THE ULTIMATE CLOUD-BASED BI TOOL

LOOKER DEMO

Looker
PART OF Google Cloud

BY: VANSHIKA AGGARWAL

WHAT'S COMING?

WHAT IS LOOKER?

Looker is a cloud-based Business Intelligence (BI) tool that helps you explore, share, and visualize data that drive better business decisions. Looker is now a part of the Google Cloud Platform. It allows anyone in your business to analyze and find insights into your datasets quickly.

Features of Looker

- Automated Modeling
- Intuitive Visualizations
- Self-Service BI
- Pre-Built Analytics Code
- LookML Data Modeling Language
- Time Zone Handling

What is a LookML?

LookML is a language used to describe dimensions, calculations, aggregates, and data relationships in a SQL database. Looker practices a model in LookML for SQL queries construction against a specific database.

Overview of LookML structures

Projects

which are libraries of LookML code.A project is composed of one or more models.

Model

is a set of Explores by business area or need. Each model contains one or more Explores.

View

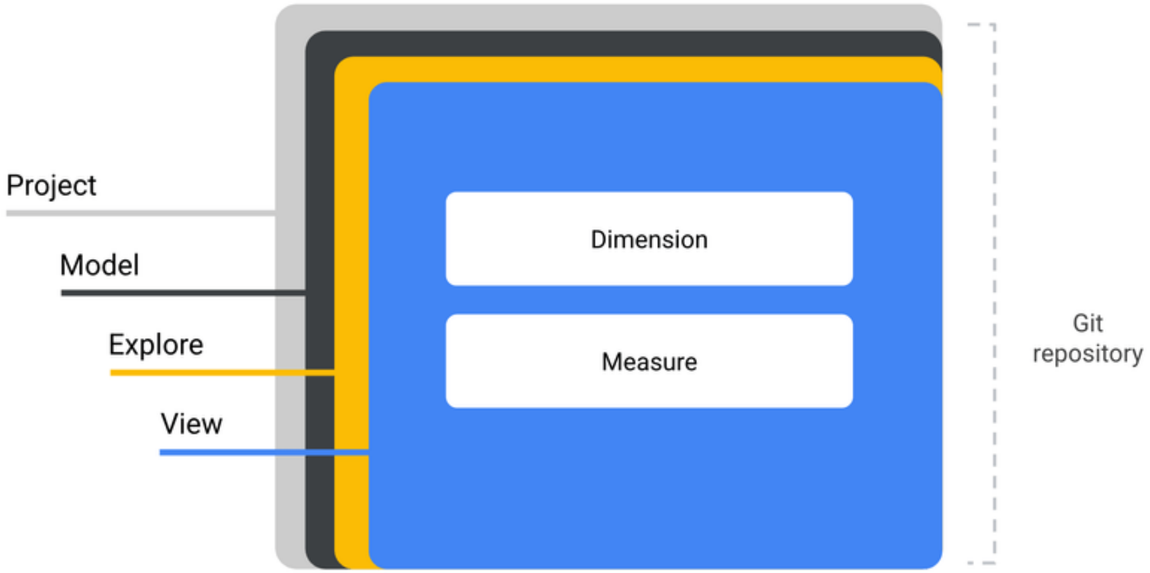
in LookML is a database table or a logical representation of one. Each view includes dimensions and measures

Dimentions

are database columns or logical representations of them

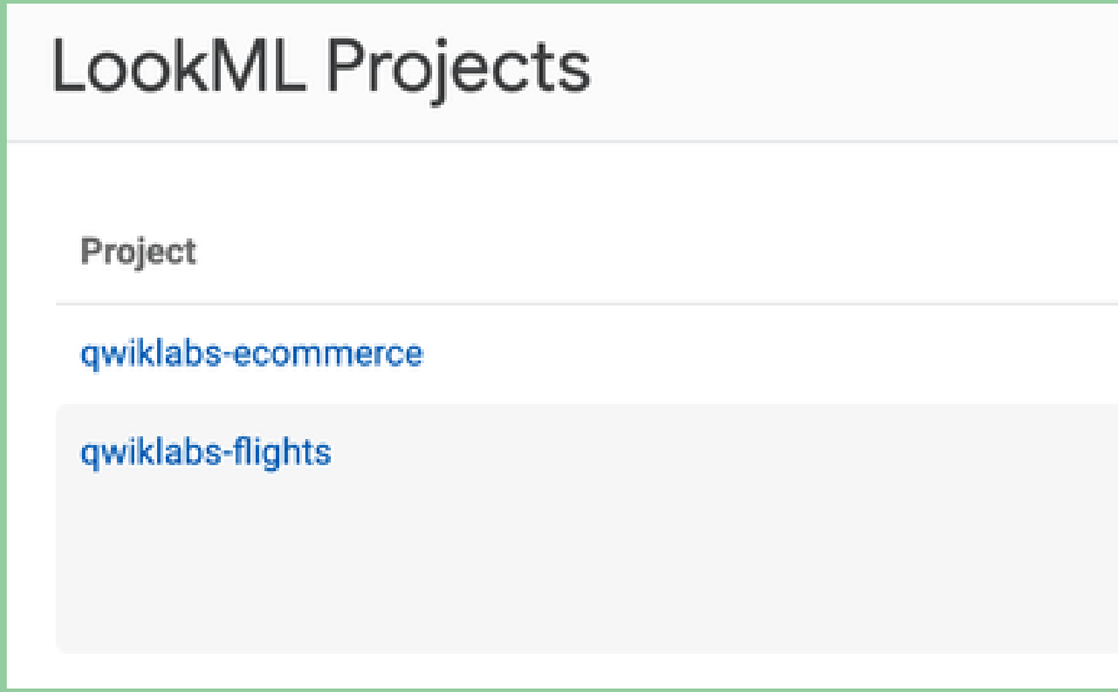
Measures

which are aggregate functions on dimensions, such as a COUNT of customers or a SUM of cost



PROJECT

In Looker, a project is a collection of files that describe the objects, connections, and user interface elements that will be used to carry out SQL queries for a Looker user. At the most basic level, these files describe how your database tables relate to each other and how Looker should interpret them.



MODEL

training_ecommerce.model

```
1  connection: "bigquery_public_data_looker"
2
3  # include all the views
4  include: "/views/*.view"
5  include: "/z_tests/*.lkml"
6  include: "/**/*.dashboard"
7
8  datagroup: training_ecommerce_default_datagroup {
9    # sql_trigger: SELECT MAX(id) FROM etl_log;;
10   max_cache_age: "1 hour"
11 }
12
13 persist_with: training_ecommerce_default_datagroup
14
15 label: "E-Commerce Training"
16
17 explore: order_items {
18   join: users {
19     type: left_outer
20     sql_on: ${order_items.user_id} = ${users.id} ;;
21     relationship: many_to_one
22   }
23
24   join: inventory_items {
25     type: left_outer
26     sql_on: ${order_items.inventory_item_id} = ${inventory_items.id} ;;
27     relationship: many_to_one
28   }
29
30   join: products {
31     type: left_outer
32     sql_on: ${inventory_items.product_id} = ${products.id} ;;
33     relationship: many_to_one
34   }
```

Models contain data connection information and definitions of Explores. Models can be used to restrict user access to certain Explores and separate and organize Explores by business area.Models are the next level of hierarchy and contain:

- The database connection you are using, as defined in the image by line 1.
- View files that are accessible to this model, as defined by lines 4, 5, and 6.
- Definitions of Explores and their join logic.

EXPLORE

Add an option to the Explore menu based on the view called `users`:

```
explore: users {
  # additional explore parameters go here
}
```

Add an option to the Explore menu called **Events** based on the view called `user_events`:

```
explore: events {
  from: user_events
}
```

explore adds an existing view to Looker's menu of Explores as described on the LookML terms and concepts documentation page. As a best practice, an Explore should be defined inside of a model file.

Explores are typically named after an existing view. However, if you want to have multiple Explores based on the same view, you can add a from parameter to the Explore. In that case, the Explore can be given any valid name, which includes only lowercase letters (a-z), digits (0-9), and underscores.

