

SELF-STUDY HANDBOOK

Data Analytics & Visualization with Qlik and Tableau

CommunityForce Inc.

Modules in this edition:

- **QT-01** BI Foundations: From Raw Data to Decisions
- **QT-02** Dimensional Modelling: Facts, Dimensions, Star & Snowflake
- **QT-03** Your First Qlik Cloud App
- **QT-04** Load Script Essentials & the Data Load Editor
- **QT-05** Qlik Architecture & the QVD Layer
- **QT-06** Synthetic Keys, Circular References & Link Tables

Screens are from recorded training sessions. Names, avatars and account details are replaced or removed; replaced values are marked DEMO DATA in the videos.

BI Foundations: From Raw Data to Decisions

Video: 01_QT-01_BI_Foundations.mp4 · 4:59 · Stage: Kick-off & Requirements · Audience: Anyone new to business intelligence; no tool experience needed

Learning objectives

- Tell the three forms of data apart
- Describe the five stages of BI
- Place Qlik, Tableau and Power BI in the BI landscape

Three forms of data

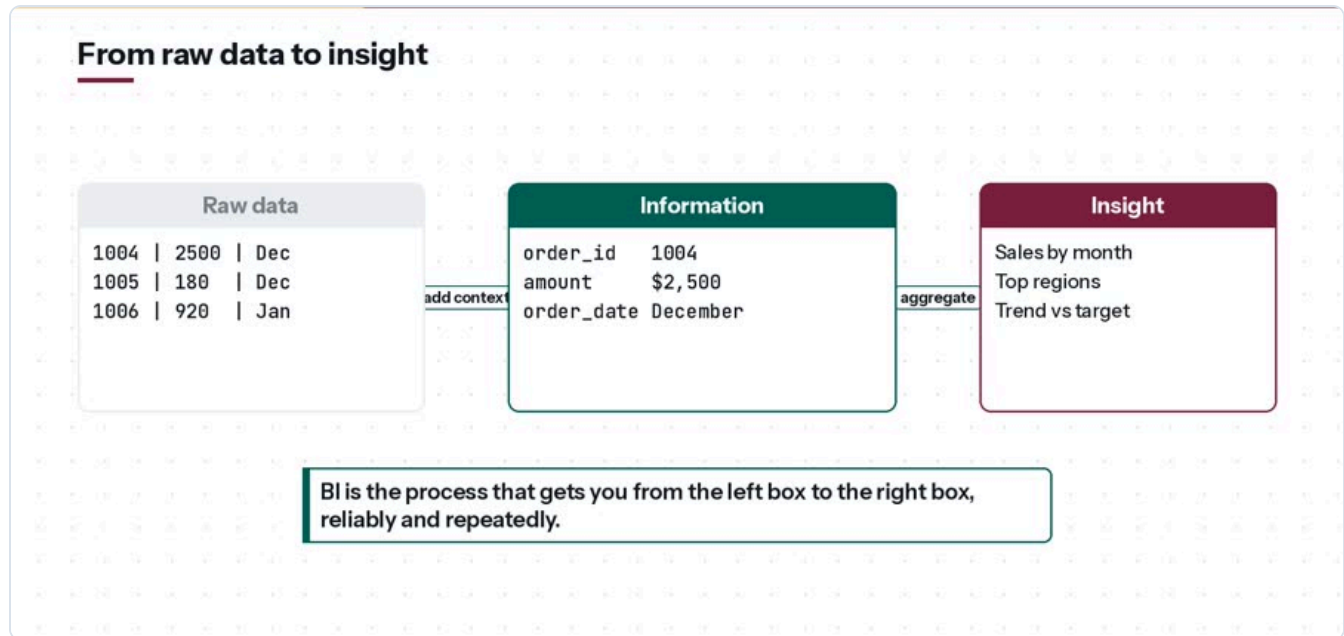
Three forms of data

Data is information about something: a number, a word, a date, a click.

1 Structured	2 Semi-structured	3 Unstructured
Rows and columns with a fixed schema.	Tagged, but no fixed table.	No predefined model.
Example: an orders table in a database (order_id, amount, order_date).	Examples: JSON, XML, log files.	Examples: images, video, free text, documents.
Easy to query and aggregate.	Transform it into rows and columns before you query it.	Needs extra processing, often AI, before analysis.

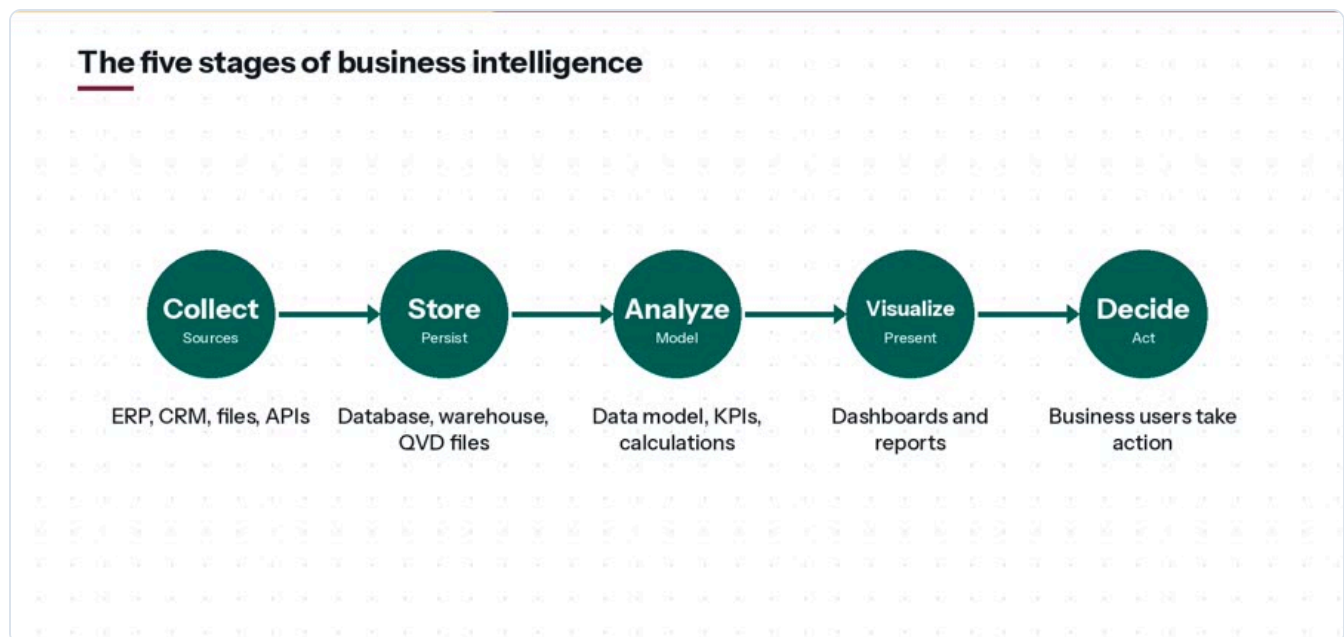
Data is simply information about something: a number, a word, a date, a click. It comes in three forms. Structured data lives in rows and columns, like an orders table in a database, so it's easy to query. Semi-structured data, such as JSON, XML, or log files, has tags but no fixed table, so we transform it before we query it. And unstructured data, like images, video, and free text, is most of the world's data, and it needs extra processing before it can be analyzed.

From raw data to insight



Here's why context matters. On its own, a raw record like one thousand and four, twenty-five hundred, December, means very little. Add field names and it becomes information: order one thousand and four, for twenty-five hundred dollars, placed in December. Summarize thousands of those records and you get an insight a manager can act on.

Business intelligence is the process of turning raw data into meani...



Business intelligence is the process of turning raw data into meaningful insights that support faster, better decisions. Every BI project follows five stages. Collect the data from source systems. Store it in a database, a warehouse, or Qlik's own QVD files. Analyze it by modelling and calculating key measures. Visualize it in dashboards. And finally, people decide and act.

How BI evolved

How BI evolved		
Era	Typical tools	What it was like
1990s	Spreadsheets, static reports	Manual calculations, slow, no interaction
2000s	Central data warehouses	Single source of truth, but new reports took weeks
2000s–2010s	Enterprise reporting suites	Powerful, but built and owned by IT
2010s	Qlik Sense, Tableau, Power BI	Self-service, interactive, increasingly cloud
Today	AI-assisted analytics	Ask in plain language; still needs a good data model

BI tools have evolved quickly. In the nineteen nineties, teams relied on spreadsheets and static reports, with lots of manual work. In the two thousands, central data warehouses arrived, but a single new report could take weeks. Enterprise reporting suites followed, mostly built and owned by IT. In the twenty tens, self-service tools like Qlik Sense, Tableau, and Power BI let analysts explore data interactively, increasingly in the cloud. And today, AI assistants can suggest charts and answer questions in plain language, but they still depend on a well-built data model.

Where the leading tools fit

Where the leading tools fit			
Tool	Engine & approach	Data preparation	Known for
Qlik Sense	Associative, in-memory engine	Built-in load script (ETL), QVD files	Exploring relationships across many sources
Tableau	Visual query (drag fields to shelves); live or extract	Tableau Prep, data source page	Visual analysis and storytelling
Power BI	In-memory import or DirectQuery	Power Query (M) and DAX	Microsoft 365 and Azure integration
Licensing models differ by vendor and change often. Always check current vendor pricing.			

So where do today's leading tools fit? Qlik Sense is built on an associative, in-memory engine, and its load script handles extract, transform, and load inside the product. Tableau is built around visual analysis: you drag fields onto shelves, and it can query data live or from an extract. Power BI is tightly integrated with Microsoft three sixty-five and Azure, and uses Power Query and the DAX language. All three are capable. The right choice depends on your data, your skills, and your existing platforms.

The associative experience (illustration)

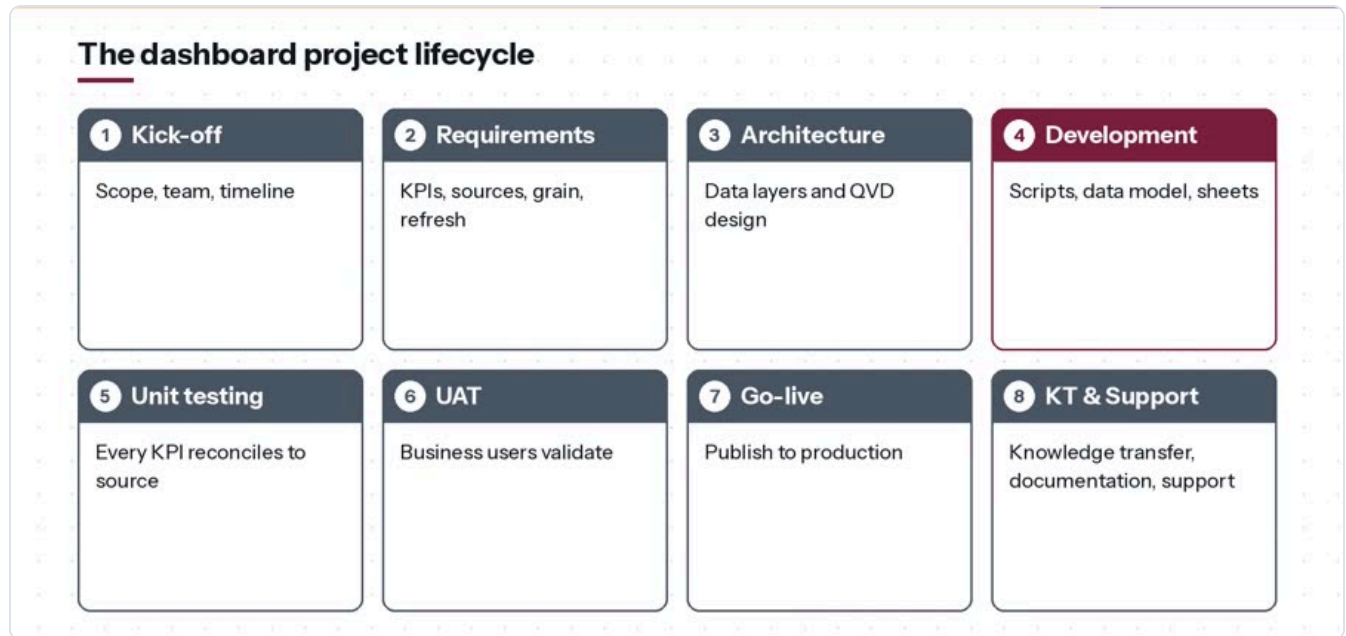
The associative experience (illustration)

Region	Product category	Sales rep
East	Furniture	Rep A
North	Office supplies	Rep B
South	Technology	Rep C
West	Toys	Rep D

☒ Selected ☐ Possible (related) ☐ Alternative (same field) ☐ Excluded (unrelated)

Qlik's associative engine is worth seeing early, because it shapes how you'll work. When you select a value, Qlik instantly shows you what's related and what isn't. Select the West region, and it's highlighted in green. Related values stay white, and unrelated values turn gray. Seeing what's excluded is often as revealing as seeing what's included.

The dashboard project lifecycle



Dashboards are delivered as projects, and this course follows the same eight-stage lifecycle. Kick-off agrees scope and timelines. Requirements captures the measures, sources, and refresh needs. Architecture designs the data layers. Development builds the scripts, model, and sheets. Unit testing checks every measure against its source. User acceptance testing lets the business validate. Then go-live, and finally handover and support.

Identify your data: five questions

Identify your data: five questions

Question	Example answer: revenue dashboard
1 What decision will it support?	Which regions need action to hit target
2 Which KPI measures it?	Monthly revenue vs target, by region
3 What grain is needed?	One row per order line
4 Which dimensions filter it?	Date, region, product, customer, channel
5 Where does the data live?	ERP sales tables · daily 6 AM refresh · finance team, read-only

Example adapted from the trainer's data inventory worksheet.

During requirements, identify your data by working backwards from the decision. Ask five questions. What decision will this support? What measure, or KPI, tracks it? What grain is needed, for example one row per order line? Which dimensions will people filter by? And where does the data live, who owns it, how often does it refresh, and who may see it? Here's an example answer for a revenue dashboard.

Key takeaways

- Three forms: structured, semi-structured, unstructured
- Collect, store, analyze, visualize, decide
- Qlik, Tableau and Power BI take different approaches
- Start every dashboard with the five questions

Practice

1. Pick a report you use at work. Write down the decision it supports, its main KPI, its grain, the dimensions you filter by, and where the data lives.
2. Classify five data sources you know (for example an Excel export, an email inbox, a database table, a JSON API response, scanned invoices) as structured, semi-structured or unstructured.

Check your understanding

1. Which stage of BI comes directly after Analyze?
2. A JSON file is an example of which form of data?
3. In Qlik's associative experience, what colour shows values unrelated to your selection?
4. Which lifecycle stage checks every KPI against its source?

Answers at the back of this handbook.

