From	Service agent answer
To	CRM case note (a new record)
Mechanism and schedule	The conversation and the agent's answer are saved to the customer's case
Format change; what was lost or assumed	MEANING Free text with a timestamp and the agent's tool calls. **A wrong fact (lapsed for non-payment) now has its own lineage and can be read by the next agent**
Accountable owner	Head of customer service
Who can change it	Agents (human and AI); CRM administrators
Classification; retention or erasure duty	Personal data; complaint-handling retention
Read by (people, apps, models, agents)	Claims handlers, the next agent, the complaints team
Cost per month; basis	CRM licence share (illustrative)

A claims handler corrected the answer the next morning, Friday 13 February, from row 7 of the STATUS_CODES table in DB2 and the scan of register line 417. The insurer's first instinct was to delete the wrong note. It did not, for two reasons the Ledger made plain. The case note is kept under the insurer's complaint-handling retention policy (illustrative), so the record of what the customer was told had to stay. And deleting it would remove the only record that ties the answer to the tool calls behind it, since the call log kept calls but not rows. So the correction became a record of its own, written as one more row in the Ledger:

THE HOP · 9B · 2026 · CLAIMS HANDLER, FROM THE STATUS_CODES TABLE IN DB2 AND THE 1898 REGISTER PAGE SCAN TO CRM CASE NOTE THAT SUPERSEDES THE AGENT'S NOTE (THE AGENT'S NOTE IS FLAGGED, NOT DELETED)

Record	Policy 1898/0417 (perpetual fire policy), hop 9b, 2026
From	Claims handler, from the STATUS_CODES table in DB2 and the 1898 register page scan
To	CRM case note that supersedes the agent's note (the agent's note is flagged, not deleted)
Mechanism and schedule	Manual correction on Friday 13 February 2026, linked to the wrong note by its identifier
Format change; what was lost or assumed	Free text citing its evidence: register line 417 and STATUS_CODES row 7. **No material loss: the correction carries its evidence**
Accountable owner	Head of customer service
Who can change it	Claims handlers; CRM administrators
Classification; retention or erasure duty	Personal data; kept under the insurer's complaint-handling retention policy, so the wrong note is flagged, not deleted
Read by (people, apps, models, agents)	Claims handlers, the complaints team, and every later agent, which now reads the flag before the note
Cost per month; basis	Ten minutes of a claims handler (illustrative)

The correction is not the end of the work, because the note was only the last hop. The rows behind it got their own fixes in the same 90 days. The status code table now travels with the status column into the warehouse, closing the MEANING loss at hop 6 for this column. The scanned register page was re-indexed with the word "perpetual" as a searchable field, not only as text in a low-ranking chunk (hop 8). The tool's metadata now marks which columns are coded, and the agent's SQL tool refuses a coded column without its code table. Each of those fixes is a change to a row, approved by the row's owner, which is the kind of control level 4 asks for. GDPR's Article 16 gives a person the right to obtain rectification of inaccurate personal data "without undue delay"; the insurer is not bound by the GDPR, but the Ledger gives it the same ability: it can say who was told the wrong fact, and tell them.

> *Do not silently delete the wrong answer. Unless a duty requires erasure, flag it, supersede it, link it, and record the correction as a hop of its own.*

ERA CARD

Reader. Every reader in this book at once: the person at the desk with a workbook, the batch job, the query, the stream, and the agent that now reads and writes.

Security. The agent access worksheet is the control: purpose, meaning, write-back and logged rows, reviewed by a named owner on a date.

Cost. Half a lead, part of an engineer's time and two hours a month per owner for 90 days (illustrative); cheaper than one wrong answer repeated by every agent.

AI. An agent that writes back is a hop: give its writes a row, an owner and a limit before it ships.

Monday rule: pick the five datasets your agents already read, write one owner's name next to each, and book the first 30-minute row for this week.

CLOSING

Epilogue: The Record Outlives the System



THE ERAS OF THE RECORD

On 14 March 1898 a clerk at a mutual fire insurer wrote line 417 in the policy register: a perpetual policy on a brick warehouse on the river front, issued to a wool merchant, paid once by a deposit of 412.50 dollars. The clerk could not have known the readers who would come after: a keypunch operator, a tape drive, a hierarchical database, a relational one, a clerk with a spreadsheet, a cloud warehouse, a stream, and in 2026 a service agent that read a bare number and answered a question nobody in 1898 had imagined. The tabulators, the tape and the IMS database that carried it have been retired, and the systems that carry it now will be retired in turn. The line is still in the bound volume. The record has outlived every system retired so far, and that is the lesson this book ends on.

Eight readers, one promise

Chapter One defined a record as a promise from the writer to a future reader. Policy 1898/0417 kept that promise for 128 years, but not because any system kept it. The card dropped the slash and two digits of the year. The tape dropped the remarks column. The hierarchical database kept the status code and left its meaning on paper; the relational one gave that meaning a table in 1999. The Y2K window made the policy a hundred years younger, and a hand-typed workbook made it old again, for this policy and not for every other. The warehouse kept the number 7 and lost what it meant. Each of those hops was reasonable on the day it was made. Each was made by competent people who knew what their own reader needed and did not know what the next one would.

That is the pattern the Ledger is built to see. Most failures of meaning in this book happened at a hop, where one reader's knowledge did not travel to the next. Some records were wrong at birth, as Chapter Fourteen showed, and some systems failed on their own. But the hop is where a company can look, one row at a time, and the Ledger is the place to write down what it finds.

What the AI reader changes

No machine reader in this thread asked the writer what a value meant. The batch program could not, and the model cannot. A COBOL job that met status 7 in 1974 did exactly what its code said: if the code had no branch for 7, it did nothing, or it stopped. The model is different in one way that matters. It **interprets instead of executing**. Where meaning is missing it supplies a plausible one, and unless it is built to stop, it does not. On 12 February 2026 the service agent saw no premium since 1898 and a status it could not decode, and it told a building owner the policy had lapsed. It was fluent, polite and wrong.

OUR VIEW, AS OF NOVEMBER 2026

What changes. Three things. First, the reader fills gaps: a missing code table used to produce an error, a blank or a silent default, and now it can produce a fluent answer. Second, the reader writes: an agent's answer is saved as a case note, a rate, a ticket or a summary, and that write is a new record with its own lineage, as hop 9 and the pricing agent showed. Third, the hops are faster and more numerous: a hotel booking passes through seven of them in under a minute, and an agent can read a dataset no one on the data team knew it could reach. Together these make lineage the **evidence** that the record behind an answer is the record that was written, and who changed it on the way.

What does not change. The six questions are the same ones a company secretary could have asked of the 1898 register: where is it written, what does it mean, who can change it, what does it cost, who reads it now, and what proves it is true and current. Meaning still has to travel with the value, as a code table, a unit, a time basis. A copy still needs an owner. Deleting a record still means finding every place it went. In our experience, people are still the largest line in the cost of keeping data right. And lineage still cannot prove a record was true when it was written; it can only show where it came from.

What I expect. Teams that already know their hops will adopt agents faster and with fewer public mistakes, because they can answer "where did that come from" in minutes. Teams that do not will learn their hops from their customers, one wrong answer at a time. I do not expect a model to fix missing meaning by itself; I expect it to make missing meaning visible sooner, and more expensive.

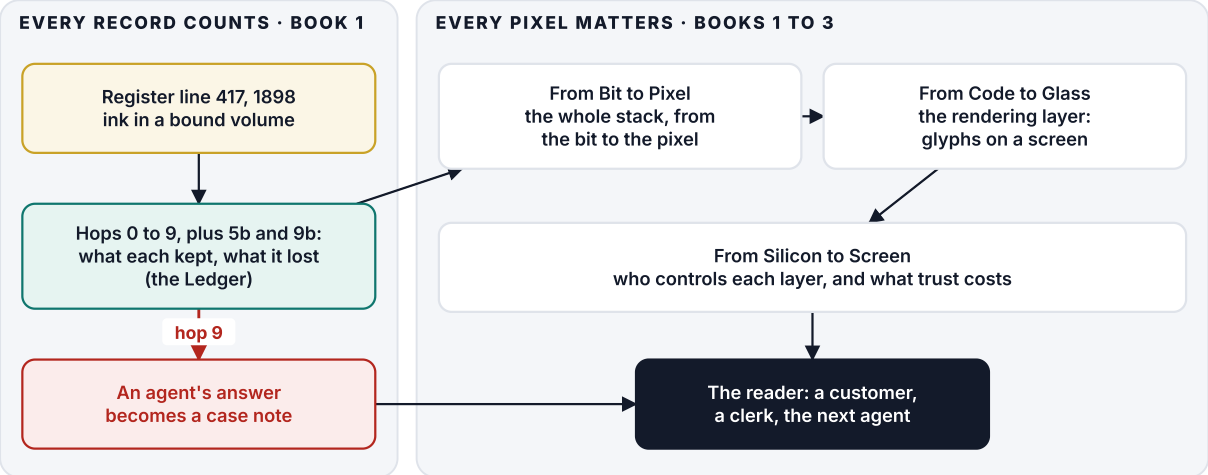
The record from bit to pixel

This book followed one record across eras. The sister series *Every Pixel Matters* follows a different axis: what happens to any record between the bits on a disk and the light on a screen. Figure EP.1 puts the two together for line 417. Book 1 of that series, *From Bit to Pixel*, covers the whole stack, from the bit to the pixel; Book 2, *From Code to Glass*, covers

the rendering layer that draws the glyphs a reader sees; Book 3, *From Silicon to Screen*, covers who controls each layer and what trust costs. A record can be correct in the warehouse and wrong on the screen, or correct on both and changed by someone with access to a layer nobody reviewed.

FIGURE EP.1

Line 417 from bit to pixel: this series follows the meaning, the sister series follows the layers



The same record crosses the Ledger's hops (this book) and the layers from the bit to the pixel (Every Pixel Matters, Books 1 to 3).

Author's diagram.

The two series answer different questions about the same record. This one asks what the record means and who changed it on the way. The other asks how the record is stored, drawn and controlled. A CTO needs both, because an agent's answer passes through every layer on its way to a customer.