VIEWS

users.view

```
view: users {
  sql_table_name: `cloud-training-demos.looker_ecomm.users`
    ;;
  drill_fields: [id]

  dimension: id {
    primary_key: yes
    type: number
    sql: ${TABLE}.id ;;
  }

  dimension: age {
    type: number
    sql: ${TABLE}.age ;;
  }
```

Views are where you define dimensions (which are the data attributes) and measures (which are aggregations of dimensions). Think of views as tables that bundle related fields. There are a few different types of views:

DIMENTION

The lowest level of a LookML object are fields, which can be dimensions or measures. Dimensions are created for any columns that are already in your database tables when the view files are generated from a table by Looker.

users.view

```
85    dimension: traffic_source {
86      type: string
87      sql: ${TABLE}.traffic_source ;;
88    }
89
90    dimension: zip {
91      type: zipcode
92      sql: ${TABLE}.zip ;;
93    }
94
95    measure: count {
96      type: count
97      drill_fields: [id, last_name, first_name,
98    }
99  }
```

```
90    dimension: zip {
91      type: zipcode
92      sql: ${TABLE}.zip ;;
93    }
94
95    measure: count {
96      type: count
97      drill_fields: [id, last_name, first_name, events.count, order_items.count]
98    }
99  }
```

MEASURE

Measures are aggregates that do not live explicitly in your database tables. They must be created in LookML. They aggregate dimensions into values like sums or counts.

Labels

This page refers to the label parameter that is part of a model.
label can also be used as part of an Explore, as described on the label (for Explores) parameter documentation page.
label can also be used as part of a view, as described on the label (for views) parameter documentation page.
label can also be used as part of a field, as described on the label (for fields) parameter documentation page.
label can also be used as part of a reference line, described on the Dashboard reference line parameters documentation page.

usage

```
label: "desired label"
```

Hierarchy

Model File

`label`

Default Value

The name of the model file, capitalized with spaces instead of underscores

Accepts

A string

Definition

label helps make Explores more user-friendly by allowing you to set the model names that appear in the Explore menu.
If you do not explicitly add a label to a model definition, the label defaults to the name of the model, but nicely formatted. Underscores are changed to spaces, and each word is capitalized.

Examples

If your model file is called user_data.model, by default the Explore menu will use the filename, capitalized and with spaces instead of underscores. So the model's entry in the Explore would be rendered as User Dat

E

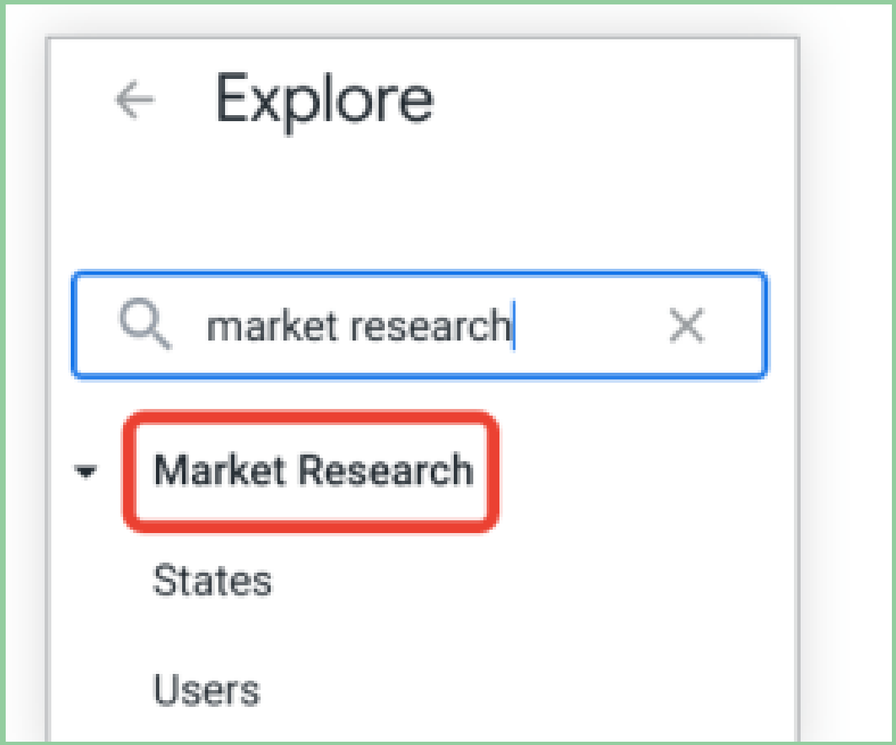
label: "Market Research"

The label parameter goes at the top level of the model file:

user_data.model

```
connection: "faa"
label: "Market Research"

include: "/views/*.view.lkml"

explore: users {}
explore: states {}

```

The model now shows up in the Explore menu as Market Research:



IMPORTING EXTERNAL CSV IN LOOKER CAN ONLY BE DONE BY THE ADMIN

Create Table

Create from Prev

Source Data

Location: Google Drive — https://docs.google.com/spreadsheets/d/17nMtWNlaH

File format: Google Sheets — View Files

Destination Table

Table name: namesheet . names

Table type: External table

Schema

Name	Type	Mode
event_time	TIMESTAMP	NULLABLE
name	STRING	NULLABLE

Add Field

Edit as Text

Options

Number of errors allowed: 0

Create Table

You are in Development Mode. Exit Development Mode

Looker

Explore
Develop
Admin

Home
Recently Viewed
Favorites
Boards
No boards added yet

Folders
Blocks

Development Mode

Welcome, Developer! See recent updates and more from your colleagues below.

Your favorite dashboards and Looks

No looks yet

Looks are saved query visualizations created in the Explore section. You can add looks to a dashboard.

No dashboards yet

Dashboards provide a view of related content by allowing you to place multiple tables, graphs, or looks on one page.

Recently viewed at your organization

Group: Everyone

There is no recent activity for the group you have selected.

Popular at your organization

Group: Everyone

Develop

Content Validator

SQL Runner

Projects

Find a Project

qwiklabs-ecommerce

qwiklabs-flights

Looker

qwiklabs-ecommerce dev-developer-student-dwx9 personal branch

Recheck Errors Up to Date

File Browser

- models
  - training_ecommerce.model
- views
  - distribution_centers.view
  - event_session_facts.view
  - event_session_funnel.view
  - events.view
  - inventory_items.view
  - order_items.view
  - products.view
  - users.view
- z_tests
- business_pulse.dashboard

training_ecommerce.model

```
connection: "bigquery_public_data_looker"

# include all the views
include: "/views/*.view"
include: "/z_tests/*.lkml"
include: "/**/*.dashboard"

datagroup: training_ecommerce_default_datagroup {
  # sql_trigger: SELECT MAX(id) FROM etl_log;;
  max_cache_age: "1 hour"
}

persist_with: training_ecommerce_default_datagroup

label: "E-Commerce Training"

explore: order_items {
  join: users {
    type: left_outer
    sql_on: ${order_items.user_id} = ${users.id} ;;
    relationship: many_to_one
  }

  join: inventory_items {
    type: left_outer
    sql_on: ${order_items.inventory_item_id} = ${inventory_items.id} ;;
    relationship: many_to_one
  }
```

Quick Help Metadata

A `model` references a combination of related explores. Unlike other LookML elements, a model is not declared explicitly with the `model` keyword.

```
model: {
  access_grant: identifier
  case_sensitive: yes or no
  connection: "string"
  datagroup: identifier
  explore: identifier
  fiscal_month_offset: number
  include: "string"
  label: possibly-localized-
  string
  map_layer: identifier
  named_value_format: identifier
  persist_for: "string"
  persist_with: datagroup-ref
  test: identifier
  view: identifier
  week_start_day: monday or ...
}
```

