

SELF-STUDY HANDBOOK

Data Analytics & Visualization with Qlik and Tableau

CommunityForce Inc.

Modules in this edition:

- **QT-07** Script Variables & Transformation Functions
- **QT-08** Building a Master Calendar
- **QT-09** Incremental Loads with QVDs
- **QT-10** Designing Qlik Sheets & Visualizations
- **QT-11** Set Analysis & Alternate States
- **QT-12** Securing, Optimising & Deploying Qlik Apps

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

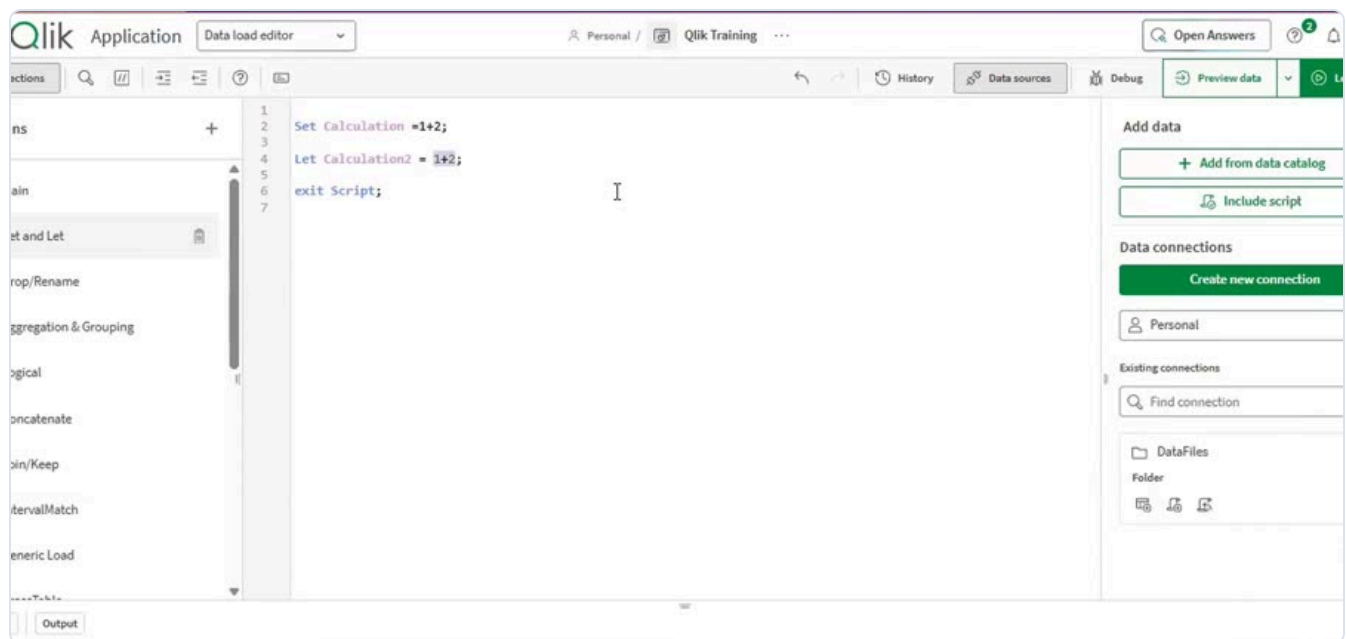
Script Variables & Transformation Functions

Video: 07_QT-07_Script_Variables_and_Functions.mp4 · 4:10 · Stage: Development · Audience: Dashboard developers who completed QT-04 to QT-06

Learning objectives

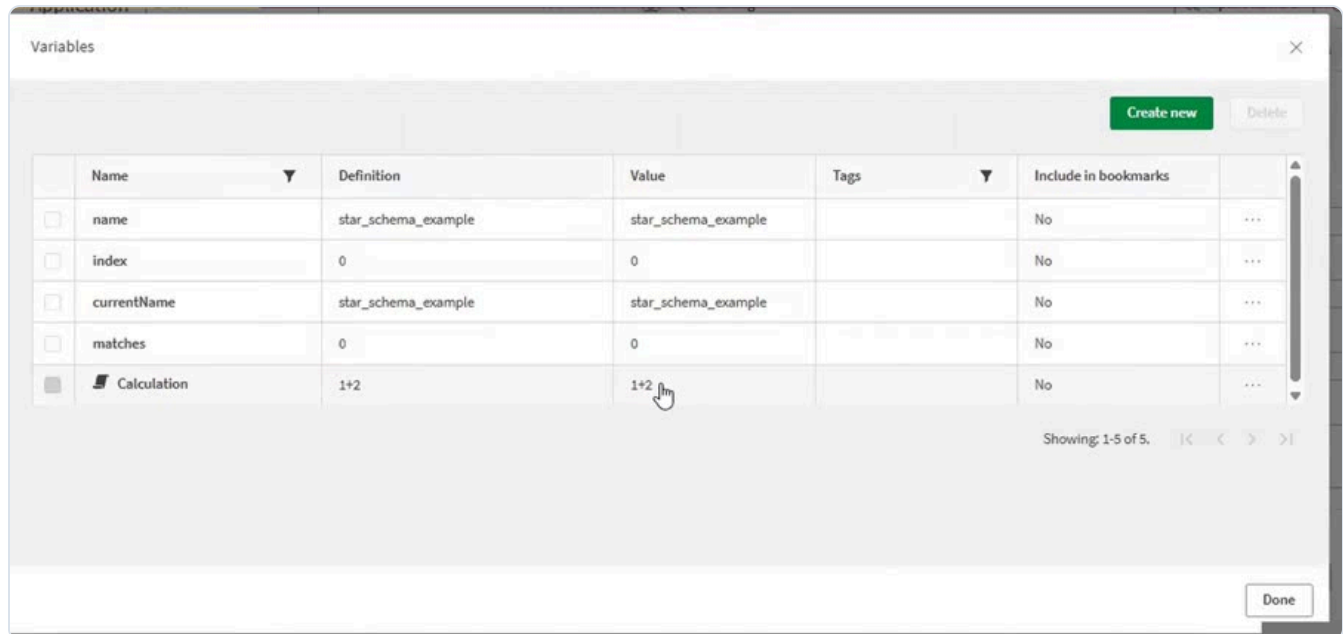
- Use SET and LET variables and dollar-sign expansion
- Format, shift and build dates with date functions
- Clean text with string functions

A variable holds a value you can reuse anywhere in the app



A variable holds a value you can reuse anywhere in the app. There are two ways to assign one. SET stores the text exactly as written. LET evaluates the expression first and stores the result. Here the trainer assigns one plus two both ways.

After reloading, the Variables panel shows what SET stored



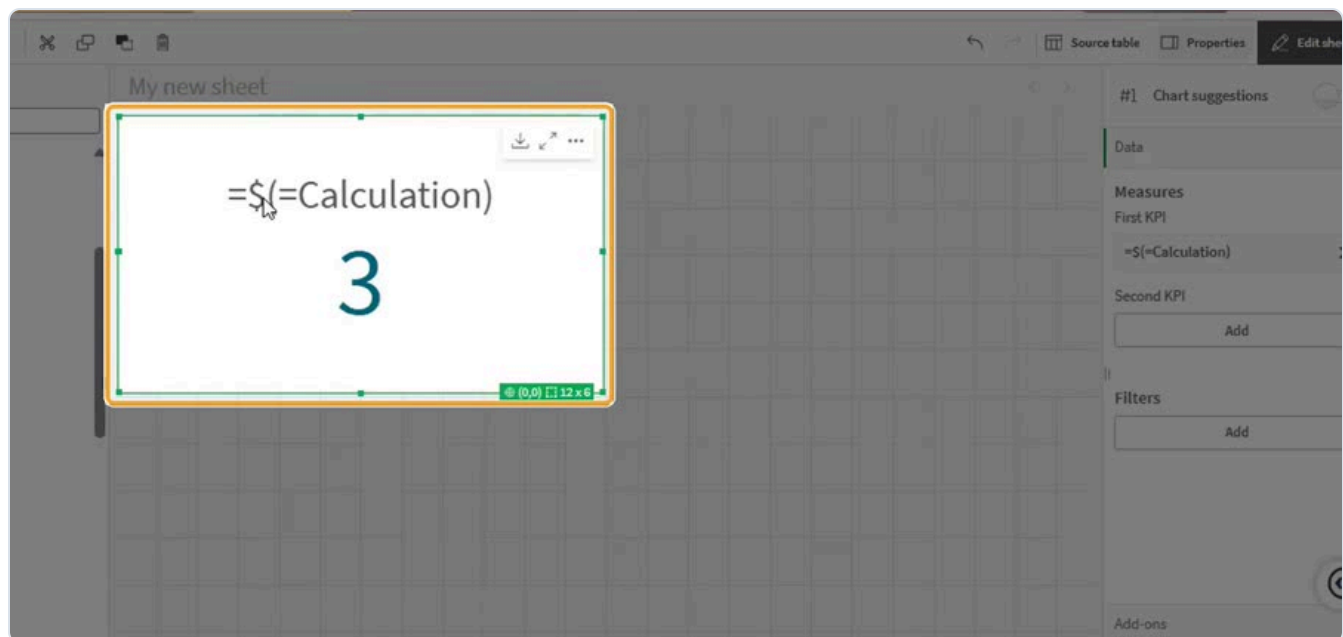
	Name	Definition	Value	Tags	Include in bookmarks	
☐	name	star_schema_example	star_schema_example		No	...
☐	index	0	0		No	...
☐	currentName	star_schema_example	star_schema_example		No	...
☐	matches	0	0		No	...
☒	Calculation	1+2	1+2		No	...

Showing: 1-5 of 5.

Done

After reloading, the Variables panel shows what SET stored. The variable named Calculation holds the text one plus two, not the number three. The definition and the value are both just text.

To evaluate stored text in a chart, use dollar-sign expansion



To evaluate stored text in a chart, use dollar-sign expansion. Dollar sign, then brackets, tells Qlik to replace the variable with its contents before calculating. In this KPI, dollar sign equals Calculation returns three.

SET, LET and \$()

SET, LET and \$()		
	Example	Stores / does
SET	SET vFormat = 'DD/MM/YYYY';	The text exactly as written
LET	LET vToday = Today();	The evaluated result
\$()	Sum({<Year=\$(vYear)>} Sales)	Replaces the variable with its contents

Here's the rule of thumb. Use SET for fixed text, like a format string or a whole expression you want to reuse. Use LET when you need a calculated result, such as today's date or a maximum value. And wrap a variable in dollar-sign expansion wherever you want its contents used.

QUALIFY and UNQUALIFY

QUALIFY and UNQUALIFY	
<pre>1 QUALIFY *; 2 Table2: 3 LOAD * INLINE [A, B, D 4 1, DD, DDDD]; 5 UNQUALIFY *; 6 7 // Fields become Table2.A, Table2.B, Table2.D</pre>	UNQUALIFY * stops qualifying for the tables that follow.

QUALIFY is another script statement you met in QT-06. It prefixes field names with their table name, so shared fields stop linking. UNQUALIFY star switches it off again, so later tables load normally.

Everyday date functions

Everyday date functions

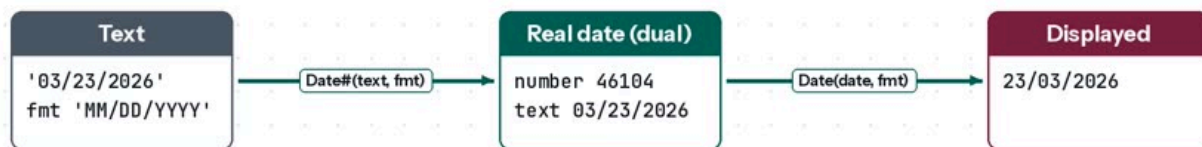
Function	Example	Result
Date()	Date(Today(), 'DD/MM/YYYY')	25/06/2026
AddMonths()	AddMonths('25/06/2026', -3)	25/03/2026
AddYears()	AddYears('25/06/2026', 2)	25/06/2028
WeekStart()	WeekStart('25/06/2026', -1)	Start of the previous week
MakeDate()	MakeDate(2026, 4, 23)	23/04/2026

Results depend on the DateFormat and FirstWeekDay settings at the top of the script.

Now the date functions the trainer used most. Date formats a date for display. AddMonths and AddYears shift a date forward, or back with a negative number. WeekStart returns the first day of the week, and an offset moves to earlier or later weeks. And MakeDate builds a date from a year, month, and day.

Date#() interprets, Date() formats

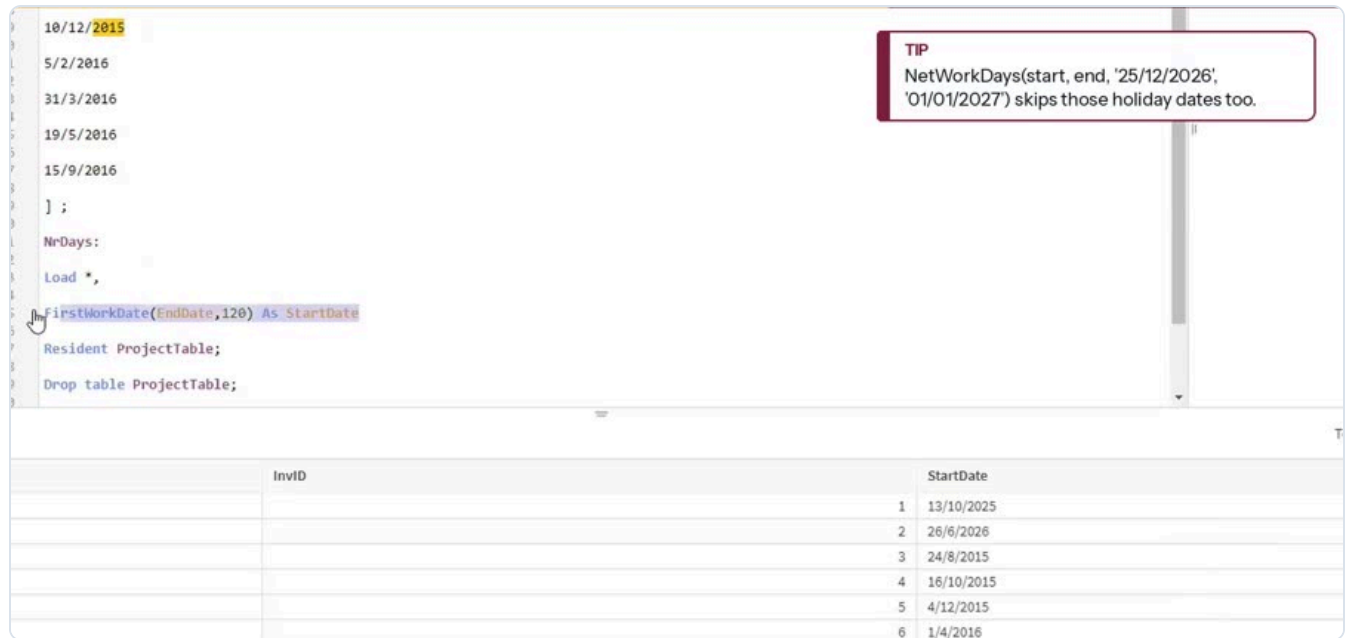
Date#() interprets, Date() formats



Format doesn't match the text? Date#() returns null.

One distinction catches everyone out. Date hash interprets text as a date: you tell Qlik what format the text is in, and it becomes a real date with a number behind it. Date, without the hash, only formats a value that is already a date. If the format you give Date hash doesn't match the text, the result is null.