Back to line 417

When the claims handler corrected the case note on 13 February 2026, the evidence was not in the warehouse, the stream or the vector index. It was the scan of the register page, which had ranked below newer documents, and the STATUS_CODES table in DB2, keyed from the paper code book in 1999 and never carried past it. The oldest record in the chain, the ink, still held the meaning, and so did the 1999 table that had copied the code book from paper. The ink had outlived every reader that was supposed to replace it.

That is not an argument for ink. It is an argument for treating every important record as if it will outlive the system that holds it, because the important ones usually do. Write down where it goes. Carry what it means. Name who owns it. Record what each hop lost. The next reader is already on its way, and it will not ask.

> *Systems are retired. Records are inherited. Write each hop down for the reader you will never meet.*

Monday rule: find the oldest record your agents can reach, and check that its code tables, units and owner reached the agent with it.

APPENDICES

APPENDIX A

The Data Lineage Ledger: Template, Loss Codes and Mappings

The Ledger as a working tool: the blank template, a note on each column, the worked example and a map to OpenLineage and W3C PROV. The blank Ledger is a free download, with the loss codes and this worked example: XLSX workbook and PDF (blog.jcardena.com/ebooks/from-tally-to-token-ledger.xlsx and .pdf).

The blank template and how to fill it

The Data Lineage Ledger: one row per hop

Record or dataset, hop	From	To	Mechanism and schedule	Format change; what was lost or assumed	Accountable owner	Who can change it	Classification; retention or erasure duty	Read by (people, apps, models, agents)	Cost per month; basis

Loss codes

MEANING	a code or definition did not travel with the value
CONTEXT	notes or surrounding knowledge stayed behind
PRECISION	type or rounding changed the value
TIME	century, time zone, or event versus processing time changed
IDENTITY	the key was renumbered, merged or split
OWNERSHIP	a copy has no accountable owner
RIGHTS	purpose, consent or legal basis did not travel
COMPLETENESS	rows or fields were dropped

Column	Write
1. Record or dataset, hop	A name a business user knows, the grain (record, field, event, dataset, derived artifact), the hop number
2 and 3. From, To	The system, store or person; include copies, indexes and agents
4. Mechanism and schedule	The job, tool or hand that moves it, and how often
5. What was lost or assumed	Loss codes, then one sentence; "no material loss" is valid
6. Accountable owner	A person by job title; "none recorded" is a finding (OWNERSHIP)
7. Who can change it	People, roles, service accounts and agents with write access
8. Classification; duty	Data class; for personal data, the purpose and legal basis (RIGHTS marks a hop where they did not travel); the retention, erasure or hold duty for this copy
9. Read by	People, applications, models and agents that read this copy now
10. Cost per month	Store, move, query, keep, delete, people; say if estimated

Do not merge two hops because one tool runs both. When an agent writes, the write is a hop with its own row.

The worked example: One Record, Since 1898

The insurer and its numbers are illustrative; the practice at each hop is sourced in its chapter.

THE LEDGER, PART 1 OF 2

One Record, Since 1898: the full Ledger: movement and meaning

Hop, year	From	To	Mechanism and schedule	Format change; what was lost or assumed
0 1898	Applicant and the insurer's surveyor	Policy register, line 417 (ink, bound volume)	Clerk writes the line by hand from the application and the survey report	CONTEXT Ink; fields in ruled columns: number, date, insured, property, sum insured, deposit, remarks. **The surveyor's notes on the building (brick, slate roof, distance to the next warehouse) stay in a separate survey book**
1 1931	Policy register	IBM 80-column card, one card per policy	Keypunch operator transcribes the register; cards sorted and tabulated monthly	IDENTITY TIME Holes in 80 columns; numbers only in most fields; a fixed card layout is the schema. **Key 1898/0417 cannot be punched with a slash: renumbered 980417; the year shrinks to two digits to save columns**
2 1962	Card master deck	Tape master file on the insurer's first computer	One conversion run reads every card; COBOL batch programs then update the tape each night	COMPLETENESS Fixed-length records described by a COBOL copybook; six-bit BCD character code on tape; numbers as decimal digits. **The remarks field (card columns 61 to 80) is dropped as 'unused'; it held remark code 9 in column 61, which marked perpetual policies**
3 1974	Tape master file	IMS hierarchical database on MVS	Conversion programs load policy, building and deposit segments into an IMS hierarchy	MEANING Policy root segment with child segments; status as a one-digit code. **Status codes are kept as digits; the code book that says what 7 means stays on paper in the policy department**
4 1999	IMS database	DB2 for OS/390 on the mainframe (new policy system after the merger)	Merger conversion: both insurers' policies loaded into one DB2 schema; Y2K date windowing applied in the same project	TIME IDENTITY Relational tables; issue date as DATE; key widened with a prefix; the paper status code book is keyed into a DB2 lookup table (STATUS_CODES) for the first time. **98 is read as 1998 by the date window; the merged insurer's numbers collide, so keys get prefixes (A-980417)**
5 2004	DB2 policy tables	Excel workbook on a shared drive (branch)	A clerk exports policies with odd dates to check them, and types corrected years by hand	OWNERSHIP Spreadsheet; dates typed as text and as Excel dates. **A copy with no accountable owner; the workbook is copied and emailed**
5b 2011	Excel workbook	DB2 policy tables (correction load)	The workbook is loaded back as a one-off correction	TIME CSV export of the workbook, loaded by a utility. **Hand-typed dates overwrite the windowed ones for some 3,400 policies (illustrative count): this policy is fixed to 1898, a few hundred others are made wrong**

Hop, year	From	To	Mechanism and schedule	Format change; what was lost or assumed
6 2021	DB2 for z/OS	Cloud data warehouse tables (loaded via Parquet files)	Nightly unload job, file transfer and load into warehouse tables	MEANING EBCDIC and packed decimal converted to UTF-8 text and DECIMAL(18,2); status as an integer column. **The STATUS_CODES lookup table is not carried: in the warehouse 7 is a bare number**
7 2023	Policy system events	Event stream (change data capture) into the warehouse	CDC reads the DB2 log and publishes policy change events within seconds; the topic is not keyed by policy	TIME JSON events with an event time and a processing time. **Endorsements are ordered by processing time, not event time: when an original change arrives late, the correction that follows it can appear first**
8 2025	Warehouse tables and the scanned register page	Service agent: a SQL tool for policy rows, a retrieval index for documents	The agent calls a read-only SQL tool; scanned documents are embedded into a vector index for retrieval	RIGHTS MEANING Rows returned as text to the model; register page scan as OCR text chunks and vectors. **The purpose of processing is not attached to the tool; the scanned page that says perpetual exists but ranks below newer documents**
9 2026	Service agent answer	CRM case note (a new record)	The conversation and the agent's answer are saved to the customer's case	MEANING Free text with a timestamp and the agent's tool calls. **A wrong fact (lapsed for non-payment) now has its own lineage and can be read by the next agent**
9b 2026	Claims handler, from the STATUS_CODES table in DB2 and the 1898 register page scan	CRM case note that supersedes the agent's note (the agent's note is flagged, not deleted)	Manual correction on Friday 13 February 2026, linked to the wrong note by its identifier	Free text citing its evidence: register line 417 and STATUS_CODES row 7. **No material loss: the correction carries its evidence**

THE LEDGER, PART 2 OF 2

One Record, Since 1898: the full Ledger: control, readers and cost

Hop	Accountable owner	Who can change it	Classification; retention or erasure	Read by	Cost per month; basis
0	Company secretary	Register clerk; corrections initialled by the secretary	Commercial record; kept for the life of the policy	Clerks, the auditor, the board	Clerk time, about 10 minutes a month (2 hours a year) for the register line and the annual audit (illustrative)
1	Head of the tabulating department	Keypunch operators; the card layout set by the tabulating supervisor	Commercial record; master deck kept in steel cabinets	Sorter and tabulator; accounting clerks reading printed reports	Card stock, machine rental from the vendor, two operators (illustrative)
2	Data processing manager	Programmers who change the copybook; the night operator	Commercial record; tape generations kept (grandfather, father, son)	Batch COBOL programs; printed renewal and audit reports	Computer lease, tape reels, programmers and operators (illustrative)

Hop	Accountable owner	Who can change it	Classification; retention or erasure	Read by	Cost per month; basis
3	Data processing manager; the database administrator (a new role)	DBA and application programmers; online clerks through screens	Commercial record	Online clerks on terminals; batch reports	Mainframe, IMS licence, DBA and programmers (illustrative)
4	Head of policy administration	DBAs, the conversion team, batch IDs	Commercial record; state record retention rules apply	Policy system screens; SQL reports; actuaries' extracts	Conversion project, mainframe, DB2 licence, staff (illustrative)
5	None recorded	Anyone with access to the shared drive	Contains personal and commercial data; no retention rule applied	Clerks; later an assistant that can search the shared drive	Near zero in storage; real cost in rework later (illustrative)
5b	Head of policy administration (signed the change)	One DBA running the load	Commercial record	Policy system	One day of a DBA; years of later reconciliation (illustrative)
6	Head of data platform	Pipeline service account; three data engineers; the warehouse load role	Commercial record; the policy domain also holds personal data of individual policyholders under US federal (GLBA) and state insurance privacy rules; retention duties	Finance and actuarial dashboards; data scientists	About 2,800 dollars a month to store, move and query this domain (illustrative)
7	Head of data platform	Stream platform team; connector configuration	Commercial record; the policy domain also holds personal data of individual policyholders under US federal (GLBA) and state insurance privacy rules; retention duties	Real-time dashboards; from 2025 the service agent's tools, through the warehouse	Stream platform share, about 600 dollars a month (illustrative)
8	Head of customer service (agent owner); head of data platform (data)	Agent platform team; prompt and tool configuration; the vector index job	Commercial record; personal data of individual policyholders under US federal (GLBA) and state insurance privacy rules; agent access logged	The service agent and, through it, customers	Model and retrieval costs, about 1,100 dollars a month for this use (illustrative)
9	Head of customer service	Agents (human and AI); CRM administrators	Personal data; complaint-handling retention	Claims handlers, the next agent, the complaints team	CRM licence share (illustrative)
9b	Head of customer service	Claims handlers; CRM administrators	Personal data; kept under the insurer's complaint-handling retention policy, so the wrong note is flagged, not deleted	Claims handlers, the complaints team, and every later agent, which now reads the flag before the note	Ten minutes of a claims handler (illustrative)

Mapping to OpenLineage and W3C PROV

OpenLineage (LF AI & Data Foundation) models a job, a run and a dataset; a run event carries `eventType`, `eventTime`, `run`, `job`, `inputs`, `outputs`, `producer` and `schemaURL`, with detail in facets. PROV-O, a W3C Recommendation of 30 April 2013, starts from `prov:Entity`, `prov:Activity` and `prov:Agent`.