Shift + ? for keyboard shortcuts

ADDING DIMENTIONS

```
dimension: id {
  primary_key: yes
  type: number
  sql: ${TABLE}.id ;;
}
dimension: country {
  type: string
  map_layer_name: countries
  sql: ${TABLE}.country ;;
}
dimension: email {
  type: string
  sql: ${TABLE}.email ;;
}
dimension: first_name {
  type: string
  sql: ${TABLE}.first_name ;;
}
dimension: last_name {
  type: string
  sql: ${TABLE}.last_name ;;
}
```

ADDING MEASURES

```
measure: count {
  type: count
  drill_fields: [id, last_name, first_name]
}
```

File Browser

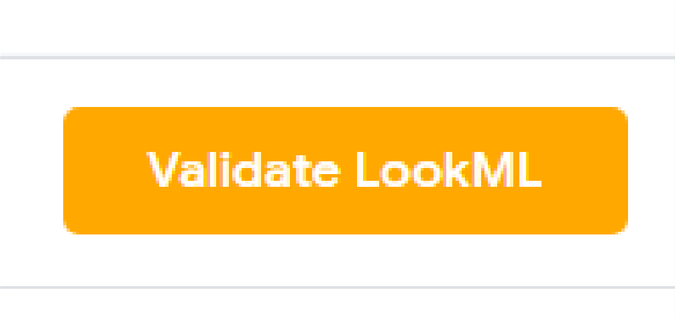
- models
  - training_ecommerce.model
- views
  - distribution_centers.view
  - event_session_facts.view
  - event_session_funnel.view
  - events.view
  - inventory_items.view
  - order_items.view
  - products.view
  - users.view
  - users_limited.view
- z_tests
- business_pulse.dashboard

users_limited.view

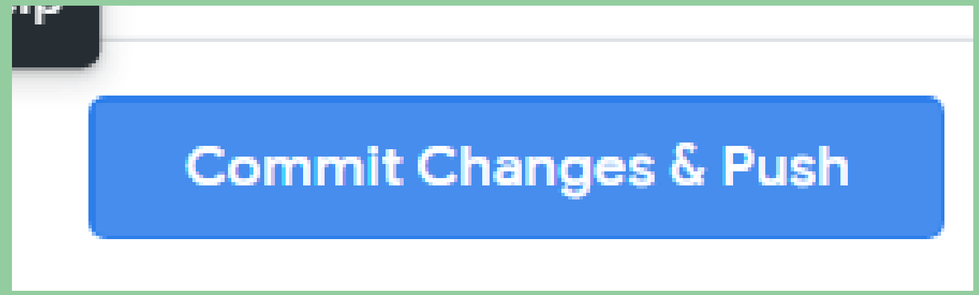
```
view: users_limited {
  sql_table_name: `cloud-training-demos.looker_ecomm.users` ;;

  dimension: id {
    primary_key: yes
    type: number
    sql: ${TABLE}.id ;;
  }

  dimension: country {
    type: string
    map_layer_name: countries
    sql: ${TABLE}.country ;;
  }

  dimension: email {
    type: string
    sql: ${TABLE}.email ;;
  }

  dimension: first_name {
    type: string
    sql: ${TABLE}.first_name ;;
  }

  dimension: last_name {
    type: string
    sql: ${TABLE}.last_name ;;
  }

  measure: count {
    type: count
    drill_fields: [id, last_name, first_name]
  }
}
```