Dimensional Modelling: Facts, Dimensions, Star & Snowflake

Video: 02_QT-02_Dimensional_Modelling.mp4 · 3:57 · Stage: Architecture · Audience: Analysts and new dashboard developers; basic familiarity with tables

Learning objectives

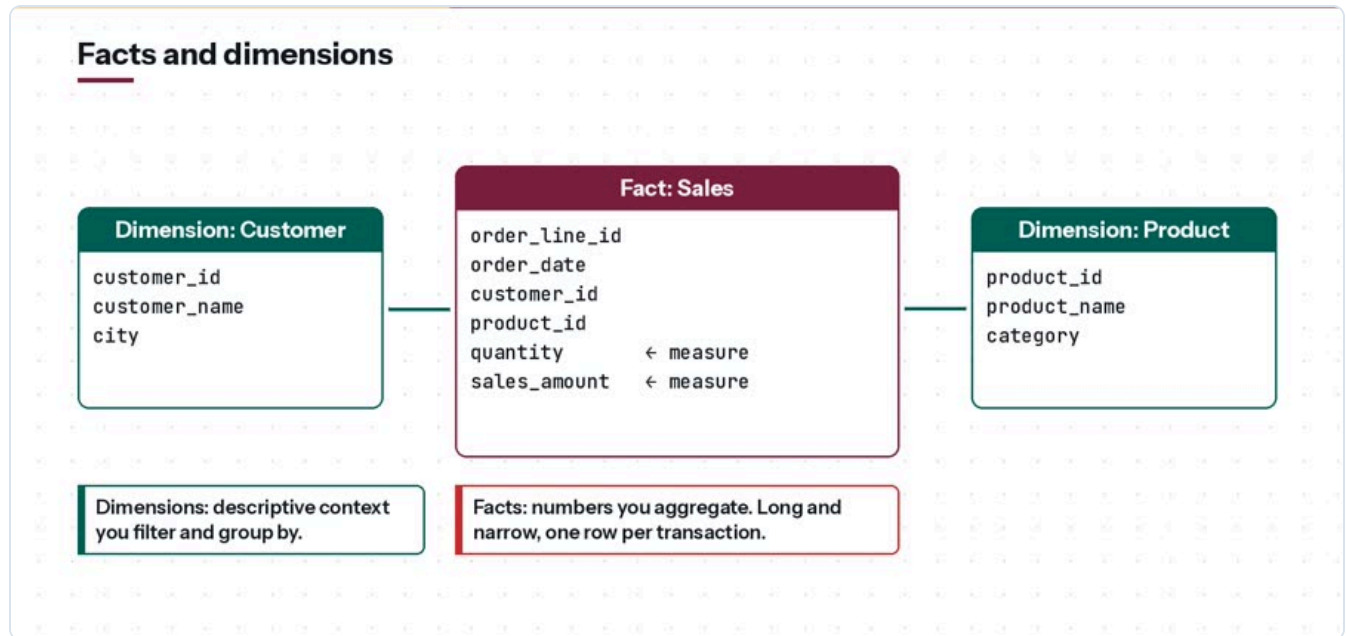
- Tell fact tables from dimension tables
- Build a star schema, and know when to snowflake
- Recognise a good model from a poor one

Good model vs poor model

Good model vs poor model		
	Good model	Poor model
Reload	Runs in seconds	Takes minutes
Totals	Reconcile with the source	Double counting
Keys	Clean, single-field keys	Synthetic keys and loops
Change	Add a field safely	Risky rework, retest, re-run UAT

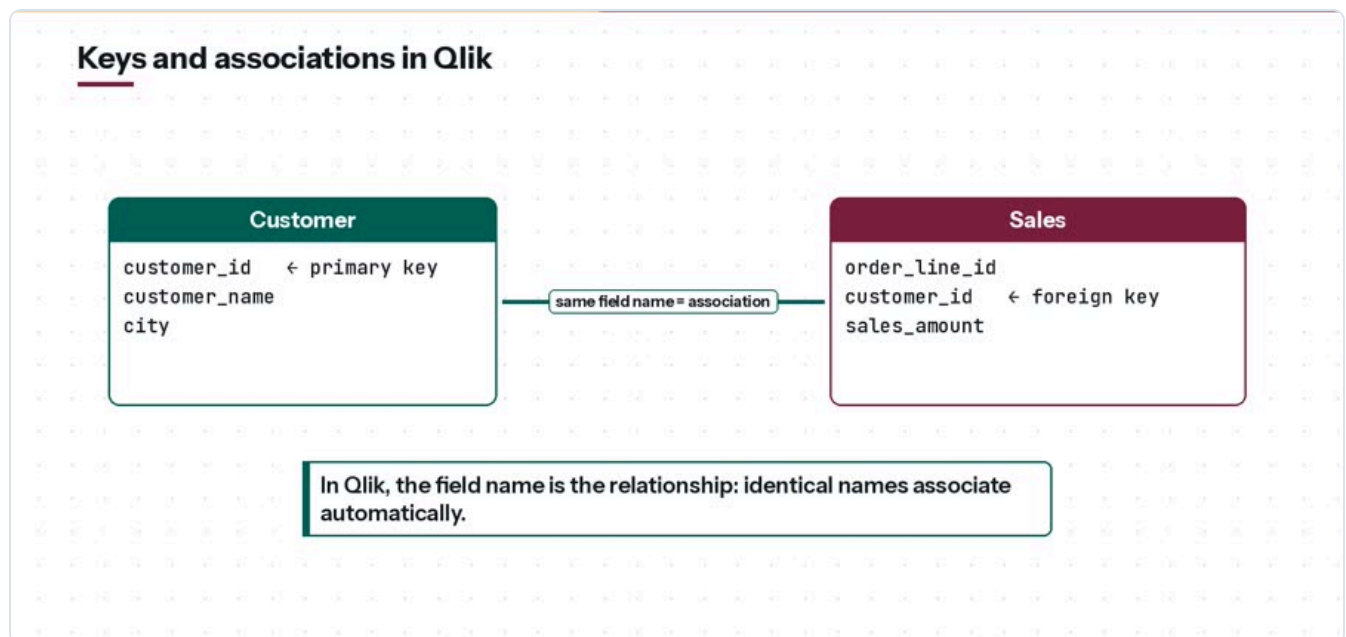
Why does modelling matter so much? Compare a good model with a poor one. A good model reloads in seconds; a poor one can take minutes. In a good model, totals reconcile with the source; in a poor one, rows get double counted. A good model uses clean, single-field keys; a poor one creates synthetic keys and loops. And in a good model you can safely add a field, while in a poor one every change means risky rework, retesting, and another round of user acceptance.

Facts and dimensions



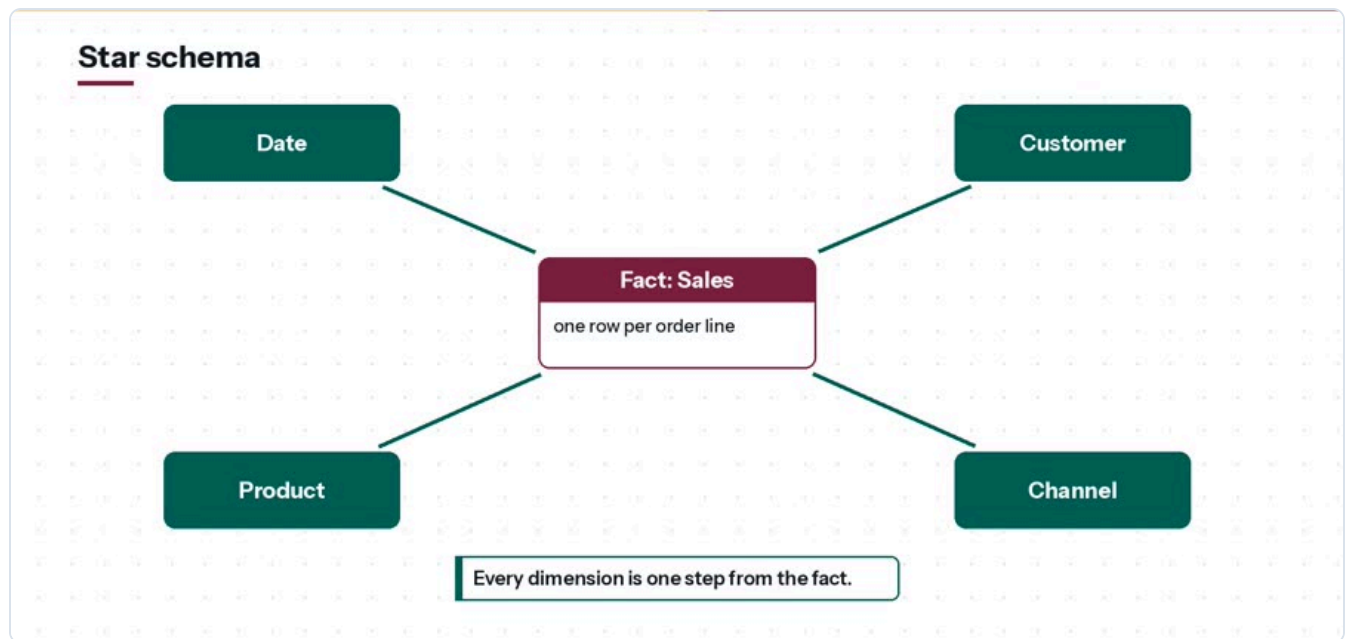
Think about a shop receipt. The numbers you'll add up, like quantity and sales amount, belong in a fact table. Fact tables are long and narrow, with one row per transaction. The descriptive context, like who the customer is and what the product is called, belongs in dimension tables. Dimensions are what you filter and group by.

Keys and associations in Qlik



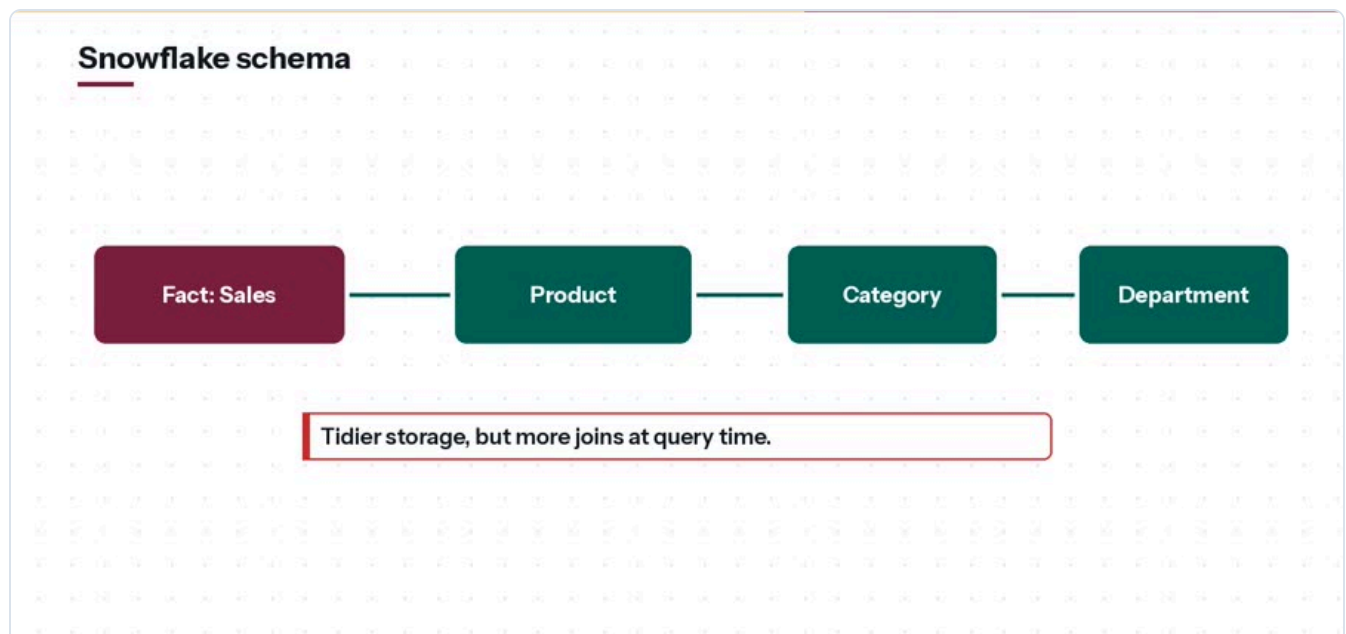
Tables connect through keys. Customer ID is the primary key of the customer table, and a foreign key in the sales table. Here's something special about Qlik: it associates tables automatically whenever they share a field with exactly the same name. There's no join to draw. The field name is the relationship, which is why naming fields carefully matters so much.

Star schema



The preferred design is the star schema. One central fact table is surrounded by dimension tables, each joined directly to the fact. Every dimension is just one step away, so there are fewer joins, the model is easy to read, and calculations stay fast.

Snowflake schema



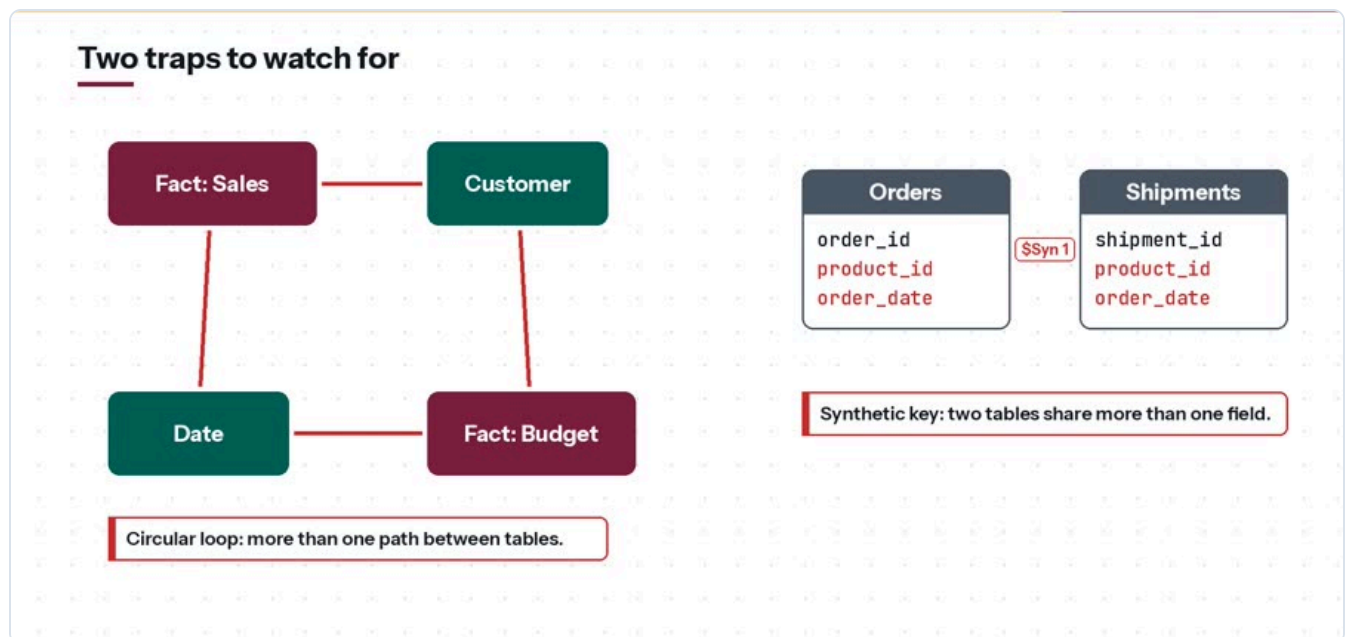
A snowflake schema normalizes the dimensions further. Product splits out into category, and category into department. Storage is tidier, but every calculation now has to travel through more joins, which can slow things down.

Star vs snowflake

Star vs snowflake		
	Star schema	Snowflake schema
Structure	Denormalized dimensions	Normalized dimensions
Speed	Fewer joins, faster	More joins, can be slower
Storage	Some repeated values	Tidier, less repetition
In Qlik	Preferred	Flatten lookups in the load script

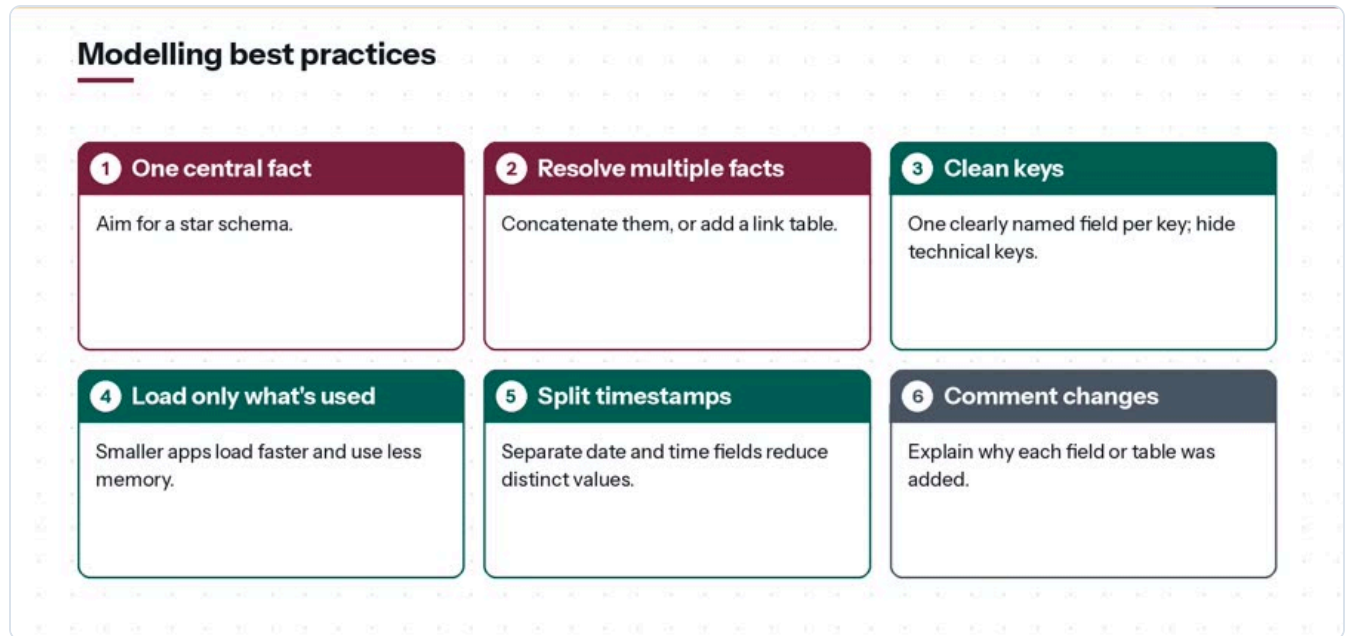
Here's the comparison side by side. A star schema has fewer joins and faster queries, at the cost of some repeated values. A snowflake saves storage but adds joins and complexity. In Qlik, prefer the star. If your source is snowflaked, flatten those small lookup tables into the dimension in your load script.

Two traps to watch for



Two traps to watch for. When two fact tables share the same dimensions, the model can form a circular loop, and results become unpredictable. And when two tables share more than one field, Qlik silently builds a synthetic key to join them. Both are signs of a modelling mistake, and module QT-06 shows how to fix them.

Modelling best practices



To finish, here are the trainer's best practices. Build around one central fact table. Resolve multiple facts by concatenating them or adding a link table. Keep keys to a single, clearly named field, and hide technical keys from users. Load only the fields you actually use. Split timestamps into separate date and time fields. And comment every change in your script.