Ledger column	OpenLineage	W3C PROV-O
1. Record, hop	Dataset; the hop is a job, each execution a run	Entity; the hop is an Activity
2. From	`inputs`	`prov:used`
3. To	`outputs`	`prov:wasGeneratedBy`, `prov:wasDerivedFrom`
4. Mechanism	Job and run facets; `eventTime`	`prov:startedAtTime`, `prov:generatedAtTime`
5. Lost or assumed	`schema` and `columnLineage` facets show the change; no field for loss: use a custom facet	No term; `prov:wasRevisionOf` marks a revision
6. Owner	`ownership` dataset facet	`prov:wasAttributedTo`
7. Who can change it	No field; `producer` names only the tool	`prov:wasAssociatedWith`, `prov:actedOnBehalfOf`: who took part in a recorded change, not who may make one
8. Class; duty	`tags` facet; `lifecycleStateChange` records events, not duties	`prov:wasInvalidatedBy` records a deletion after the fact
9. Read by	Later runs that list it in `inputs` (instrumented readers only)	Later activities that `prov:used` it
10. Cost	No field	No term

The mismatch. OpenLineage works at dataset, job and run grain, with column detail in a facet; the Ledger can also hold one record or field. Neither standard records what was lost. OpenLineage sees only readers that emit events, so a clerk's workbook (hop 5) and an agent's tool call (hop 8) are missing unless added; PROV can state them, but someone must

write the statements. A custom facet carries its own fields plus the required `_producer` and `_schemaURL`, so the loss code can travel with each run event. Column names match the Data Bill of Materials in *From Bronze to Gold*.

APPENDIX B

Timeline

Twenty-six dated points from the book, each checked against a primary source or an institutional history. Where only a month or a year is certain, only that is given. A longer version, with the thread's hops beside the real events, is on the book's web page.

Date	Event	Ch	Source
1494	Pacioli's *Summa de arithmetica* printed in Venice: the earliest printed description of double entry known to historians, "the method of Venice"	3	National Library of Scotland; ICAEW Library
16 Oct 1834	Obsolete Exchequer tally sticks burned in stoves under the House of Lords; the fire destroys most of the Palace of Westminster	3	UK Parliament, "The Great Fire of 1834"
1890	Hollerith's tabulating machines process the US census	4	US Census Bureau, "The Hollerith Machine"
10 Jan 1921	Fire in the Commerce Building damages the 1890 population schedules; Congress authorizes destruction of the remains in 1933	13	National Archives, *Prologue*, Spring 1996
1928	IBM introduces the 80-column card with rectangular holes	4	IBM History, "The punched card"
31 Mar 1951	By its own history, the Census Bureau signs for UNIVAC I (other histories call the day its acceptance)	5	US Census Bureau, "UNIVAC I"
1956	IBM introduces the 305 RAMAC with the 350 disk unit; the first 350 ships in June	5, 13	IBM History, "RAMAC"; Computer History Museum
Apr 1960	*COBOL: Initial Specifications* published after the CODASYL meeting of May 1959	5	CODASYL report, US Department of Defense
14 Aug 1968	ICS/DL/I, the system IBM renamed IMS, sends its first "READY" message at Rockwell Space Division, Downey; IBM announces the commercial release in 1968	5	IBM History, "Information Management System"

Date	Event	Ch	Source
Jun 1970	Codd, "A Relational Model of Data for Large Shared Data Banks", *CACM* 13(6)	6	doi.org/10.1145/362384.362685
Apr 1971	CODASYL Data Base Task Group report (network model)	5	ACM, DBTG April 71 Report
7 Jun 1983	IBM announces DB2 for MVS; Version 1.1 launched on MVS in 1985	6	IBM History, "The relational database"
16 Oct 1986	ANSI X3.135-1986, the first SQL standard, approved	6	NIST, FIPS PUB 127
Oct 1991	Unicode 1.0, volume 1	11	Unicode Consortium, publication dates
4 Jun 1996	Ariane 501 breaks up about 40 seconds after the flight sequence began	11	Inquiry Board report, 19 July 1996
23 Sep 1999	Mars Climate Orbiter lost at Mars arrival	11	NASA MIB Phase I report, 10 Nov 1999
Jun 2011	Kreps, Narkhede and Rao present Kafka at the NetDB workshop	9	NetDB'11 paper
16 Jan 2013	JPMorgan Chase publishes its Management Task Force report on the 2012 CIO losses	7	JPMorgan Chase Form 8-K, 16 Jan 2013
30 Apr 2013	PROV-O becomes a W3C Recommendation	2	W3C, PROV-O
Jun 2017	Vaswani et al., "Attention Is All You Need" (the Transformer)	10	arXiv 1706.03762
22 Apr 2018	TSB goes live after migrating its customers to a new platform built and hosted by SABIS	13	Slaughter and May, *TSB Review*, dated 31 October 2019, para 1.1
25 May 2018	The GDPR applies (Article 99)	12	Regulation (EU) 2016/679
16 Dec 2019	Bates v Post Office (No 6) "Horizon Issues" judgment	14	[2019] EWHC 3408 (QB)
17 Dec 2020	Dutch parliamentary report *Ongekend onrecht* on childcare benefits	12	Tweede Kamer, Kamerstuk 35 510, nr. 2
7 Jul 2023	Report of the Royal Commission into the Robodebt Scheme signed and presented	12	The report, vol. 1, letter of transmittal
25 Nov 2024	Anthropic announces the Model Context Protocol	10	Anthropic, "Introducing the Model Context Protocol"

Regulation (EU) 2026/1744, which amends the EU AI Act's high-risk dates, was published in the Official Journal on 24 July 2026; Chapter Twelve gives the dates it sets. Full references for every row are in Appendix E.

APPENDIX C

Failure Library

The main failure cases in the book, with the official report, judgment or primary paper behind it. The loss code is the author's reading of the case through the Ledger, not a finding of the report; where none of the eight codes fits, the table says so, because some systems failed on their own and some records were wrong at birth. Each case is told once, in the chapter named; this table only points to it. A machine-readable copy of the library is part of the free download on the book's web page.

Case, year	What was lost (loss code)	Official report or judgment	Ch
Exchequer tallies, 1834	COMPLETENESS Two cartloads of tally sticks, records of an accounting system not used since 1826, were burned in stoves under the House of Lords on 16 October; the fire destroyed both Houses of Parliament. Deletion as a hop, done without a safe method.	"The Great Fire of 1834", UK Parliament, Living Heritage. parliament.uk/about/living-heritage/building/palace/architecture/palacestructure/great-fire/	3
1890 census schedules, 1921	COMPLETENESS A fire in the Commerce Building on 10 January 1921 destroyed about a quarter of the 1890 population schedules and damaged about half the rest; Congress authorized destruction of the remains on 21 February 1933. No fewer than 6,160 names survive.	Kellee Blake, "First in the Path of the Firemen: The Fate of the 1890 Population Census", *Prologue* 28(1), US National Archives, Spring 1996. archives.gov/publications/prologue/1996/spring/1890-census-1.html	13
Ariane 501, 1996	PRECISION On 4 June, software reused from Ariane 4 raised an exception while converting a 64-bit floating-point value to a 16-bit signed integer; both inertial reference systems failed and the launcher broke up about 40 seconds into the flight sequence.	*ARIANE 5 Flight 501 Failure: Report by the Inquiry Board*, chaired by J. L. Lions, set up by ESA and CNES, Paris, 19 July 1996. esa.int/Newsroom/Press_Releases/Ariane_501_-_Presentation_of_Inquiry_Board_report	11
Mars Climate Orbiter, 1999	MEANING Thruster performance data in English units was used in a ground software file, "Small Forces", where metric units were required; the unit did not travel with the number. The spacecraft was lost on 23 September.	*Mars Climate Orbiter Mishap Investigation Board Phase I Report*, NASA, 10 November 1999. llis.nasa.gov/llis_lib/pdf/1009464main1_0641-mr.pdf	11
JPMorgan Chase CIO losses, 2012	PRECISION OWNERSHIP The new value-at-risk model "operated through a series of Excel spreadsheets" filled "by a process of copying and pasting data from one spreadsheet to another" (p. 124); one spreadsheet "divided by their sum instead of their average" (p. 128), likely muting volatility.	*Report of JPMorgan Chase & Co. Management Task Force Regarding 2012 CIO Losses*, JPMorgan Chase, 16 January 2013 (announced in Form 8-K, sec.gov, same date). elischolar.library.yale.edu/ypfs-documents/454	7