Working-day functions skip weekends



The screenshot shows a SQL script editor with the following code:

```
10/12/2015
5/2/2016
31/3/2016
19/5/2016
15/9/2016
];
NrDays:
Load *,
FirstWorkDate(EndDate,120) As StartDate
Resident ProjectTable;
Drop table ProjectTable;
```

A tip box in the top right corner states: **TIP** NetWorkDays(start, end, '25/12/2026', '01/01/2027') skips those holiday dates too.

Below the script, a table displays the results of the query:

Invid	StartDate
1	13/10/2025
2	26/6/2026
3	24/8/2015
4	16/10/2015
5	4/12/2015
6	1/4/2016

Working-day functions skip weekends. Here, FirstWorkDate takes each end date and counts back one hundred and twenty working days to find the start date. NetWorkDays does the reverse, counting the working days between two dates, and both accept a list of holidays.

Tip NetWorkDays(start, end, '25/12/2026', '01/01/2027') skips those holiday dates too.

String functions clean up text



The screenshot shows a SQL script editor with the following code:

```
1
2 Let vChar = Capitalize ( 'my little pony girl' );
3
4 Let vChr = Chr(65);
5
6 Let vCahr = Evaluate(5*8);
7
8 let vFindOf = FindOneOf( 'my example text string', 'et%s');
9
10 let vKeepChar = KeepChar ( 'a2b1c4d3', '123' );
11 /*
12
13
```

The line `let vKeepChar = KeepChar ( 'a2b1c4d3', '123' );` is highlighted with a yellow box.

String functions clean up text. The trainer tried several with LET variables: Capitalize, Chr, Evaluate, FindOneOf, and KeepChar.

Everyday string functions

Everyday string functions

Function	Example	Result
Capitalize()	Capitalize('my little pony')	My Little Pony
Upper() / Lower()	Upper('qlik')	QLIK
Trim()	Trim(' Sales ')	Sales
Left()	Left('ABCDEF', 3)	ABC
Mid()	Mid('ABCDEF', 3, 2)	CD
Len()	Len('Data Model')	10

Here's what the most useful string functions return. Capitalize sets the first letter of each word in upper case. Upper and Lower change the whole string. Trim removes leading and trailing spaces. Left takes characters from the start, and Mid takes characters from any position. And Len counts characters, including spaces.

Conditional functions

Conditional functions

```
1 Orders:
2 LOAD order_id,
3   price_usd,
4   If(price_usd ≥ 100, 'Large', 'Small') AS OrderSize,
5   Match(Region, 'North', 'South') AS RegionNo,
6   Alt(discount, 0) AS Discount
7 FROM [lib://DataFiles/orders.csv]
8 (txt, embedded labels, delimiter is ',' );
```

Alt(value, fallback) replaces a missing number with 0.

Conditional functions add business rules. If returns one value when a condition is true and another when it's false. Match returns the position of a value in a list, or zero, which makes it a tidy alternative to many nested ifs. And Alt returns the first valid number, so it's a clean way to replace missing values with zero.

Key takeaways

- SET stores text; LET stores a result
- \$() expands a variable's contents
- Date functions format, shift and build dates
- String and conditional functions clean and classify

Practice

1. Create SET vExpr = Sum(Sales); and use =\$ (vExpr) in a KPI. Then try LET with the same text and compare.
2. Load an inline table of order dates and add fields with Year(), Month(), WeekStart() and AddMonths(date, 1).
3. Clean a list of customer names with Trim() and Capitalize().

Check your understanding

1. What does SET store?
2. What does LET store?
3. What does \$(vName) do?
4. Which function turns text into a real date?
5. What does Mid('ABCDEF', 3, 2) return?

Answers at the back of this handbook.

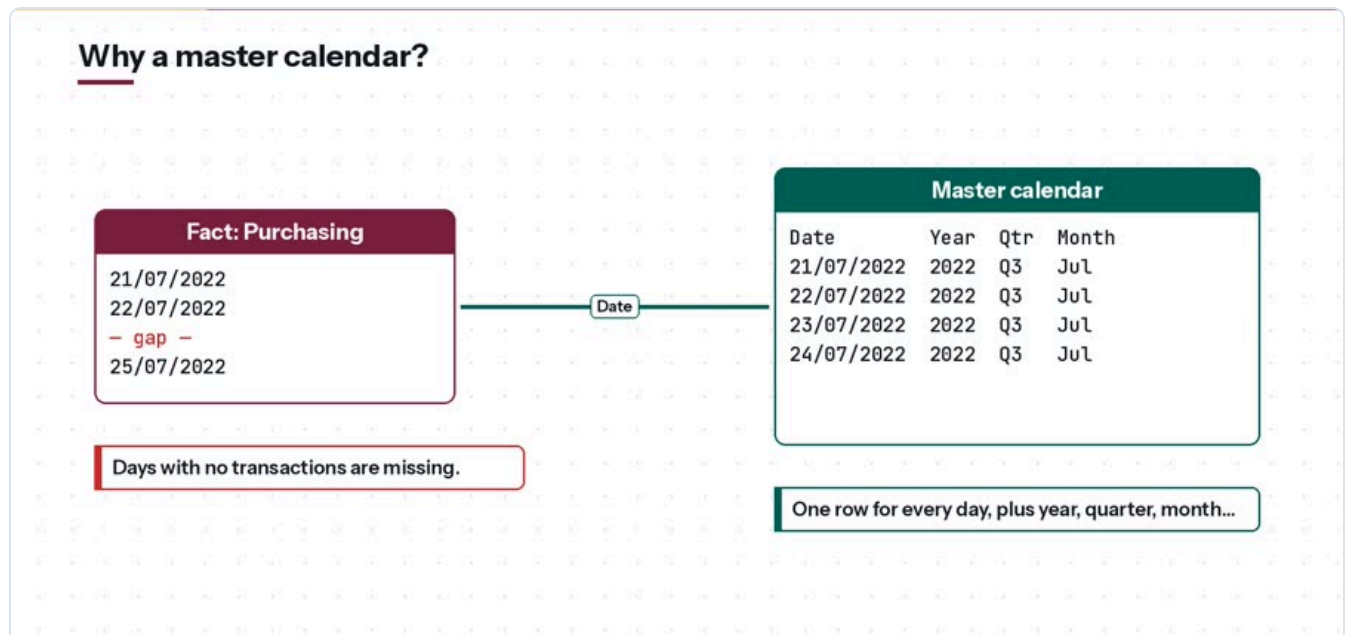
Building a Master Calendar

Video: 08_QT-08_Building_a_Master_Calendar.mp4 · 3:58 · Stage: Development · Audience: Dashboard developers who completed QT-07

Learning objectives

- Generate rows with IterNo and WHILE
- Find the date range with Min and Max
- Build and link a master calendar

Why a master calendar?



Why not just use the dates in the fact table? Because days with no transactions are missing, so charts skip them and trends look wrong. A master calendar is a separate dimension with one row for every day between the first and last date, linked to the fact on the date field.

Generating rows with IterNo and WHILE

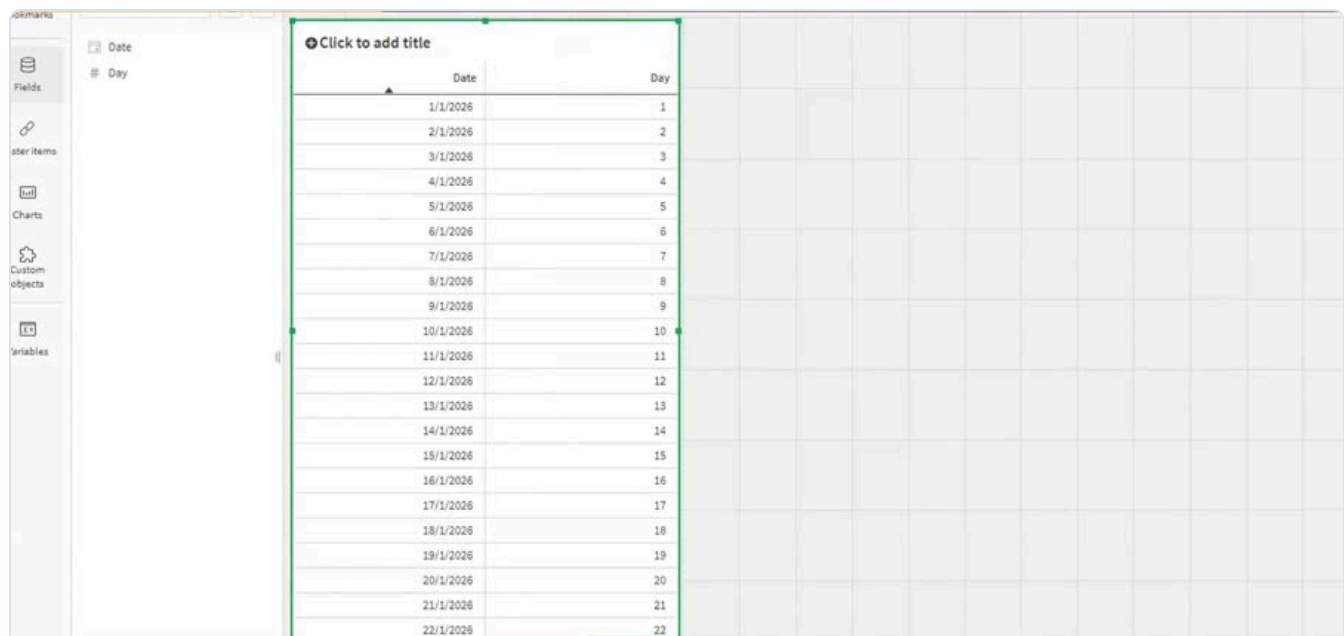
Generating rows with IterNo and WHILE

```
1 Days:
2 LOAD
3   IterNo() AS Day,
4   Date($(vStart) + IterNo() - 1) AS Date
5 AUTOGENERATE 1
6 WHILE $(vStart) + IterNo() - 1 ≤ $(vEnd);
```

Start date + IterNo() - 1 = one row per day.

The key tool is the IterNo function, used with a WHILE clause. WHILE repeats a LOAD for as long as its condition is true, and IterNo counts the iterations, starting at one. So starting from a start date, adding IterNo minus one gives one row per day until the end date.

Here's the trainer's first test with a short date range



Date	Day
1/1/2026	1
2/1/2026	2
3/1/2026	3
4/1/2026	4
5/1/2026	5
6/1/2026	6
7/1/2026	7
8/1/2026	8
9/1/2026	9
10/1/2026	10
11/1/2026	11
12/1/2026	12
13/1/2026	13
14/1/2026	14
15/1/2026	15
16/1/2026	16
17/1/2026	17
18/1/2026	18
19/1/2026	19
20/1/2026	20
21/1/2026	21
22/1/2026	22

Here's the trainer's first test with a short date range. Each row has a Day number from IterNo, and a Date that moves forward one day at a time.

RecNo vs RowNo

RecNo vs RowNo

	RecNo()	RowNo()
Counts	Records read from the source	Rows in the resulting table
WHERE clause	Counted before filtering	Counted after filtering
Use for	Tracing source records	A clean running number

Two related functions are easy to confuse. RecNo returns the record number read from the source, before any WHERE clause. RowNo returns the row number in the resulting table, after filtering. Use RowNo when you need a clean running number.

RecNo vs RowNo with a WHERE clause

RecNo vs RowNo with a WHERE clause

```
1 T:
2 LOAD A, B, RecNo() AS RecNo, RowNo() AS RowNo
3 INLINE [
4 A, B
5 1, aa
6 2, bb
7 3, cc
8 4, dd
9 ] WHERE A > 1;
```

```
11 // A B RecNo RowNo
12 // 2 bb 2 1
13 // 3 cc 3 2
14 // 4 dd 4 3
```

RowNo: 1, 2, 3. Rows in the result.

Here's the difference in action. The inline table has four records, and the WHERE clause drops the first one. RecNo keeps the source numbers: two, three, and four. RowNo rennumbers the rows that survived: one, two, and three.

Now the real calendar

```
12 TempTable:
13
14 Load
15 Date(Max(LastDate)) as Max_LastDate,
16 Date(Min(LastDate)) as Min_LastDate
17
18 Resident Purchasing;
19
20 Drop table Purchasing;
21
22
23
24
25 Iterno:
26
27 LOAD
28
29 Iterno() as Day,
30
31 Date( StartDate + Iterno() - 1 ) as Date
32
33 While StartDate + Iterno() - 1 <= AddMonths(EndDate,1);
34
35
36
```

Max_LastDate	Min_LastDate
21/10/2025	21/7/2022

Now the real calendar. First, find the date range. A resident load of the purchasing table takes the Max and Min of the last receipt date, wrapped in Date so they display as dates.

Then a preceding load on top generates the dates

```
ModifiedDate
FROM Adventureworks2025.Purchasing.ProductVendor;

TempTable:
Load *,
IterNo() as Day,
Date(Min_LastDate+IterNo()-1) as Date
While Min_LastDate+IterNo()-1 <= Max_LastDate;

Load
Date(Max(LastDate)) as Max_LastDate,
Date(Min(LastDate)) as Min_LastDate

Resident Purchasing;

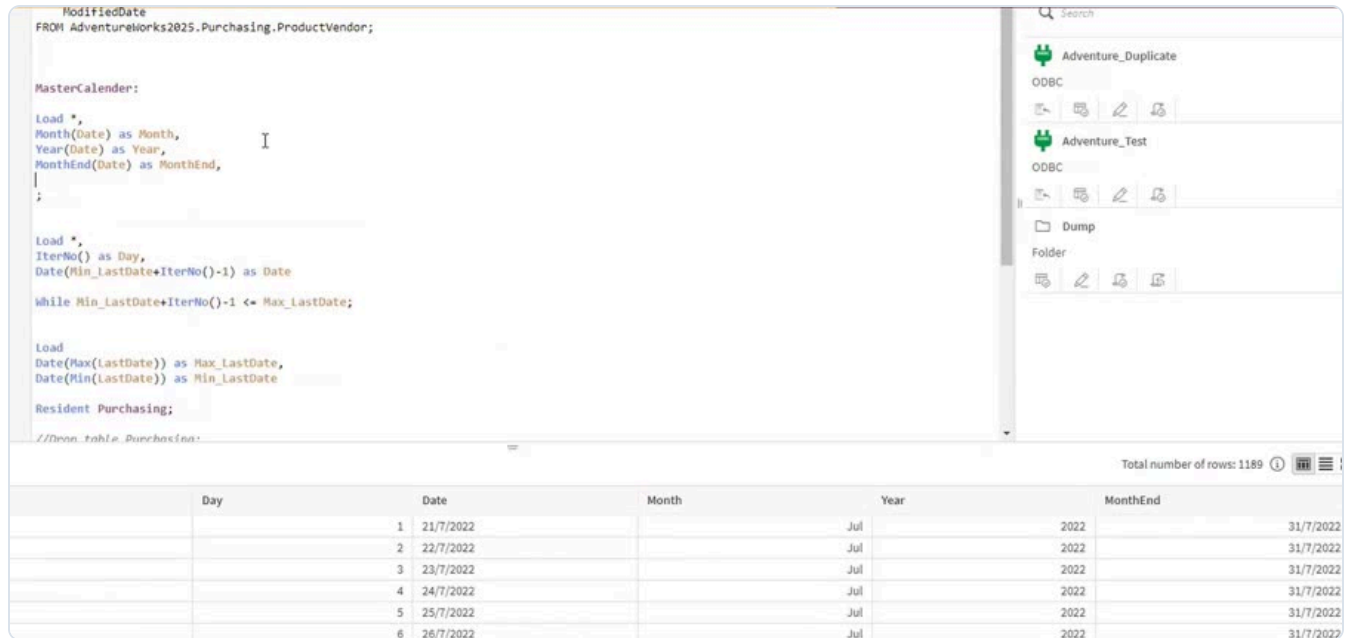
Drop table Purchasing;

Iterno:
LOAD
```

Min_LastDate	Day	Date
21/7/2022		1 21/7/2022
21/7/2022		2 22/7/2022
21/7/2022		3 23/7/2022
21/7/2022		4 24/7/2022
21/7/2022		5 25/7/2022
21/7/2022		6 26/7/2022

Then a preceding load on top generates the dates. Date is Min Last Date plus IterNo minus one, while that value is less than or equal to Max Last Date. The preview shows one row for every day, more than three years of dates, with no gaps.

