www.cloudskillsboost.google /games/4423/labs/28617

Modularizing LookML Code with Extends | Google Cloud Skills Boost

Qwiklabs : 15-19 minutes

GSP936

Overview

Looker is a modern data platform in Google Cloud that you can use to analyze and visualize your data interactively. You can use Looker to do in-depth data analysis, integrate insights across different data sources, build actionable data-driven workflows, and create custom data applications.

In this lab, you learn how to modularize your LookML code with extends by extending views and Explores.

Prerequisites

Familiarity with LookML is necessary. It is recommend that you complete the Understanding LookML in Looker skill badge quest before you begin this lab.

What are LookML Extends?

Extends allow you to modularize code by creating copies of LookML objects that can then be integrated into other LookML objects and modified independently from the original LookML object. In Looker, you can extend views, Explores, and LookML-defined dashboards. By modularizing your code, extends allow you to treat pieces of code as modules or units that you can then build upon to expand your model.

Why use extends?

Writing DRY code

Easier and faster to make changes

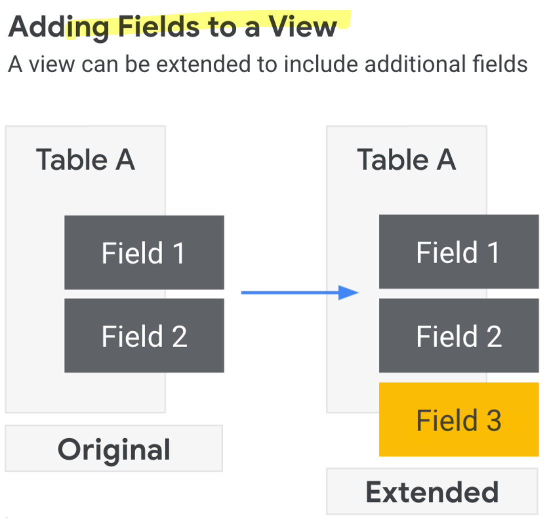
Consistency

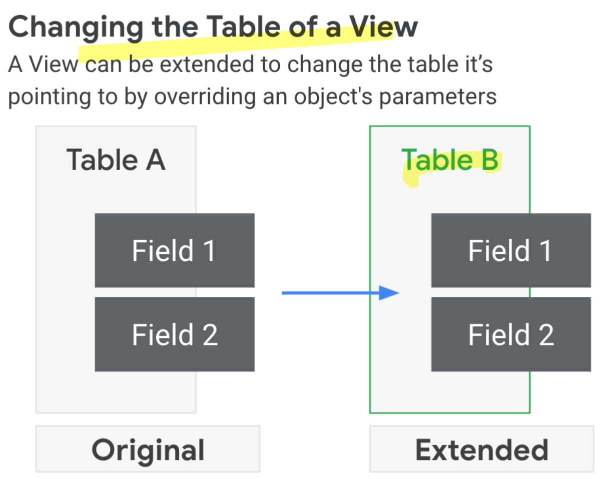
Easier management of different field sets for different users

Extends help you write D.R.Y. (Don't Repeat Yourself) code. By copying preexisting objects and sections of code, you can more easily add or modify logic. This is critical for scaling your model as your organization and use cases expand. It also maximizes consistency in your model, because you aren't manually rewriting code all the time. And it makes it easier to manage field access for different groups of users, which is also important for scalability.