Case, year	What was lost (loss code)	Official report or judgment	Ch
Reinhart and Rogoff replication, 2013	COMPLETENESS PRECISION The authors replicated a widely cited result and attributed its errors to "coding errors, selective exclusion of available data, and unconventional weighting of summary statistics"; the coding error was in a spreadsheet.	Thomas Herndon, Michael Ash and Robert Pollin, "Does High Public Debt Consistently Stifle Economic Growth? A Critique of Reinhart and Rogoff", PERI Working Paper 322, University of Massachusetts Amherst, April 2013; later *Cambridge Journal of Economics* 38(2), 2014. peri.umass.edu	7
TSB migration, 2018	MEANING IDENTITY TSB migrated its customers to a new platform built and hosted by SABIS, live on 22 April 2018 (para 1.1). The records moved and the platform failed; later, some customers got an incorrect reason code and were recorded as deceased, and false "also known as" links let a few see each other's accounts (para 19.19).	*TSB Review: An Independent Review Following TSB's Migration onto a New IT Platform in April 2018*, Slaughter and May, dated 31 October 2019, published by the TSB Board on 19 November 2019. tsb.co.uk/news-releases/slaughter-and-may/	13
Post Office Horizon, judgment 2019	No single code fits: the court found that Fujitsu could change branch data remotely, that an injected transaction "would look as though the SPM had done it" (para 1004), that the powerful roles were "effectively unaudited" (para 1008), and Horizon's figures were believed over people. It found "Legacy Horizon was not remotely robust" (para 975), and that Fujitsu "could access all transaction data recorded by Horizon" remotely (para 991).	*Bates and Others v Post Office Ltd*, Judgment (No 6) "Horizon Issues", [2019] EWHC 3408 (QB), Fraser J, 16 December 2019. judiciary.uk/judgments/bates-others-v-post-office	14
Robodebt, report 2023	TIME MEANING Income employers reported to the tax office for a period was spread evenly across fortnights and compared with fortnightly declarations to raise debts; the Royal Commission called the scheme "neither fair nor legal".	*Report of the Royal Commission into the Robodebt Scheme*, Commissioner Catherine Holmes AC SC, presented 7 July 2023. robodebt.royalcommission.gov.au; averaging described in vol. 1, p. xxiv	12

Case, year	What was lost (loss code)	Official report or judgment	Ch
Dutch childcare benefits, report 2020	RIGHTS The committee found that in administering the benefit fundamental principles of the rule of law were violated; the Data Protection Authority found nationality processed unlawfully, including in a risk model (report of 16 July 2020). The purpose and limits of the records did not protect the parents.	*Ongekend onrecht* (Unprecedented Injustice), Parlementaire ondervragingscommissie Kinderopvangtoeslag, Tweede Kamer, Kamerstuk 35 510, nr. 2, 17 December 2020. tweedekamer.nl, final report PDF (pages as cited in Chapter Twelve)	12

How to use the library

Read the cases by loss code, not by industry. MEANING and PRECISION cases (Mars Climate Orbiter, Ariane 501, the spreadsheets) are the ones a Type Contract and a carried code table are built to catch; TSB's reason codes add an IDENTITY loss. COMPLETENESS cases (the tallies, the 1890 schedules) are deletion and retention decisions, and belong in the deletion map of Chapter Fifteen. TIME, MEANING and RIGHTS cases (Robodebt, the childcare benefits) are the ones where a derived record was used for a purpose its form could not support, and they are the cases most like an agent reading a dataset it was never meant to read. The case with no single code (Horizon) is a reminder that the Ledger is not the whole of proof: Chapter Fourteen sets out what lineage cannot show.

APPENDIX D

Glossary

Terms used in this book, in plain words, as of November 2026.

Term	Meaning
ACID	Atomicity, consistency, isolation, durability: the guarantees of a database transaction.
Agent	An AI system that calls tools, reads data and can act or write on someone's behalf.
Audit trail	A record of who changed a record, when, and from what.
Change data capture (CDC)	Reading a database's log to publish each change as an event.
Code table	The list that says what each code in a field means, such as status 7.
Column-level lineage	Lineage that shows which input columns feed each output column.
Copybook	A COBOL file that describes the layout of a record for every program that reads it.
Data independence	Codd's idea that programs see a logical view of data, not its access path.
Dataset	A named collection of records, such as a table or a file.
Date window	A Y2K fix that reads two-digit years in a range as one century or the next.
Deletion duty	A legal or contractual obligation to erase a record, everywhere it went.
Derived artifact	Something computed from records: a report, a feature, an embedding, a model.
Double entry	Bookkeeping in which every transaction is recorded with equal debits and credits, so many errors show.
EBCDIC	IBM's eight-bit character encoding for mainframes.
Embedding	A list of numbers (a vector) that a model uses to place text by meaning.
Era card	The box at the end of each chapter: reader, security, cost and AI for that era.
Event time	When something happened in the world, as opposed to when a system processed it.
Facet	In OpenLineage, a named block of extra detail on a job, run or dataset.
Field	One named value in a record, such as an issue date.
Grain	What one Ledger row describes: a record, field, event, dataset or derived artifact.
Hop	A movement or transformation after which a different reader holds the record.
IMS	IBM's hierarchical database, announced in 1968.
Key	The value that identifies a record, such as policy 1898/0417.

Term	Meaning
Legal hold	A duty to keep records that may be evidence, suspending normal deletion.
Lineage	Where a record came from and what happened to it on the way.
Loss code	One of eight Ledger labels for what a hop lost: MEANING, CONTEXT, PRECISION, TIME, IDENTITY, OWNERSHIP, RIGHTS, COMPLETENESS.
Maturity scale	The author's five levels for lineage: unknown, written, captured, governed, evidential.
Model Context Protocol (MCP)	An open protocol, announced by Anthropic in November 2024, for connecting models to tools and data.
OpenLineage	An open specification for lineage events about jobs, runs and datasets.
Owner (accountable)	A person who approves changes to a dataset and answers when it is wrong.
Packed decimal	Decimal digits one per half-byte, sign in the last half-byte.
Parquet	An open columnar file format for analytics.
Perpetual policy	A fire policy paid once by a deposit; in this book's example, in force while the building stands.
Processing time	When a system handled an event, which may differ from event time.
PROV	The W3C family of provenance specifications: entity, activity, agent.
Purpose of processing	The stated reason personal data is used; under the GDPR it limits further use. The Ledger records it in column 8.
Record	A promise from a writer to a future reader: values written to be read later.
Retention	How long a record must, or may, be kept.
Retrieval	Finding documents and passing them to a model as context for an answer.
Schema	The declared structure of a record: fields, types, keys.
Schema on read	Structure applied by the reader when data is read, not when it is written.
Status code	A short code that stands for a state, meaningful only with its code table.
Tally stick	A notched stick split in two, so each party holds a matching half.
Token	A piece of text, as a number, that a model reads; also, in payments, a stand-in for card data.
Tokenizer	The program that splits text into a model's tokens.
Type Contract	A field's seven-line box: type and precision, encoding, unit, time basis, identity scope, absence, authority.
Unit-record	Punched-card data processing: one card per record, machines per step.
Vector index	A store that finds embeddings close to a query's embedding.
Write-ahead log	A log a database writes before changing data, so it can recover.

Term	Meaning
Write-back	An agent writing to a system of record; it is a hop with its own row.
Zoned decimal	One digit per byte in character form, with the sign in the last byte's zone.

APPENDIX E

Sources

Every dated fact in this book was checked against the source listed for its chapter in November 2026 (edition 1.1 rechecks on 20 November 2026). Primary sources come first: original papers, standards, statutes, judgments, official reports, vendor announcements and institutional histories. This list holds 307 sources; the claim-by-claim record behind it is kept with the book and is available on request.

Reader's Brief: One Record, Six Questions

- Library Company of Philadelphia, PEAES Guide: Philadelphia Contributionship https://librarycompany.org/Economics/PEAESGuide/PDF/CS-PhilaContributionship.pdf

One Record, Since 1898

- Library Company of Philadelphia, PEAES Guide: Philadelphia Contributionship https://librarycompany.org/Economics/PEAESGuide/PDF/CS-PhilaContributionship.pdf
- Library Company of Philadelphia, PEAES Guide (as above
- The Philadelphia Contributionship Digital Archives, "The Past" https://www.philadelphiabuildings.org/contributionship/essay-past.cfm

Chapter 1. What a Record Is

- Library Company of Philadelphia, PEAES Guide: Philadelphia Contributionship https://librarycompany.org/Economics/PEAESGuide/PDF/CS-PhilaContributionship.pdf
- Library Company of Philadelphia, PEAES Guide (as above
- 44 U.S.C. 3301(a)(1)(A), statute text via the Legal Information Institute https://www.law.cornell.edu/uscode/text/44/3301
- US Census Bureau, "The Hollerith Machine" https://www.census.gov/about/history/bureau-history/census-innovations/technology/hollerith-machine.html
- National Library of Scotland https://www.nls.uk/collections/rare-books/collections/icas/pacioli; ICAEW Library https://www.icaew.com/library/library-collection/historical-accounting-literature/highlights-of-the-collection/summa-di-arithmetica
- US Census Bureau, "UNIVAC I" https://www.census.gov/about/history/bureau-history/census-innovations/technology/univac-i.html
- IBM History, "RAMAC" https://www.ibm.com/history/ramac
- IBM History, "Information Management System" https://www.ibm.com/history/information-management-system
- ACM Digital Library https://doi.org/10.1145/362384.362685
- IBM, "Celebrating 40 years of Db2" https://www.ibm.com/new/announcements/celebrating-30-years-of-db2-running-the-worlds-mission-critical-workloads
- Computer History Museum catalog, VisiCalc (Software Arts, 1979) https://www.computerhistory.org/
- Open Library record, Building the Data Warehouse (QED Technical Publishing Group, 1992, ISBN 0-89435-404-3) https://openlibrary.org/books/OL1560174M/Building_the_data_warehouse
- LinkedIn https://blog.linkedin.com/2011/01/11/open-source-linkedin-kafka
- arXiv 1301.3781 https://arxiv.org/abs/1301.3781
- arXiv 1706.03762 https://arxiv.org/abs/1706.03762
- Anthropic, "Introducing the Model Context Protocol" https://www.anthropic.com/news/model-context-protocol
- Kimball Group, "Declaring the Grain" https://www.kimballgroup.com/2003/03/declaring-the-grain/
- US Census Bureau, "Establishment of the Permanent U.S. Census Bureau" https://www.census.gov/about/history/stories/monthly/2017/march-2017.html