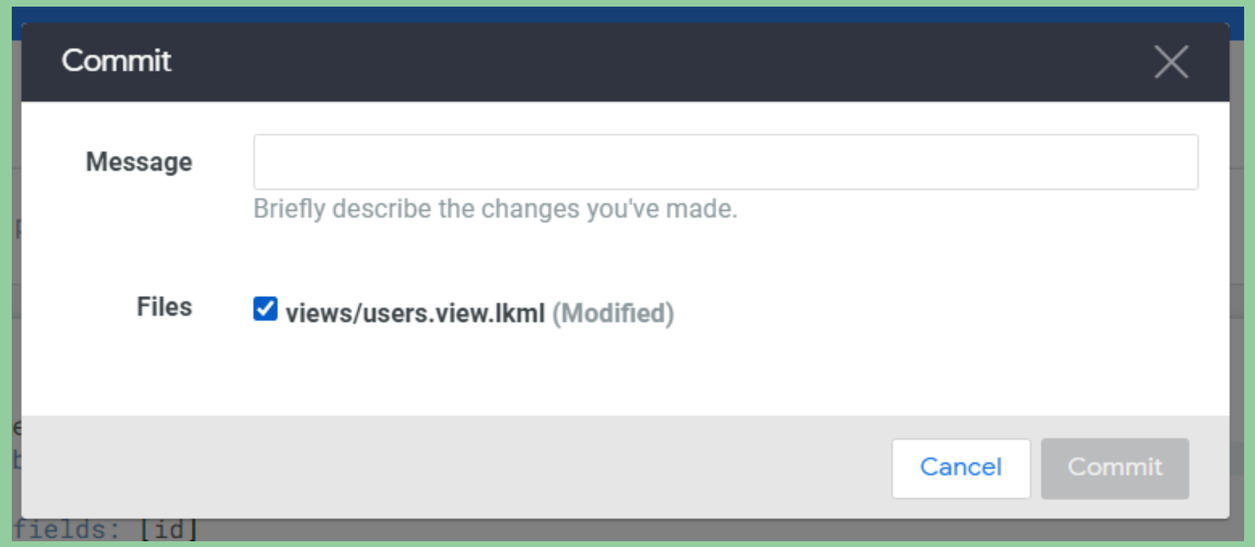
CLICK ON SAVE CHANGES

THEN VALIDATE LOOKML



THEN COMMIT





Join a view to an existing explore

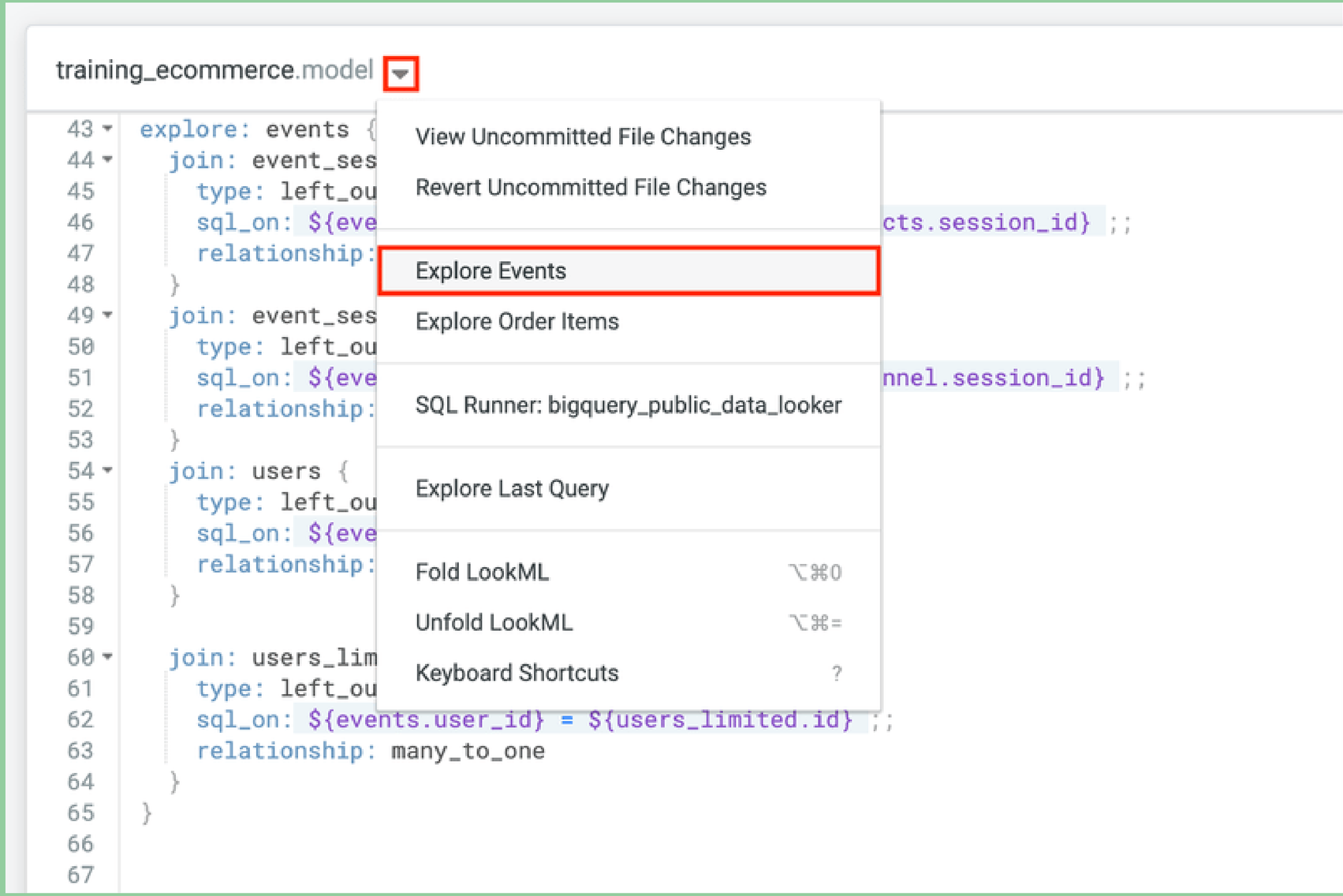
File Browser

- models
  - training_ecommerce.model
- views
  - distribution_centers.view
  - event_session_facts.view
  - event_session_funnel.view
  - events.view
  - inventory_items.view
  - order_items.view
  - products.view
  - users.view
  - users_limited.view
- z_tests

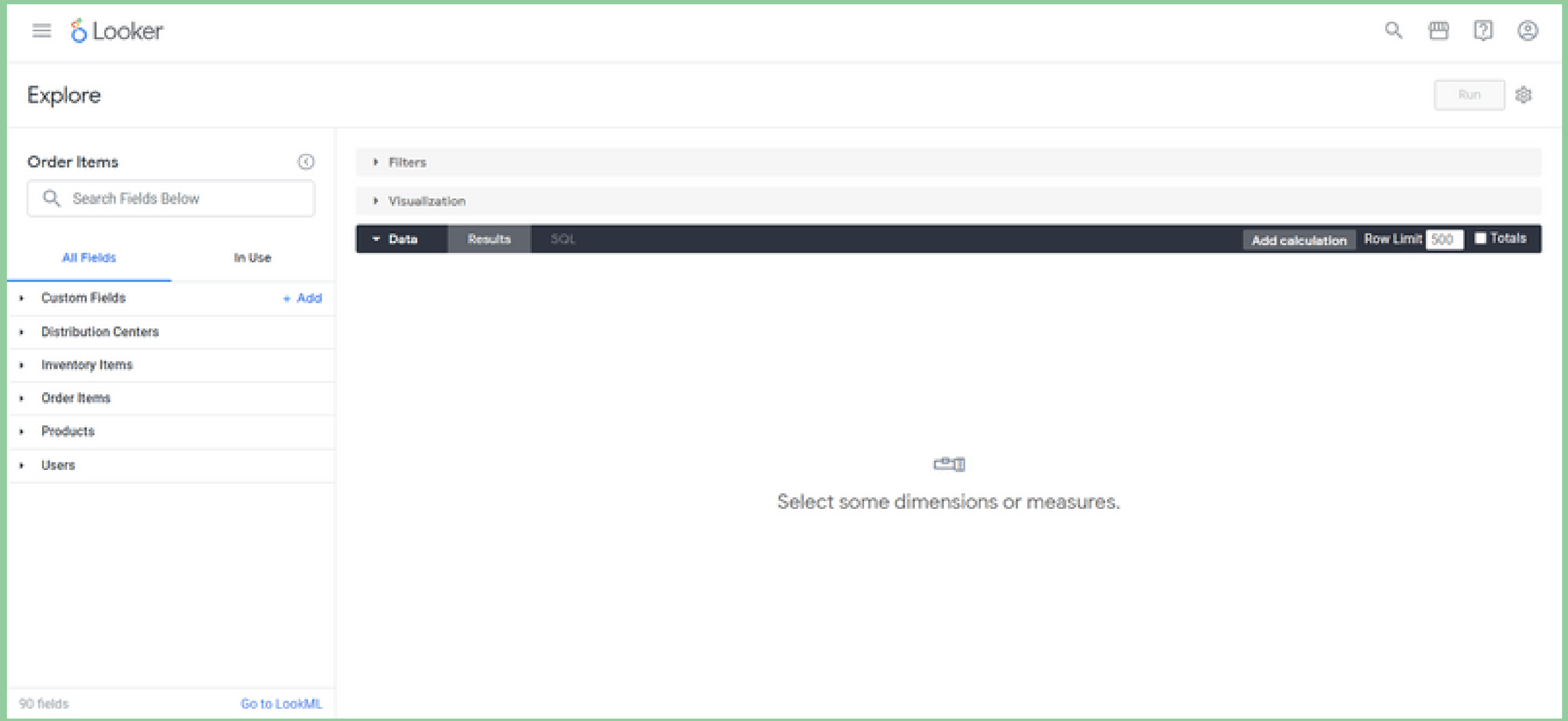
training_ecommerce.model

```
43 explore: events {
44   join: event_session_facts {
45     type: left_outer
46     sql_on: ${events.session_id} = ${event_session_facts.session_id} ;;
47     relationship: many_to_one
48   }
49   join: event_session_funnel {
50     type: left_outer
51     sql_on: ${events.session_id} = ${event_session_funnel.session_id} ;;
52     relationship: many_to_one
53   }
54   join: users {
55     type: left_outer
56     sql_on: ${events.user_id} = ${users.id} ;;
57     relationship: many_to_one
58   }
59
60   join: users_limited {
61     type: left_outer
62     sql_on: ${events.user_id} = ${users_limited.id} ;;
63     relationship: many_to_one
64   }
65 }
```