Finally, one more preceding load adds the calendar fields: Month, Y...



```
ModifiedDate
FROM AdventureWorks2025.Purchasing.ProductVendor;

MasterCalendar:
Load *,
Month(Date) as Month,
Year(Date) as Year,
MonthEnd(Date) as MonthEnd,
;

Load *,
IterNo() as Day,
Date(Min_LastDate+IterNo()-1) as Date
While Min_LastDate+IterNo()-1 <= Max_LastDate;

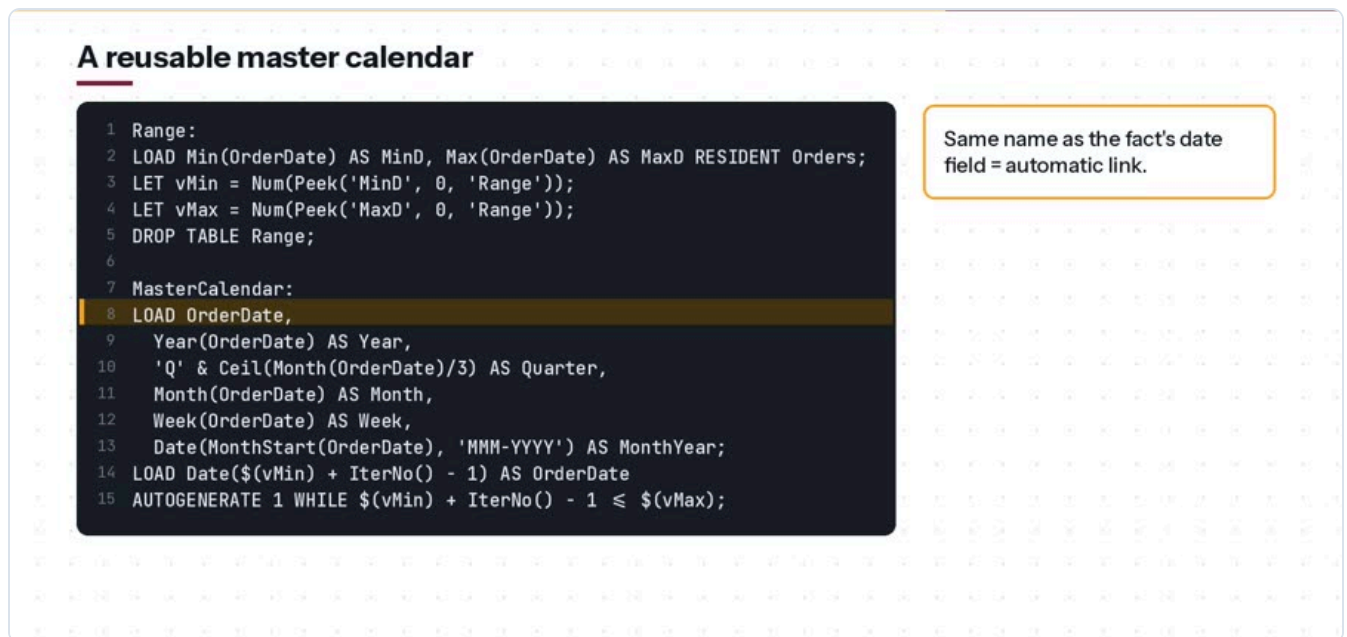
Load
Date(Max_LastDate) as Max_LastDate,
Date(Min_LastDate) as Min_LastDate
Resident Purchasing;

//Drop table MasterCalendar;
```

Day	Date	Month	Year	MonthEnd
1	21/7/2022	Jul	2022	31/7/2022
2	22/7/2022	Jul	2022	31/7/2022
3	23/7/2022	Jul	2022	31/7/2022
4	24/7/2022	Jul	2022	31/7/2022
5	25/7/2022	Jul	2022	31/7/2022
6	26/7/2022	Jul	2022	31/7/2022

Finally, one more preceding load adds the calendar fields: Month, Year, and MonthEnd, each derived from Date. In the preview, every date now carries its month, its year, and the last day of its month.

A reusable master calendar



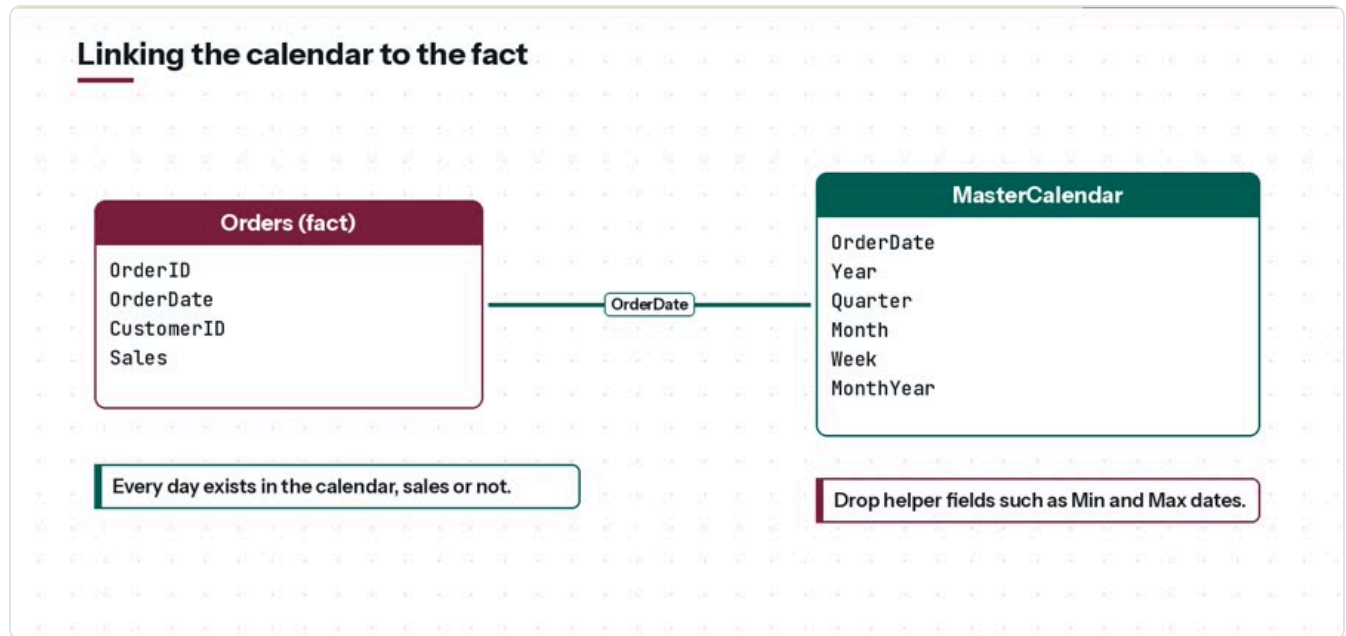
```
A reusable master calendar

1 Range:
2 LOAD Min(OrderDate) AS MinD, Max(OrderDate) AS MaxD RESIDENT Orders;
3 LET vMin = Num(Peek('MinD', 0, 'Range'));
4 LET vMax = Num(Peek('MaxD', 0, 'Range'));
5 DROP TABLE Range;
6
7 MasterCalendar:
8 LOAD OrderDate,
9 Year(OrderDate) AS Year,
10 'Q' & Ceil(Month(OrderDate)/3) AS Quarter,
11 Month(OrderDate) AS Month,
12 Week(OrderDate) AS Week,
13 Date(MonthStart(OrderDate), 'MMM-YYYY') AS MonthYear;
14 LOAD Date$(vMin) + IterNo() - 1 AS OrderDate
15 AUTOGENERATE 1 WHILE $(vMin) + IterNo() - 1 <= $(vMax);
```

Same name as the fact's date field = automatic link.

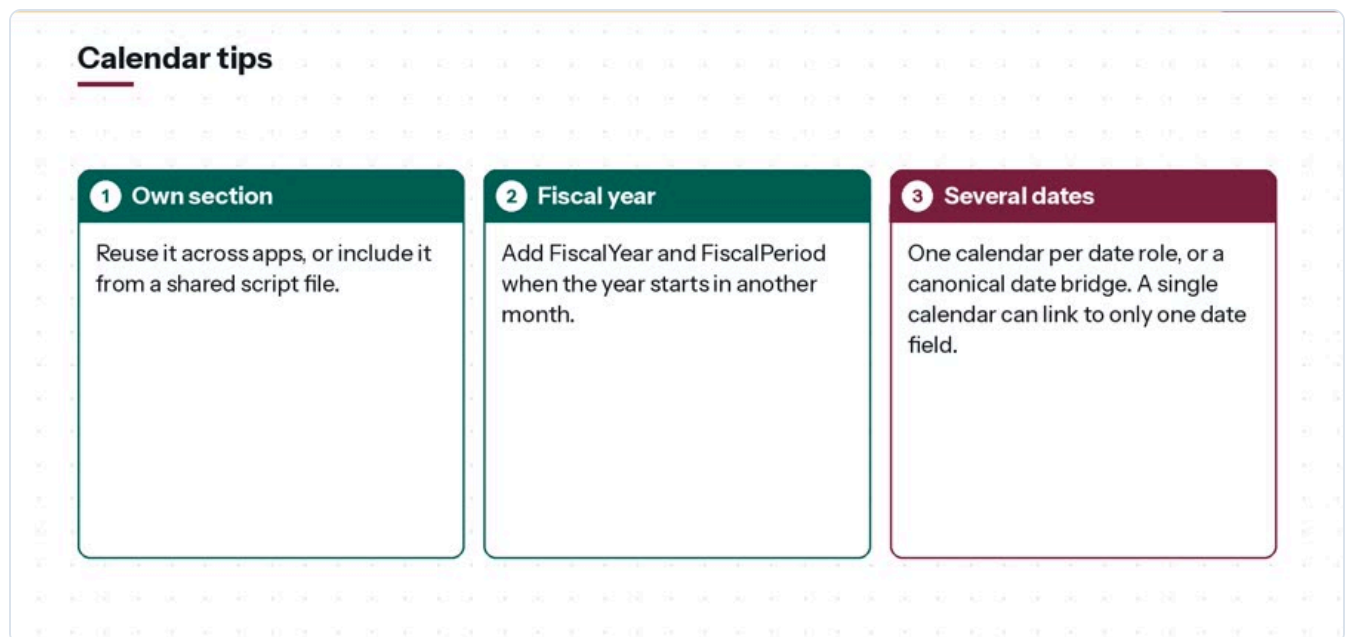
Here's a complete, reusable version. Find the range into variables with Peek. Generate one row per day. Then derive year, quarter, month, week, and a month-year field for trend charts. Name the date field to match the fact table, so the calendar links automatically.

Linking the calendar to the fact



In the data model, the calendar sits beside the fact table as one more dimension, associated on the shared date field. Any chart can now use Year, Quarter, or MonthYear, and filters list every period, even ones with no sales. Keep helper fields, like the min and max dates left in the recorded calendar, out of the final version.

Calendar tips



Three tips from real projects. Keep the calendar in its own script section so it's easy to reuse. If your fiscal year doesn't start in January, add fiscal year and fiscal period fields. And if a fact has several dates, such as order and ship date, use one calendar per date role, or a canonical date bridge, rather than linking one calendar to both.

Key takeaways

- IterNo() + WHILE generates rows
- Min and Max give the date range
- One row per day, with year, quarter, month
- Linked on the date field name

Practice

1. Generate a table of all dates in 2026 with IterNo() and WHILE, and confirm it has 365 rows.
2. Build a master calendar from the min and max order dates of your orders table, with Year, Quarter, Month, Week and MonthYear.
3. Add FiscalYear for a fiscal year starting in October.
4. Load a table with a WHERE clause and add both RecNo() and RowNo(); explain the difference you see.

Check your understanding

1. What does IterNo() return?
2. Why build a master calendar instead of using fact dates?
3. Which counts rows after a WHERE clause: RecNo or RowNo?
4. How does the calendar link to the fact table?

Answers at the back of this handbook.

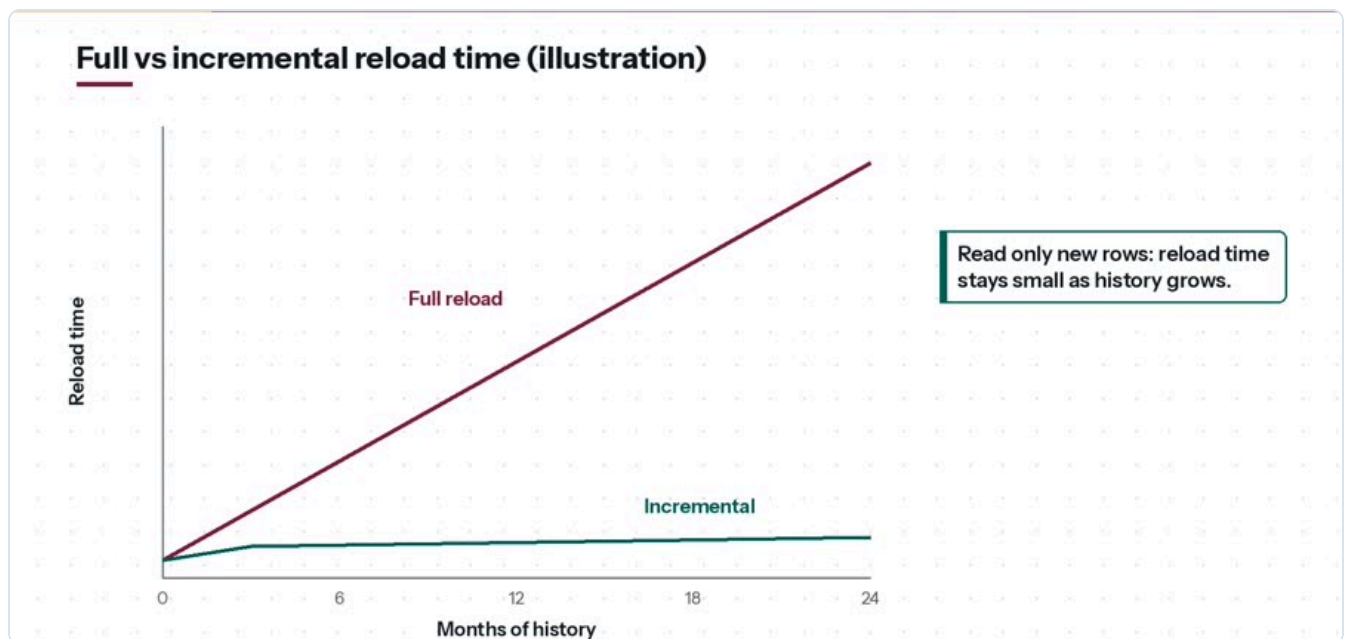
Incremental Loads with QVDs

Video: 09_QT-09_Incremental_Loads_with_QVDs.mp4 · 3:55 · Stage: Development · Audience: Dashboard developers who completed QT-05 and QT-07