Chapter 2. The Lineage Ledger

- OpenLineage https://openlineage.io/
- OpenLineage specification, Object model https://openlineage.io/docs/spec/object-model
- OpenLineage specification, Run cycle https://openlineage.io/docs/spec/run-cycle
- OpenLineage specification, Dataset facets https://openlineage.io/docs/spec/facets/dataset-facets/
- OpenLineage specification, Column level lineage dataset facet https://openlineage.io/docs/spec/facets/dataset-facets/column_lineage_facet
- OpenLineage specification, Custom facets https://openlineage.io/docs/spec/facets/custom-facets
- W3C, PROV-Overview https://www.w3.org/TR/prov-overview/
- W3C, PROV-DM: The PROV Data Model (Recommendation, 30 April 2013) https://www.w3.org/TR/prov-dm/
- From Bronze to Gold (Medallion, Book 1), Reader's Brief, same author

Chapter 3. Before the Computer: Tallies, Tokens and Ledgers

- UK Parliament, Living Heritage, "The Great Fire of 1834" https://www.parliament.uk/about/living-heritage/building/palace/architecture/palacestructure/great-fire/
- UK Parliament, "The Great Fire of 1834" (as above
- UK Parliament, "The Great Fire of 1834", "Did you know?" note (as above
- Science Museum Group Collection, "Stock from exchequer split tally, English, 1822" https://collection.sciencemuseumgroup.org.uk/objects/co8457645/stock-from-exchequer-split-tally-english-1822
- Science Museum Group Collection, "Medieval Exchequer tally sticks" https://collection.sciencemuseumgroup.org.uk/objects/co60506/medieval-exchequer-tally-sticks
- Denise Schmandt-Besserat, "From Accounting to Writing", University of Texas https://sites.utexas.edu/dsb/tokens/from-accounting-to-writing/
- Lucy E. Bennison-Chapman, "Reconsidering 'Tokens': The Neolithic Origins of Accounting or Multifunctional, Utilitarian Tools?", Cambridge Archaeological Journal 29(2), 2019, 233 to 259 https://www.cambridge.org/core/journals/cambridge-archaeological-journal/article/abs/reconsidering-tokens-the-neolithic-origins-of-accounting-or-multifunctional-utilitarian-tools/7E6C04CB040AD8AA0EA84B94D4D275C4
- ICAEW Library, Historical accounting literature, "Summa de arithmetica" https://www.icaew.com/library/library-collection/historical-accounting-literature/highlights-of-the-collection/summa-di-arithmetica
- National Library of Scotland, ICAS collection, Pacioli (catalogue record) https://www.nls.uk/collections/rare-books/collections/icas/pacioli
- Alan Sangster and Franco Rossi, "Benedetto Cotrugli on Double Entry Bookkeeping", De Computis 15(2), 2018, 22 to 38 https://www.upo.es/revistas/index.php/decomputis/article/view/6934
- US National Archives, "1790 Census Records" https://www.archives.gov/research/census/1790
- US Constitution, Article I, Section 2, Clause 3, National Archives transcript https://www.archives.gov/founding-docs/constitution-transcript
- US Census Bureau, "1790 Overview" https://www.census.gov/history/www/through_the_decades/overview/1790.html
- The Philadelphia Contributionship, company timeline https://www.philadelphiabuildings.org/contributionship/timeline.cfm
- Maryland Center for History and Culture, Baltimore Equitable Society collection, finding aid https://mdhistory.libraryhost.com/repositories/2/resources/929

Chapter 4. Cards and the Unit-Record Shop

- US Census Bureau, "Herman Hollerith" (Notable Alumni) https://www.census.gov/library/photos/1880/herman-hollerith.html
- US Patent and Trademark Office, Journeys of Innovation, "Count me in" https://www.uspto.gov/learning-and-resources/journeys-innovation/historical-stories/count-me
- US Census Bureau, "The Hollerith Machine" https://www.census.gov/about/history/bureau-history/census-innovations/technology/hollerith-machine.html
- IBM, "The punched card" https://www.ibm.com/history/punched-card
- US Senate Special Committee on the Year 2000 Technology Problem, "Investigating the Impact of the Year 2000 Problem" (S. Prt. 106-10) https://www.govinfo.gov/content/pkg/GPO-CPRT-106sprt10/pdf/GPO-CPRT-106sprt10-6-1.pdf

- US Census Bureau, "Establishment of the Permanent U.S. Census Bureau" https://www.census.gov/about/history/stories/monthly/2017/march-2017.html
- IBM History, "The punched card tabulator" https://www.ibm.com/history/punched-card-tabulator

Chapter 5. Tape, Disk and the First Databases

- US Census Bureau, "UNIVAC I" https://www.census.gov/about/history/bureau-history/census-innovations/technology/univac-i.html
- Computer History Museum, Timeline of Computer History, 1951 https://www.computerhistory.org/timeline/1951/
- Computer History Museum, Revolution, "UNIVAC I Uniservo tape drive" https://www.computerhistory.org/revolution/artifact/100/1491
- IBM, "RAMAC" https://www.ibm.com/history/ramac
- Computer History Museum, Timeline of Computer History, 1956 https://www.computerhistory.org/timeline/1956/
- IBM, "RAMAC" (no day given) and Computer History Museum, 1956 timeline (as above
- Computer History Museum, "This Day in History: May 28" https://www.computerhistory.org/tdih/may/28/
- CODASYL, "COBOL: Initial Specifications for a Common Business Oriented Language", US Department of Defense, April 1960 (report
- IBM, Enterprise COBOL for z/OS Language Reference, "COPY statement" https://www.ibm.com/docs/en/SS6SG3_6.5/lr/ref/rlcdscop.html
- IBM, "IBM System/360 Principles of Operation", A22-6821-6, January 1967, pp. 11, 35, 36, 39 http://www.bitsavers.org/pdf/ibm/360/princOps/A22-6821-6_360PrincOpsJan67.pdf
- Computer History Museum, Fellow award, "Charlie Bachman" https://computerhistory.org/?p=4766
- IBM, "Information Management System" https://www.ibm.com/history/information-management-system
- CODASYL Data Base Task Group, "April 71 Report", ACM, New York, 1971 https://doi.org/10.1145/1387569
- IBM History, "System/360" https://www.ibm.com/history/system-360
- IBM, Enterprise COBOL for z/OS 6.2 Programming Guide, "Formats for numeric data" https://www.ibm.com/docs/SS6SG3_6.2.0/pg/concepts/cpari09.html

Chapter 6. The Relational Bet

- E. F. Codd, CACM 13(6), 1970 https://doi.org/10.1145/362384.362685 text read from the published article
- IBM, IMS Application Programming APIs, GN/GHN call and GNP/GHNP call https://www.ibm.com/support/knowledgecenter/en/SSEPH2_14.1.0/com.ibm.ims14.doc.apr/ims_gnpghnpcall.htm
- IBM, "The relational database" https://www.ibm.com/history/relational-database
- ACM Digital Library https://doi.org/10.1145/800296.811515
- Oracle Database SQL Language Reference, History of SQL https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/History-of-SQL.html
- Stonebraker, Wong, Kreps and Held, "The Design and Implementation of INGRES", ACM TODS 1(3), September 1976 https://doi.org/10.1145/320473.320476
- Harris and Nicol, "SQL/DS: IBM's First RDBMS", IEEE Annals of the History of Computing 35(2), 2013 https://doi.org/10.1109/MAHC.2013.28
- IBM, "Celebrating 40 years of Db2" https://www.ibm.com/new/announcements/celebrating-30-years-of-db2-running-the-worlds-mission-critical-workloads
- ANSI X3.135-1986 / FIPS PUB 127 (NBS, 1987) https://nvlpubs.nist.gov/nistpubs/Legacy/FIPS/fipspub127.pdf
- ISO catalogue https://www.iso.org/standard/16661.html; and https://www.iso.org/standard/16662.html
- Jim Gray, Tandem Computers, VLDB 1981 (Tandem TR 81.3, June 1981), copy read at https://www.cs.purdue.edu/homes/bb/cs542-06Spr-bb/TCVL-Gray-81.pdf
- ACM Computing Surveys https://doi.org/10.1145/289.291
- US GAO, Year 2000 Computing Crisis: An Assessment Guide, GAO/AIMD-10.1.14, September 1997 https://www.gao.gov/assets/aimd-10.1.14.pdf
- US GAO, testimony GAO/T-AIMD-99-214, 22 June 1999 https://www.govinfo.gov/content/pkg/GAOREPORTS-T-AIMD-99-214/html/GAOREPORTS-T-AIMD-99-214.htm
- IBM COBOL for AIX 5.1 Programming Guide, Using a century window (sentence verbatim, example included) https://www.ibm.com/docs/en/cobol-aix/5.1?topic=problems-using-century-window