Opening the Explore Page



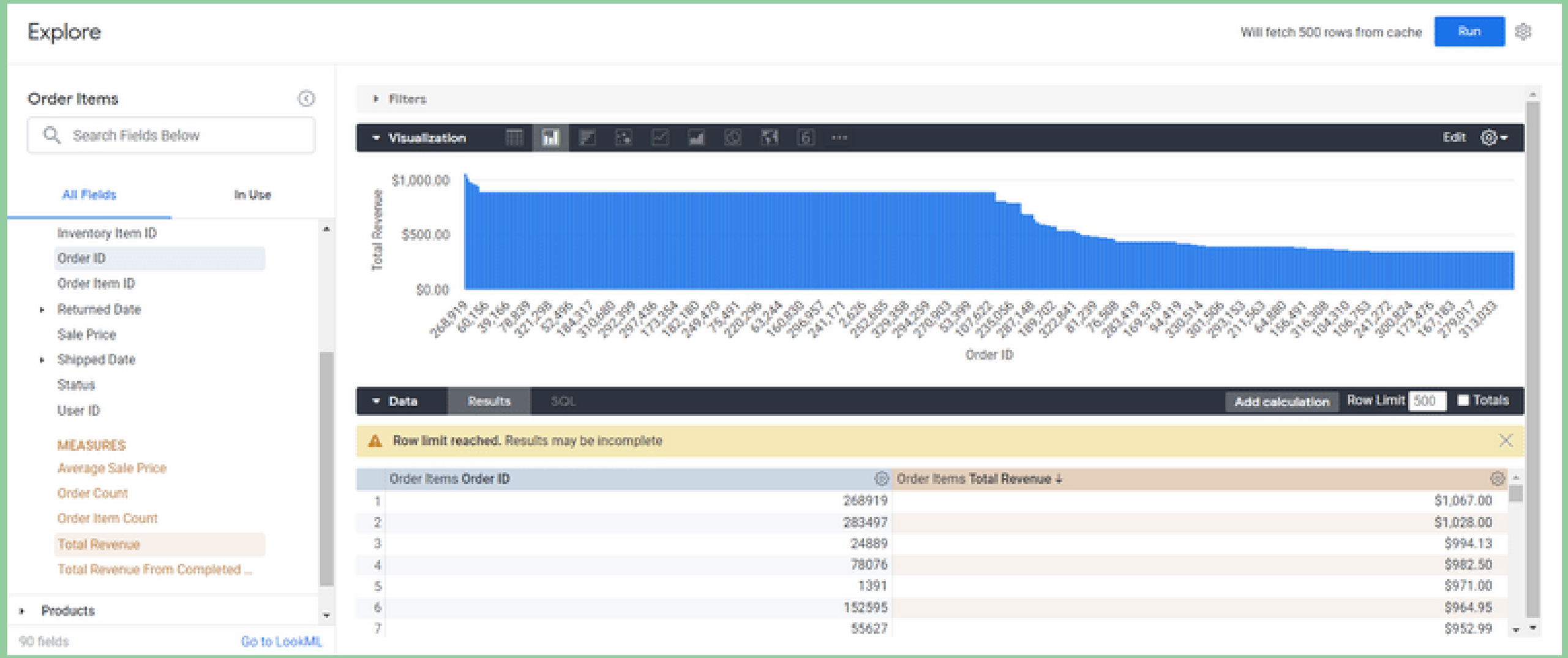
EXPLORE PAGE



ANALYSING DIMENSIONS AND MEASURES

	Users Limited First Name	Users Limited Count
1	Mary	2,078
2	James	2,034
3	John	2,023
4	Robert	1,990
5	Michael	1,699
6	William	1,508
7	David	1,506
8	Richard	1,118
9	Charles	968
10	Joseph	880

VISULIZATION



SQL QURIES IN EXPLORE

Will process 4.54 MB Run

SQL QUERY Google BigQuery Standard SQL

```
SELECT
  order_items.order_id AS order_id
  ,order_items.user_id AS user_id
  ,COUNT(*) AS order_item_count
  ,SUM(order_items.sale_price) AS order_revenue
FROM cloud-training-demos.looker_ecomm.order_items
GROUP BY order_id, user_id
LIMIT 10
```

RESULTS

	order_id	user_id	order_item_count	order_revenue
1	15	10	1	50
2	70	38	1	14.989999771118164
3	86	51	1	49.79999923706055
4	125	82	1	8.510000228881836
5	147	98	1	53.95000076293945
6	287	189	1	44.9900016784668
7	326	214	1	68.52999877929688
8	402	264	1	44.9900016784668
9	579	388	1	29.989999771118164
10	616	418	1	33.209999084472656

SQL QURIES IN DEVELOP PAGE

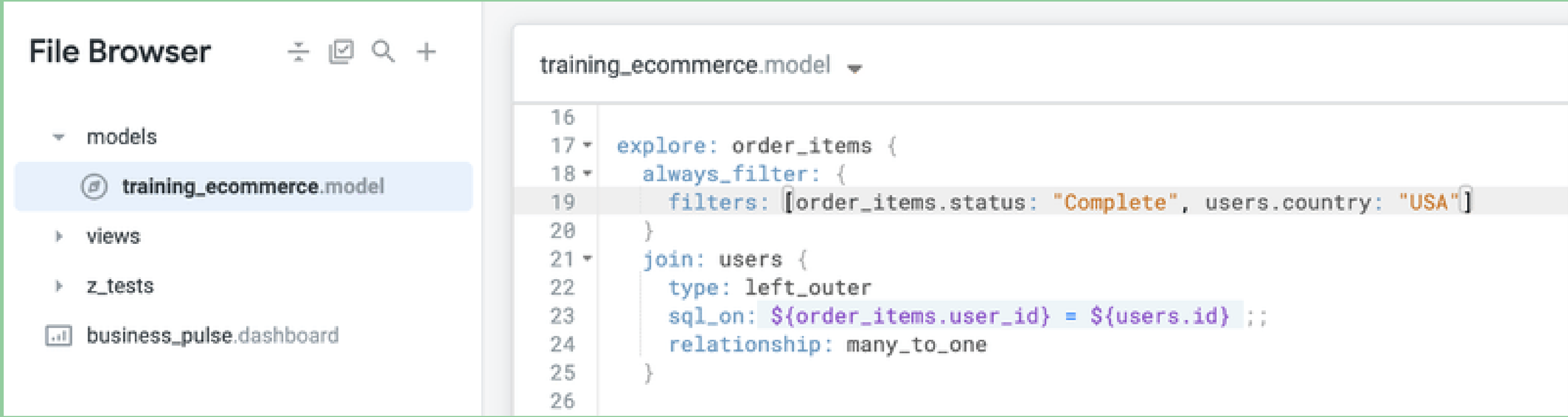
File Browser

- models
- views
  - distribution_centers.view
  - event_session_facts.view
  - event_session_funnel.view
  - events.view
  - inventory_items.view
  - order_details.view
  - order_items.view
  - products.view
  - users.view
- z_tests
- business_pulse.dashboard

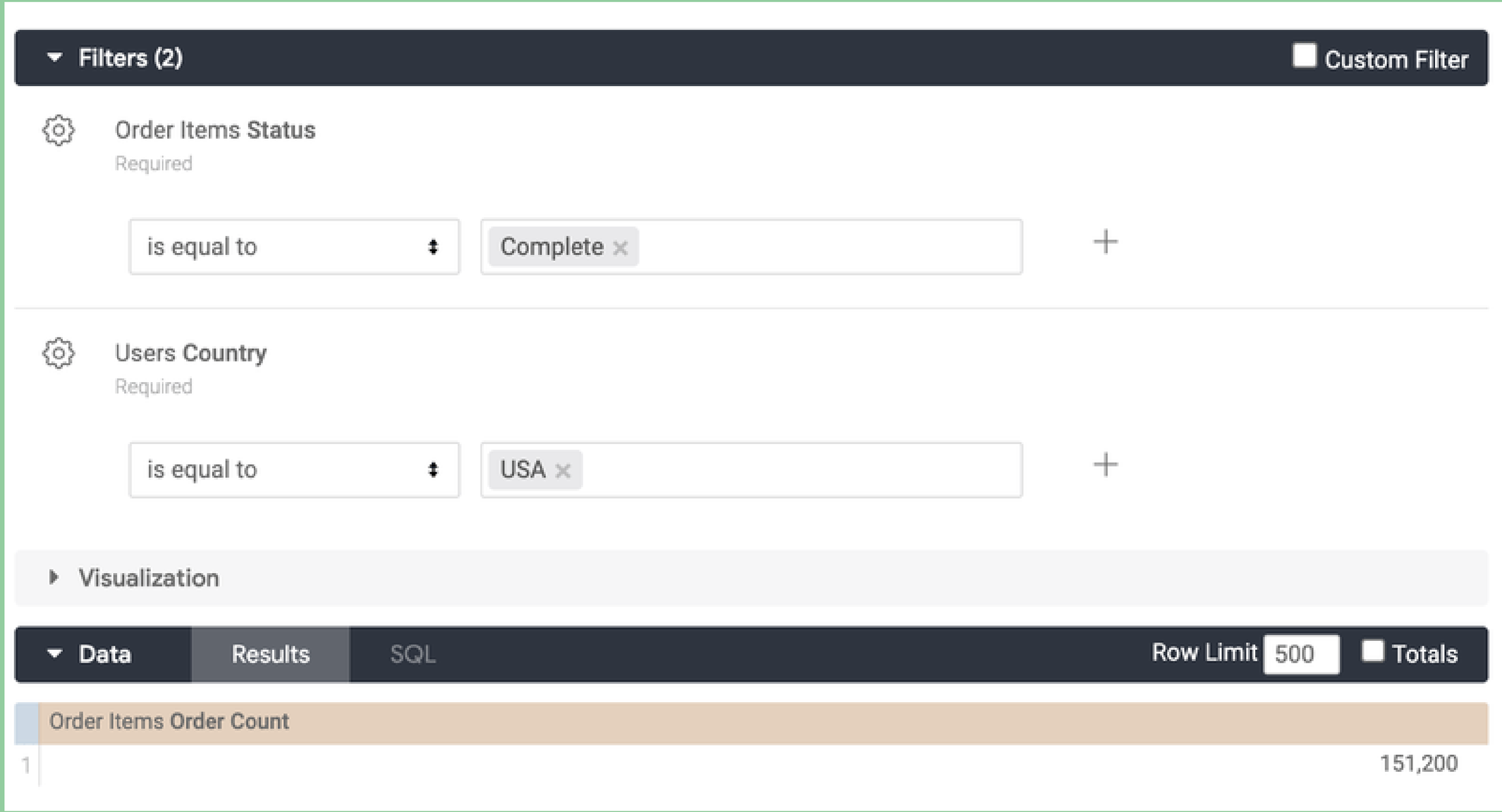
order_details.view

```
view: order_details {
  derived_table: {
    sql: SELECT
        order_items.order_id AS order_id
        ,order_items.user_id AS user_id
        ,COUNT(*) AS order_item_count
        ,SUM(order_items.sale_price) AS order_revenue
      FROM cloud-training-demos.looker_ecomm.order_items
      GROUP BY order_id, user_id
       ;;
  }

  measure: count {
    hidden: yes
    type: count
    drill_fields: [detail*]
  }

  dimension: order_id {
    primary_key: yes
    type: number
    sql: ${TABLE}.order_id ;;
  }
```