Learning objectives

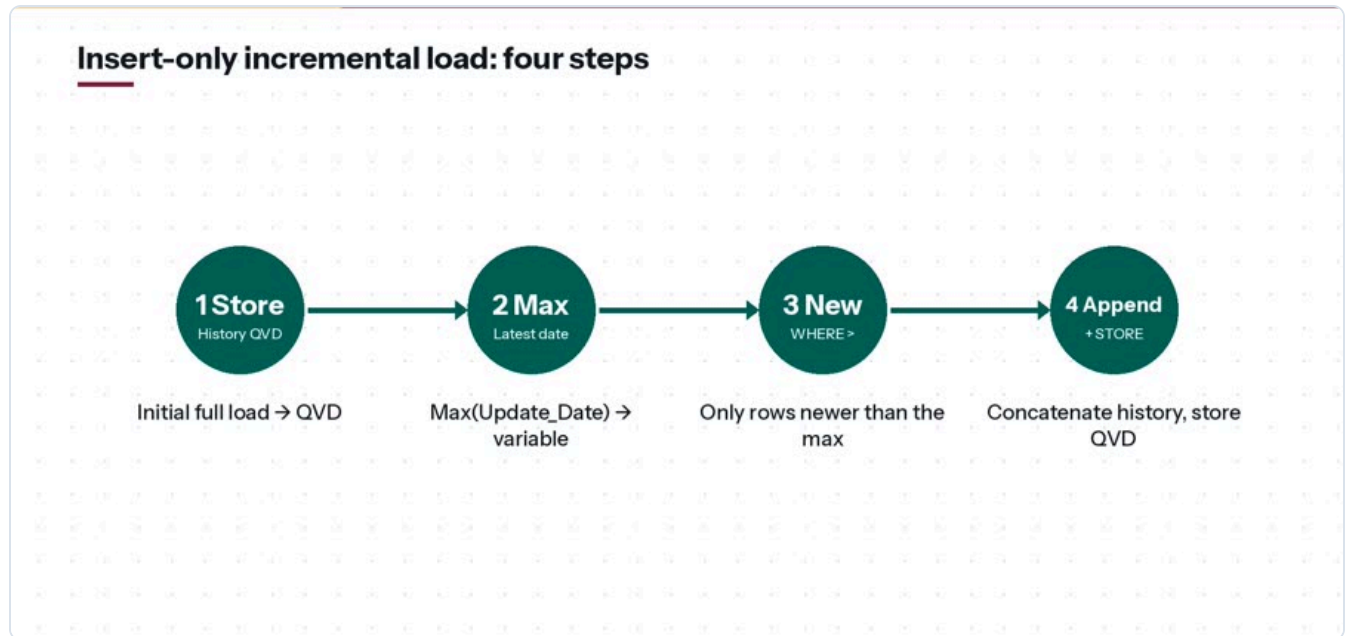
- Explain full vs incremental reloads
- Build an insert-only incremental load
- Extend it for updates and deletes

Full vs incremental reload time (illustration)



Here's why it matters. With a full reload, reload time grows with the size of the table, every single night. With an incremental load, you only read the new rows, so reload time stays small and flat as history grows.

The trainer's procedure has four steps



The trainer's procedure has four steps. One: load the history once and store it as a QVD. Two: read the QVD and capture the latest update date in a variable. Three: query the source for records newer than that date. Four: concatenate the stored history, and store the combined table back to the QVD.

Here are those steps written as comments in the script

```
3 //Step 2: Load Historical data/Historical Qvd and find Latest record(Inter date or Update date) for the filter
4 //Step 3: Query the Latest/current records with help the Latest record value(Inter date or Update date) from h
5 //Step 43: Append the with Historical table/Qvd - Final Fact table with all the transaction data
6
7 // Step 1:
8 // History_Trans:
9 // Load * Inline
10 // [
11 // A,B,C,Update_Date
12 // 1,AA,AAA,05/07/2026
13 // 2,BB,CCC,06/07/2026
14 // ];
15
16 // Store * from History_Trans into [lib://Dump/History_Trans.qvd];
17
18 // Step 2: I
19
20 Trans:
21 LOAD
22 // A,
23 // "B",
24 // C,
25 Max( Update_Date) as Max_Date
26 FROM [lib://Dump/History_Trans.qvd]
```

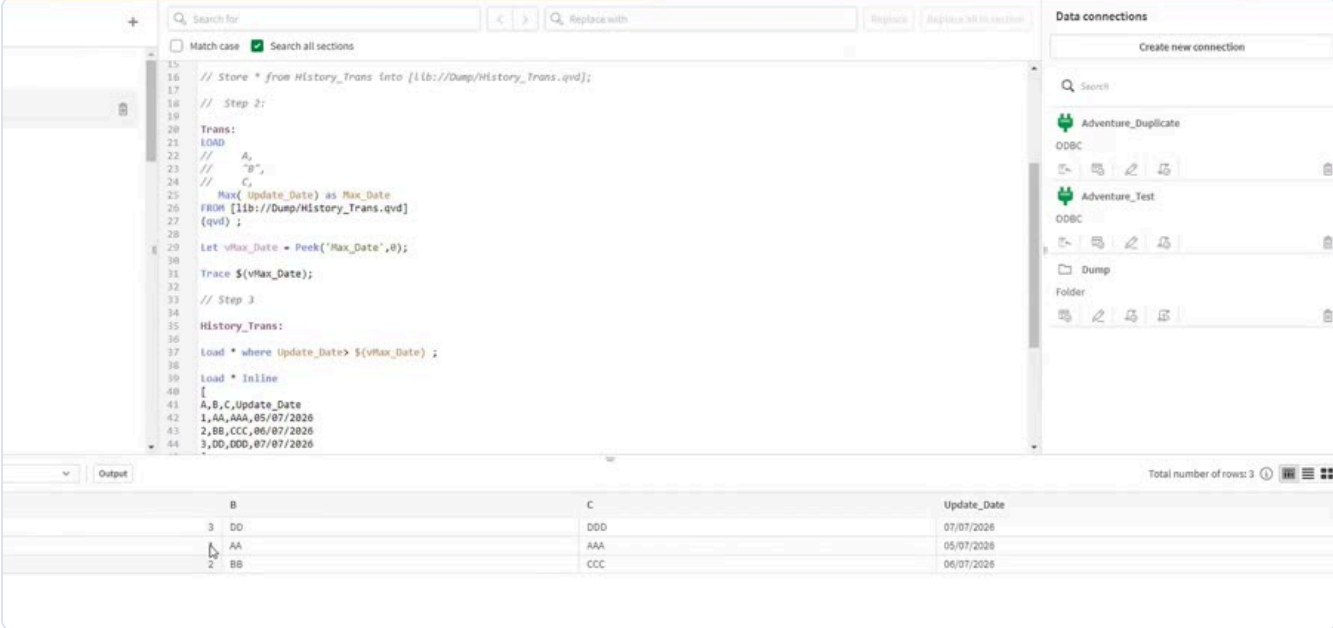
Here are those steps written as comments in the script. Step one, kept commented out after the first run, built a small history table and stored it as History_Trans dot QVD.

Step two loads the maximum update date from the QVD

```
39 Load * Inline
40 [
41 A,B,C,Update_Date
42 1,AA,AAA,05/07/2026
43 2,BB,CCC,06/07/2026
44 3,DD,DDD,07/07/2026
45 ];
46
47 // Step 4:
48 Concatenate(History_Trans)
49
50 Trans:
51 LOAD
52 A,B,C,Update_Date
53 FROM [lib://Dump/History_Trans.qvd]
54 (qvd) ;
55
56 Store * from History_Trans into [lib://Dump/History_Trans.qvd];
57
58
```

Step two loads the maximum update date from the QVD. Peek reads that value into a variable, vMax_Date, and TRACE writes it to the log. Step three loads only rows where the update date is greater than the variable. Step four concatenates the history from the QVD, and the final STORE overwrites it with the combined table.

After a reload, the preview shows the result: three rows



The screenshot shows the Qlik Sense script editor with a script that performs a reload. The script includes a 'Store' statement to save the current data to a QVD, followed by a 'Trans' block that loads the data from the QVD, finds the maximum update date, and filters the data to only include rows with an update date greater than or equal to the maximum date found. The final 'Store' statement saves the filtered data back to the QVD. The output table at the bottom shows three rows of data.

	B	C	Update_Date
3	DD	DDD	07/07/2026
1	AA	AAA	05/07/2026
2	BB	CCC	06/07/2026

After a reload, the preview shows the result: three rows. Row three, dated the seventh of July, is the new record from step three. Rows one and two came from the stored history. On a table this small you won't notice a speed difference, but with millions of rows it's significant.

Insert-only incremental load

Insert-only incremental load

```
1 MaxDate:
2 LOAD Max(UpdateDate) AS MaxD FROM [lib://QVD/Orders.qvd] (qvd);
3 LET vMaxDate = Timestamp(Peek('MaxD', 0, 'MaxDate'),
4 'YYYY-MM-DD hh:mm:ss');
5 DROP TABLE MaxDate;
6
7 Orders:
8 SQL SELECT * FROM dbo.Orders
9 WHERE UpdateDate > '$(vMaxDate)';
10
11 CONCATENATE (Orders)
12 LOAD * FROM [lib://QVD/Orders.qvd] (qvd);
13
14 STORE Orders INTO [lib://QVD/Orders.qvd] (qvd);
```

Append history, then overwrite the QVD.

Here's the same pattern with a real source. Peek reads the max date into a variable. The SQL query uses that variable in its WHERE clause, so the database only returns new rows. Then concatenate the stored history and store the result.

Insert, update and delete

Insert, update and delete

```
1 Orders:
2 SQL SELECT * FROM dbo.Orders
3 WHERE UpdateDate > '$(vMaxDate)';
4
5 CONCATENATE (Orders)
6 LOAD * FROM [lib://QVD/Orders.qvd] (qvd)
7 WHERE NOT Exists(OrderID);
8
9 INNER JOIN (Orders)
10 SQL SELECT OrderID FROM dbo.Orders;
11
12 STORE Orders INTO [lib://QVD/Orders.qvd] (qvd);
```

Keeps only keys that still exist at the source, removing deletes.

If source rows can change, an insert-only load would keep both the old and the new version. Add a primary key check: when loading the history, keep only rows whose key does not already exist in the new data. And if rows can be deleted at the source, finish with an inner join to the list of keys that still exist.

Which pattern?

Which pattern?

Pattern	Source behaviour	Script pieces
Insert only	Rows only added (logs, events)	WHERE date > max; CONCATENATE
Insert + update	Existing rows can change	+ modified date, WHERE NOT Exists(key)
Insert + update + delete	Rows can be removed	+ INNER JOIN to current keys

Choose the pattern by how the source behaves. Insert only, when rows are only ever added, like log files. Insert and update, when existing rows can change, using a modified date and Not Exists. And insert, update, and delete, when rows can also disappear, adding the inner join.

Before you go incremental

Before you go incremental

1 Reliable keys

A modified date and a primary key in the source.

2 Keep a rebuild

Keep the initial full load to recreate the QVD.

3 > not >=

Avoid loading the boundary rows twice.

4 Schedule it

Reload tasks in Qlik Cloud or the QMC.

A few cautions. Your source needs a reliable created or modified date, and a primary key. Keep the first full load in the script, commented out, so you can rebuild the QVD if it's ever lost. Use greater than, not greater than or equal, unless you also remove duplicates. And schedule the reload, in Qlik Cloud or the Management Console, at the cadence the business agreed in requirements.

Key takeaways

- Read only new or changed rows
- Peek the max date → WHERE
- Concatenate history, STORE the QVD
- NOT Exists for updates; INNER JOIN for deletes

Practice

1. Store a small inline orders table as a QVD (the initial full load), then comment that step out.
2. Add an inline table of two new rows and build steps 2–4 of the insert-only pattern; reload and check the QVD now holds all rows.
3. Change one existing row's amount and modified date, then extend the script with WHERE NOT Exists(OrderID) so only the new version is kept.

Check your understanding

1. What does Peek() return in step two?
2. Why use > rather than >= in the WHERE clause?
3. Which clause handles updated rows?
4. How are deleted source rows removed?

Answers at the back of this handbook.

Designing Qlik Sheets & Visualizations

Video: 10_QT-10_Designing_Qlik_Sheets_and_Visualizations.mp4 · 4:29 · Stage: Development · Audience: Dashboard developers who completed QT-03 to QT-06

Learning objectives

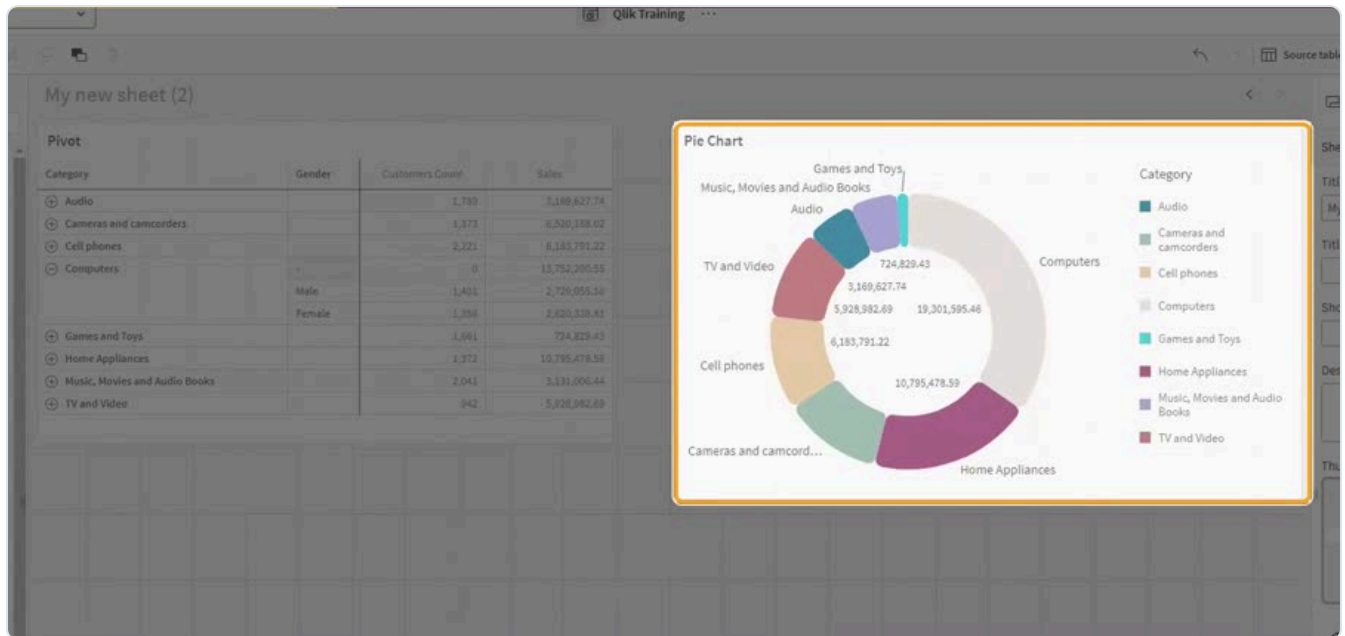
- Choose the right chart for the question
- Configure measures, limits, colours and labels
- Add reference lines, combo charts and pivots

Pick the chart from the question

Pick the chart from the question		
Question	Best chart	Tip
How do categories compare?	Bar chart	Sort by value; limit to the top N
How is it trending?	Line chart	Time on the x-axis, in order
What's the headline number?	KPI	Add a comparison, e.g. last year
What share does each part have?	Pie / donut	Five or fewer slices
Numbers by two dimensions?	Pivot table	Expand and collapse rows
Two measures, different scales?	Combo chart	Second measure on a secondary axis