Chapter 7. Data on Every Desk

- The MS-DOS Encyclopedia (Microsoft Press, 1988), Section I https://www.pcjs.org/documents/books/mspl13/msdos/encyclopedia/section1/
- Computer History Museum, Revolution, Software: Putting PCs to Work https://www.computerhistory.org/revolution/personal-computers/17/305
- Computer History Museum, Mitchell Kapor profile https://computerhistory.org/profile/mitchell-kapor/
- Microsoft Learn, The History of Microsoft, 1985 https://learn.microsoft.com/en-us/shows/history/history-of-microsoft-1985
- Microsoft Learn, The History of Microsoft, 1987 https://learn.microsoft.com/en-us/shows/history/history-of-microsoft-1987
- Microsoft Learn, The History of Microsoft, 1992 https://learn.microsoft.com/en-us/shows/history/history-of-microsoft-1992
- Shafranovich, RFC 4180, Common Format and MIME Type for CSV Files https://www.rfc-editor.org/rfc/rfc4180
- Microsoft, Differences between the 1900 and the 1904 date system in Excel https://docs.microsoft.com/office/troubleshoot/excel/1900-and-1904-date-system; Date systems in Excel https://support.microsoft.com/en-us/office/date-systems-in-excel-e7fe7167-48a9-4b96-bb53-5612a800b487
- Microsoft Learn, Excel incorrectly assumes that the year 1900 is a leap year https://learn.microsoft.com/en-us/troubleshoot/microsoft-365-apps/excel/wrongly-assumes-1900-is-leap-year
- Microsoft Support, Change the date system, format, or two-digit year interpretation https://support.microsoft.com/en-us/excel/change-the-date-system-format-or-two-digit-year-interpretation
- Google, "Introducing Google Drive… yes, really" https://drive.googleblog.com/2012/04/introducing-google-drive-yes-really.html
- JPMorgan Chase & Co., Task Force Report (as filed by the firm) https://web.archive.org/web/2013/http://files.shareholder.com/downloads/ONE/2272984969x0x628656/4cb574a0-0bf5-4728-9582-625e4519b5ab/Task_Force_Report.pdf; catalogue record https://elischolar.library.yale.edu/ypfs-documents/454
- Herndon, Ash and Pollin, "Does High Public Debt Consistently Stifle Economic Growth? A Critique of Reinhart and Rogoff" https://peri.umass.edu/images/WP322.pdf
- Microsoft Learn, Data, Privacy, and Security for Microsoft Copilot https://learn.microsoft.com/en-us/copilot/microsoft-365/microsoft-365-copilot-privacy
- Author's computation from Microsoft's date-system rules (serial 1 = 1 January 1900, with the 1900 leap-year day), 19 November 2026
- Microsoft Learn, "How shareable links work in OneDrive and SharePoint in Microsoft 365" (dated 6 November 2026) https://learn.microsoft.com/en-us/sharepoint/shareable-links-anyone-specific-people-organization

Chapter 8. Warehouse to Lakehouse

- Crossref record, DOI 10.1147/sj.271.0060 https://doi.org/10.1147/sj.271.0060
- Barry Devlin, "Data Warehousing 3.0", TDWI, 1 February 2016 https://tdwi.org/articles/2016/02/01/data-warehousing-30.aspx
- Library of Congress catalog record, LCCN 91041284 https://lccn.loc.gov/91041284
- Library of Congress catalog record, LCCN 95050557 https://lccn.loc.gov/95050557
- White paper text https://staff.icar.cnr.it/manco/Teaching/2006/datamining/articoli/olapcoddwp.pdf
- Armbrust, Ghodsi, Xin and Zaharia, "Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics", CIDR 2021 https://vldb.org/cidrdb/2021/lakehouse-a-new-generation-of-open-platforms-that-unify-data-warehousing-and-advanced-analytics.html
- AWS What's New, "Announcing Amazon Redshift" https://aws.amazon.com/about-aws/whats-new/2012/11/28/announcing-amazon-redshift
- AWS What's New https://aws.amazon.com/about-aws/whats-new/2013/02/14/amazon-redshift-available-to-all-customers/
- Google Developers Blog, "Google BigQuery brings Big Data analytics to all businesses" https://developers.googleblog.com/2012/05/google-bigquery-brings-big-data.html
- Snowflake press release, "Snowflake Redefines the Data Warehouse for the Cloud Era", 23 June 2015 https://www.snowflake.com/?p=1440
- Dmitriy Ryaboy, "Announcing Parquet 1.0: Columnar Storage for Hadoop", Twitter Blog (archived) https://web.archive.org/web/20130910203755/https://blog.twitter.com/2013/announcing-parquet-10-columnar-storage-for-hadoop

- The Apache Software Foundation blog https://blogsarchive.apache.org/foundation/entry/the_apache_software_foundation_announces75
- Apache Parquet format, LogicalTypes.md https://github.com/apache/parquet-format/blob/master/LogicalTypes.md
- Databricks press release https://www.databricks.com/company/newsroom/press-releases/databricks-open-sources-delta-lake-for-data-lake-reliability
- Apache Incubator, Iceberg project status https://incubator.apache.org/projects/iceberg.html Armbrust et al., CIDR 2021
- Apache Iceberg Table Spec https://iceberg.apache.org/spec/
- AWS What's New, "Amazon Redshift introduces RA3 nodes with managed storage" https://aws.amazon.com/about-aws/whats-new/2019/12/amazon-redshift-announces-ra3-nodes-managed-storage
- Snowflake documentation, Understanding overall cost https://docs.snowflake.com/en/user-guide/cost-understanding-overall

Chapter 9. Meaning Moves to the Reader: Web, JSON, NoSQL, Logs and Streams

- W3C, REC-xml-19980210 https://www.w3.org/TR/1998/REC-xml-19980210
- RFC 8259, The JavaScript Object Notation (JSON) Data Interchange Format, IETF, December 2017 https://www.rfc-editor.org/rfc/rfc8259
- Kenton Varda, "Protocol Buffers: Google's Data Interchange Format", Google Open Source Blog https://opensource.googleblog.com/2008/07/protocol-buffers-googles-data.html
- ACM https://doi.org/10.1145/945445.945450
- USENIX https://www.usenix.org/legacy/event/osdi04/tech/full_papers/dean/dean.pdf
- Apache archive https://archive.apache.org/dist/hadoop/core/
- Armbrust et al., "Lakehouse", CIDR 2021 https://vldb.org/cidrdb/papers/2021/cidr2021_paper17.pdf
- USENIX OSDI '06 https://www.usenix.org/legacy/event/osdi06/tech/chang.html
- ACM https://doi.org/10.1145/1294261.1294281; abstract posted by co-author Werner Vogels https://www.allthingsdistributed.com/2007/10/amazons_dynamo.html
- Paper text https://people.csail.mit.edu/matei/courses/2015/6.S897/readings/kafka.pdf
- LinkedIn Engineering (archived) https://web.archive.org/web/20131229054459/http://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying
- VLDB https://www.vldb.org/pvldb/vol8/p1792-Akidau.pdf
- Apache Kafka documentation, Introduction (topics and logs) https://kafka.apache.org/11/_print
- IBM, Db2 13 for z/OS, "Altering a table to capture changed data" https://www.ibm.com/docs/en/SSEPEK_13.0.0/admin/src/tpc/db2z_altertablefordatacapture.html

Chapter 10. The Model as Reader

- Anthropic, "Introducing the Model Context Protocol" https://www.anthropic.com/news/model-context-protocol
- NIST, TREC-3 proceedings https://trec.nist.gov/pubs/trec3/t3_proceedings.html; paper https://trec.nist.gov/pubs/trec3/papers/city.ps.gz
- Wiley via Crossref https://doi.org/10.1002/(SICI)1097-4571(199009)41:6%3C391::AID-ASI1%3E3.0.CO;2-9
- arXiv 1301.3781 https://arxiv.org/abs/1301.3781
- arXiv 1706.03762 https://arxiv.org/abs/1706.03762
- arXiv 2005.11401 https://arxiv.org/abs/2005.11401
- USENIX https://www.usenix.org/conference/usenixsecurity21/presentation/carlini-extracting arXiv 2012.07805
- arXiv 2202.07646 https://arxiv.org/abs/2202.07646; ICLR 2023 https://iclr.cc/virtual/2023/poster/11469
- Amit Singhal, "Introducing the Knowledge Graph: things, not strings", Google https://blog.google/products/search/introducing-knowledge-graph-things-not/

Chapter 11. Meaning: the Same Value in Every Representation

- NASA, Mars Climate Orbiter Mishap Investigation Board Phase I Report, 10 November 1999, Executive Summary and section 2 https://llis.nasa.gov/llis_lib/pdf/1009464main1_0641-mr.pdf
- same report, section 4, MCO Root Cause

- same report, section 2
- IEEE Standards Association https://standards.ieee.org/ieee/754/993/
- Python 3 tutorial, "Floating-Point Arithmetic: Issues and Limitations" https://docs.python.org/3/tutorial/floatingpoint.html
- Author's computation, Python 3.13.3 (IEEE 754 double), 19 November 2026
- IBM System/360 Principles of Operation, form A22-6821-0 (1964), "Decimal Arithmetic", pp. 34 to 35 https://bitsavers.org/pdf/ibm/360/princOps/A22-6821-0_360PrincOps.pdf
- same manual, pp. 34 to 35
- same manual, p. 35
- Author's computation from the manual's rules and Python's cp037 codec, 19 November 2026
- IBM Enterprise COBOL 6.2 Language Reference, "Symbols used in the PICTURE clause" https://www.ibm.com/docs/en/SS6SG3_6.2.0/lr/ref/rlddesym.html
- IBM Enterprise COBOL 6.2 Programming Guide, "Formats for numeric data" https://www.ibm.com/docs/SS6SG3_6.2.0/pg/concepts/cpari09.html
- Apache Parquet format, LogicalTypes.md, DECIMAL https://github.com/apache/parquet-format/blob/master/LogicalTypes.md
- National Bureau of Standards, FIPS PUB 1-2 (ANSI X3.32-1973) https://nvlpubs.nist.gov/nistpubs/Legacy/FIPS/fipspub1-2.pdf
- IEEE Milestone, American Standard Code for Information Interchange ASCII, 1963 https://ethw.org/Milestones:American_Standard_Code_for_Information_Interchange_ASCII,_1963
- Unicode Consortium, Publication dates https://www.unicode.org/history/publicationdates.html
- RFC 3629, UTF-8, a transformation format of ISO 10646, November 2003, section 1 https://www.rfc-editor.org/rfc/rfc3629
- RFC 3629, header and "Changes from RFC 2279"; RFC 2279 https://www.rfc-editor.org/rfc/rfc2279
- HL7 FHIR R5, Data Types https://hl7.org/fhir/R5/datatypes.html
- Ariane 501 Inquiry Board report, cover and section 1 https://esamultimedia.esa.int/docs/esa-x-1819eng.pdf
- same report, section 2.1, pp. 3 to 4
- same report, section 3 (Cause of the failure), p. 13
- FCA Final Notice to Citigroup Global Markets Limited, 17 May 2024, paragraphs 2.4 to 2.6, 4.6, 4.7, 4.10 https://www.fca.org.uk/publication/final-notices/citigroup-global-markets-limited-2024.pdf
- FCA press release, "FCA fines CGML £27,766,200 for failures in its trading systems and controls" https://www.fca.org.uk/news/press-releases/fca-fines-cgml-27-million
- openai/tiktoken README https://github.com/openai/tiktoken
- Author's computation, tiktoken 0.9.0, 19 November 2026
- Model card https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
- Author's computation, all-MiniLM-L6-v2 (revision 1110a243), mean pooling and L2 normalization, PyTorch, 19 November 2026
- RFC 2044 and RFC 2279 headers https://www.rfc-editor.org/rfc/rfc2044; and https://www.rfc-editor.org/rfc/rfc2279