ADDING FILTERS IN EXPLORE VIEW



FILTERS IN EXPLORE VIEW



CONDITIONAL FILTERS

File Browser

- models
  - training_ecommerce.model
- views

training_ecommerce.model

```
17 explore: order_items {
18   conditionally_filter: {
19     filters: [created_date: "3 years"]
20     unless: [users.id, users.state]
21   }
```

Filters (1)

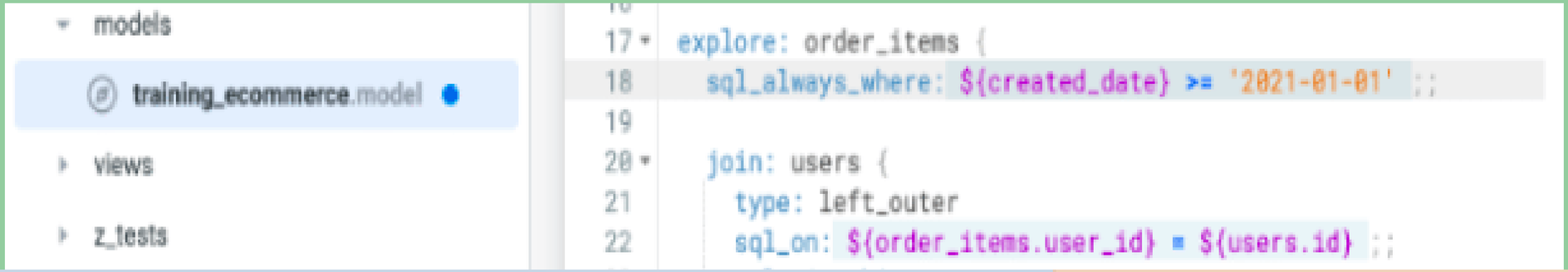
Custom Filter

Order Items Created Date
Conditionally Required

is in the past | 3 | years

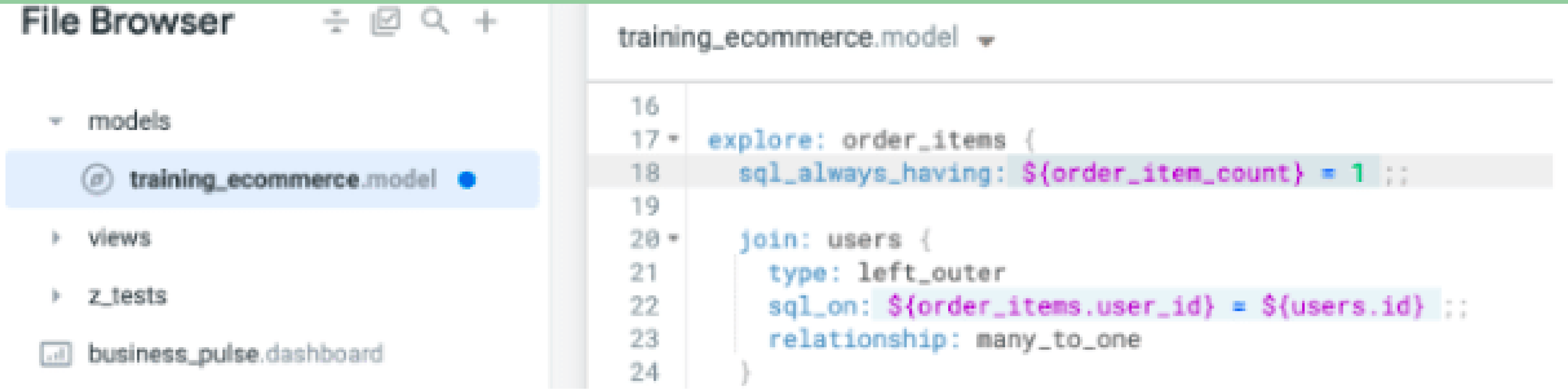
Types of Explore Filters

sql_always_where: The sql_always_where filter is used to add a WHERE clause applied to **dimensions** in a SQL query. In the following example we have added a sql_always_where filter to the Order Items Explore to only include data from the year 2021 and later.



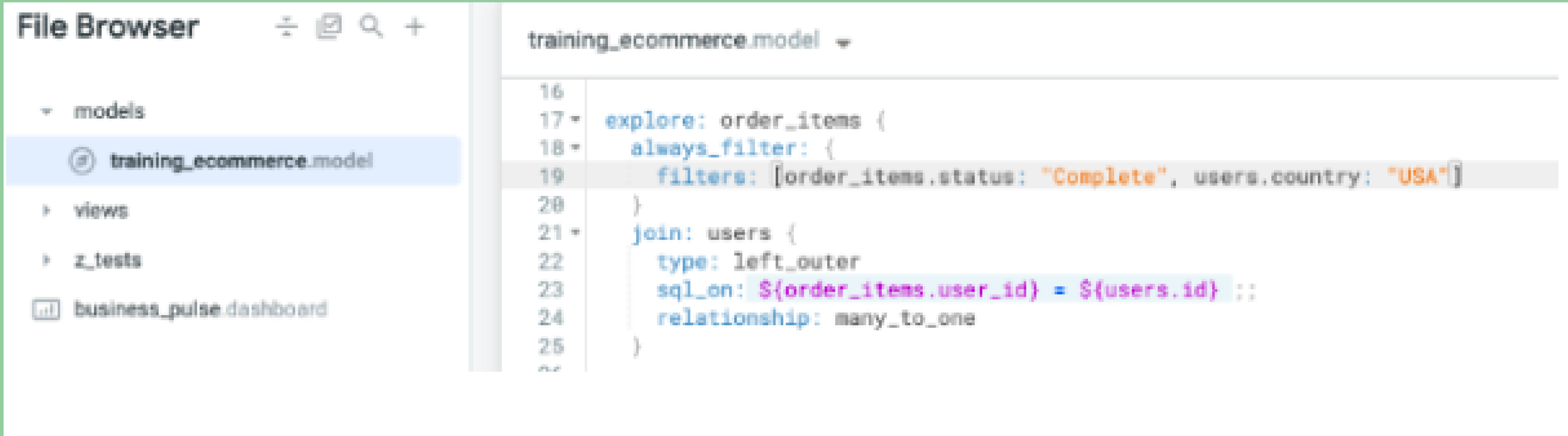
	Order Items Created Date	Order Items Order Count
1	2021-03-30	194
2	2021-03-29	342
3	2021-03-28	245
4	2021-03-27	242
5	2021-03-26	281
6	2021-03-25	326
7	2021-03-24	304
8	2021-03-23	305

sql_always_having: It is used to add a HAVING clause applied to **measures** in a SQL query. In the following example filter to the Order Items Explore to prevent users from looking at orders with more than one item. This will be used to omit any orders from Explore that have multiple items in them.