LookML view extends

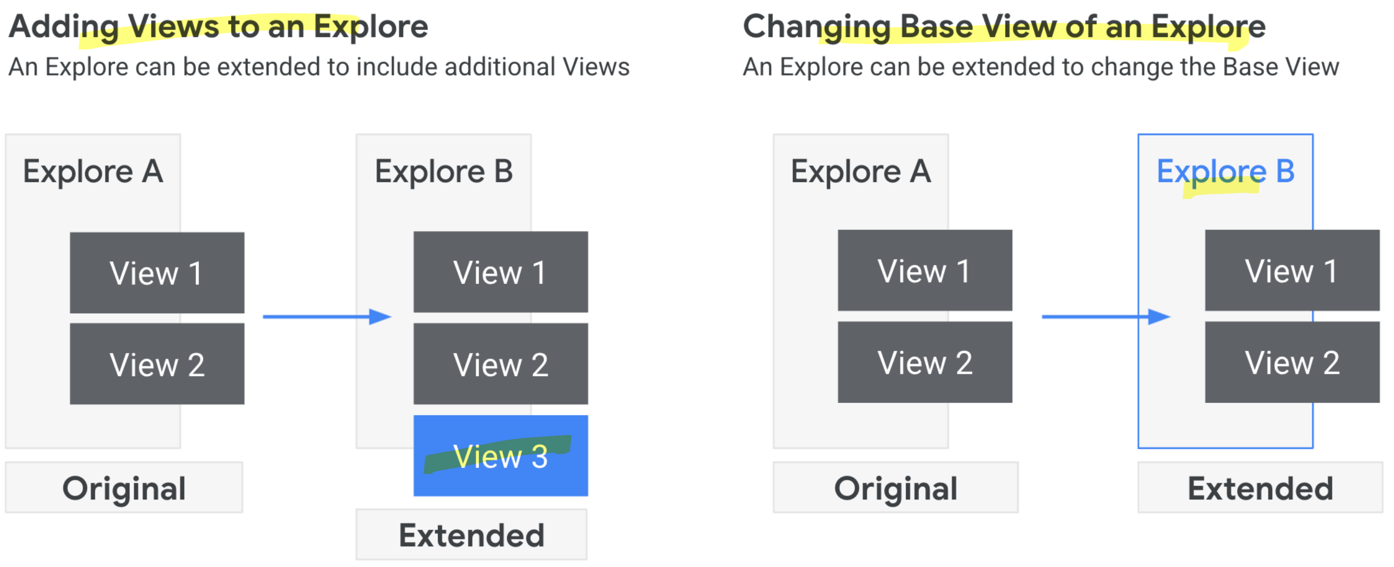
As mentioned earlier, one object you can extend is a LookML view. This is commonly done to add more fields and/or update logic to the existing fields. Another use case is to change the database table specified in the sql_table_name parameter.



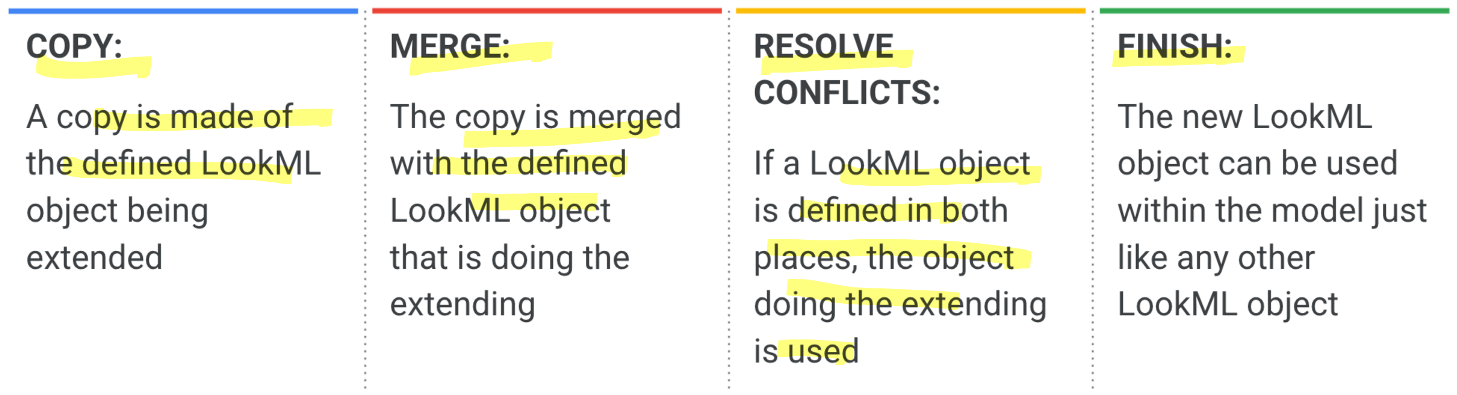

Changing the Table of a View
A View can be extended to change the table it's pointing to by overriding an object's parameters
Table A
Field 1
Field 2
Original
Table B
Field 1
Field 2
Extended

