Engineer

Data lineage

the path data takes through sources and transformations

What it is

Data lineage records how data reaches a destination through its sources and transformations. A revenue figure on a dashboard may come from an order dataset, a table of refunds and a rule for converting currencies. Lineage gives the reader a path from the figure back to those inputs. It can also show which reports depend on an input before its owner changes a column or delivery schedule. The scope ranges from a table-level diagram to a recorded run with the exact partitions and fields it read and wrote.

OpenLineage models a dataset, a named job that transforms or moves data, and a run, meaning a particular execution of that job. A run can report which datasets it read and produced, and attach details about the environment, schema or source versions. Those event records give a team a way to tell which job was responsible for a particular transition. A static view of the source code can also show possible dependencies. Both are useful when the view makes clear how it obtained its evidence.

Choose the granularity to fit the question. At dataset level an analyst can see that a report depends on two tables. At column level a reviewer can trace the origin of an email address or a currency field through selected expressions. At execution level an operator can see which version of a transformation processed which batch and where it left its output. To retrace a disputed number, the time window, partition and rule version may matter as much as the names of the tables.

Use the concept in a decision

Put “What is data lineage?” to work in a real decision. We can help relate the concept to the system you are building or buying. Numbers arriving where they are needed, on a schedule, without anyone copying a spreadsheet at the end of the month.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. One infrastructure plan for 14 departments — 14 research departments on one infrastructure plan

One infrastructure plan for 14 departments Get in touch →

Why it matters

Lineage makes a failure search smaller. When a daily figure suddenly falls, the investigator can check whether the collection job brought in fewer records, a join dropped a group, or a report selected the wrong time window. A path narrows the first place to look. The next step is to confirm each suspected break with observation, since a connected diagram alone cannot establish that the data was correct or even current.

Knowing the downstream users also helps plan a change. A developer renaming a status field can learn which reconciliation job and reports would be affected. The owners can agree who will test the new name, where a temporary compatibility field is needed and how long to keep it. Lineage enables this conversation while the change is still reversible. Let people see enough to understand the risk while protecting data contents and permissions.

Well maintained provenance supports confidence when different people use the same label for different measures. A team can compare the dataset and transformation definitions behind two numbers instead of treating the newer report or the larger figure as authoritative. Lineage makes the question precise enough to discuss with the owner. Correctness still needs a definition and a deliberate check of the result against that definition.

What it looks like in practice

Engineer ApS describes operating PostgreSQL, MS SQL and BigQuery warehouses. Work across different warehouse engines makes source boundaries and scheduled processing relevant. The example below is a separate, fictional exercise for recording evidence about a reporting flow.

Imagine a daily report that counts orders ready to ship. Orders arrive from the storefront. A different feed delivers dispatch statuses. The report’s query groups by shipping date and chooses the newest status event for each order reference. Create a lineage entry for each input and the output, and name the job that joins them. Record the job invocation identifier and its start and end times. Preserve

the input partitions and the deployed revision that selects the newest status. If a run retries, record the final accepted output and keep the failed attempt visible as its own execution.

Now trace an order that appears on a Tuesday report. Use the report date and order reference through approved queries. Follow the reported run to the chosen dispatch record. Check the source event time against the shipping cutoff and verify how tied events are resolved. If the status feed was late, make that visible in the description of the report and in the incident timeline. When the cutoff changes, use the stored dependency information to ask the affected owners to verify both an ordinary day and a late delivery.

A small implementation can begin with one critical report and a hand-reviewed set of edges. Ask an analyst to trace a row and an operator to trace a failed run. Note the point where either person reaches a guess. Add the missing identifier, version or dataset in that order. Automatic collection has more value once the useful evidence and the limits of collection are explicit.

Keep the status event time and the time the input became available. Record the publication cutoff too, so a pre-cutoff event that arrived late can be explained.

Where it goes wrong

A diagram can look comprehensive when it has silently missed a manual export, a stored procedure or a spreadsheet someone downloads and rewrites. Count what has been instrumented and state how the remaining path is documented. Give manually recorded edges a source and a date. A connected process that has changed since review needs fresh evidence before someone relies on it.

Names that change without a stable namespace can make two datasets appear to be the same. Give datasets identifiers that include their environment and storage context. An archive and a production table may have identical short names. Group them only when the relationship is known. Keep enough history to explain which identifiers were used before a move.

Finally, protect the metadata itself. Table names, dependency patterns or operational notes can reveal information about sensitive work. Keep query strings and example records out of general-purpose event fields when readers have different permissions. Test important edges by making one known safe

change and checking the output and the path recorded for that run. A new input value can change the result while the dependency stays the same. Expect a different edge when the job actually changes its source. That supports a practical claim about the path you actually observed.

Further reading

- OpenLineage object model
- W3C PROV-O provenance ontology
- Google Cloud documentation on data lineage

Glossary

Choose for your actual constraints

Bring a representative dataset, the questions it must answer and the current bottleneck. We can agree how to measure quality, throughput and recovery.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is data lineage?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/data-lineage/