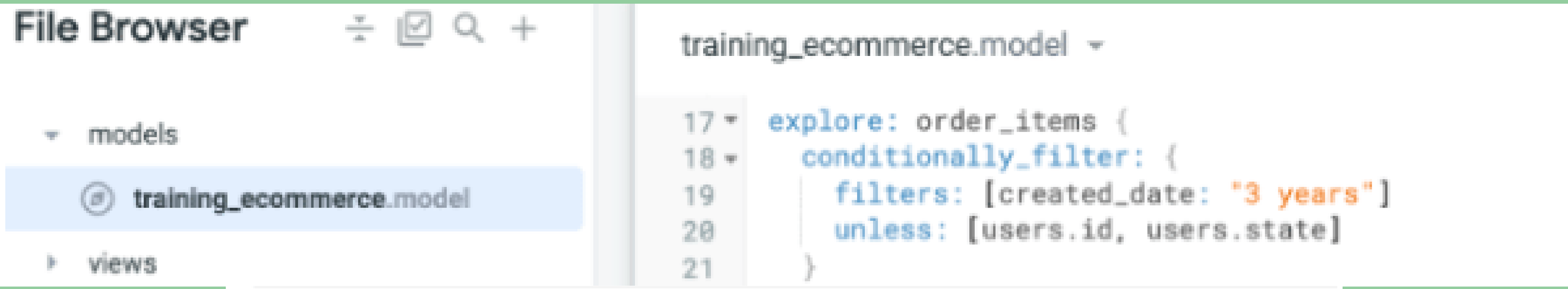


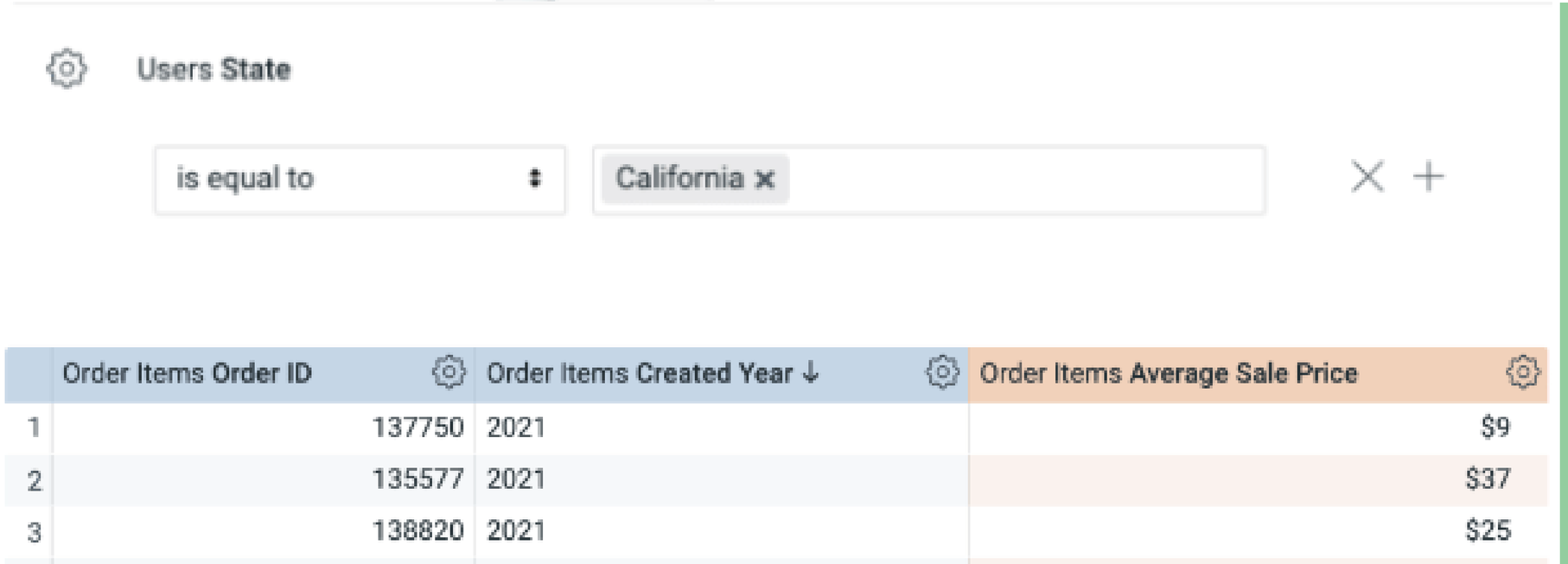
	Order Items Order ID	Order Items Order Item Count	Order Items Average Sale Price
1	287	1	$45
2	579	1	$30

- Always_filter: The always_filter enables you to require users to include a certain set of filters that you define. You also define a default value for the filters. Though users may change your default value for their query, they cannot remove the filter entirely. This is helpful when you want users to always filter by specific dimensions, such as always filtering by order status or user country, so that they do not request all of the possible data at one time.



- Conditionally_filter: The conditionally_filter adds a filter to the Explore frontend that is accessible by business users. The conditionally_filter parameter enables you to define a set of default filters that users can override *if* they apply at least one filter from a second list that you define.



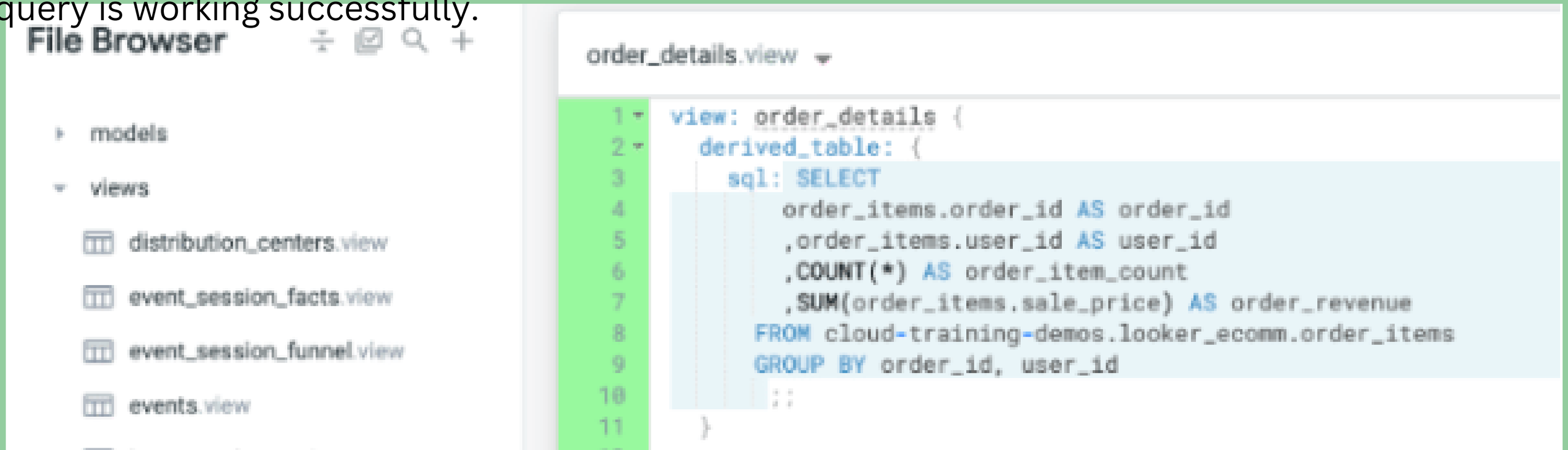


Derived Tables in LookML

In LookML, you can define derived tables using either SQL queries to define a SQL derived table or Explore queries to define a native derived table.

In the following example, the desired query selects the order_id and user_id, counts the number of items associated with each order, and then sums the price of those items.

Specifically, the **COUNT** clause is counting the number of individual order item IDs (the primary key of the order_items table), and the **SUM** clause is totaling the sale_price of the order item IDs. The **GROUP BY** clause is used to group the results by order_id and user_id, and the **LIMIT** clause is used to limit the results, as we only need to review a subset of records to ensure that our query is working successfully.