Key takeaways

- Facts = numbers; dimensions = context
- Identical field names create associations
- Prefer a star schema
- Watch for circular loops and synthetic keys

Practice

1. Sketch a star schema for a retail shop: one sales fact table and at least four dimensions. Mark the key field on every link.
2. Take the snowflake Product → Category → Department and describe how you would flatten it into one Product dimension.

Check your understanding

1. Quantity and sales amount belong in which type of table?
2. How does Qlik decide which tables are associated?
3. Which schema has fewer joins?
4. What does Qlik build when two tables share more than one field?

Answers at the back of this handbook.

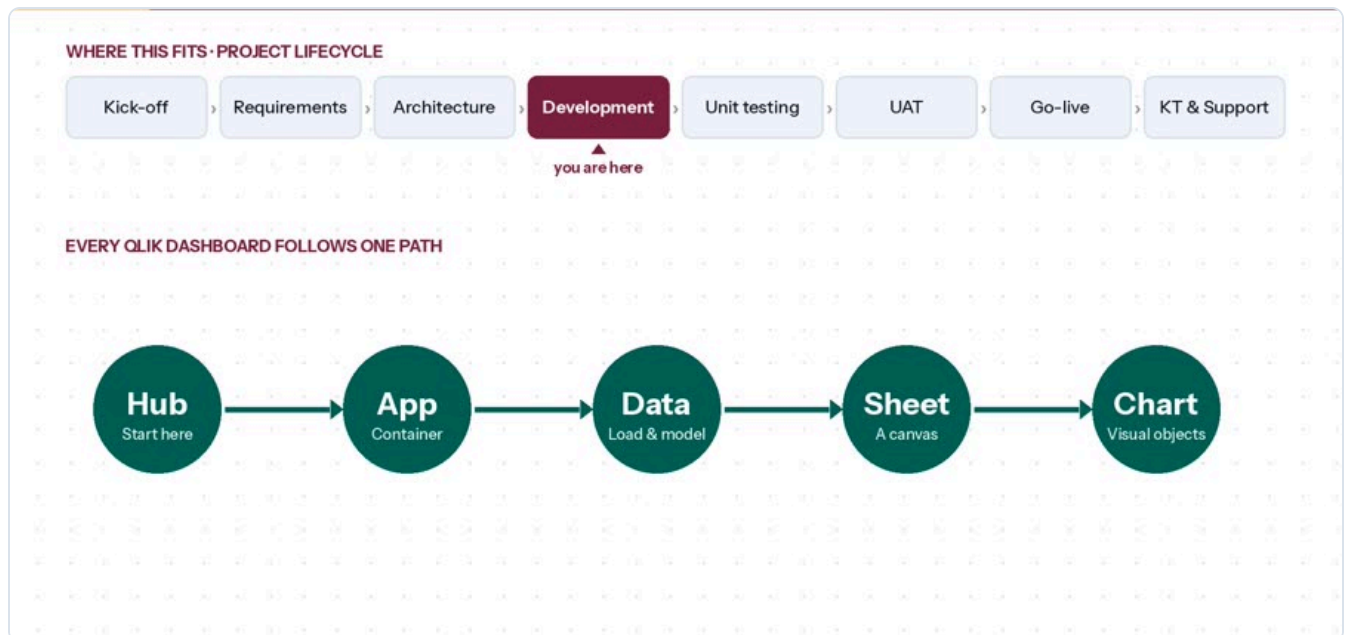
Your First Qlik Cloud App

Video: 03_QT-03_Your_First_Qlik_Cloud_App.mp4 · 4:45 · Stage: Development · Audience: New analysts and dashboard developers; no prior Qlik experience

Learning objectives

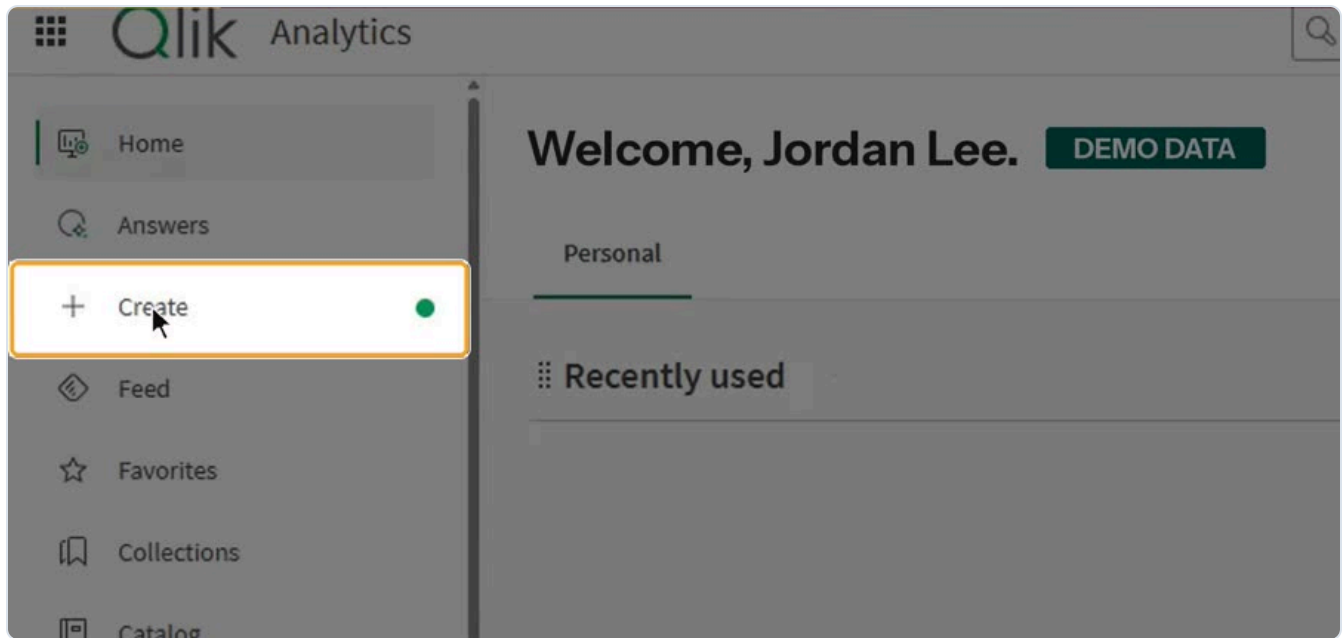
- Create an app in a space
- Load a data file with the Data manager
- Build your first chart

On a dashboard project, this work happens in the Development stage



On a dashboard project, this work happens in the Development stage. Every Qlik dashboard follows the same path. You start in the Hub. You create an App. You load Data into it. Then you add a Sheet, and place Charts on that sheet.

This is the Hub, the home page of Qlik Cloud Analytics



This is the Hub, the home page of Qlik Cloud Analytics. The left menu takes you to your content, the catalog, and every creation tool. Recently used items appear in the center. To start a new app, choose Create.

Pick Application

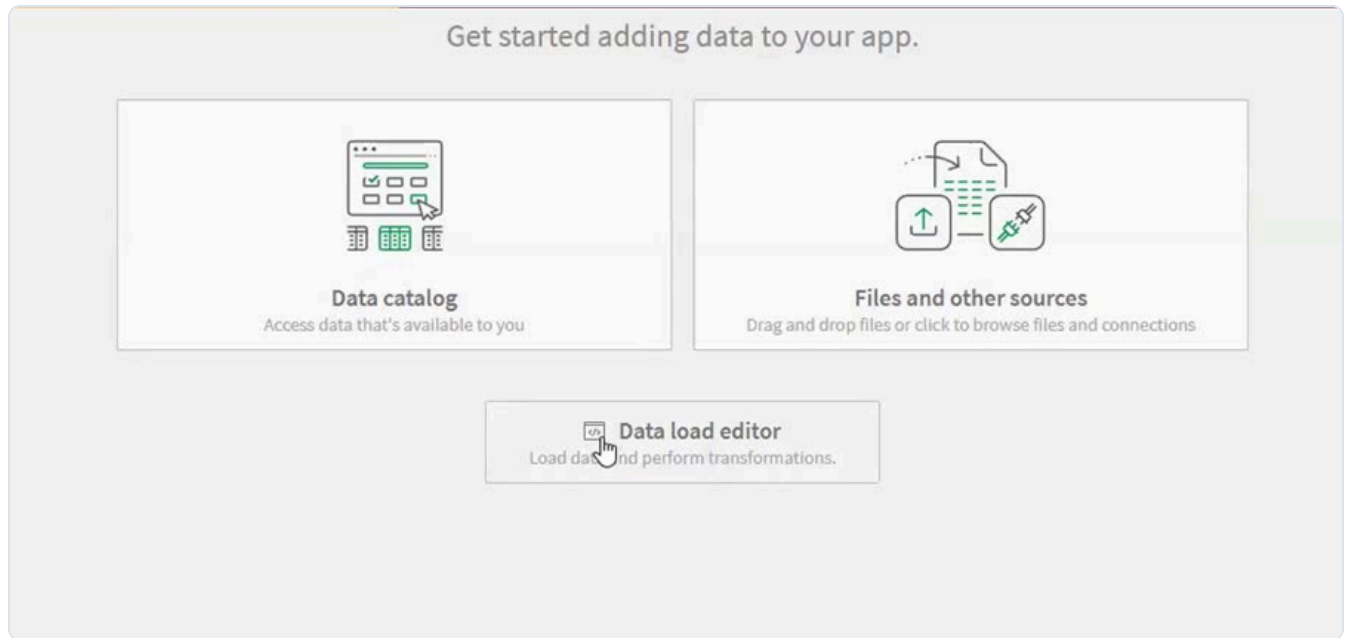
The screenshot shows the 'Create a new application' dialog box. The background is a blurred view of the Qlik Sense interface with various tiles like 'ML experiment', 'Data flow', and 'Assistant'. The dialog box is white with a title bar that says 'Create a new application' and a close button. It has the following fields and elements:

- Name:** A text input field containing 'Qlik App'. A red circle with the number '1' is next to it.
- Space:** A dropdown menu showing 'Personal'. A red circle with the number '2' is next to it.
- Description:** A large text area for optional description.
- Tags:** A search input field containing 'Search'. A red circle with the number '3' is next to it. Below the input, it says 'No tags found.'
- Open application:** A checkbox that is checked.
- Buttons:** 'Cancel' and 'Create' buttons at the bottom right. A mouse cursor is clicking the 'Create' button.

Pick Application. Give the app a clear name. Next, choose a space. A personal space is private to you, while a shared space lets teammates see and co-develop the app. Add an optional description and tags, which make the app easier to find. Then select Create.

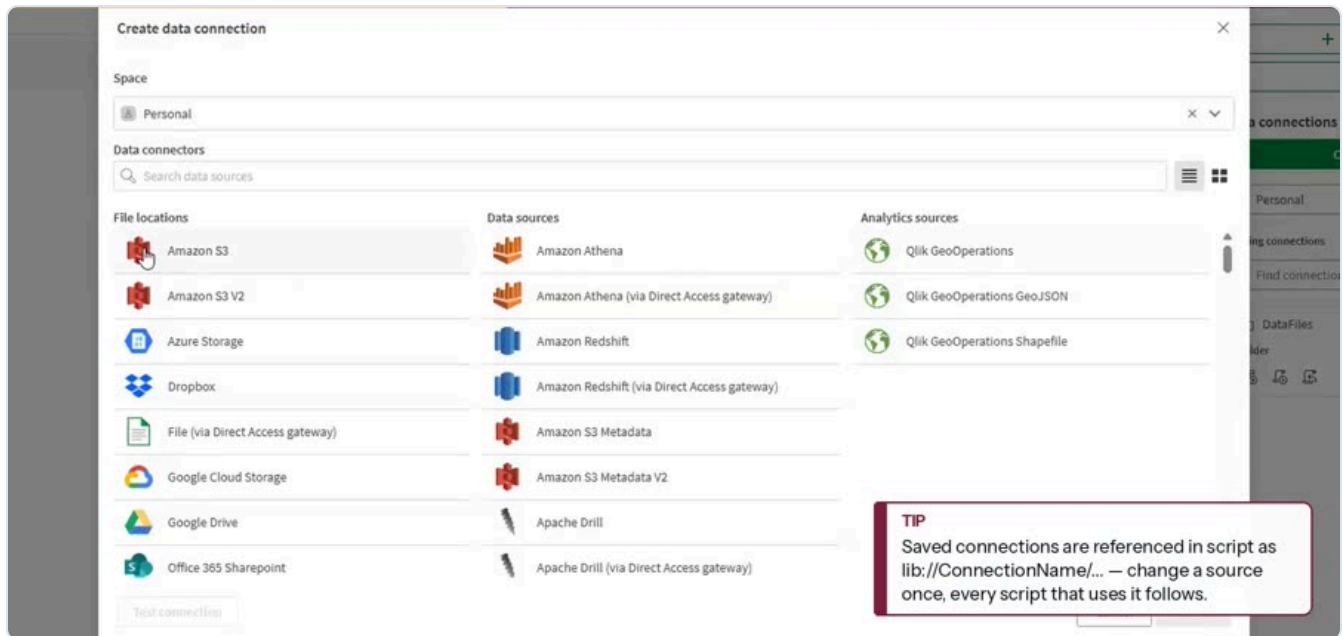
Tip Personal space = only you. Shared or managed space = your team can find and work on the app.

A new app opens empty, with three ways to add data



A new app opens empty, with three ways to add data. The Data catalog lists data already shared in your tenant. Files and other sources lets you drag in a file or connect to a system. And the Data load editor is where you write a load script by hand.

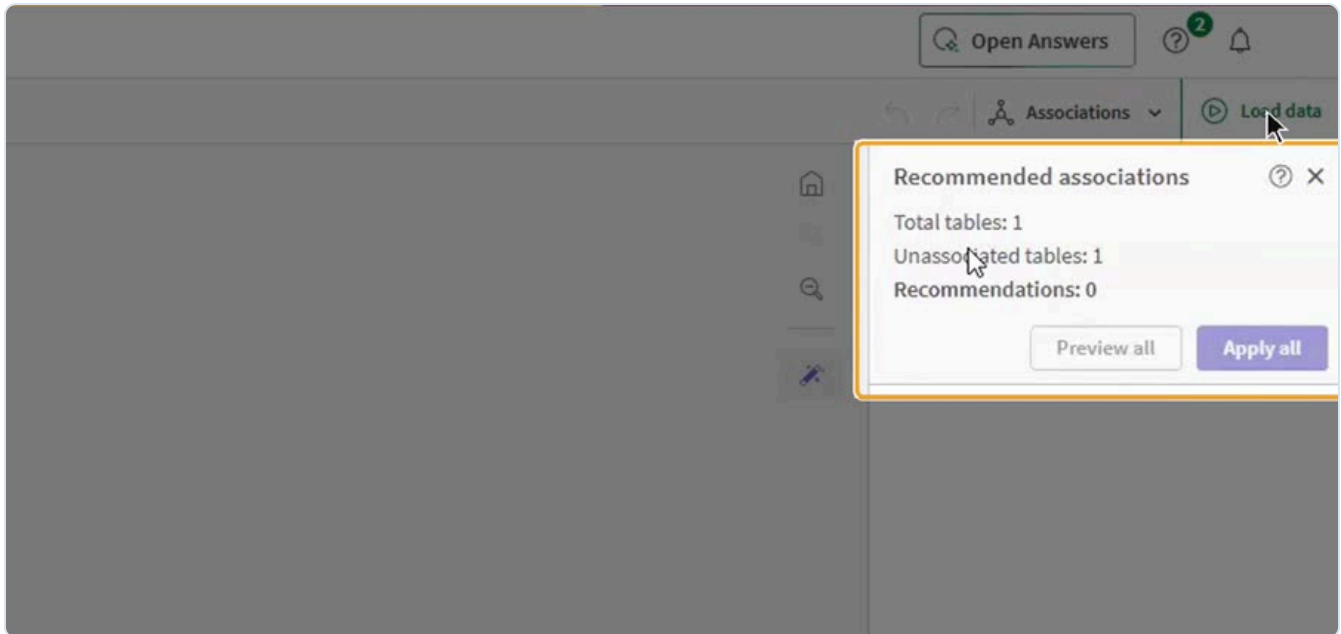
In the load editor, the first lines of script set default number an...



In the load editor, the first lines of script set default number and date formats. On the right, Create new connection shows Qlik's built-in connectors: file locations like Amazon S3, Azure Storage, and SharePoint, plus databases and other data sources. Every connection you save becomes a reusable library path.