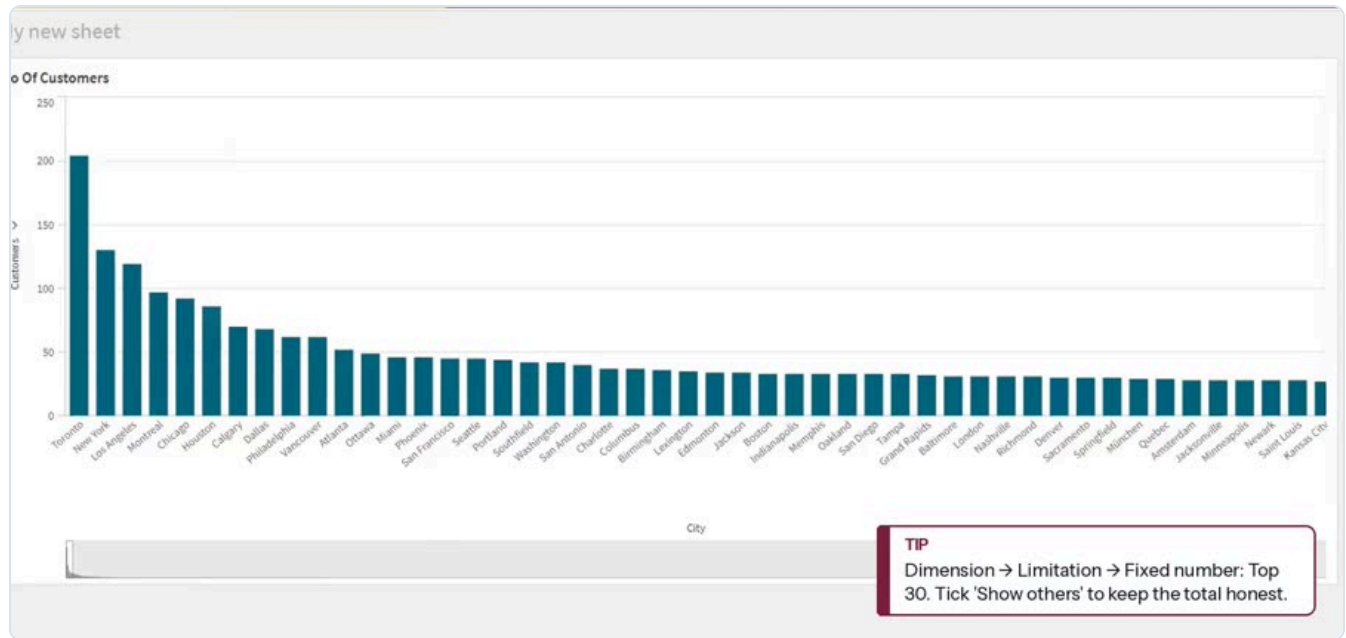
Start with the question, not the chart. To compare categories, use a bar chart. To show a trend over time, use a line chart. For a single headline number, use a KPI. To show parts of a whole with only a few slices, a pie or donut works. To slice numbers by two dimensions, use a pivot table. And to compare two measures with different scales, use a combo chart.

In edit mode, the Charts panel lists every visualization: bar, comb...



In edit mode, the Charts panel lists every visualization: bar, combo, KPI, line, pie, pivot table, and more, plus dashboard objects like buttons and containers. Drag one onto the sheet, then set it up in the properties panel. This sheet already holds a pivot table and a donut chart.

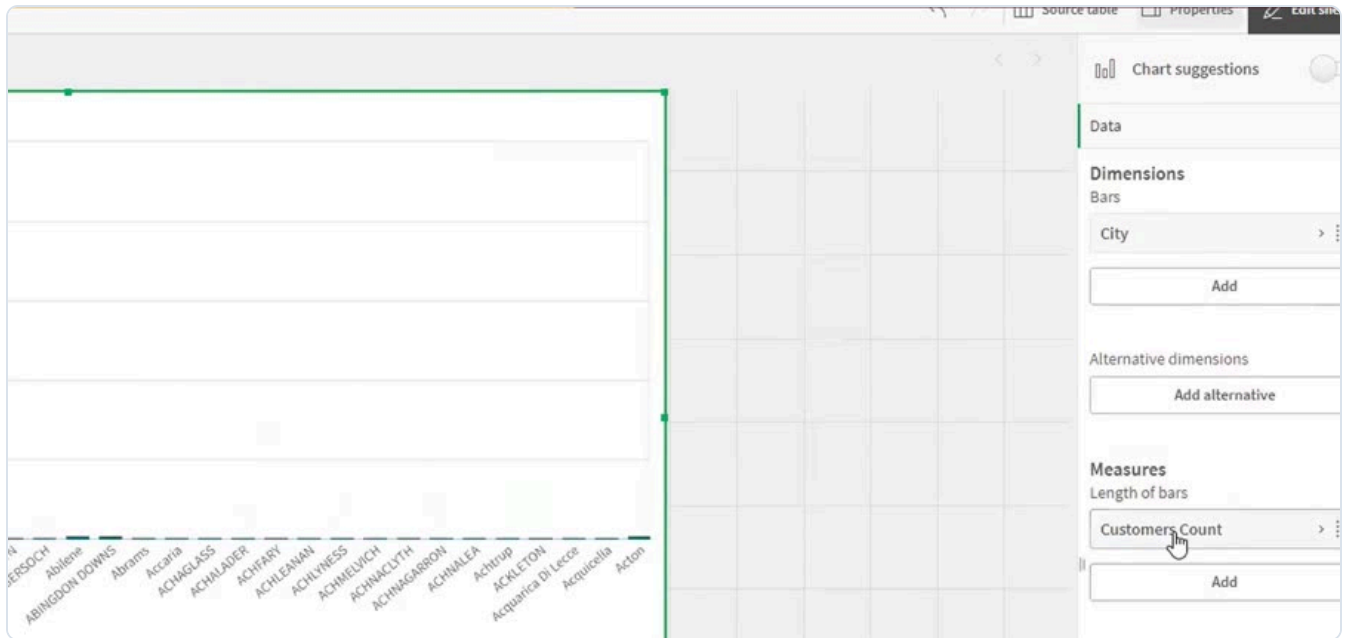
Here's a bar chart of customers by city



Here's a bar chart of customers by city. It's sorted by value, which is right for comparison. But with hundreds of cities, the bars become a long tail, and Qlik adds a scroll bar. The fix is in the dimension settings: a limit, such as the top thirty values, with everything else grouped as Others.

Tip Dimension → Limitation → Fixed number: Top 30. Tick 'Show others' to keep the total honest.

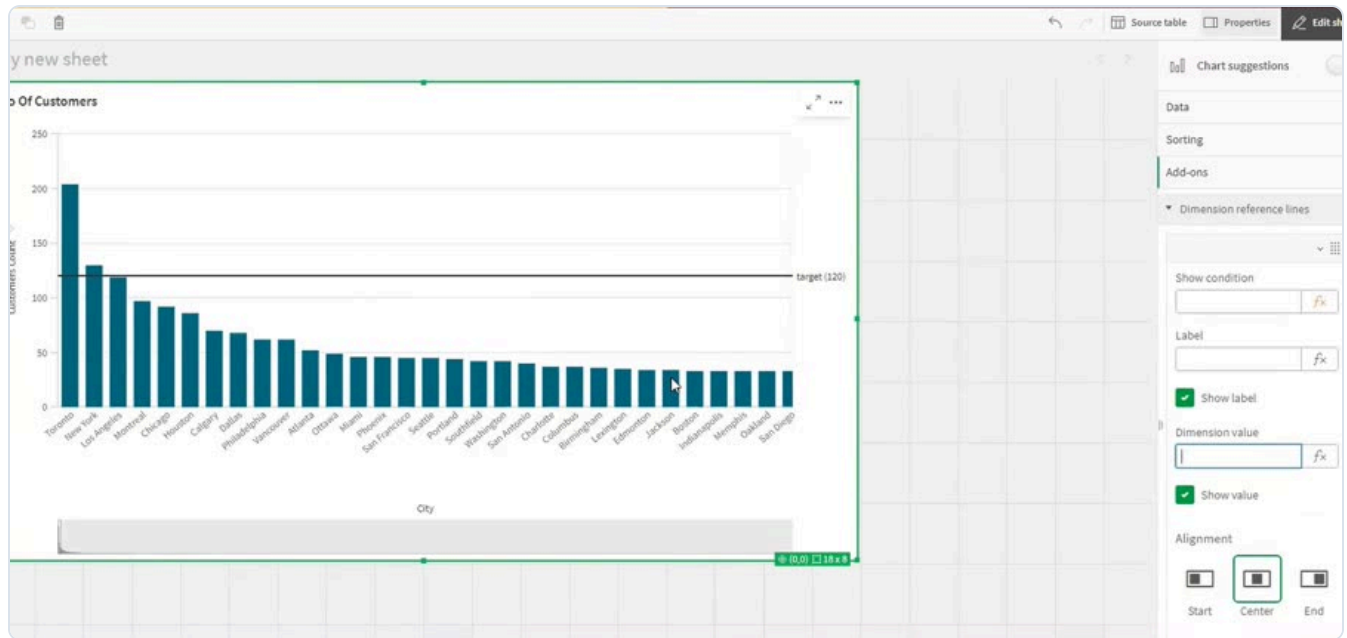
In the Data section, the bars use City as the dimension and Custome...



In the Data section, the bars use City as the dimension and Customers Count as the measure. Below, the trainer added an alternative measure, Birthday Count. Alternative dimensions and measures let users switch what the chart shows, without you building a second chart. Customers Count is also saved as a master measure, on the left, so it's defined once and reused everywhere.

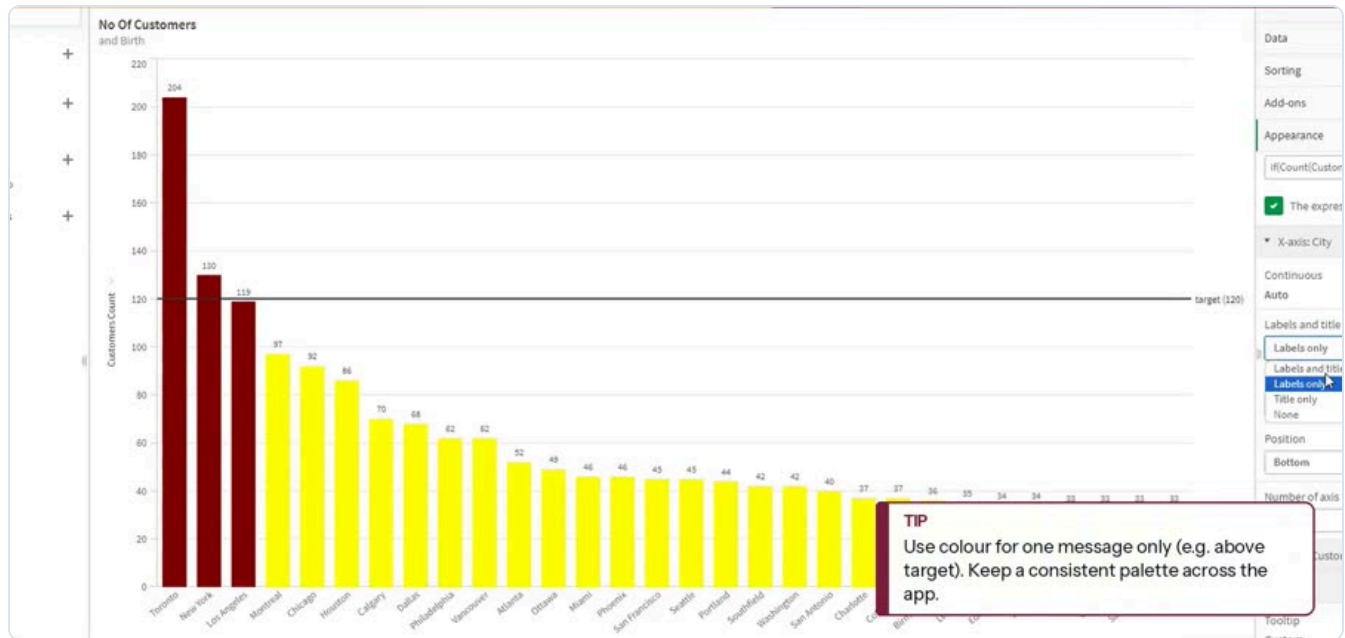
Tip Users switch alternatives from the chart's exploration menu or axis label.

Under Add-ons, a reference line draws a target across the chart



Under Add-ons, a reference line draws a target across the chart. Here it's set at one hundred and twenty customers, so it's instantly clear which cities are above target. Dimension reference lines, just below, mark a specific value on the x-axis instead.

Colour can carry meaning too



Colour can carry meaning too. In Appearance, the trainer colours by expression: if the customer count is over one hundred, the bar is dark red, otherwise yellow, and ticks the box that says the expression is a colour code. Value labels sit on top of each bar, so nobody has to read the axis.

Tip Use colour for one message only (e.g. above target). Keep a consistent palette across the app.

Reference lines can also be calculated



Reference lines can also be calculated. Here the dimension value is an expression using FirstSortedValue, which returns the region ranked first by a measure, and marks it with a dashed line that follows the data as it changes. Notice one slip: the title says customers, but the axis shows sales. Keep every title in step with what's plotted.

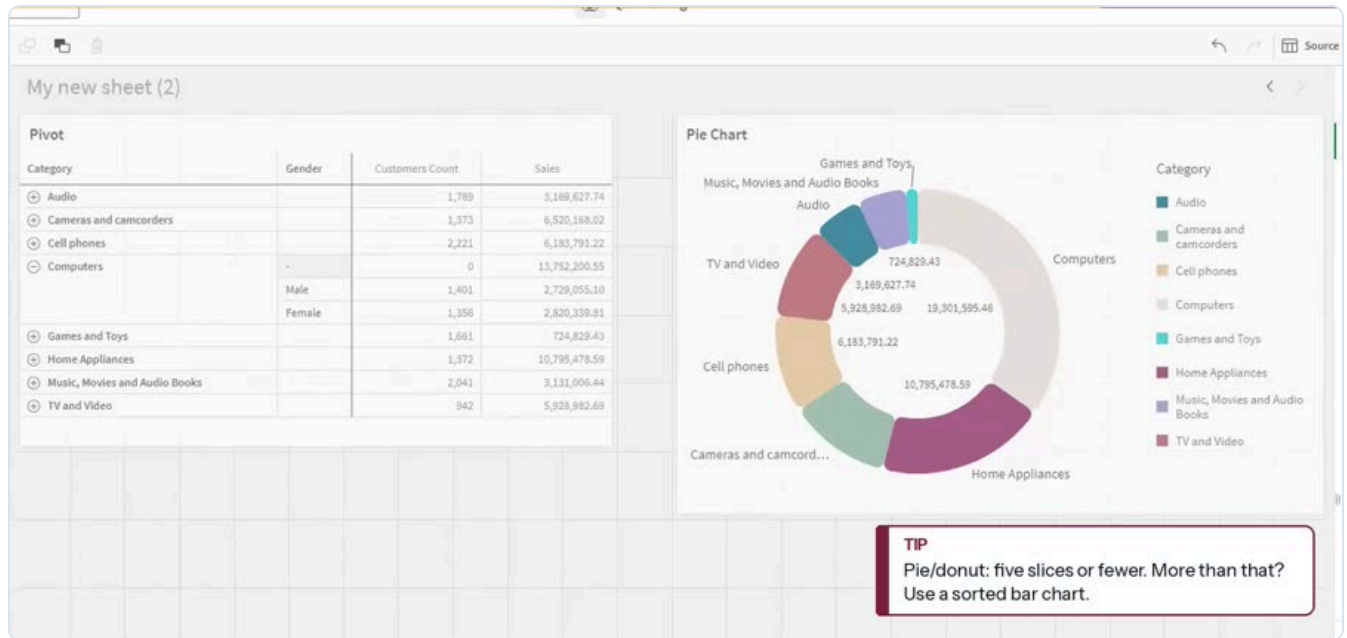
Tip FirstSortedValue(field, -weight): the minus sign returns the top-ranked value.