Native Derived Table

In contrast to SQL derived tables, native derived tables, or NDTs, are expressed entirely in LookML. Native derived tables are useful because they embody that essential LookML principle of reusability. They allow you to inherit already existing dimensions, measures, and even Explores and join logic.

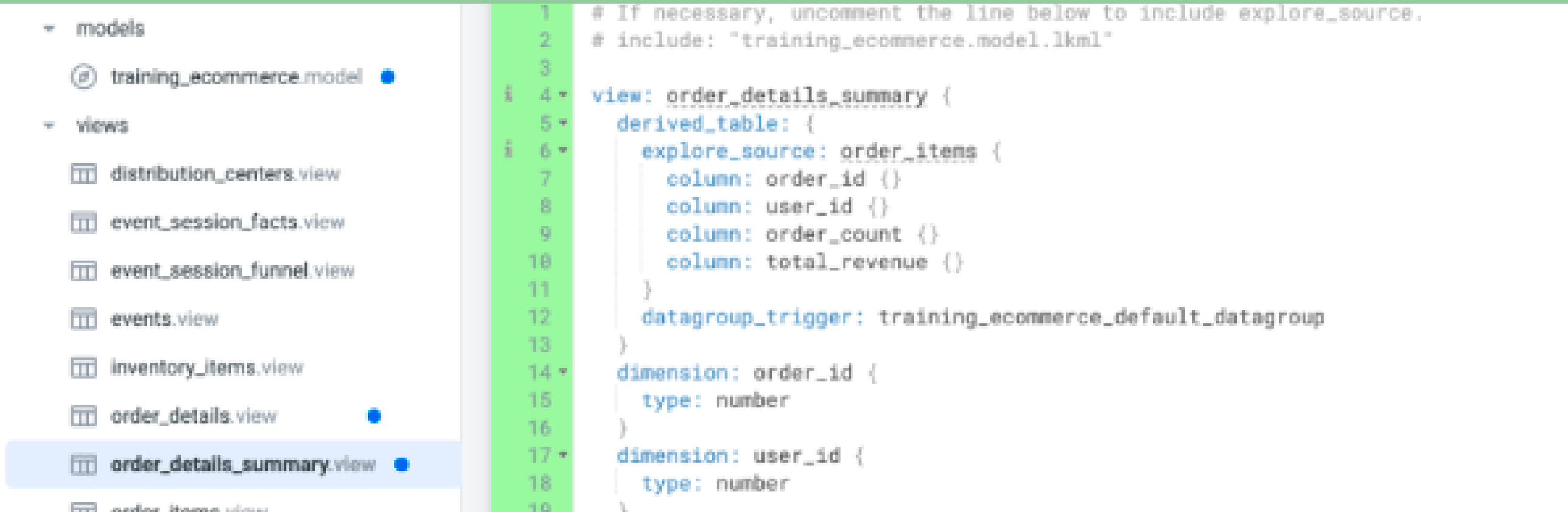
	Order Items Order ID	Order Items User ID	Order Items Order Count	Order Items Total Revenue
1	287	189	1	$44.99
2	579	388	1	$29.99
3	616	418	1	$33.21
4	402	264	1	$44.99
5	70	38	1	$14.99
6	15	10	1	$50.00

Copy the LookML code below, and paste it into your project definition.

```
view: add_a_unique_name_1623275538 {
  derived_table: {
    explore_source: order_items {
      column: order_id {}
      column: user_id {}
```

Persist a derived table

Persistent derived tables, or PDTs, are written to and stored in the connected database. The steps to persist a derived table are the same whether it is a SQL derived or native derived table. The benefit in persisting derived tables is that they are ready to go when business users need them, and therefore reduce query runtimes. The downsides are that they take up storage space in your database (which may correlate to cost), and they are more rigid.



Project manifest parameters

```
## STRUCTURAL PARAMETERS
project_name: "Current Project Name"
new_lookml_runtime: yes
local_dependency: {
project: "project_name"
override_constant: constant_name {
value: "string value"
}
}
## Possibly more local_dependency statements

remote_dependency: remote_project_name {
url: "remote_project_url"
ref: "remote_project_ref"
override_constant: constant_name {
value: "string value"
}
}
# Possibly more remote_dependency statements

constant: constant_name {
value: "string value"
export: none | override_optional | override_required
}
# Possibly more constant statements

## LOCALIZATION PARAMETERS
localization_settings: {
localization_level: strict | permissive
default_locale: locale_name
}

## EXTENSION FRAMEWORK PARAMETERS
application: application_name {
label: "Application Label"
url: "application_url"
file: "application_file_path"
entitlements: {
# Desired entitlements (described on application page)
}
}
## Possibly more application statements

## CUSTOM VISUALIZATION PARAMETERS
visualization:{
id: "unique-id"
label: "Visualization Label"
url: "visualization_url"
sri_hash: "SRI hash"
dependencies: ["dependency_url_1","dependency_url_2"]
file: "visualization_file_path"
}
## Possibly more visualization statements
```

16:49

Parameter Name	Description
Structural Parameters	
`project_name`	Specifies the name of the current project.
`new_lookml_runtime`	Enables or disables New LookML Runtime for a LookML project.
`local_dependency`	Specifies that this project depends on another project. This parameter has the `project` and `override_constant` subparameters.
`project`	Specifies a project that contains files that you want to include.
`remote_dependency`	Specifies one or more remote projects that contain files that you want to include. This parameter has `url` and `ref` subparameters, as well as the subparameter `override_constant`.
`constant`	Defines a LookML constant that can be used throughout your project. This parameter has the subparameters `value` and `export`.
Localization Parameters	
`localization_settings`	Specifies the localization information for your model. This parameter has `default_locale` and `localization_level` subparameters.
`localization_level`	Specifies whether strings with no translation are allowed in your model.
`default_locale`	Specifies the locale that will be used as your model's default for translating strings.
Extension Framework Parameters	
`application`	Defines an application for Looker's extension framework.
`label`	Specifies the name of the application that is displayed to the user.
`url`	Provides the URL of the application.
`file`	Provides the path to a JavaScript file (with a `.js` extension) that defines the application. The path is relative to the project root.
`entitlements`	Specifies the entitlements to control access to the extension features of the application. This parameter has many subparameters listed on the `application` parameter page.

Parameter Name	Description
Structural Parameters	
`access_grant`	Creates an access grant that limits access of LookML structures to only those users who are assigned an approved user attribute value. This parameter has the `user_attribute` and `allowed_values` subparameters.
`explore`	Exposes a view in the Explore menu. For more information about Explores and their parameters, see the Explore Parameter Reference page.
`include`	Adds files to a model
`test`	Creates a data test to verify your model's logic. The project settings include an option to require data tests. When this is enabled for a project, developers on the project must run data tests before deploying their changes to production. This parameter has the `explore_source` and `assert` subparameters.
Display Parameters	
`label` (for model)	Changes the way a model appears in the Explore menu
Filter Parameters	
`case_sensitive` (for model)	Specifies whether filters are case-sensitive for a model
Query Parameters	
`connection`	Changes the database connection for a model
`datagroup`	Creates a datagroup-caching policy for a model. This parameter has the `label`, `description`, `max_cache_age`, and `sql_trigger` subparameters.
`fiscal_month_offset`	Specifies the month your fiscal year begins (if it differs from the calendar year)
`persist_for` (for model)	Changes the cache settings for a model
`persist_with` (for model)	Specifies the datagroup to use for the model's caching policy
`week_start_day`	Specifies the day of the week on which week-related dimensions should start
Visualization and Formatting Parameters	
`map_layer` (for model)	Creates custom maps to be used with `map_layer_name`
`named_value_format`	Creates a custom value format to be used with `value_format_name`. This parameter has the `value_format` and `strict_value_format` subparameters.
Parameters to Avoid	
`scoping`	REMOVED 3.52 No longer required
`template`	REMOVED 3.30 No longer required

THANK YOU!