Chapter 12. Control and Law: Who Could Change It, Who Could Read It

- Report of the Royal Commission into the Robodebt Scheme, vol. 1, letter of transmittal https://robodebt.royalcommission.gov.au/publications/report PDF read via the Internet Archive copy of robodebt_report_volume_1.pdf
- same report, vol. 1, executive summary, p. xxix
- same report, vol. 1, p. xxiv
- same report, vol. 1, p. xiii
- same report, vol. 2, ch. 14, p. 402
- same report, vol. 2, ch. 14, p. 401
- Tweede Kamer https://www.tweedekamer.nl/sites/default/files/atoms/files/20201217_eindverslag_parlementaire_onderzoekscommissie_kinderopvangtoeslag.pdf
- same report, p. 7 of the Tweede Kamer PDF (pagina 7/132
- same report, Tweede Kamer PDF
- Autoriteit Persoonsgegevens, Belastingdienst/Toeslagen: De verwerking van de nationaliteit van aanvragers van kinderopvangtoeslag (z2018-22445), ch. 4, pp. 58 to 59 https://autoriteitpersoonsgegevens.nl/uploads/imported/onderzoek_belastingdienst_kinderopvangtoeslag.pdf

- Records, Computers and the Rights of Citizens, DHEW Publication No. (OS)73-94, Summary and Recommendations https://www.justice.gov/opcl/docs/rec-com-rights.pdf
- OECD/LEGAL/0188 https://legalinstruments.oecd.org/en/instruments/OECD-LEGAL-0188
- 15 U.S.C. 1681, source note https://www.govinfo.gov/content/pkg/USCODE-2023-title15/html/USCODE-2023-title15-chap41-subchapIII-sec1681.htm
- 15 U.S.C. 6801 to 6809 https://www.govinfo.gov/content/pkg/USCODE-2023-title15/html/USCODE-2023-title15-chap94-subchapI-sec6801.htm
- NAIC model laws 670 and 672, legislative history notes https://content.naic.org/sites/default/files/model-law-670.pdf; https://content.naic.org/sites/default/files/model-law-672.pdf
- https://www.govinfo.gov/content/pkg/PLAW-104publ191/html/PLAW-104publ191.htm
- 45 CFR 164.534 https://www.ecfr.gov/current/title-45/part-164
- 45 CFR 164.530(j)(2) https://www.ecfr.gov/current/title-45/part-164/section-164.530
- 42 CFR 493.1105(a)(6) https://www.law.cornell.edu/cfr/text/42/493.1105
- HHS, HIPAA FAQ 580 https://www.hhs.gov/hipaa/for-professionals/faq/580/does-hipaa-require-covered-entities-to-keep-medical-records-for-any-period/index.html
- Regulation (EU) 2016/679 https://eur-lex.europa.eu/eli/reg/2016/679/oj
- Regulation (EU) 2024/1689 https://eur-lex.europa.eu/eli/reg/2024/1689/oj
- Regulation (EU) 2026/1744, Art. 1 points 6, 9 and 40, Art. 4 https://eur-lex.europa.eu/eli/reg/2026/1744/oj
- Tweede Kamer https://www.tweedekamer.nl/nieuws/kamernieuws/eindverslag-onderzoek-kinderopvangtoeslag-overhandigd
- Seltzer and Anderson, "The Dark Side of Numbers", Social Research 68(2), 2001, p. 486 https://www.jstor.org/stable/40971467
- same article, p. 492
- Anderson and Seltzer, Journal of Privacy and Confidentiality 1(1), 2009, pp. 9 and 17 https://journalprivacyconfidentiality.org/index.php/jpc/article/view/563
- US Census Bureau, History of privacy protection https://www2.census.gov/library/visualizations/2019/communications/history-privacy-protection.pdf
- US Census Bureau, Disclosure Avoidance for the 2020 Census: An Introduction (November 2021), ch. 1 and 2 https://www2.census.gov/library/publications/decennial/2020/2020-census-disclosure-avoidance-handbook.pdf
- US Census Bureau press release CB21-CN.42 https://www.census.gov/newsroom/press-releases/2021/2020-census-key-parameters.html
- US Census Bureau, 2020 DAS updates https://www.census.gov/programs-surveys/decennial-census/decade/2020/planning-management/process/disclosure-avoidance/2020-das-updates/2021-08-12.html
- Prygodicz v Commonwealth of Australia (Robodebt settlement) [2026] FCA 1029, Beach J, orders of 23 June 2026 https://www.judgments.fedcourt.gov.au/judgments/Judgments/fca/single/2026/2026fca1029
- Prygodicz v Commonwealth of Australia (No 2) [2021] FCA 634, judgment summary https://www.judgments.fedcourt.gov.au/judgments/Judgments/fca/single/2021/2021fca0634/summaries/2021fca0634-summary
- 50 U.S.C. App., Second War Powers Act, 1942, codification note to former section 644a https://www.govinfo.gov/content/pkg/USCODE-2009-title50/html/USCODE-2009-title50-app-secondwar.htm
- Regulation (EU) 2026/1744, English text https://eur-lex.europa.eu/eli/reg/2026/1744/oj/eng

Chapter 13. Cost and Death: Store, Move, Query, Keep, Delete

- Kellee Blake, "First in the Path of the Firemen: The Fate of the 1890 Population Census", Prologue 28:1 (Spring 1996), National Archives https://www.archives.gov/publications/prologue/1996/spring/1890-census-1
- Blake, Prologue 1996, same URL
- Computer History Museum, "1956: First commercial hard disk drive shipped", The Storage Engine https://www.computerhistory.org/storageengine/first-commercial-hard-disk-drive-shipped/
- IBM, "RAMAC", IBM History https://www.ibm.com/history/ramac
- Amazon press release, "Amazon Web Services Launches Amazon S3", 14 March 2006 https://press.aboutamazon.com/2006/3/amazon-web-services-launches
- Amazon press release, "Amazon Web Services Announces Amazon Glacier", 21 August 2012 https://press.aboutamazon.com/2012/8/amazon-web-services-announces-amazon-glacier
- AWS Price List API, AmazonS3 offer, us-east-1 https://pricing.us-east-1.amazonaws.com/offers/v1.0/aws/AmazonS3/current/us-east-1/index.json

- AWS Price List API, AWSDataTransfer offer, us-east-1 (publication 2026-11-19) https://pricing.us-east-1.amazonaws.com/offers/v1.0/aws/AWSDataTransfer/current/us-east-1/index.json
- Zubulake v. UBS Warburg LLC, 220 F.R.D. 212, 218 (S.D.N.Y. 22 Oct. 2003) (Zubulake IV
- Zubulake IV, 220 F.R.D. at 216 to 218; Zubulake v. UBS Warburg LLC, 229 F.R.D. 422, 427, 431 (S.D.N.Y. 2004) (Zubulake V), CourtListener https://www.courtlistener.com/opinion/8757285/zubulake-v-ubs-warburg-llc/
- Zubulake V, 229 F.R.D. at 425 to 426
- Zubulake V, 229 F.R.D. at 432 to 434
- Zubulake V, 229 F.R.D. at 429, 436, 437
- Federal Rules of Civil Procedure, Rule 37(e), Cornell LII https://www.law.cornell.edu/rules/frcp/rule_37
- AWS, "Locking objects with Object Lock", Amazon S3 User Guide https://docs.aws.amazon.com/AmazonS3/latest/userguide/object-lock.html
- Library of Congress, "Sustainability of Digital Formats: Sustainability Factors" https://www.loc.gov/preservation/digital/formats/sustain/sustain.shtml
- Slaughter and May, "Independent Review of TSB's 2018 Migration to a New IT Platform" (report dated 31 October 2019), paragraphs 1.1, 2.1, 2.2 https://www.tsb.co.uk/content/dam/tsb-public/documents/media-centre/Slaughter-and-May-final-report.pdf
- Slaughter and May report, paragraphs 2.2, 2.3
- Slaughter and May report, paragraph 2.4
- Slaughter and May report, paragraph 2.7
- Slaughter and May report, paragraph 2.12
- Slaughter and May report, paragraph 19.19(F) and (G
- TSB, "TSB Board publishes independent review of 2018 IT Migration", 19 November 2019 https://www.tsb.co.uk/news-releases/slaughter-and-may/
- Zubulake v. UBS Warburg LLC, 217 F.R.D. 309 (S.D.N.Y. 2003) (Zubulake I
- AWS Price List https://pricing.us-east-1.amazonaws.com/offers/v1.0/aws/AmazonS3/current/us-east-1/index.csv

Chapter 14. Proof: What Shows a Record Is True and Current

- Judiciary of England and Wales https://www.judiciary.uk/wp-content/uploads/2019/12/bates-v-post-office-judgment.pdf
- Bates (No 6), paragraph 1
- Bates (No 6), paragraphs 928 to 929
- Bates (No 6), paragraphs 975 to 978
- Bates (No 6), paragraph 985
- Bates (No 6), paragraph 988
- Bates (No 6), paragraph 991
- Bates (No 6), paragraphs 994 to 995
- Bates (No 6), paragraph 1000
- Bates (No 6), paragraph 1003
- Bates (No 6), paragraph 1004
- Bates (No 6), paragraph 1005
- Bates (No 6), paragraph 1006
- Bates (No 6), paragraphs 1008, 1011
- Bates (No 6), paragraph 1013
- Bates (No 6), paragraph 1016
- Bates (No 6), paragraph 1018
- Post Office Horizon IT Inquiry, Final Report Volume 1 (8 July 2025), paragraphs 1.7 to 1.9, 1.12 https://www.postofficehorizoninquiry.org.uk/sites/default/files/2025-07/Post%20Office%20Horizon%20IT%20Inquiry%20Final%20Report%20Volume%201%20-%20Accessible%20Version.pdf
- ISO https://www.iso.org/standard/81745.html
- UK Government, "The Government Data Quality Framework", 3 December 2020 https://www.gov.uk/government/publications/the-government-data-quality-framework/the-government-data-quality-framework
- OpenLineage, "Column Level Lineage Dataset Facet" https://openlineage.io/docs/spec/facets/dataset-facets/column_lineage_facet
- UK Government, The Government Data Quality Framework, 3 December 2020