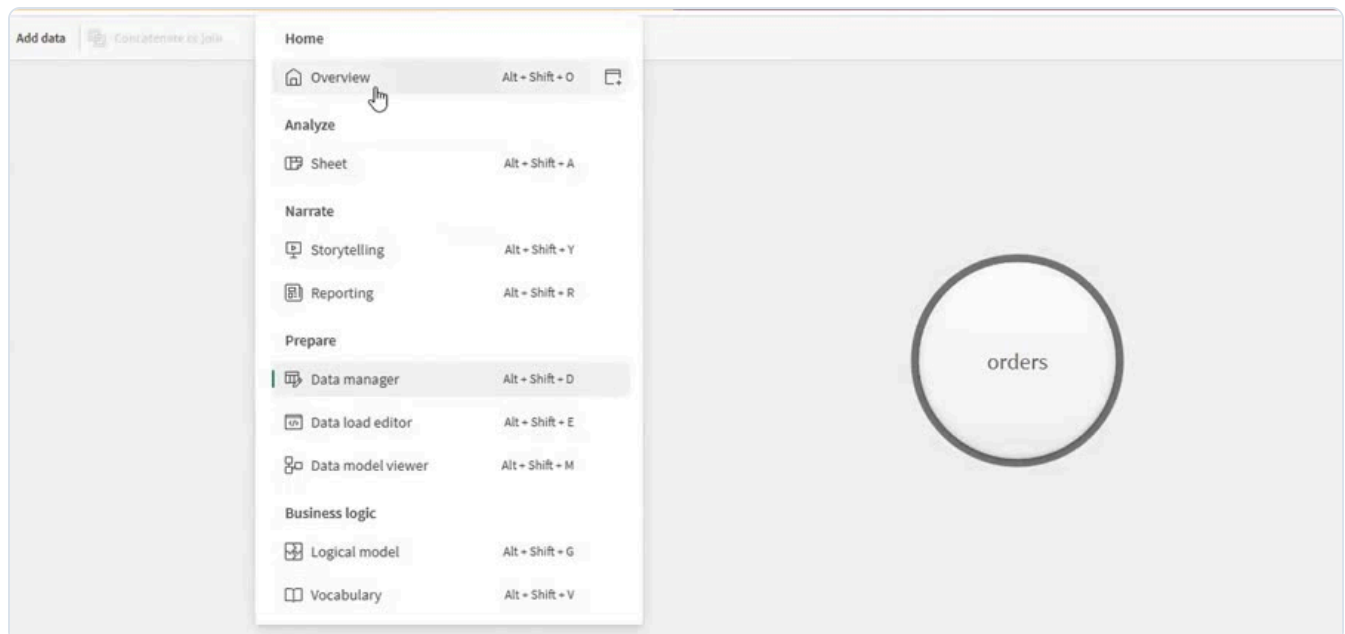
Tip Saved connections are referenced in script as `lib://ConnectionName/...` — change a source once, every script that uses it follows.

For a quick start, the trainer used the Data manager instead and ad...



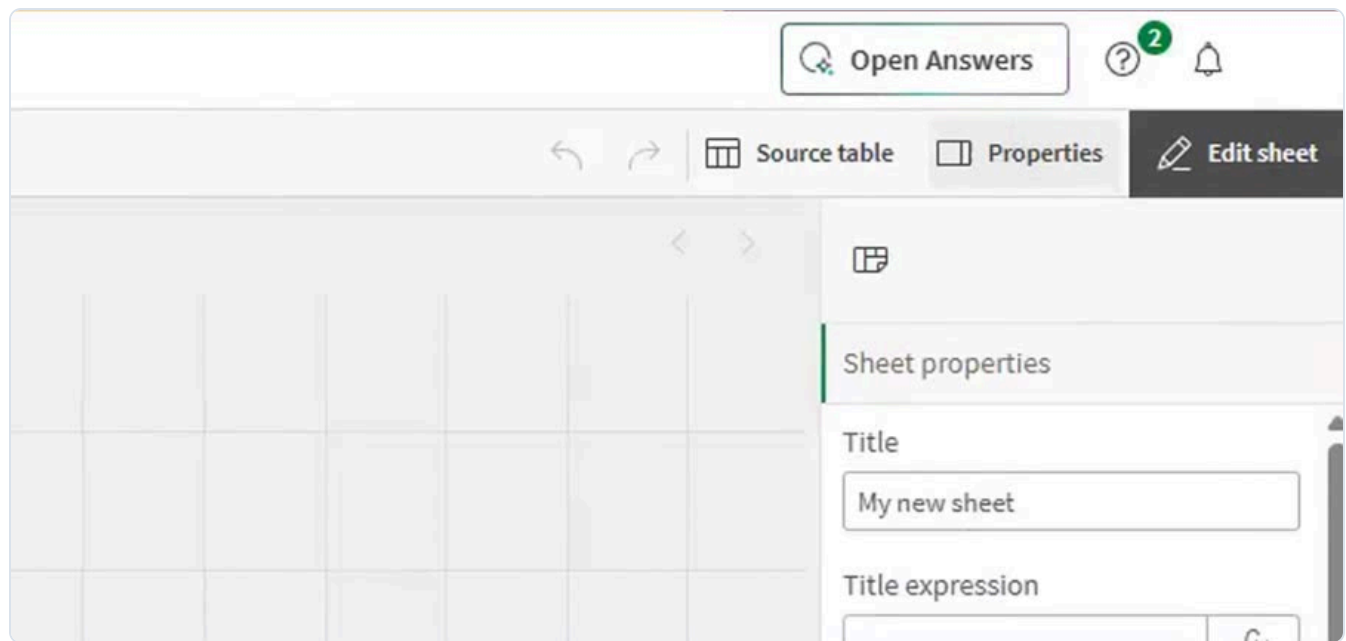
For a quick start, the trainer used the Data manager instead and added a file called orders.csv. Select the table to preview its fields: order ID, created at, items purchased, price, and more. With only one table there's nothing to associate yet, so select Load data.

The app's navigation menu groups every view



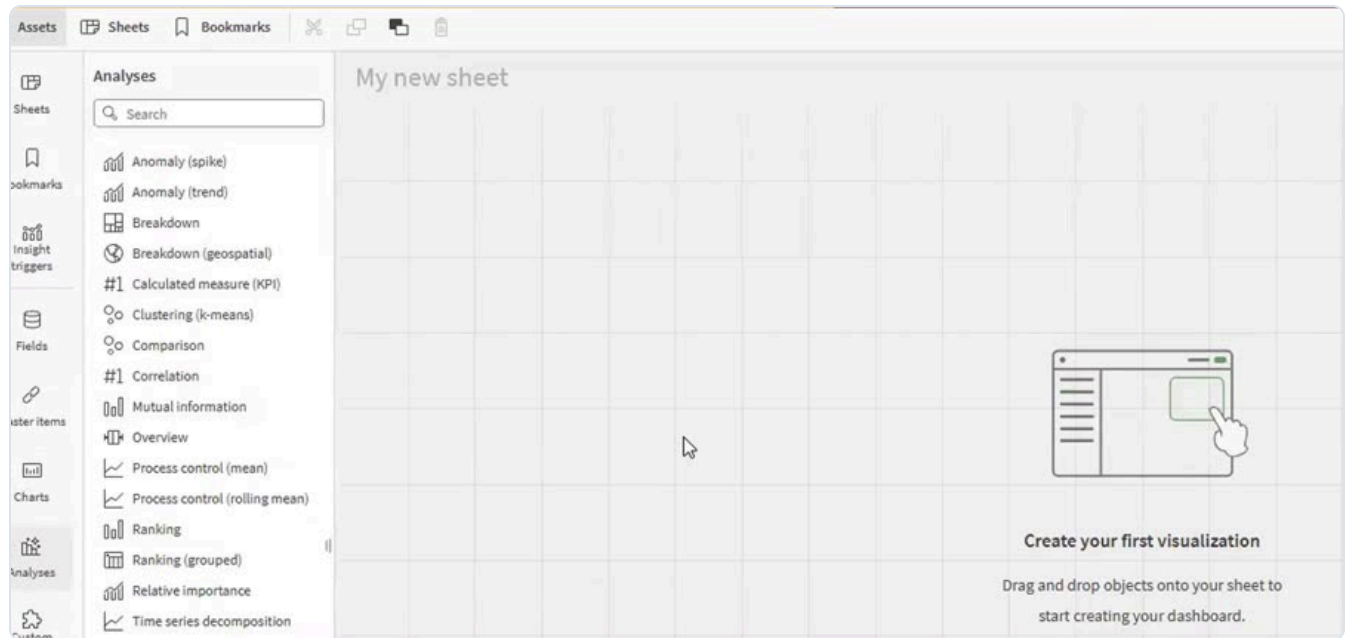
The app's navigation menu groups every view. Analyze holds your sheets. Narrate covers storytelling and reporting. Prepare holds the Data manager, Data load editor, and Data model viewer.

Once data is loaded, sheets unlock



Once data is loaded, sheets unlock. Qlik creates My new sheet for you. It's empty, so select Edit sheet in the top right to start building.

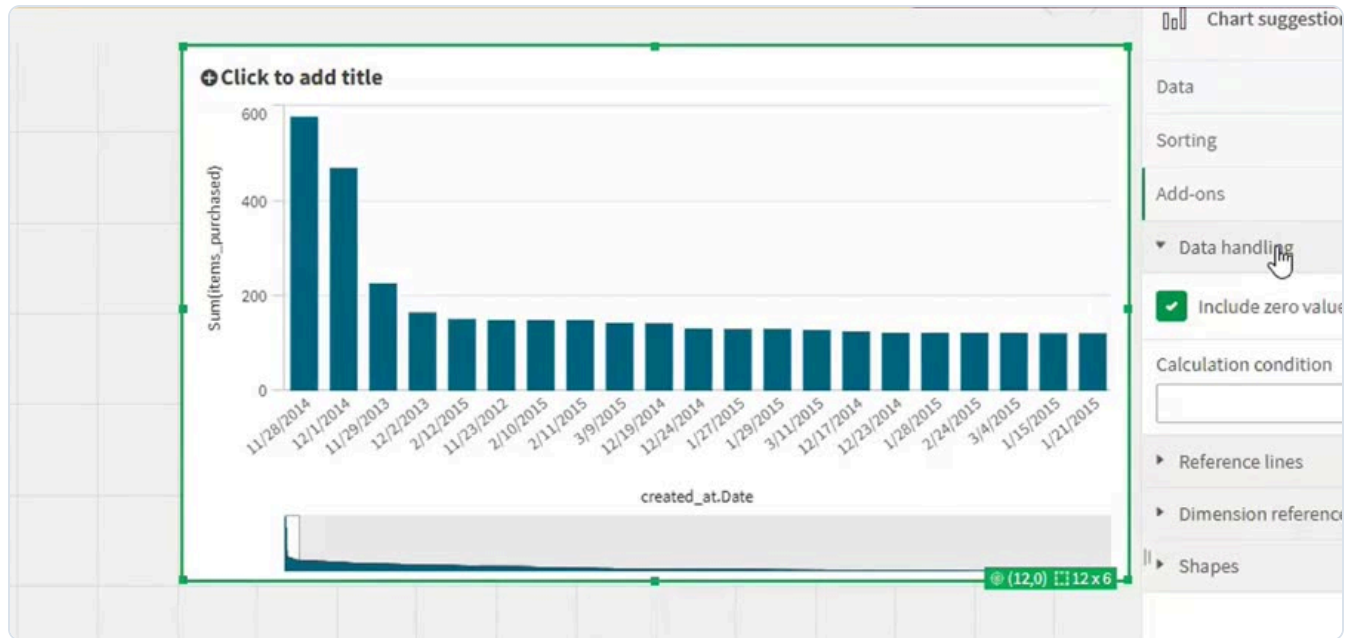
Edit mode opens the assets panel on the left



Edit mode opens the assets panel on the left. Fields lists every field you loaded, and you can filter by table. Master items store dimensions and measures you define once and reuse across the whole app. Charts holds the visualization library. Analyses offers ready-made patterns such as trend over time and ranking.

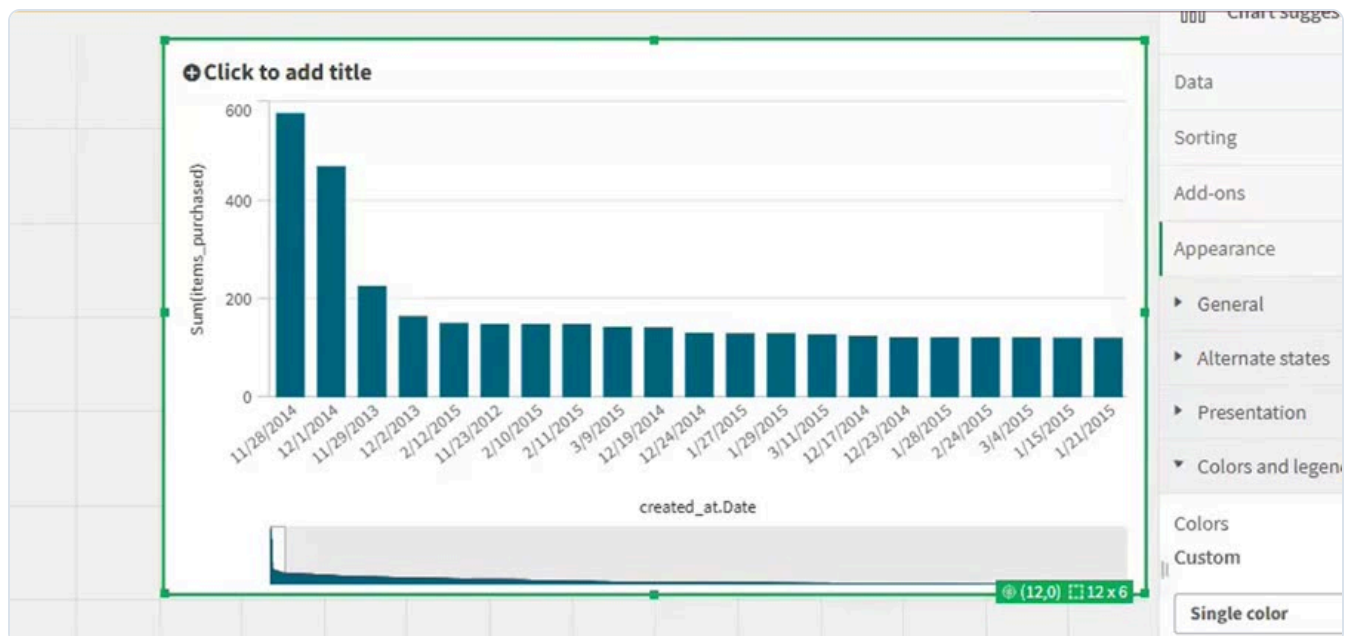
Tip Define a measure once as a master item. Change it there and every chart that uses it updates.

Drag a Bar chart onto the sheet



Drag a Bar chart onto the sheet. The chart asks for two things: a dimension for the bars, and a measure for their length. Add the dimension `created_at.Date`. Then add the measure: `Sum(items_purchased)`. The chart draws instantly.

With the chart selected, the properties panel on the right has four...



With the chart selected, the properties panel on the right has four sections. Data holds your dimensions and measures. Sorting controls the order. Notice the bars here are ranked by value, so the dates are out of sequence; Sorting fixes that. Add-ons add reference lines and data handling. And Appearance changes titles, colors, axes, and tooltips.

Behind the scenes, the Data manager wrote a load script for you

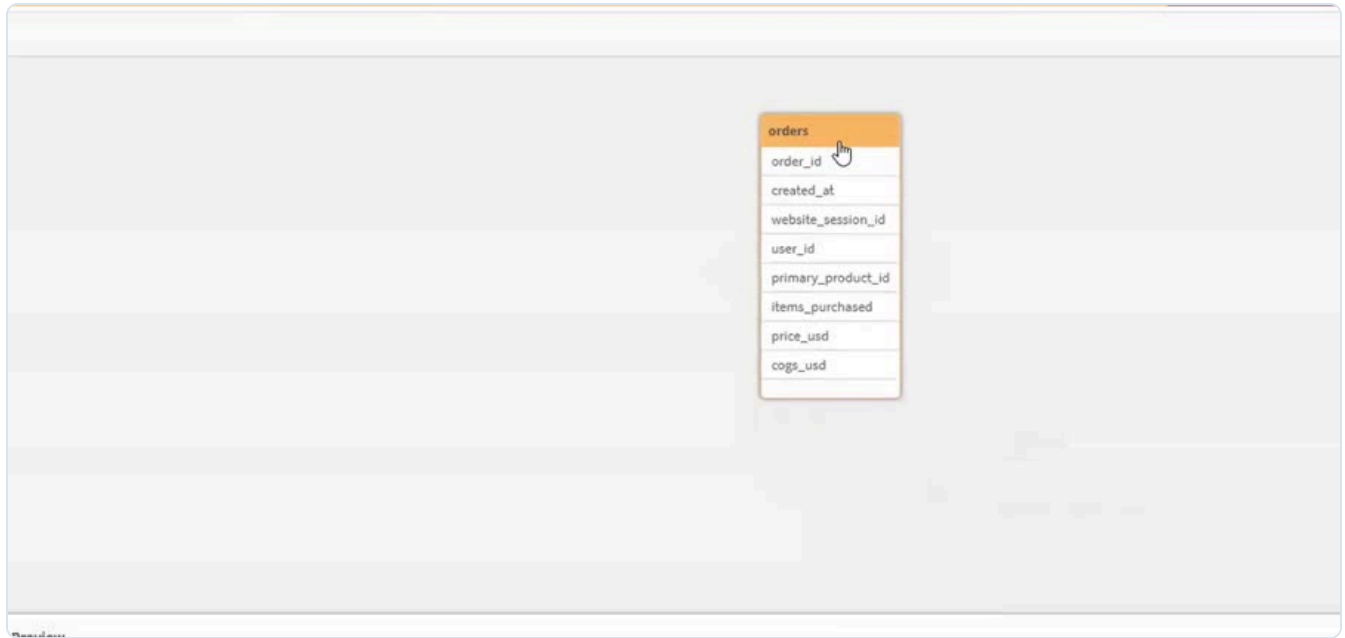
```
22 Unqualify *;  
23 |  
24 [orders]:  
25 LOAD  
26     [order_id],  
27     [created_at],  
28     [website_session_id],  
29     [user_id],  
30     [primary_product_id],  
31     [items_purchased],  
32     [price_usd],  
33     [cogs_usd]  
34 FROM [lib://DataFiles/orders.csv]  
35 (txt, codepage is 28591, embedded labels, delimiter is ',', msq);  
36  
37  
38  
39  
40 [autoCalendar]:  
41 DECLARE FIELD DEFINITION Tagged ('$date')  
42 FIELDS  
43     Dual(Year($1), YearStart($1)) AS [Year] Tagged ('$year'),  
44     Dual('Q' & Num(Ceil(Num(Month($1))/3)), Num(Ceil(Num(Month($1))/3))) AS [Quarter] Tagged ('$quarter'),  
45     Dual(Year($1) & '-' & Q' & Num(Ceil(Num(Month($1))/3)), Quarter($1)) AS [Date] Tagged ('$date');
```

TIP
Trainer's rule: use the Data manager to add files quickly; use the Data load editor for transformations and clean-up.