A combo chart combines bars and a line



A combo chart combines bars and a line. Here, sales are drawn as a line and customer count as bars, by region. Because the two measures have very different scales, one uses the secondary axis on the right. Each measure's settings decide whether it's a bar or a line, and which axis it uses.

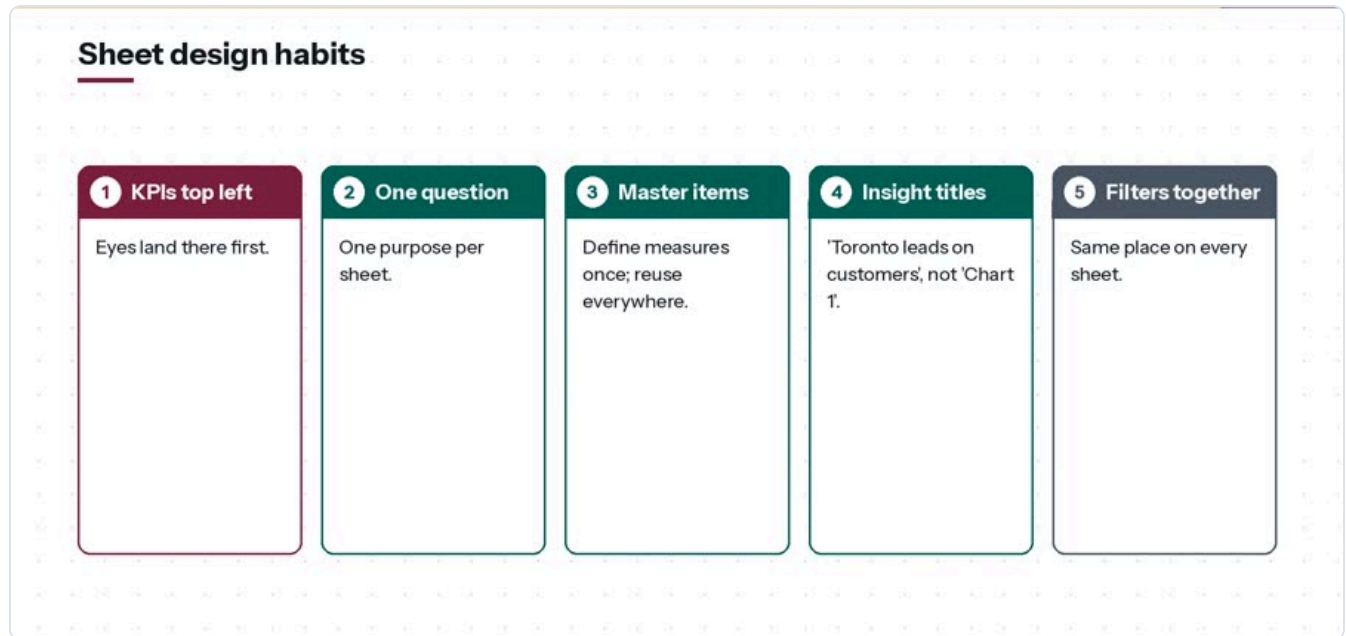
The pivot table shows category and gender



The pivot table shows category and gender. Expand a row with the plus sign to drill into it, and collapse it with the minus sign. The donut beside it shows each category's share of sales, with labels on the slices. With eight categories it's already busy, which is why bars are often clearer.

Tip Pie/donut: five slices or fewer. More than that? Use a sorted bar chart.

Sheet design habits



Finally, a few sheet design habits. Put the most important KPIs at the top left, where eyes land first. Keep one question per sheet. Use master items so measures are defined once and look the same everywhere. Give every chart a clear title that states the insight. And keep filters together in a consistent place.

Key takeaways

- Pick the chart from the question
- Limit long dimensions; offer alternatives
- Reference lines and colour show what matters
- Combo charts and pivots for richer views

Practice

1. Build a bar chart of customers by city, limited to the top 15 with Others shown.
2. Add a reference line at your target value and colour bars above target by expression.
3. Build a combo chart with sales as bars and order count as a line on a secondary axis.

Check your understanding

1. Which chart suits a trend over time?
2. Where do you add a target line?
3. What lets users switch a chart's measure without a second chart?
4. Why use a secondary axis in a combo chart?

Answers at the back of this handbook.

Set Analysis & Alternate States

Video: 11_QT-11_Set_Analysis_and_Alternate_States.mp4 · 4:26 · Stage: Development · Audience: Dashboard developers comfortable with Qlik expressions (QT-10)

Learning objectives

- Read and write set expressions
- Build current year vs previous year KPIs
- Compare two selections with alternate states

Anatomy of a set expression

The diagram illustrates the components of a Qlik set expression within an aggregation. It shows the following structure:

```
1 Sum( {$ <Year_Order = {2021}> } Quantity )
```

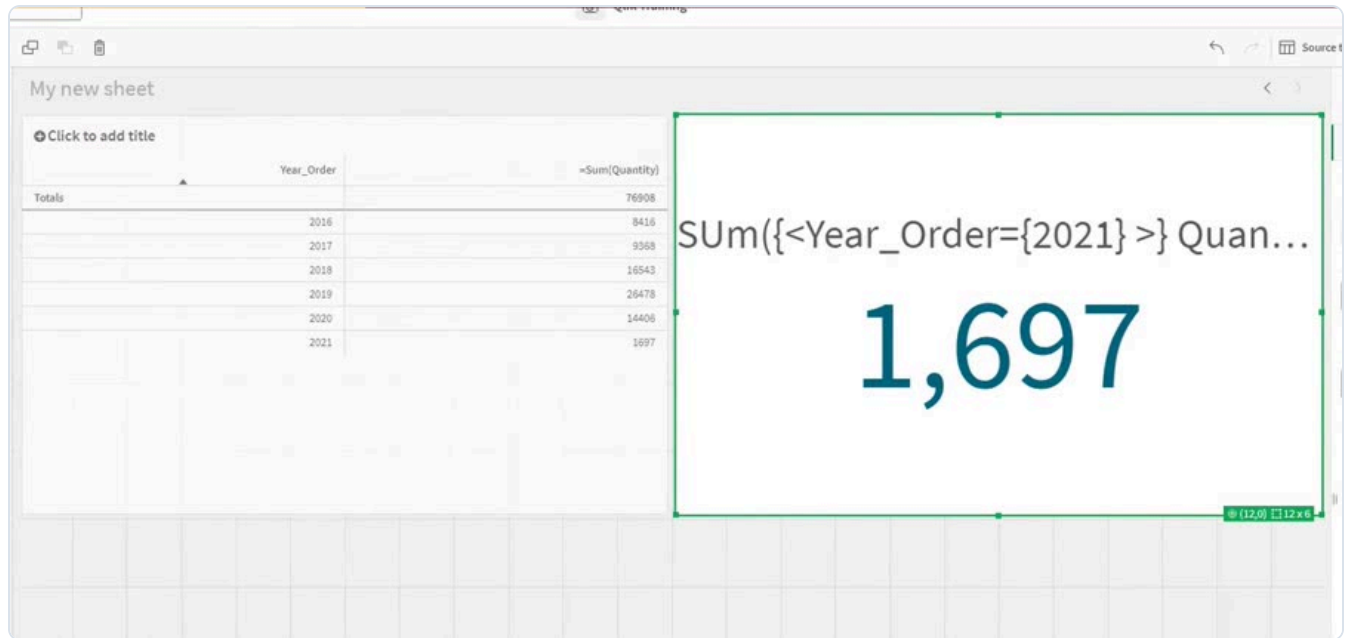
The components are identified as follows:

- set expression**: The curly braces `{ ... }` containing the set definition.
- identifier**: The dollar sign `$` indicating the current selection state.
- modifier**: The angle brackets `< ... >` used to override selections for specific fields.
- Quantity field**: The field being aggregated.

A callout box states: "Then the field, as in any aggregation."

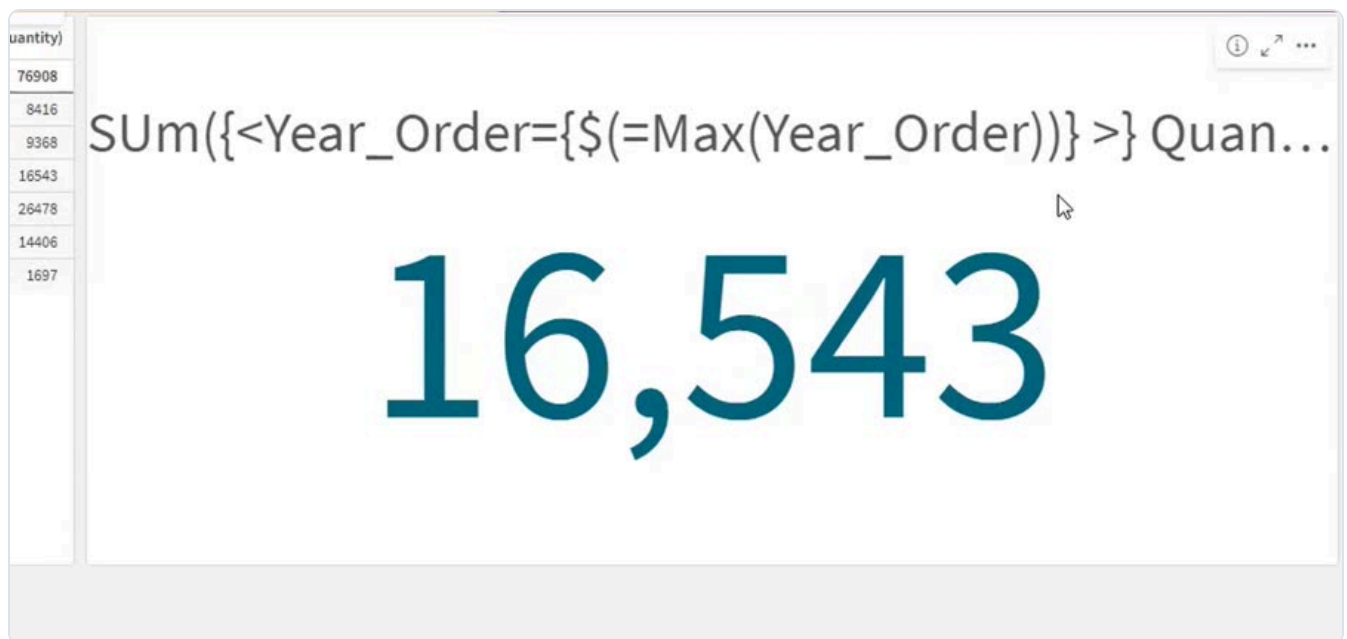
Here's the anatomy of a set expression. It sits inside the aggregation, wrapped in curly braces, before the field. The identifier, a dollar sign, means start from the current selections. The modifier, in angle brackets, overrides selections for specific fields: here, year equals twenty twenty-one. Then comes the field to aggregate, quantity.

In the recording, the trainer builds a KPI with exactly that expres...



In the recording, the trainer builds a KPI with exactly that expression, leaving out the dollar sign because it's the default: sum of quantity where year order is twenty twenty-one. The table on the left shows quantity for every year, and the KPI shows only the twenty twenty-one figure: one thousand six hundred ninety-seven.

Hard-coding a year means editing the expression every January



Hard-coding a year means editing the expression every January. Instead, use dollar-sign expansion. Dollar, open bracket, equals Max of year order, is calculated first, then pasted into the modifier. With twenty sixteen and twenty eighteen selected, the latest selected year is twenty eighteen, so the KPI shows sixteen thousand five hundred forty-three.

Current year vs previous year

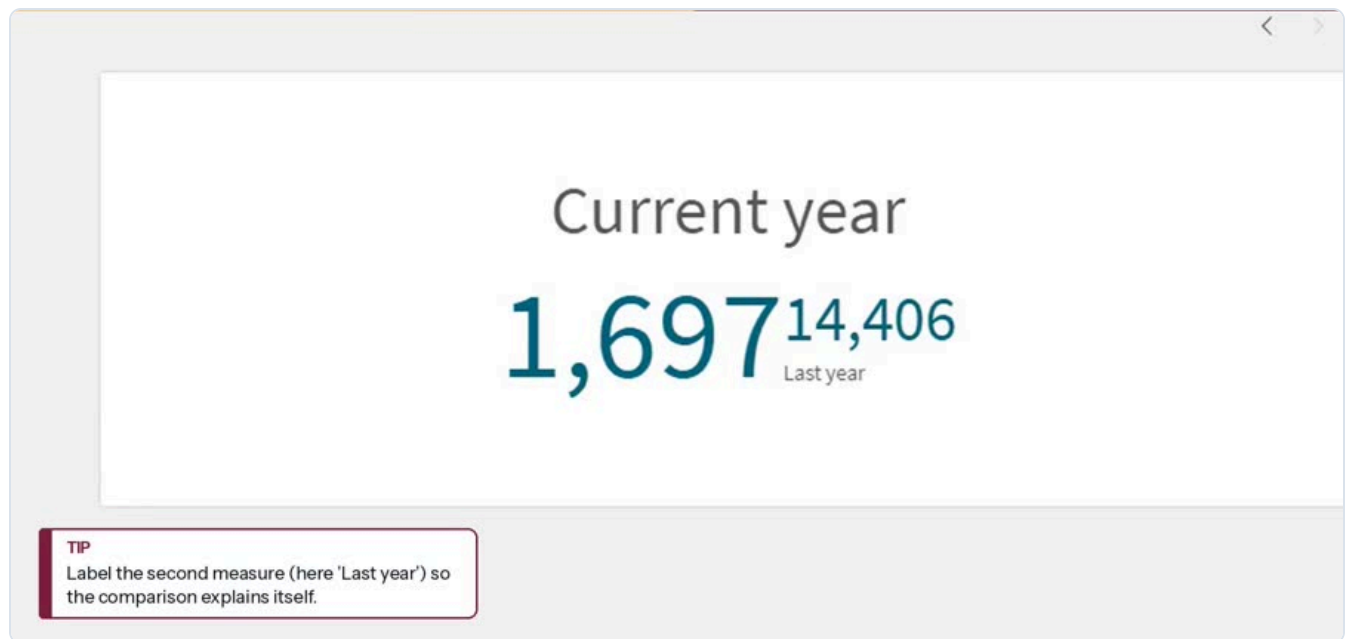
Current year vs previous year

```
1 // Current year
2 Sum({<Year_Order = {$(=Max(Year_Order))}>} Quantity)
3
4 // Previous year
5 Sum({<Year_Order = {$(=Max(Year_Order)-1)}>} Quantity)
```

Save both as master measures.

The same idea gives you current year and previous year side by side. The previous-year measure simply subtracts one inside the dollar expansion. Wrap the field name in square brackets if it contains spaces, and keep both measures in master items so every sheet uses the same definition.

Here's the result: a KPI titled Current year, showing one thousand ...



Here's the result: a KPI titled Current year, showing one thousand six hundred ninety-seven, with last year's fourteen thousand four hundred six as a second measure underneath. Users get the comparison without making any selection at all.