Chapter 15. Filling the Ledger in 90 Days

- OpenLineage, "About OpenLineage" https://openlineage.io/docs/
- LF AI & Data Foundation, OpenLineage project page https://lfaidata.foundation/projects/openlineage/
- OpenLineage, "Object Model" https://openlineage.io/docs/spec/object-model
- OpenLineage, "The Run Cycle" https://openlineage.io/docs/spec/run-cycle
- OpenLineage specification, spec/OpenLineage.md https://github.com/OpenLineage/OpenLineage/blob/main/spec/OpenLineage.md
- OpenLineage, "Column Level Lineage Dataset Facet" https://openlineage.io/docs/spec/facets/dataset-facets/column_lineage_facet
- OpenLineage, "Custom Facets" https://openlineage.io/docs/spec/facets/custom-facets
- Regulation (EU) 2016/679, Article 17 https://eur-lex.europa.eu/eli/reg/2016/679/oj
- Author's scale, defined in this chapter

Epilogue: The Record Outlives the System

- Juan Cardena, Every Pixel Matters series https://blog.jcardena.com/en/ebooks/
- Author's view, labeled "Our view, as of November 2026"

Appendix A. The Data Lineage Ledger: Template, Loss Codes and Mappings

- OpenLineage, "Object Model" https://openlineage.io/docs/spec/object-model
- OpenLineage specification https://github.com/OpenLineage/OpenLineage/blob/main/spec/OpenLineage.md
- OpenLineage, "Dataset Facets" https://openlineage.io/docs/spec/facets/dataset-facets/; https://openlineage.io/docs/spec/facets/dataset-facets/lifecycle_state_change; https://openlineage.io/docs/spec/facets/dataset-facets/ownership
- OpenLineage, "Column Level Lineage Dataset Facet" https://openlineage.io/docs/spec/facets/dataset-facets/column_lineage_facet
- OpenLineage, "Custom Facets" https://openlineage.io/docs/spec/facets/custom-facets
- W3C, "PROV-O: The PROV Ontology" https://www.w3.org/TR/prov-o/

Appendix B. Timeline

- National Library of Scotland https://www.nls.uk/collections/rare-books/collections/icas/pacioli; ICAEW Library https://www.icaew.com/library/library-collection/historical-accounting-literature/highlights-of-the-collection/summa-di-arithmetica
- UK Parliament, "The Great Fire of 1834" https://www.parliament.uk/about/living-heritage/building/palace/architecture/palacestructure/great-fire/
- US Census Bureau, "The Hollerith Machine" https://www.census.gov/about/history/bureau-history/census-innovations/technology/hollerith-machine.html
- Kellee Blake, "First in the Path of the Firemen", Prologue 28(1), Spring 1996 https://www.archives.gov/publications/prologue/1996/spring/1890-census-1.html
- IBM History, "The punched card" https://www.ibm.com/history/punched-card
- US Census Bureau, "UNIVAC I" https://www.census.gov/about/history/bureau-history/census-innovations/technology/univac-i.html
- IBM History, "RAMAC" https://www.ibm.com/history/ramac; Computer History Museum https://www.computerhistory.org/storageengine/first-commercial-hard-disk-drive-shipped/
- CODASYL, COBOL: Initial Specifications for a Common Business Oriented Language, US Department of Defense, April 1960
- IBM History, "Information Management System" https://www.ibm.com/history/information-management-system
- https://doi.org/10.1145/362384.362685
- CODASYL Data Base Task Group April 71 Report, ACM, New York, 1971
- IBM, "Celebrating 40 years of Db2" https://www.ibm.com/new/announcements/celebrating-30-years-of-db2-running-the-worlds-mission-critical-workloads
- NIST, FIPS PUB 127 https://nvlpubs.nist.gov/nistpubs/Legacy/FIPS/fipspub127.pdf
- Unicode Consortium, publication dates https://www.unicode.org/history/publicationdates.html

- ARIANE 5 Flight 501 Failure: Report by the Inquiry Board, Paris, 19 July 1996; ESA press release https://www.esa.int/Newsroom/Press_Releases/Ariane_501_-_Presentation_of_Inquiry_Board_report
- NASA, Mars Climate Orbiter Mishap Investigation Board Phase I Report, 10 November 1999 https://llis.nasa.gov/llis_lib/pdf/1009464main1_0641-mr.pdf
- https://people.csail.mit.edu/matei/courses/2015/6.S897/readings/kafka.pdf
- JPMorgan Chase Form 8-K, 16 January 2013 https://www.sec.gov/Archives/edgar/data/19617/000001961713000077/0000019617-13-000077.txt
- W3C https://www.w3.org/TR/prov-o/
- https://arxiv.org/abs/1706.03762
- Slaughter and May, TSB Review, dated 31 October 2019, para 1.1 https://www.tsb.co.uk/content/dam/tsb-public/documents/media-centre/Slaughter-and-May-final-report.pdf
- Regulation (EU) 2016/679, Article 99(2) https://eur-lex.europa.eu/eli/reg/2016/679/oj
- [2019] EWHC 3408 (QB) https://www.judiciary.uk/judgments/bates-others-v-post-office
- Tweede Kamer, Kamerstuk 35 510, nr. 2 https://www.tweedekamer.nl/sites/default/files/atoms/files/20201217_eindverslag_parlementaire_ondervragingscommissie_kinderopvangtoeslag.pdf
- Report of the Royal Commission into the Robodebt Scheme, vol. 1, letter of transmittal https://robodebt.royalcommission.gov.au/publications/report
- Anthropic, "Introducing the Model Context Protocol" https://www.anthropic.com/news/model-context-protocol
- EUR-Lex https://eur-lex.europa.eu/eli/reg/2026/1744/oj

Appendix C. Failure Library

- UK Parliament, "The Great Fire of 1834" https://www.parliament.uk/about/living-heritage/building/palace/architecture/palacestructure/great-fire/
- Kellee Blake, "First in the Path of the Firemen: The Fate of the 1890 Population Census", Prologue 28(1), Spring 1996 https://www.archives.gov/publications/prologue/1996/spring/1890-census-1.html
- ARIANE 5 Flight 501 Failure: Report by the Inquiry Board; ESA press release 33-1996 https://www.esa.int/Newsroom/Press_Releases/Ariane_501_-_Presentation_of_Inquiry_Board_report
- NASA, Mars Climate Orbiter Mishap Investigation Board Phase I Report https://llis.nasa.gov/llis_lib/pdf/1009464main1_0641-mr.pdf; NASA press release 99-134 https://nssdc.gsfc.nasa.gov/planetary/text/mco_pr_19991110.txt
- Report of JPMorgan Chase & Co. Management Task Force Regarding 2012 CIO Losses, 16 January 2013 https://elischolar.library.yale.edu/ypfs-documents/454; Form 8-K https://www.sec.gov/Archives/edgar/data/19617/000001961713000077/0000019617-13-000077.txt
- PERI https://peri.umass.edu/publication/does-high-public-debt-consistently-stifle-economic-growth-a-critique-of-reinhart-and-rogoff/; https://econpapers.repec.org/RePEc:oup:cambje:v:38:y:2014:i:2:p:257-279
- Slaughter and May, TSB Review https://www.tsb.co.uk/content/dam/tsb-public/documents/media-centre/Slaughter-and-May-final-report.pdf; TSB https://www.tsb.co.uk/news-releases/slaughter-and-may/
- Bates and Others v Post Office Ltd, Judgment (No 6) "Horizon Issues", [2019] EWHC 3408 (QB) https://www.judiciary.uk/wp-content/uploads/2022/07/bates-v-post-office-judgment-1.pdf
- Prime Minister of Australia https://www.pm.gov.au/media/final-report-royal-commission-robodebt-scheme; Royal Commission https://robodebt.royalcommission.gov.au/
- Report of the Royal Commission into the Robodebt Scheme, vol. 1, p. xxiv https://robodebt.royalcommission.gov.au/publications/report
- Tweede Kamer https://www.tweedekamer.nl/nieuws/kamernieuws/eindverslag-onderzoek-kinderopvangtoeslag-overhandigd; https://www.tweedekamer.nl/sites/default/files/atoms/files/20201217_eindverslag_parlementaire_ondervragingscommissie_kinderopvangtoeslag.pdf
- Author's reading through the Ledger's eight loss codes

THE SERIES

Every Record Counts

Every Record Counts is a series about data: where each record was born, what it means, who can change it, what it costs and who reads it now, from the tally stick to the training set. This is its first book. Each book stands on its own.

Also by the author:

Every Record Counts	**The Model as Reader**: data for training, retrieval and agents
Every Record Counts	**From Warehouse to Lakehouse**: data on every platform, and the price of exit
Every Record Counts	**Data on Every Desk**: files, spreadsheets and the PC era
Every Record Counts	**What a Value Means**: types, encodings and formats
Every Pixel Matters	**From Bit to Pixel**: the whole stack, from the bit to the pixel
Every Pixel Matters	**From Code to Glass**: the rendering layer
Every Pixel Matters	**From Silicon to Screen**: who controls each layer, and what trust costs
Medallion	**From Bronze to Gold**: why data layers decide what AI can do

Juan Cardena

Enterprise AI & Agentic Systems Architect · blog.jcardena.com

Download the latest edition and the blank Ledger at blog.jcardena.com/en/ebooks/from-tally-to-token.

Every Record Counts · Book 1 · From Tally to Token.