Behind the scenes, the Data manager wrote a load script for you. Open the Data load editor and you'll find it in the Auto-generated section, including the path to orders.csv and a generated calendar.

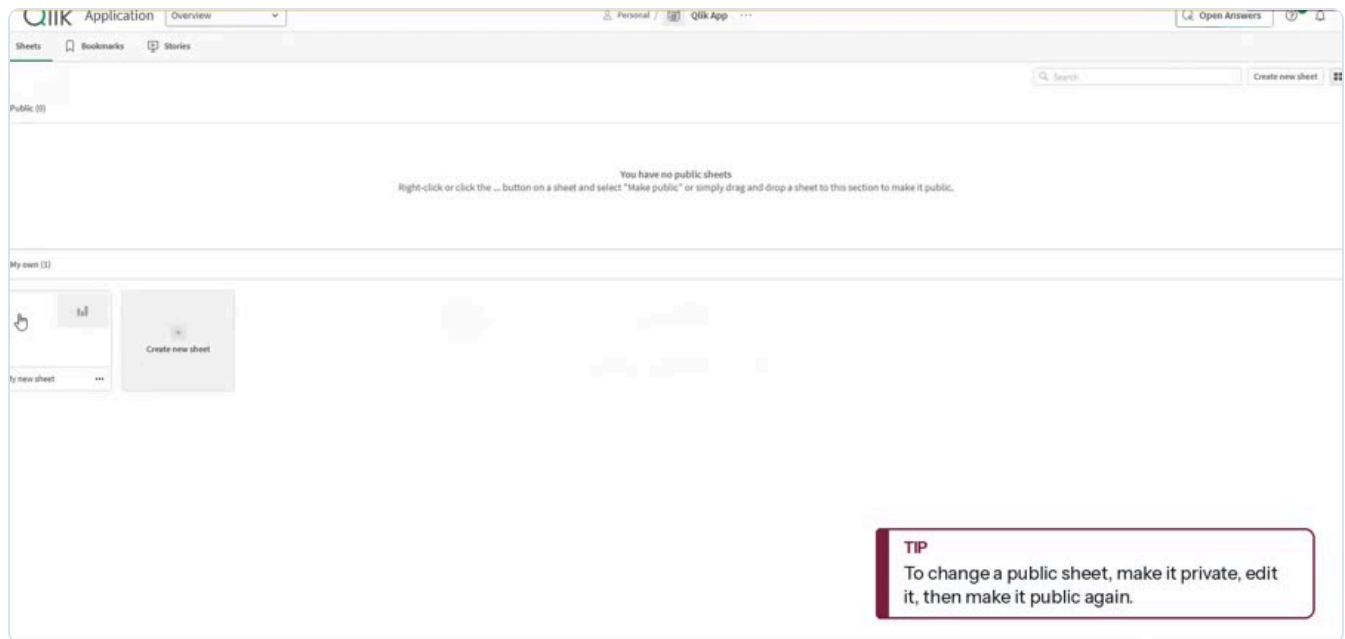
Tip Trainer's rule: use the Data manager to add files quickly; use the Data load editor for transformations and clean-up.

The Data model viewer shows every table you loaded and how tables c...



The Data model viewer shows every table you loaded and how tables connect. Select a table to preview its fields and row counts. You'll use this view constantly once your app has more than one table.

Back in the Hub, your app and its data file now appear under Recent...



Back in the Hub, your app and its data file now appear under Recently used. Open the app to see its sheets. Sheets you build start under My own. When a sheet is ready for others, make it public. To change it again, make it private first.

Tip To change a public sheet, make it private, edit it, then make it public again.

Key takeaways

- Created an app in a space
- Added orders.csv with the Data manager
- Found the script it generated
- Explored the assets & properties panels
- Built a bar chart

Practice

1. Create a new app in your personal space and load a CSV of your choice with the Data manager.
2. Build a bar chart with one dimension and one measure, then use Sorting to put the dimension in date order.
3. Open the Data load editor and find the auto-generated section.

Check your understanding

1. Where do you go to start a new app?
2. Which panel lists dimensions, measures, sorting, add-ons and appearance?
3. What must exist before sheets unlock?

Answers at the back of this handbook.

Load Script Essentials & the Data Load Editor

Video: 04_QT-04_Load_Script_Essentials.mp4 · 4:46 · Stage: Development · Audience: Dashboard developers who completed QT-03

Learning objectives

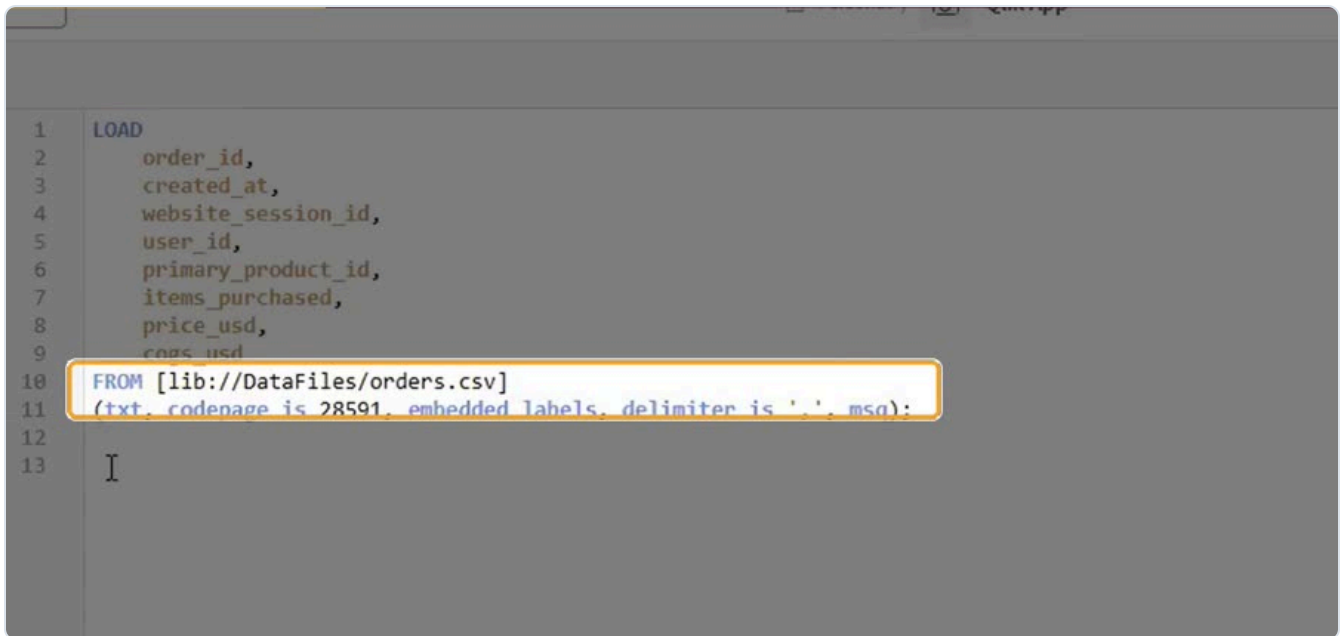
- Load files with the Data load editor
- Find errors with Preview and Debug
- Use STORE, RESIDENT, INLINE and preceding loads

Data manager or Data load editor?

Data manager or Data load editor?		
	Data manager	Data load editor
How	Drag and drop; Qlik writes the script	You write the script
Control	Basic profiling and associations	Transformations, clean-up, joins, reuse
Use it for	Adding a file fast	Everything else

First, when should you use which tool? The Data manager is quick: drag in a file and Qlik builds the script. The Data load editor gives you full control: transformations, clean-up, joins, and reusable logic. The trainer's rule is simple. Use the Data manager to add a file fast, and use the Data load editor for everything else.

Here's the Data load editor



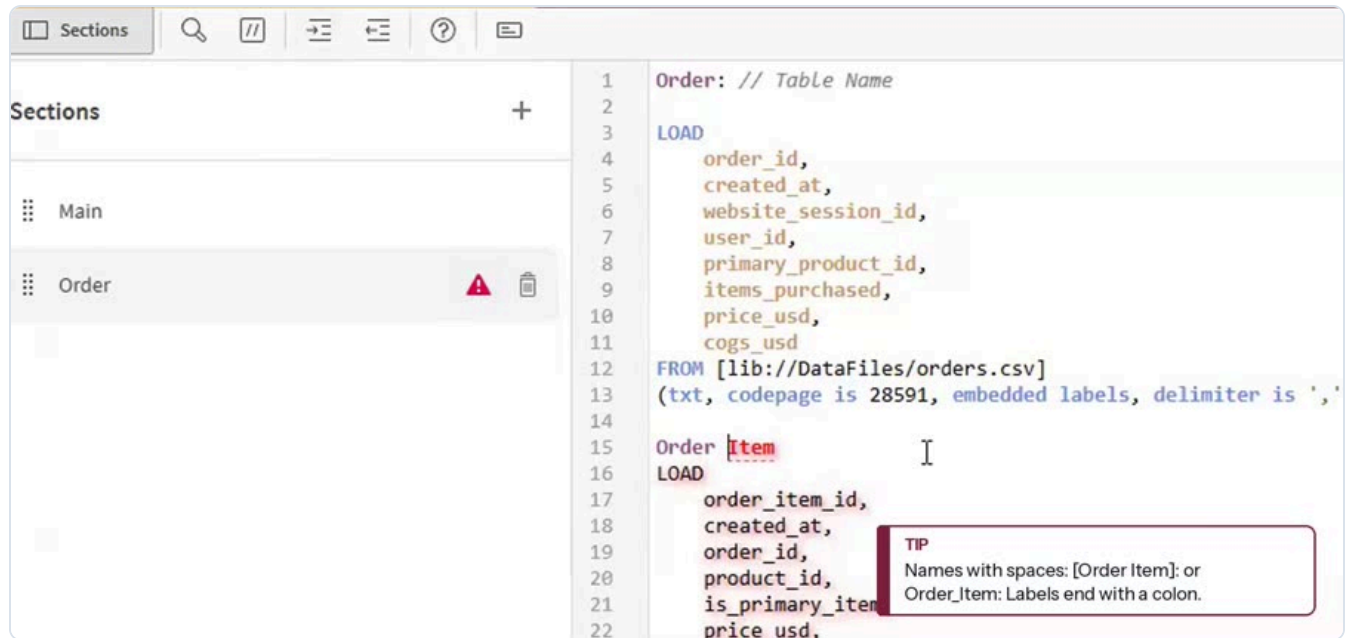
Here's the Data load editor. On the left, the script is split into sections. Qlik runs them from top to bottom, so order matters. The trainer added a section called Order, then used Select data on the DataFiles connection to choose orders .csv. Qlik wrote this LOAD statement: a list of fields, a FROM clause pointing to the file's library path, and the file format in brackets.

This is the Select data dialog that created it



This is the Select data dialog that created it. Qlik detects the file format, the field names, and the delimiter automatically. Embedded field names means the first row holds the column headers. If a file has extra lines above the headers, raise the Header size to skip them. Untick any fields you don't need, then select Add.

Give every table a label, followed by a colon, so you control its n...



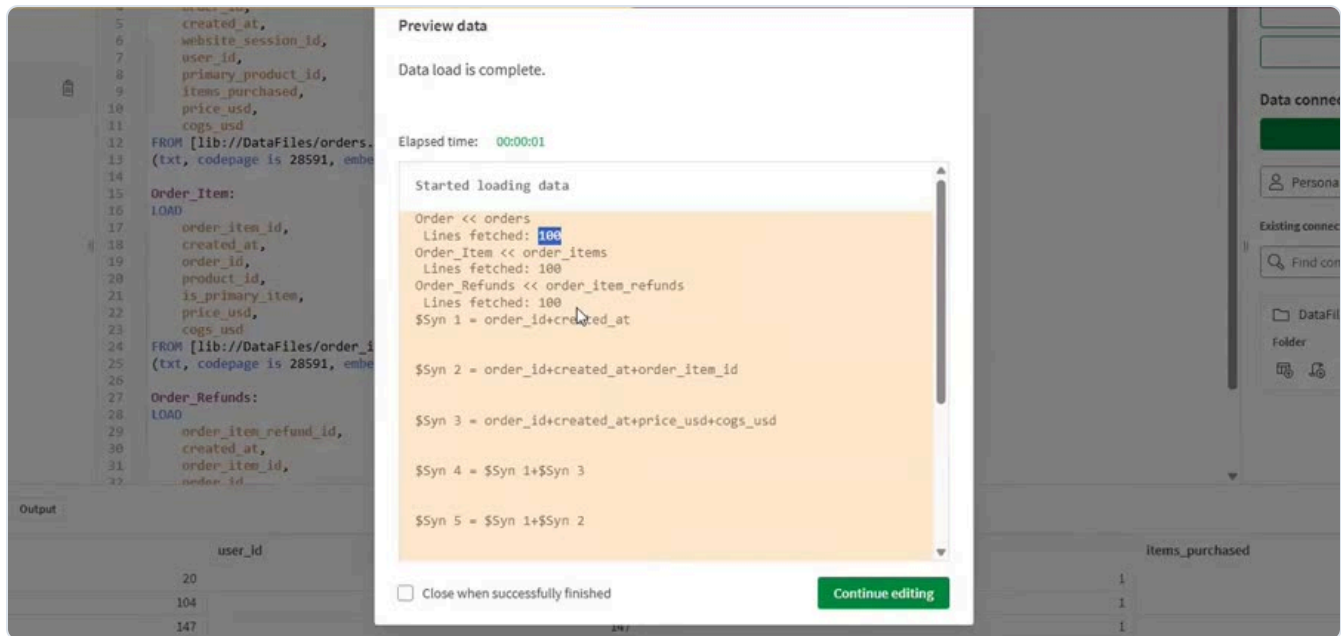
```
1 Order: // Table Name
2
3 LOAD
4     order_id,
5     created_at,
6     website_session_id,
7     user_id,
8     primary_product_id,
9     items_purchased,
10    price_usd,
11    cogs_usd
12 FROM [lib://DataFiles/orders.csv]
13 (txt, codepage is 28591, embedded labels, delimiter is ',',
14
15 Order Item
16 LOAD
17     order_item_id,
18     created_at,
19     order_id,
20     product_id,
21     is_primary_item,
22     price_usd,
```

TIP
Names with spaces: [Order Item]: or Order_Item: Labels end with a colon.

Give every table a label, followed by a colon, so you control its name in the data model. Here the second label, Order Item, contains a space, so Qlik shows it in red, and a warning triangle appears beside the section. Fix it with square brackets around the name, or use an underscore instead.