Tip Label the second measure (here 'Last year') so the comparison explains itself.

Set identifiers

Set identifiers		
Identifier	Starts from	Example
\$	Current selections (default)	Sum({\$} Sales)
1	All data; ignores selections	Sum({1} Sales) → grand total
\$1	Previous selection state (Back)	Sum({\$1} Sales)
[Group 1]	An alternate state	Sum({[Group 1]} Sales)

The identifier decides which selections you start from. Dollar sign means the current selections. One means the whole data set, ignoring every selection. Dollar one is the previous selection state, the one you'd return to with Back. And a state name, like Group one, starts from that alternate state's selections.

The P and E functions select values indirectly

TIP
Sum({<Name = P(Name)>} Quantity) ·
Sum({<Name = E(Name)>} Quantity)

Currency Code	Excluded USD	Possible	Exclusion
Totals	4	9	56146
AUD	0	0	4603
CAD	0	0	3929
EUR	0	0	10667
GBP	4	4	4669
USD	0	5	32278

The P and E functions select values indirectly. P returns the possible values of a field, and E returns the excluded values. With five customers selected, this table shows quantity for the possible customers, and for everyone excluded, in separate columns by currency.

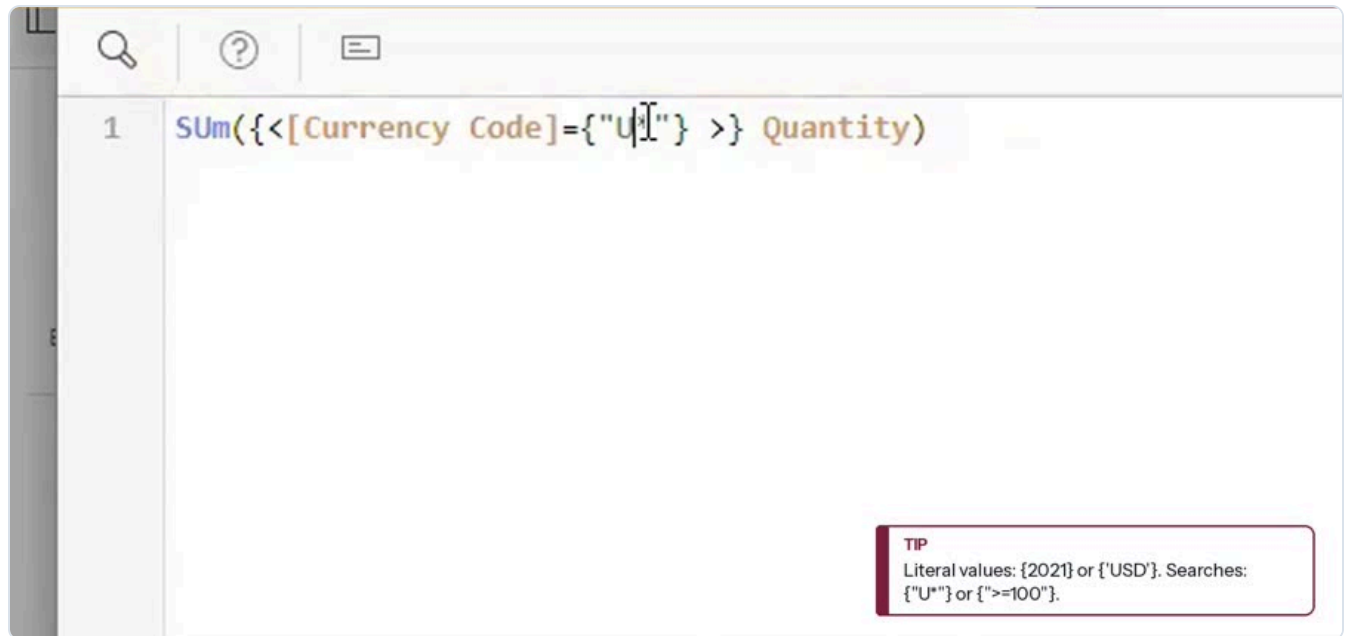
Tip Sum({<Name = P(Name)>} Quantity) · Sum({<Name = E(Name)>} Quantity)

Watch the expression editor as you type

Error in expression
Error in set modifier ad hoc element list: ';' or ')' expected
Sum({<Name={A*}>} Quantity)

Watch the expression editor as you type. Here the trainer searched for names starting with A, but left out the quotes. The panel at the bottom reports an error in the set modifier element list.

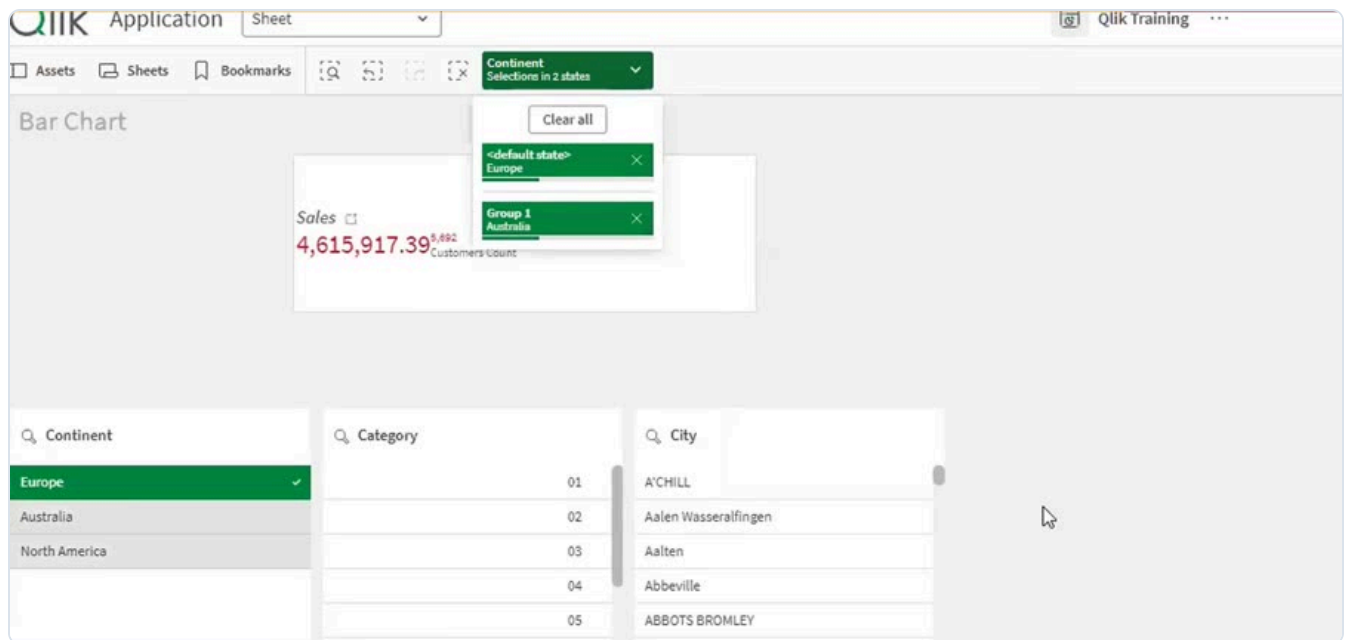
Search strings go in double quotes



Search strings go in double quotes. With currency code equals U star in quotes, the editor shows OK, and the expression returns only currencies starting with U.

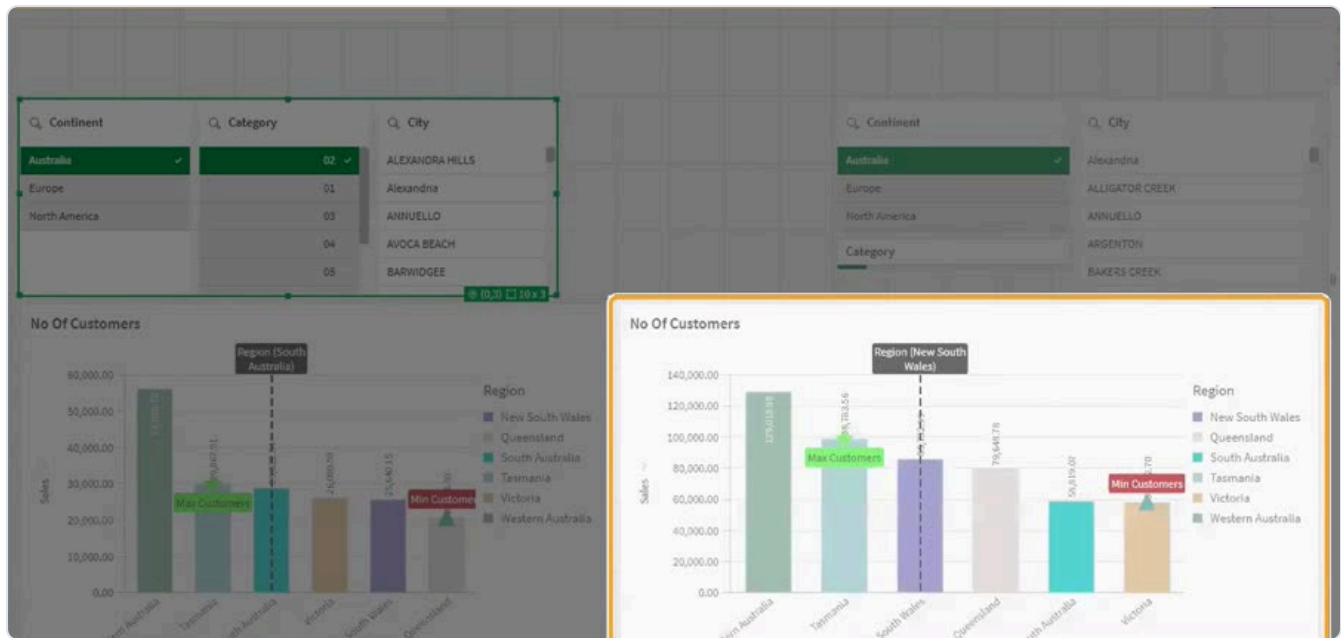
Tip Literal values: {2021} or {'USD'}. Searches: {"U*"} or {">=100"}.

Alternate states let two sets of selections exist at once



Alternate states let two sets of selections exist at once. Here the left filter panes use the default state, with Europe selected, and the right filter panes use Group one, with Australia selected. The selections bar lists both states separately.

Create states under Master items, Alternate states



Create states under Master items, Alternate states. Then assign a chart or filter pane to a state in its Appearance settings. The two identical charts below now show different numbers: one follows the default state, and the other follows Group one.

Key takeaways

- {identifier <modifier>} field
- $\$(=Max(Year))$ instead of hard-coded years
- P() possible, E() excluded
- Alternate states compare two selections

Practice

1. Build a KPI for the current year's sales using $\$(=Max(Year))$, with last year as a second measure.
2. Add a 'share of total' measure using $Sum(Sales)/Sum(\{1\} Sales)$.
3. Create an alternate state, assign one filter pane and one chart to it, and compare two regions side by side.

Check your understanding

1. What does the identifier 1 mean?
2. What does $\$(=\text{Max}(\text{Year}))$ do?
3. Which function returns excluded values?
4. Why did $\{A^*\}$ fail?

Answers at the back of this handbook.

Securing, Optimising & Deploying Qlik Apps

Video: 12_QT-12_Securing_Optimising_and_Deploying_Qlik_Apps.mp4 · 4:17 · Stage: Unit testing, UAT & Go-live ·

Audience: Dashboard developers preparing an app for production

Learning objectives

- Apply front-end and script performance practices
- Restrict data with Section Access
- Promote an app from development to production

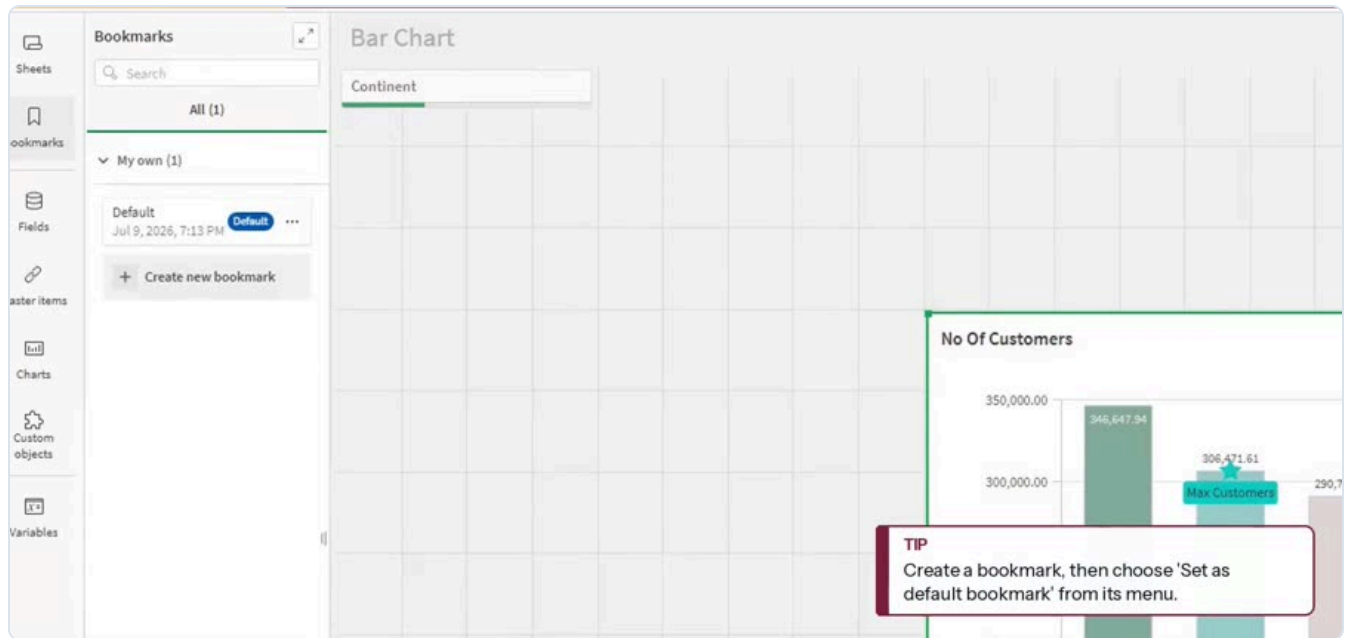
Why is my dashboard slow? The trainer's three fixes

Why is my dashboard slow? The trainer's three fixes

- 1 Set analysis, not nested IFs**
Sum({<Status={'Open'}>} Amount)
instead of
Sum(If(Status='Open', Amount))
- 2 No calculated dimensions**
Build the field once in the load script,
not on every click.
- 3 Default bookmark**
Open the app filtered to a sensible
starting point.

When a learner asked why dashboards load slowly, the trainer gave three answers. Replace nested if statements in chart expressions with set analysis. Avoid calculated dimensions, and create those fields in the load script instead. And set a default bookmark, so the app opens with sensible selections rather than calculating everything at once.