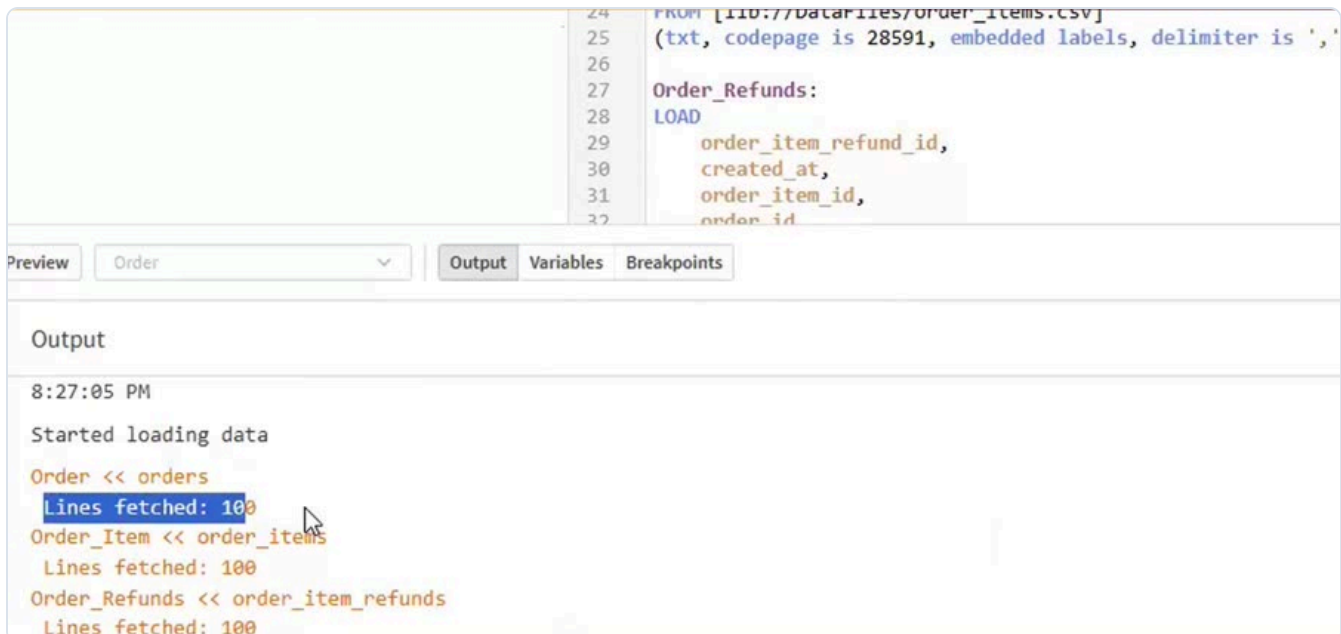
Tip Names with spaces: [Order Item]: or Order_Item: Labels end with a colon.

Before a full reload, test your script



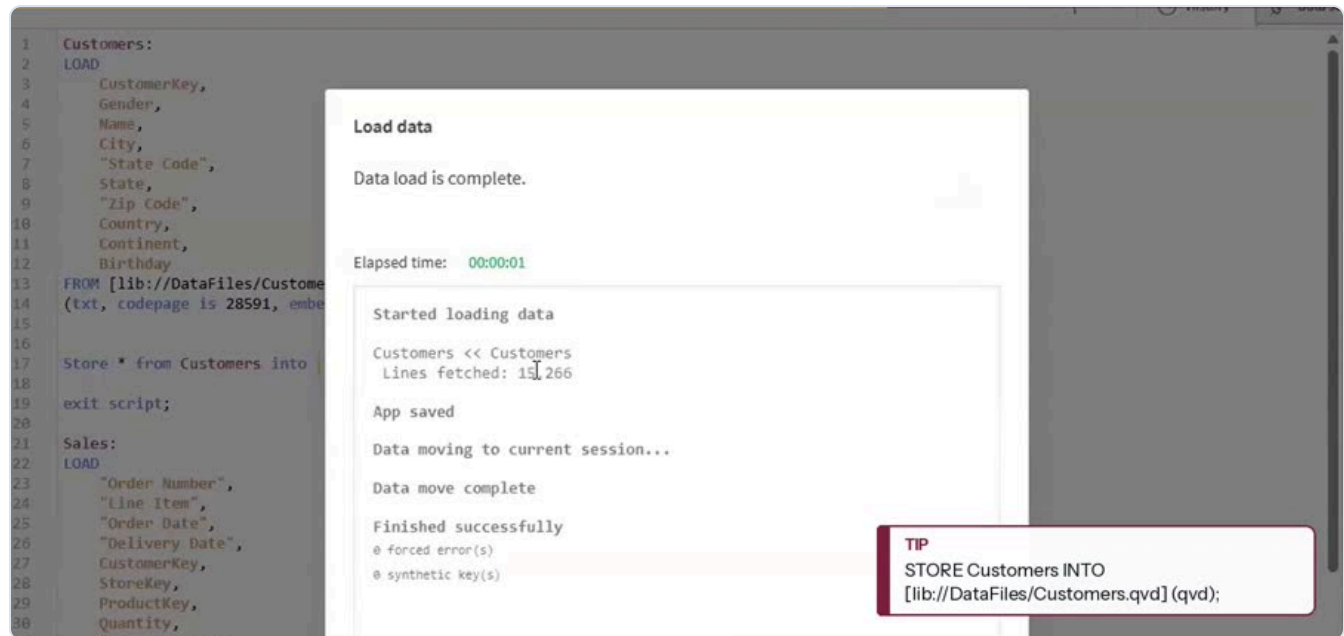
Before a full reload, test your script. Preview data loads only the first hundred rows of each table, so mistakes show up in seconds, even on millions of rows. The log confirms each table and the lines fetched. These dollar Syn lines are synthetic keys, which we'll fix in module QT-06.

Debug mode goes further



Debug mode goes further. It runs the script step by step, with an optional row limit, and you can set breakpoints on any line. The Output panel shows each table as it loads, and the Variables tab shows variable values as they change. When you're done, close Debug and select Load data for the full reload.

Once a table is loaded, the STORE statement saves it as a Q V D fil...



Once a table is loaded, the STORE statement saves it as a QVD file, Qlik's own compressed format. Watch what happens when the trainer stores Customers to just a file name. The reload fails with no qualified path for file. In Qlik Cloud, STORE needs the full library path, so he adds lib:// DataFiles. Reload again, and the load completes. QVD files load extremely fast, and they're the building block of module QT-05.

Tip STORE Customers INTO [lib://DataFiles/Customers.qvd] (qvd);

RESIDENT load: reuse a table in memory

RESIDENT load: reuse a table in memory

```
1 Customers_Raw:
2 LOAD CustomerKey, Name, Gender, City, Country
3 FROM [lib://DataFiles/Customers.csv]
4 (txt, embedded labels, delimiter is ',' );
5
6 Customers:
7 LOAD CustomerKey, Upper(Name) AS Name, Country
8 RESIDENT Customers_Raw;
9
10 DROP TABLE Customers_Raw;
```

3. Drop the raw table so fields aren't duplicated.

A RESIDENT load reads from a table that's already in memory, instead of going back to the source. Load the customers file once. Then load from Resident Customers to transform it, and drop the original table so you don't end up with duplicates.

INLINE load: small tables typed into the script

INLINE load: small tables typed into the script

```
1 Region_Map:
2 LOAD * INLINE [
3 Country, Region
4 United States, Americas
5 Canada, Americas
6 Germany, Europe
7 India, Asia
8 ];
```

Each following row = one record.

An INLINE load types the data straight into the script. It's perfect for small lookup or mapping tables, and for quick tests. The first row holds the field names, and each following row is a record. You can also create one from the Data manager with Manual entry.

Preceding load: transform in one pass

Preceding load: transform in one pass

```
1 Customers:
2 LOAD *,
3   SubField(Name, ' ', 1) AS FirstName,
4   SubField(Name, ' ', -1) AS LastName;
5 LOAD CustomerKey, Name, City, Country
6 FROM [lib://DataFiles/Customers.csv]
7 (txt, embedded labels, delimiter is ',');
```

SubField(text, delimiter, n): n = 1 is the first part, -1 the last.

A preceding load stacks one LOAD on top of another. The bottom LOAD reads the source, and the LOAD above it transforms those rows in the same pass, with no extra table and no resident step. Here, the full name is split into a first name and a last name using the SubField function.

Script hygiene

Script hygiene

1 Comment

```
// one line
/* a block */
REM a statement;
```

2 EXIT SCRIPT

Stops the reload at that line while you test.

3 Sections

Split by source or step. They run top to bottom.

4 Test first

Preview data or Debug before a full reload.

Four habits will keep your scripts readable. Comment with two slashes for one line, or slash star for a block. Use EXIT SCRIPT to stop the reload at a line while you test. Split long scripts into named sections. And always test with Preview or Debug before a full reload.

Key takeaways

- Loaded a file with Select data
- Labelled tables; fixed a syntax error
- Tested with Preview data and Debug
- Used STORE, RESIDENT, INLINE and preceding loads

Practice

1. Load two CSV files in the Data load editor, each in its own section, with table labels.
2. Add a STORE statement that saves one table as a QVD in your DataFiles connection, then reload.
3. Rewrite a name split using a preceding load instead of a RESIDENT load.

Check your understanding

1. What does Preview data load?
2. Why did STORE fail in the demo?
3. What reads a table already in memory?
4. In a preceding load, which LOAD runs first?

Answers at the back of this handbook.

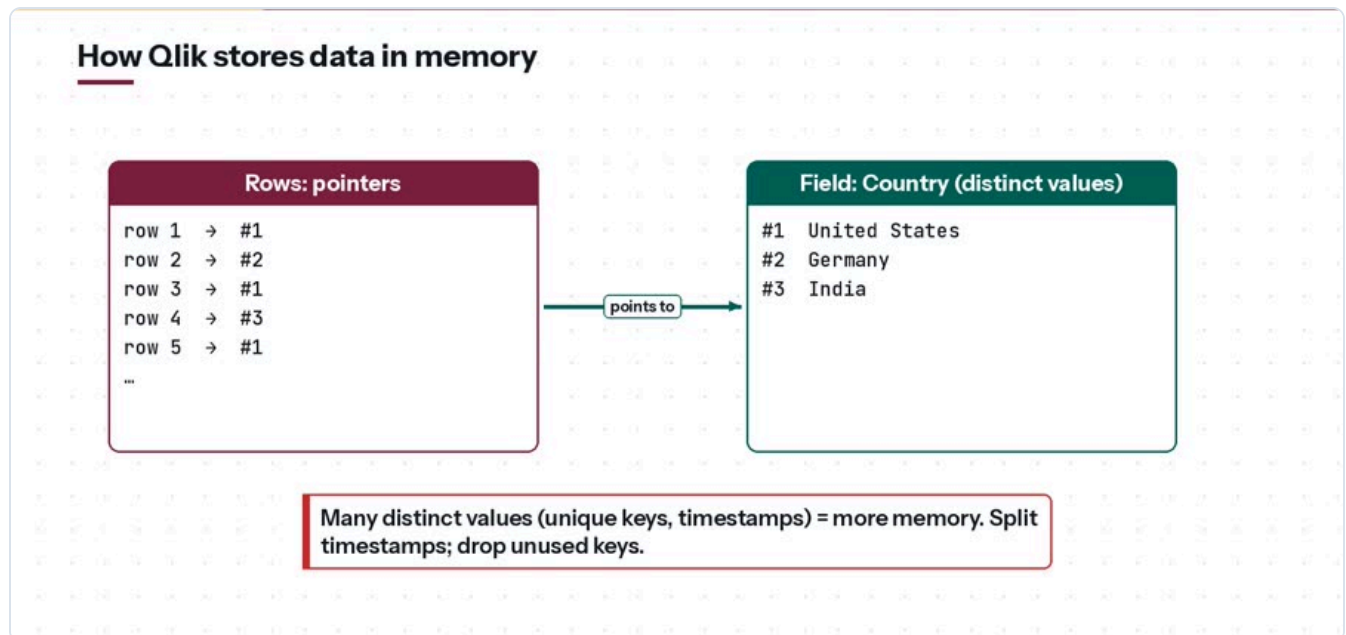
Qlik Architecture & the QVD Layer

Video: 05_QT-05_Qlik_Architecture_and_QVDs.mp4 · 3:28 · Stage: Architecture · Audience: Dashboard developers who completed QT-04

Learning objectives

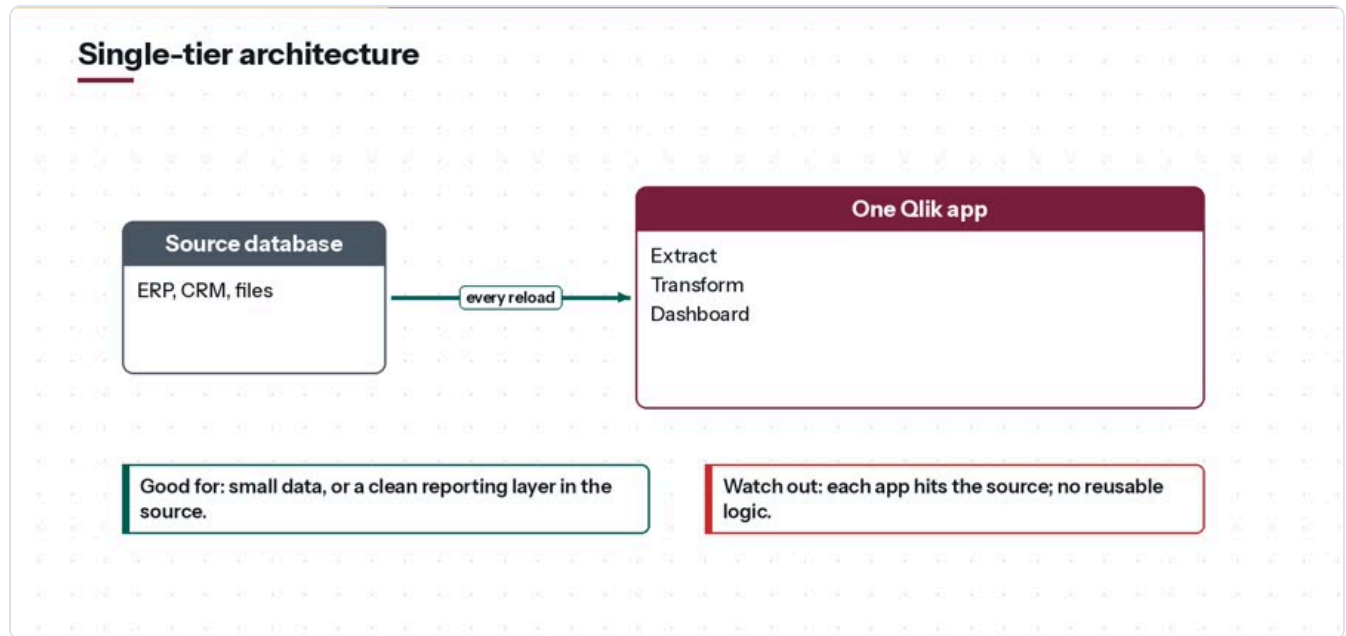
- Explain why Qlik's in-memory engine is fast
- Compare one-, two- and three-tier designs
- Plan extract, transform and UI layers with QVDs

How Qlik stores data in memory



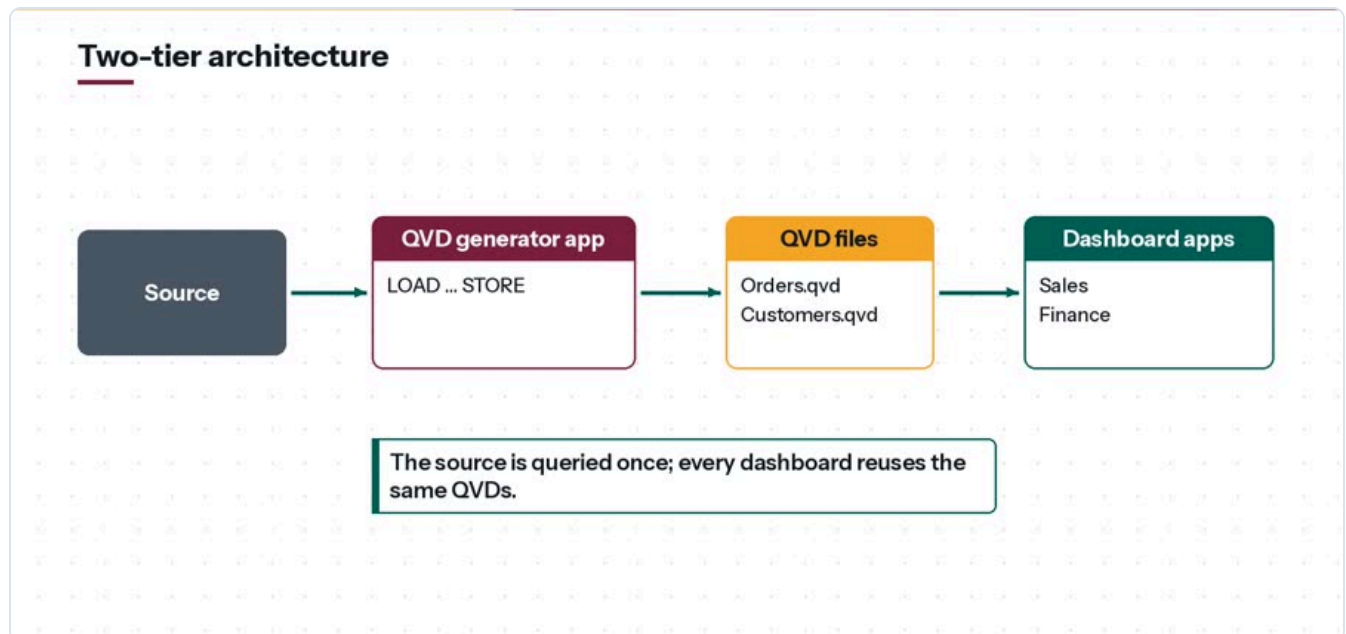
Qlik holds the whole data model in memory. Inside, each field keeps a list of its distinct values just once, and every row simply points to those values. So a country that appears a million times is stored once. That's why calculations stay fast, and why long, unique keys and timestamps cost the most memory.

Single-tier architecture



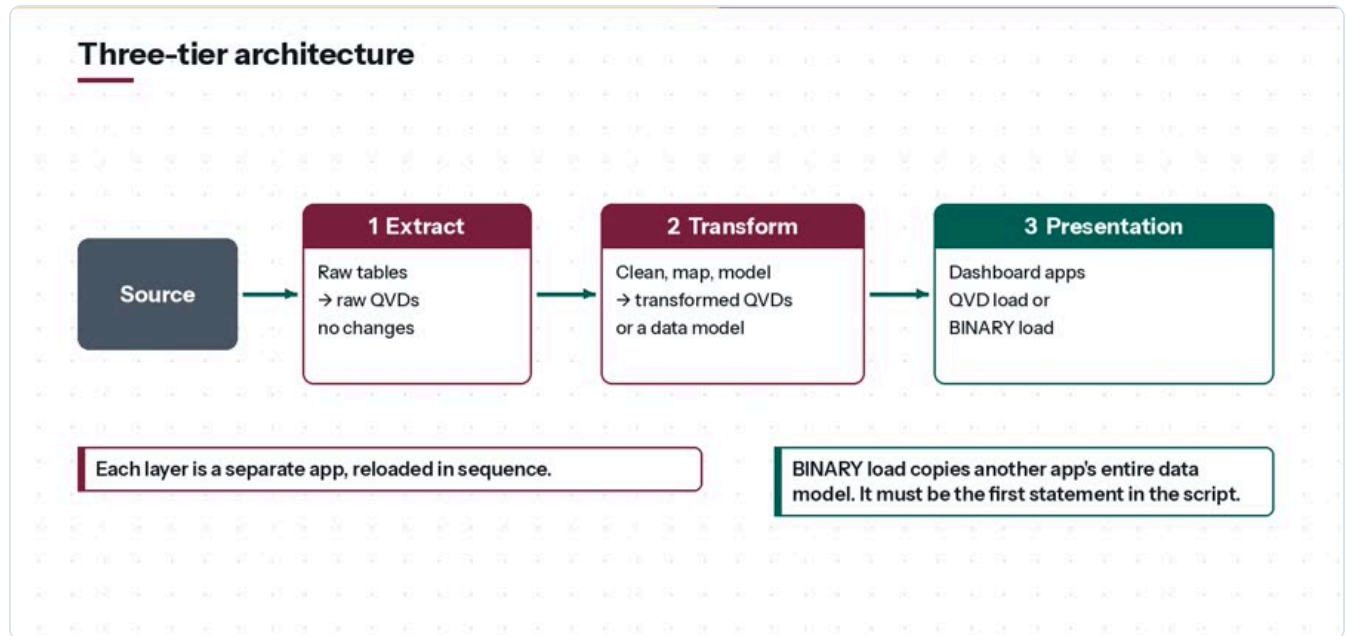
The simplest design is single tier. One app connects to the database, transforms the data, and shows the dashboard. It's quick to build, and it works well when the source already has a clean reporting layer. But every app reloads from the source, and logic can't be reused.

Two-tier architecture



Two tier adds a QVD layer. A generator app extracts tables from the source and stores them as QVD files. Dashboard apps then load those QVDs instead of hitting the database, so the source is queried once, and reloads are much faster.

Three-tier architecture



Most enterprise projects use three tiers. The extract layer pulls raw tables and stores them as QVDs, unchanged. The transform layer reads those raw QVDs, cleans, maps, and models the data, then stores transformed QVDs or a finished data model. The presentation layer holds the dashboards. A dashboard app can even use a binary load to copy the entire data model from another app in one step.

Which architecture?

Which architecture?

Design	Choose it when...	Trade-off
Single tier	Small data, or a clean reporting layer already exists in the source	Simple, but no reuse
Two tier	No ETL in the source; several apps share the same tables	Faster reloads, one extra app
Three tier	Enterprise: heavy transformation, many apps, large volumes	Most robust; more apps to schedule

So which should you choose? Use single tier for small data, or when the database already provides a clean reporting layer. Use two tier when the source has no E T L, and several apps share the same tables. And use three tier for most enterprise projects, with heavy transformations, many apps, and large volumes.

The layers in script

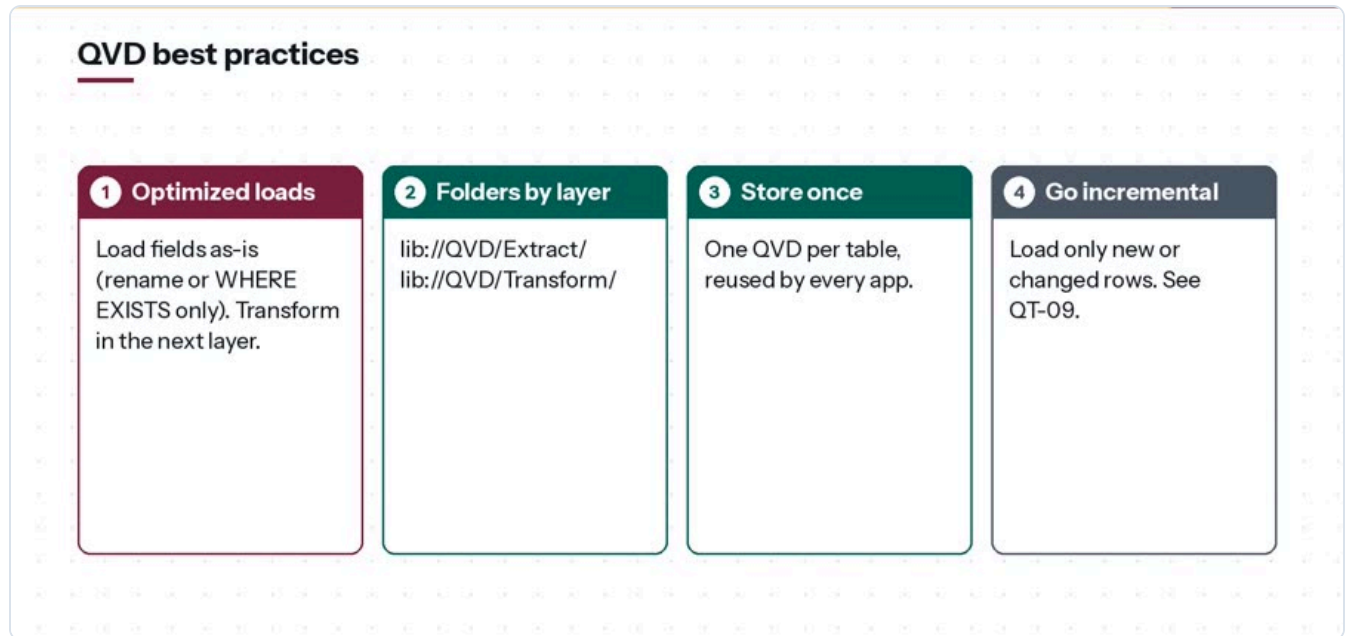
The layers in script

```
1 // 1 Extract app
2 Orders:
3 LOAD * FROM [lib://Source/orders.csv]
4 (txt, embedded labels, delimiter is ',' );
5 STORE Orders INTO [lib://QVD/Extract/Orders.qvd] (qvd);
6
7 // 2 Transform app
8 Orders:
9 LOAD order_id, Date(Floor(created_at)) AS OrderDate,
10 price_usd
11 FROM [lib://QVD/Extract/Orders.qvd] (qvd);
12
13 // 3 Dashboard app (first line of the script)
14 BINARY [<transform app>;
```

Presentation: BINARY copies the whole model. App reference syntax differs between Qlik Cloud and client-managed Qlik Sense.

Here's how the layers look in script. The extract app loads the source and stores a raw QVD. The transform app reads that QVD and shapes it. And the dashboard app starts with a binary load of the transform app, or loads the transformed QVDs directly.

QVD best practices



A few QVD best practices. Keep loads from QVD optimized: loading fields as they are, with only renaming or a where exists clause, is much faster than transforming during the load. Organize QVD folders by layer. Store each table once and reuse it everywhere. And when volumes grow, switch to incremental loads, covered in module QT-09.

Key takeaways

- Distinct values drive memory use
- Single, two and three tier designs
- Extract → Transform → Presentation

Practice

1. Draw the three-tier architecture for a project with two source systems and three dashboards. Name each app and each QVD folder.
2. List which transformations belong in the extract layer and which in the transform layer for a sales feed.

Check your understanding

1. What drives memory use in a Qlik model?
2. What does BINARY load copy?
3. Where must a BINARY statement appear?
4. Which design suits small data with a clean reporting layer?

Answers at the back of this handbook.

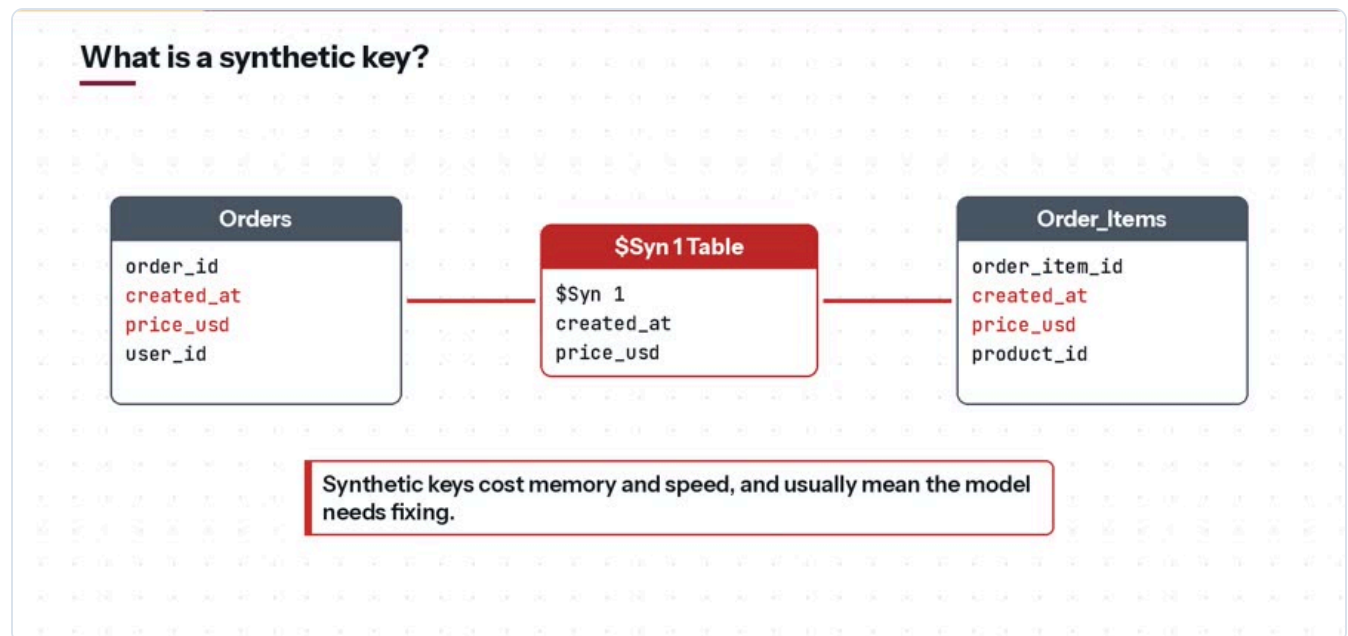
Synthetic Keys, Circular References & Link Tables

Video: 06_QT-06_Synthetic_Keys_and_Link_Tables.mp4 · 4:38 · Stage: Development · Audience: Dashboard developers who completed QT-04 and QT-05

Learning objectives

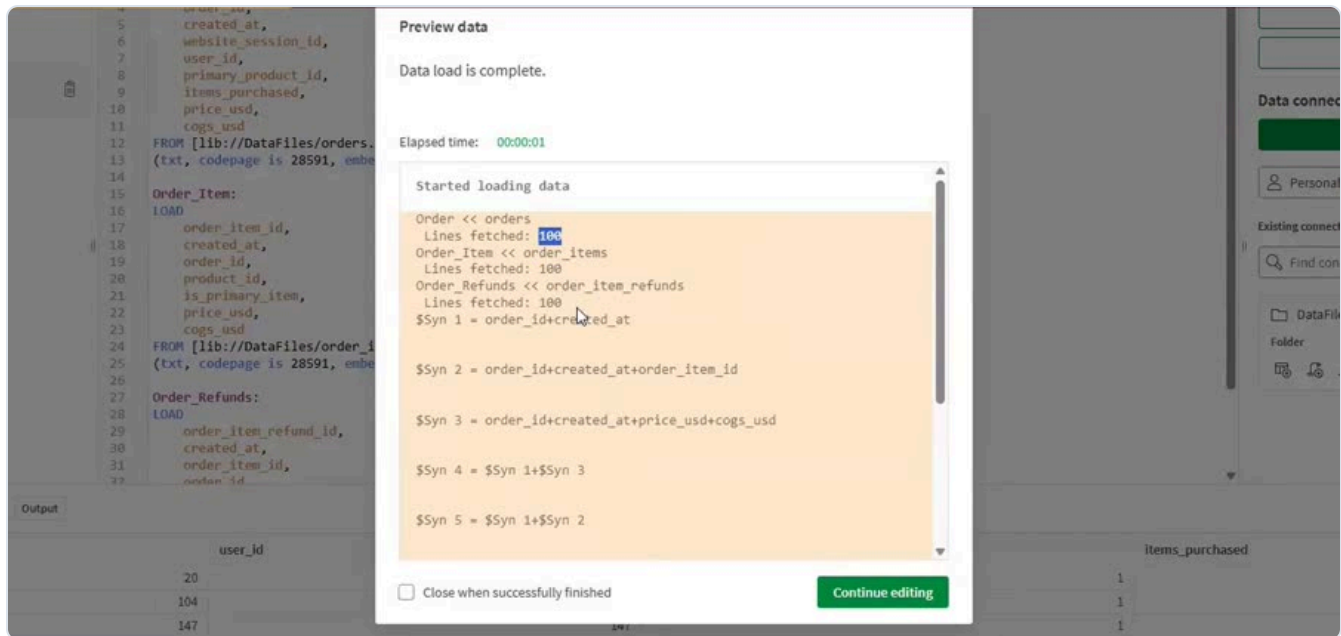
- Spot synthetic keys and circular references
- Explain why they happen
- Fix them: rename, remove, composite key, link table

What is a synthetic key?



Remember, Qlik associates tables on every field name they share. When two tables share more than one field, Qlik can't pick one, so it silently builds a synthetic key: an extra table holding every combination of the shared fields. Synthetic keys slow the app down, use memory, and often signal that the model is wrong.

Here the trainer deliberately loads three order tables blindly: ord...



Here the trainer deliberately loads three order tables blindly: orders, order items, and refunds. Look at the load log. Qlik has created one synthetic key after another. The first combines order ID and created at, and later ones combine price and cost too.

The Data model viewer makes it obvious

The Data model viewer makes it obvious. Instead of three tables linked by one key, there's a synthetic table in the middle, holding dollar Syn fields that combine order ID, created at, price, and cost. Every table now connects through it.

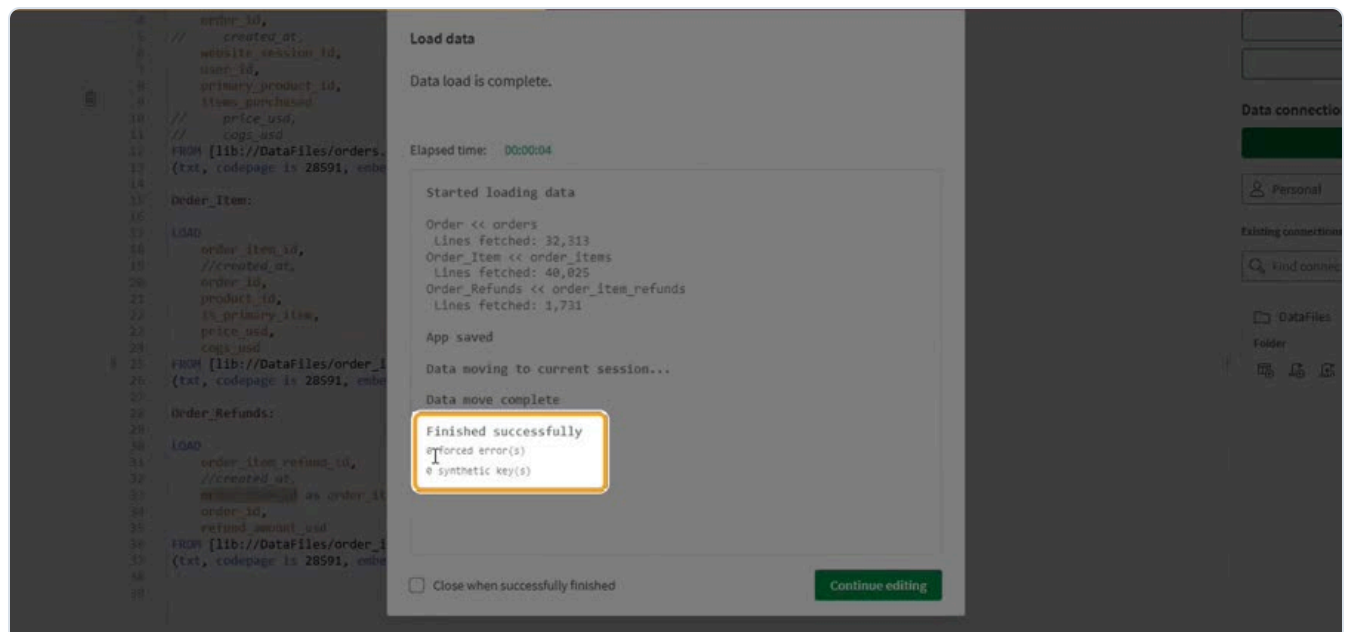
The fix starts in the script



```
1 Order:
2
3 LOAD
4     order_id,
5     // created_at,
6     website_session_id,
7     user_id,
8     primary_product_id,
9     items_purchased
10    // price_usd,
11    // cogs_usd
12 FROM [lib://DataFiles/orders.csv]
13 (txt, codepage is 28591, embedded labels, delimiter is ',', msq);
14
15 Order_Item:
16
17 LOAD
18     order_item_id,
19     //created_at,
20     order_id,
21     product_id,
22     is_primary_item,
23     price_usd,
24     cogs_usd
25 FROM [lib://DataFiles/order_items.csv]
26 (txt, codepage is 28591, embedded labels, delimiter is ',', msq);
27
```

The fix starts in the script. Ask which tables really need each shared field. Created at only needs to live in one table, so the trainer comments it out of the others with two slashes. Price and cost are repeated in orders and order items, so they're removed from orders.