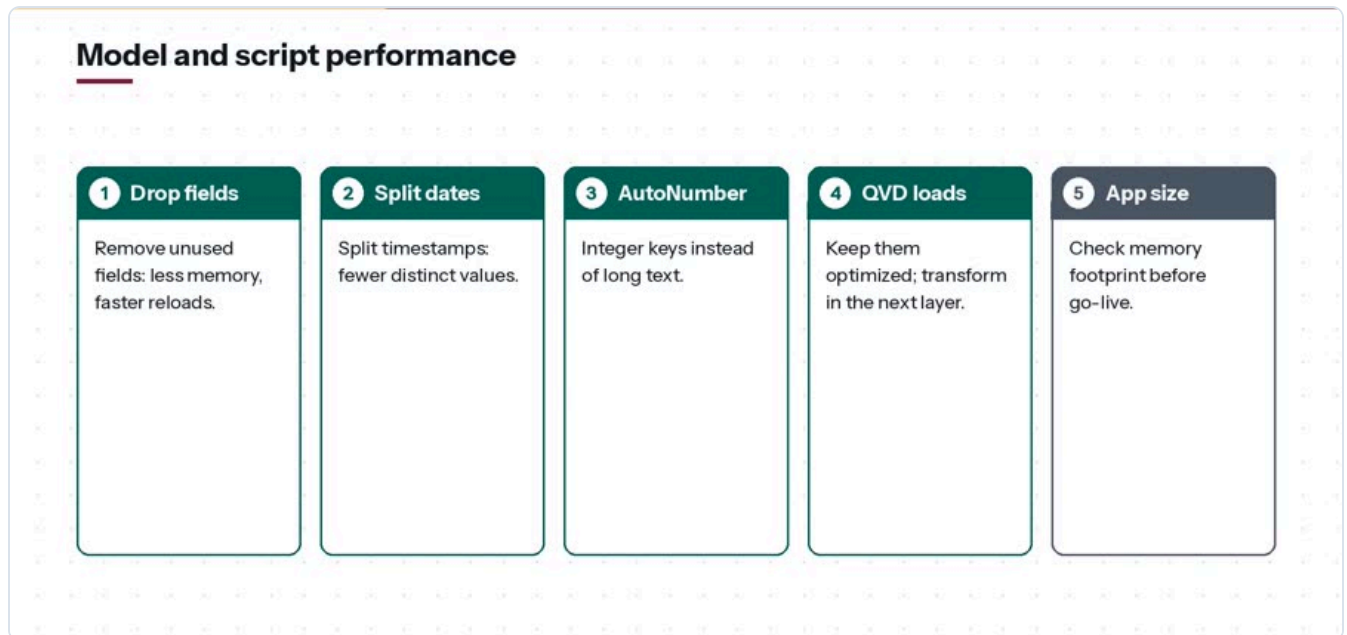
Here's the default bookmark in the recording



Here's the default bookmark in the recording. In the Bookmarks panel, the trainer's bookmark carries the Default label, so every user who opens the app starts from those selections.

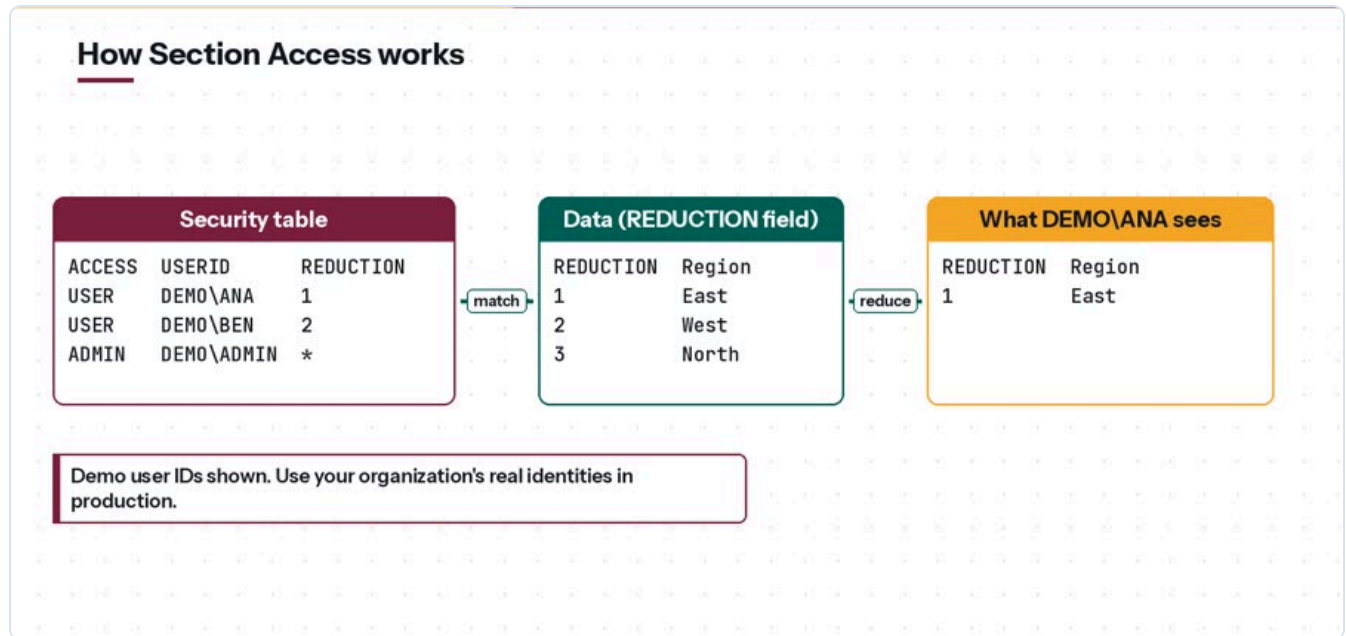
Tip Create a bookmark, then choose 'Set as default bookmark' from its menu.

Model and script performance



Script and model habits matter just as much. Drop fields nobody uses. Split timestamps into date and time. Replace long text keys with AutoNumber. Keep QVD loads optimized. And check the app's size and memory footprint before it goes live.

How Section Access works



Section Access controls who can open an app and which rows each person sees. It's a security table loaded in the script. Each row names a user, their access level, and a reduction value. When a user opens the app, Qlik keeps only the rows whose reduction field matches their value, and removes everything else before they see a single chart.

Section Access in script (demo IDs)

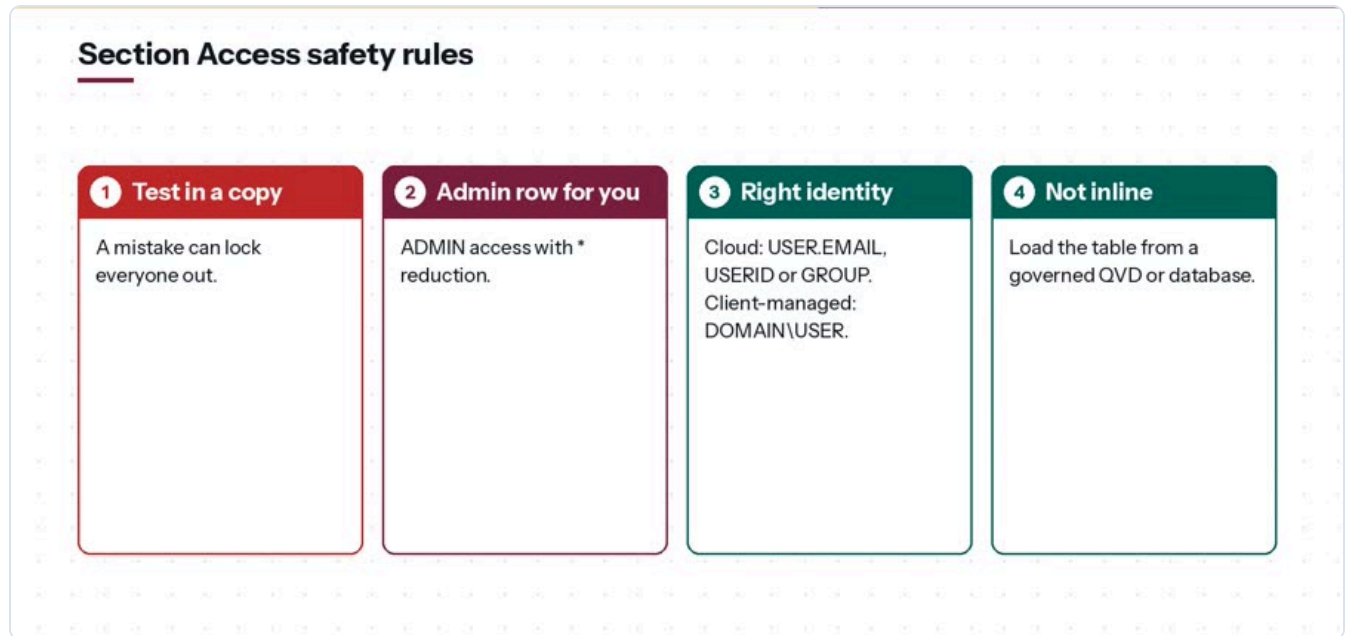
Section Access in script (demo IDs)

```
1 Section Access;
2 LOAD * INLINE [
3 ACCESS, USERID, REDUCTION
4 USER, DEMO\ANA, 1
5 USER, DEMO\BEN, 2
6 ADMIN, DEMO\ADMIN, *
7 ADMIN, INTERNAL\SA_SCHEDULER, *
8 ];
9 Section Application;
10
11 Sales:
12 LOAD *, RegionID AS REDUCTION
13 FROM [lib://QVD/Transform/Sales.qvd] (qvd);
```

Client-managed only: keep
INTERNAL\SA_SCHEDULER or
reloads fail.

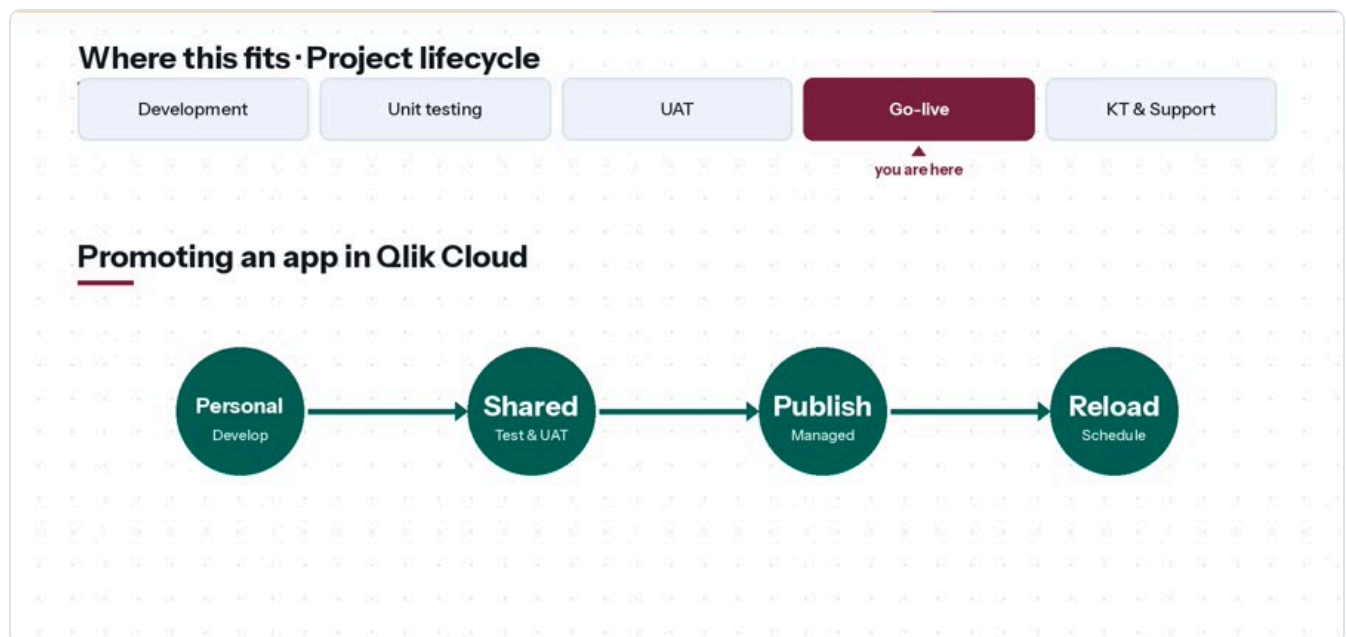
In script, the security table sits between Section Access and Section Application. Field names and values must be upper case. The reduction field must also exist, with the same upper-case name, in your data. A star in the reduction column means all values listed in this table, not every value in the data. In client-managed Qlik Sense, keep the scheduler account as an admin row, or scheduled reloads will fail.

Section Access safety rules



Section Access can lock you out of your own app, so follow a few rules. Always test in a copy of the app. Include an admin row for yourself. In Qlik Cloud, identify users with USER.EMAIL, USERID, or GROUP; in client-managed Qlik Sense, use domain and user name. And keep the security table in a QVD or database table, not typed inline, in production.

Now, deployment



Now, deployment. Build in a personal space. Share the app in a shared space so testers can validate it. Then publish it to a managed space, where business users consume it, and changes are controlled. Finally, schedule reloads so the data stays current, in the right order across your QVD layers.

Qlik Cloud vs client-managed Qlik Sense

Qlik Cloud vs client-managed Qlik Sense

Step	Qlik Cloud	Client-managed Qlik Sense
Organize	Personal, shared, managed spaces	My work, streams
Publish	Publish to a managed space	Publish to a stream (QMC)
Reload	Scheduled reloads, task chains	QMC reload tasks with triggers
Access	Space roles + Section Access	Security rules + Section Access

If your organization runs client-managed Qlik Sense instead of Qlik Cloud, the same steps use different tools. Spaces become streams. Publishing goes to a stream through the QMC, the management console. Reloads are scheduled as reload tasks, chained so each layer waits for the one before. And access is controlled by security rules, plus Section Access in both platforms.

Go-live checklist

Go-live checklist

1 KPIs reconcile Match the source totals.	2 Access tested Each role sees only its rows.	3 Reloads on time Right order across layers.	4 Sheets ready Public, titled, ordered.	5 User guide Smooth handover to support.
---	---	--	---	--

Before go-live, run through a short checklist. Every KPI reconciles to its source. Section Access is tested with real user accounts. Reloads succeed on schedule, in the right order. Sheets are public, titled, and in a sensible order. And users have a short guide, so the handover to support is smooth.

Key takeaways

- Set analysis and script-side fields for speed
- A default bookmark for a fast first view
- Section Access reduces data per user
- Personal → shared → managed, with scheduled reloads

Practice

1. Find one nested IF in a chart expression and rewrite it with set analysis.
2. In a copy of an app, add Section Access with two demo users and an admin row, reduced by region. Test that each user sees only their region.
3. Write the promotion path for one of your apps: which space or stream it lives in at each stage, and who can see it.

Check your understanding

1. Why avoid calculated dimensions?
2. Which section ends the security table?
3. What case must Section Access fields and values use?
4. Where do business users consume production apps in Qlik Cloud?

Answers at the back of this handbook.

Answer key

QT-07 • Script Variables & Transformation Functions

1. The text exactly as written
2. The evaluated result
3. Replaces the variable with its contents before calculation
4. Date#()
5. CD

QT-08 • Building a Master Calendar

1. The iteration count of the current WHILE loop, starting at 1
2. Fact dates have gaps; a calendar has every day
3. RowNo()
4. By sharing the same date field name

QT-09 • Incremental Loads with QVDs

1. The maximum update date, read into a variable
2. To avoid loading boundary rows twice
3. WHERE NOT Exists(key) on the history load
4. INNER JOIN to the keys that still exist

QT-10 • Designing Qlik Sheets & Visualizations

1. Line chart
2. Add-ons → Reference lines
3. Alternative measures
4. The measures have very different scales

QT-11 • Set Analysis & Alternate States

1. All data, ignoring selections
2. Calculates the latest selected year and pastes it into the expression
3. E()
4. Search strings must be in double quotes

QT-12 • Securing, Optimising & Deploying Qlik Apps

1. They recalculate on every click; build the field in the script instead
2. Section Application

3. Upper case

4. A managed space