One more shared field remained: order item ID, in both order items ...



Load data

Data load is complete.

Elapsed time: 00:00:04

Started loading data

- Order << orders
Lines fetched: 32,313
- Order_Item << order_items
Lines fetched: 40,025
- Order_Refunds << order_item_refunds
Lines fetched: 1,731

App saved

Data moving to current session...

Data move complete

Finished successfully

- 0 enforced error(s)
- 0 synthetic key(s)

☐ Close when successfully finished

Continue editing

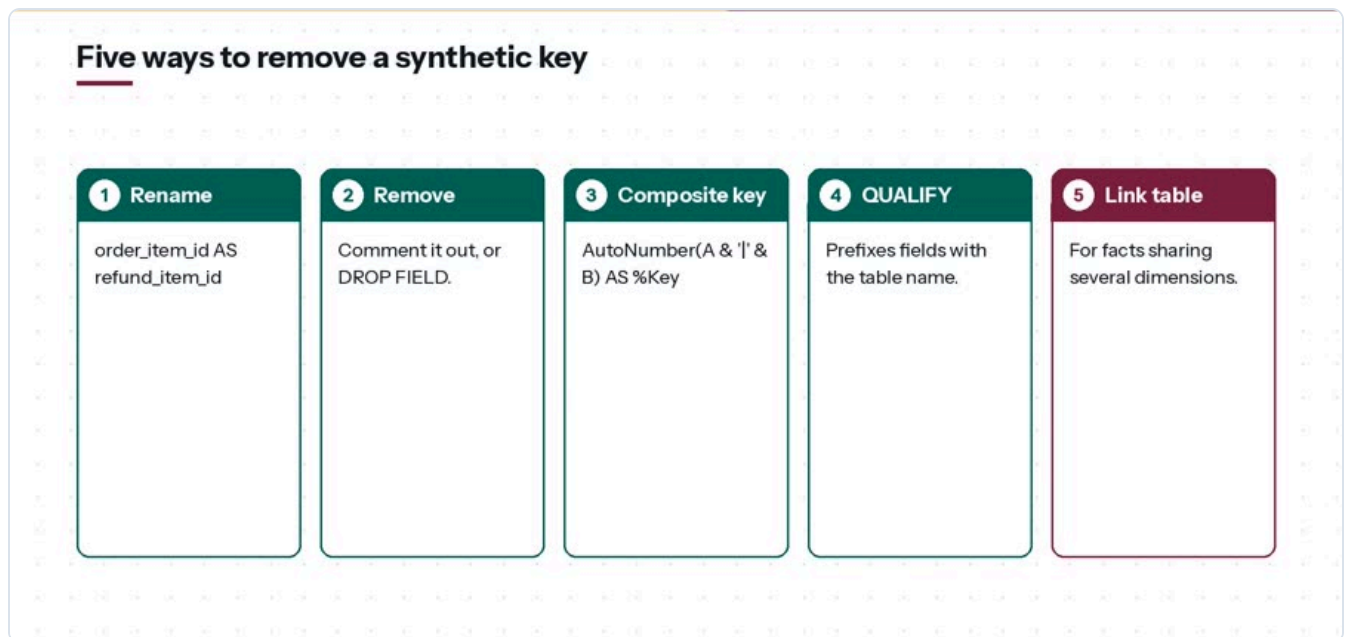
One more shared field remained: order item ID, in both order items and refunds. The trainer renamed it in the refunds table with AS, so it no longer matches. Reload, and the log reports zero synthetic keys.

Now the model is clean



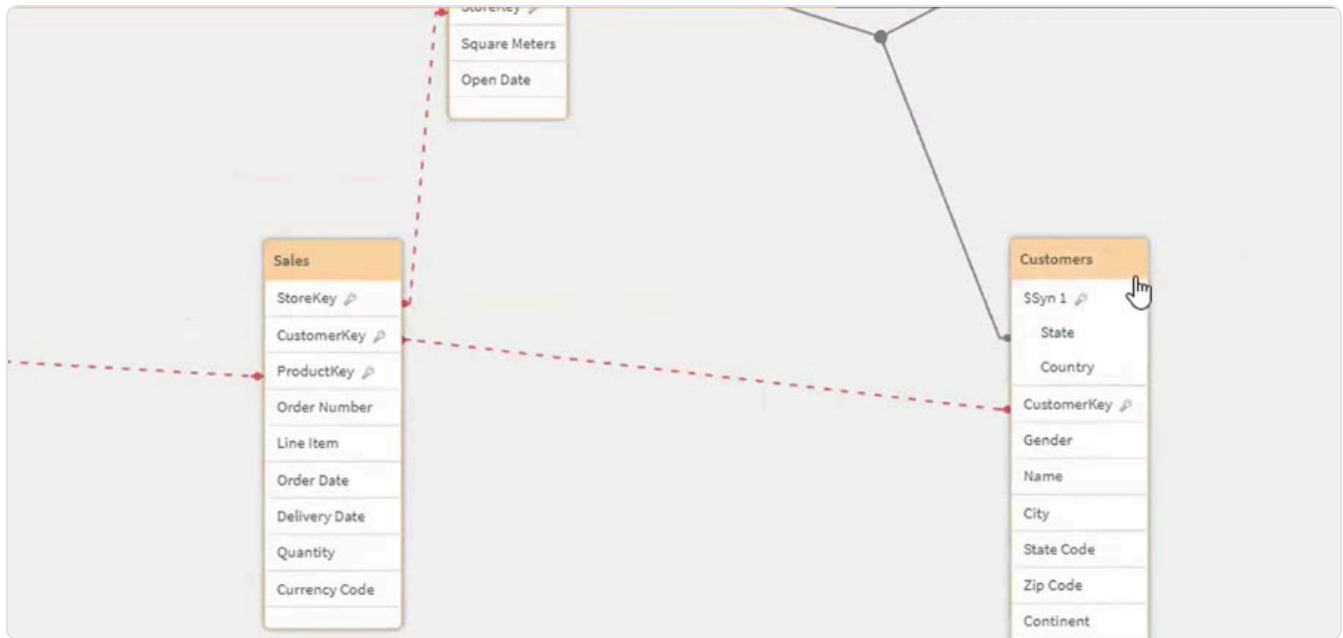
Now the model is clean. Orders, order items, and refunds associate on a single field, order ID, exactly as designed.

Five ways to remove a synthetic key



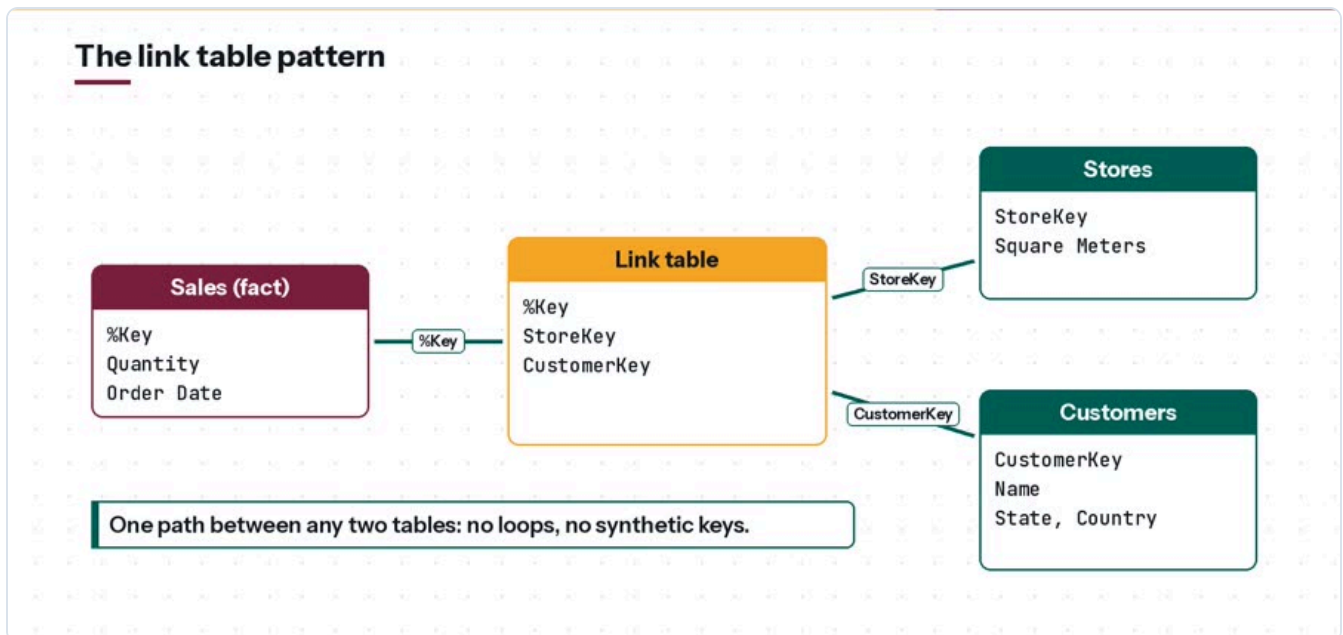
You have five ways to remove a synthetic key. Rename a field with AS so it no longer matches. Remove a field you don't need, by commenting it out or dropping it. Combine several fields into one composite key, often wrapped in AutoNumber to keep it small. Use QUALIFY to prefix field names with their table name. Or, when two fact tables share several dimensions, build a link table.

Now the second problem: a circular reference



Now the second problem: a circular reference. In this model, sales links to stores and to customers, but stores and customers also share state and country. That means there's more than one path between the tables. Qlik breaks the loop by making one link loose, shown as a red dotted line, and your totals become unreliable.

The link table pattern



The link table pattern breaks the loop. Keep shared descriptive fields like state and country in one dimension only. Then build a link table that holds the keys, and connect it to the fact table through a single composite key. Every table now has exactly one path to every other table.

Building a link table in script

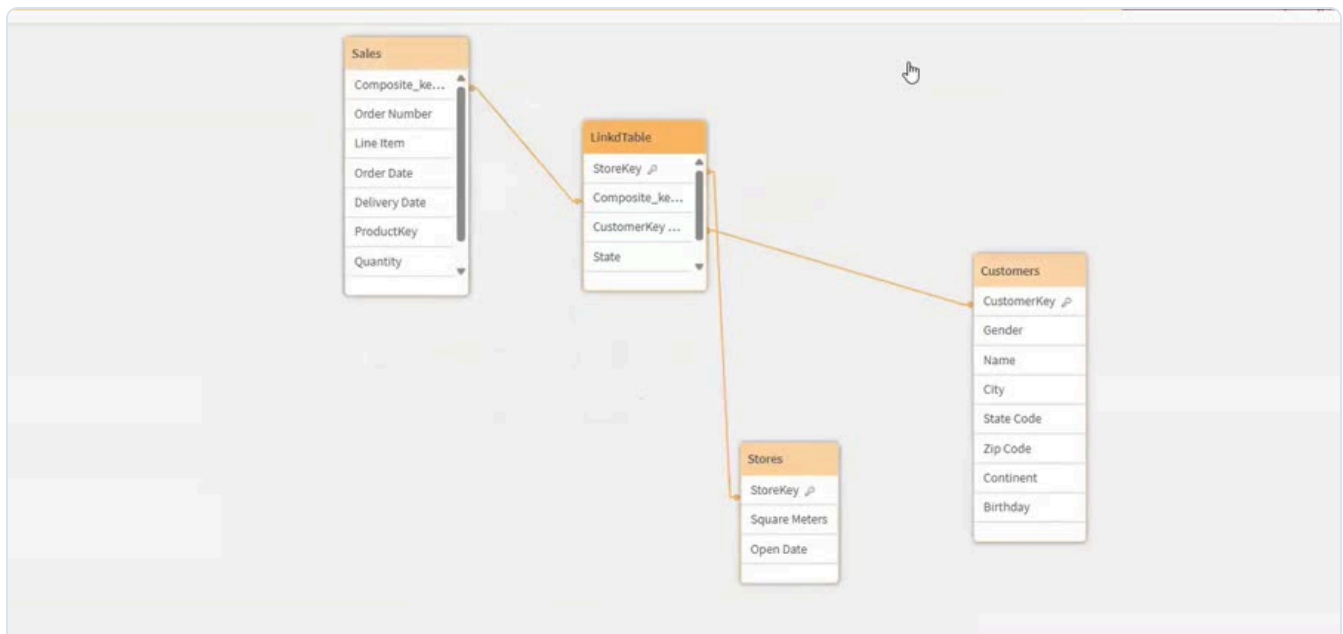
Building a link table in script

```
1 Sales:
2 LOAD *,
3     StoreKey & '|' & CustomerKey AS %Key
4 FROM [lib://DataFiles/Sales.csv]
5 (txt, embedded labels, delimiter is ',');
6
7 LinkTable:
8 LOAD DISTINCT %Key, StoreKey, CustomerKey
9 RESIDENT Sales;
10
11 DROP FIELDS StoreKey, CustomerKey FROM Sales;
```

3. Keys now live only in the link table.

In script, it takes three steps. Add a composite key to the fact table by joining the store and customer keys. Build the link table with a distinct resident load of the composite key and the individual keys. Then drop the individual keys from the fact table, so they live only in the link table.

Here's the trainer's result



Here's the trainer's result. Sales connects to the link table through its composite key. The link table carries the store key and the customer key, and stores and customers each hang off it with a single path. The loop and the synthetic key are both gone.

Two fact tables? Choose a strategy

Two fact tables? Choose a strategy

Strategy	Use when	Watch out for
Concatenate	Facts share most fields (e.g. actuals and budget)	Missing fields become null; add a flag field
Join	Rows match one-to-one	Row explosion and double counting
Link table	Facts share several dimensions at different grains	More script; more tables to maintain

When a model has more than one fact table, you have three options. Concatenate them into one table when they share most fields; missing fields simply become null. Join them only when rows match one to one, or you'll multiply rows. And use a link table when the facts share several dimensions at different grains.

Key takeaways

- Synthetic key = more than one shared field
- Rename, remove, composite key, QUALIFY
- Circular reference = more than one path
- A link table gives one path

Practice

1. Load orders, order items and refunds without editing fields, then count the synthetic keys in the load log.
2. Remove every synthetic key using rename and comment-out until the log shows zero synthetic keys.
3. Build a link table for a sales fact that links to stores and customers.

Check your understanding

1. What causes a synthetic key?
2. What causes a circular reference?
3. What does LOAD DISTINCT do in a link table?
4. When should you concatenate two fact tables?

Answers at the back of this handbook.

Answer key

QT-01 • BI Foundations: From Raw Data to Decisions

1. Visualize
2. Semi-structured
3. Gray (excluded)
4. Unit testing

QT-02 • Dimensional Modelling: Facts, Dimensions, Star & Snowflake

1. A fact table
2. Fields with exactly the same name
3. Star schema
4. A synthetic key

QT-03 • Your First Qlik Cloud App

1. Create in the Hub
2. The properties panel
3. At least one table in the data model

QT-04 • Load Script Essentials & the Data Load Editor

1. The first 100 rows of each table
2. No qualified path; a full lib:// path is required
3. A RESIDENT load
4. The bottom LOAD

QT-05 • Qlik Architecture & the QVD Layer

1. The number of distinct values per field
2. Another app's entire data model
3. First statement in the script
4. Single tier

QT-06 • Synthetic Keys, Circular References & Link Tables

1. Two tables sharing more than one field
2. More than one path between tables
3. Keeps one row per key combination
4. When they share most fields