LookML Explore extends

Another object you can extend is Explores. You may have multiple tables that must always be joined together, especially in a more normalized database architecture. To avoid rewriting the same joins repeatedly, you can make a "base" Explore that already joins them together and then extend it to create additional Explores that need to join in more views. Or you may need the same set of joined views, but with the new Explore starting from a different base view.


Adding Views to an Explore
An Explore can be extended to include additional Views
Explore A
View 1
View 2
Original
Explore B
View 1
View 2
View 3
Extended
Changing Base View of an Explore
An Explore can be extended to change the Base View
Explore A
View 1
View 2
Original
Explore B
View 1
View 2
Extended

The four steps involved in Extend execution



COPY:
A copy is made of the defined LookML object being extended

MERGE:
The copy is merged with the defined LookML object that is doing the extending

RESOLVE CONFLICTS:
If a LookML object is defined in both places, the object doing the extending is used

FINISH:
The new LookML object can be used within the model just like any other LookML object

"Behind the scenes" with an Explore:

1. Looker makes a copy of the LookML object being extended.
2. The copy, or *extending object*, is merged with the new or modified definitions.
3. If any conflicts are detected (which happens if you modified definitions), the extending object controls.
4. The extending object can be used in your LookML model just like any other object.

Note: Although implementing extends is a simple process, knowing these details is useful if you encounter unexpected behavior.

Objectives

In this lab, you learn how to:

- Describe how extends allow you to modularize and easily reuse LookML code.
- Extend a view by integrating columns defined in another view.

- Extend an Explore by integrating joins defined in another Explore.

Setup

Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended).

Note: Use an Incognito or private browser window to run this lab. This prevents any conflicts between your personal account and the Student account, which may cause extra charges incurred to your personal account.

- Time to complete the lab---remember, once you start, you cannot pause a lab.

Note: If you already have your own personal Google Cloud account or project, do not use it for this lab to avoid extra charges to your account.

How to start your lab and sign in to Looker

1. When ready, click Start Lab .

   A new panel will appear with the temporary credentials that you must use for this lab.

   If you need to pay for the lab, a pop-up will open for you to select your payment method.

2. Note your lab credentials in the left pane. You will use them to sign in to the Looker instance for this lab.

   Note: If you use other credentials, you will get **errors or incur charges**.

3. Click **Open Looker**.

4. Enter the provided Username and Password in the Email and Password fields.

   Important: You must use the credentials from the Connection Details panel on this page. Do not use your Google Cloud Skills Boost credentials. If you have your own Looker account, do not use it for this lab.

5. Click **Log In**.

After a successful login, you will see the Looker instance for this lab.

Task 1. Extend a view to add columns from another view

Instead of copying/pasting the same code across multiple views, you can create one centralized view that contains definitions for commonly used dimensions and measures. Then, using extends, you can integrate those dimensions and measures into multiple views. You can simply use specific parameters for extends to identify the view that you want Looker to copy from.

From a business perspective, this is very practical because you can have one centralized code base that is reused by multiple teams that can extend the core code and customize it for their own needs. The benefit of abstracting the location dimensions is that you can update them once, and the update is then propagated to any of the views that are extended from that location view.

In this task, you create a new view that contains location dimensions (e.g., city, country) that can be reused by extending other views such as the users and events views.

Create a new view

1. Click the toggle button to enter **Development mode**.

2. On the **Develop** tab, select the **qwiklabs-ecommerce** LookML project.

3. Click (**+**) next to File Browser, and select **Create View**.

4. Name the view `location`, drag it under the **views** folder, and add the following code to it:

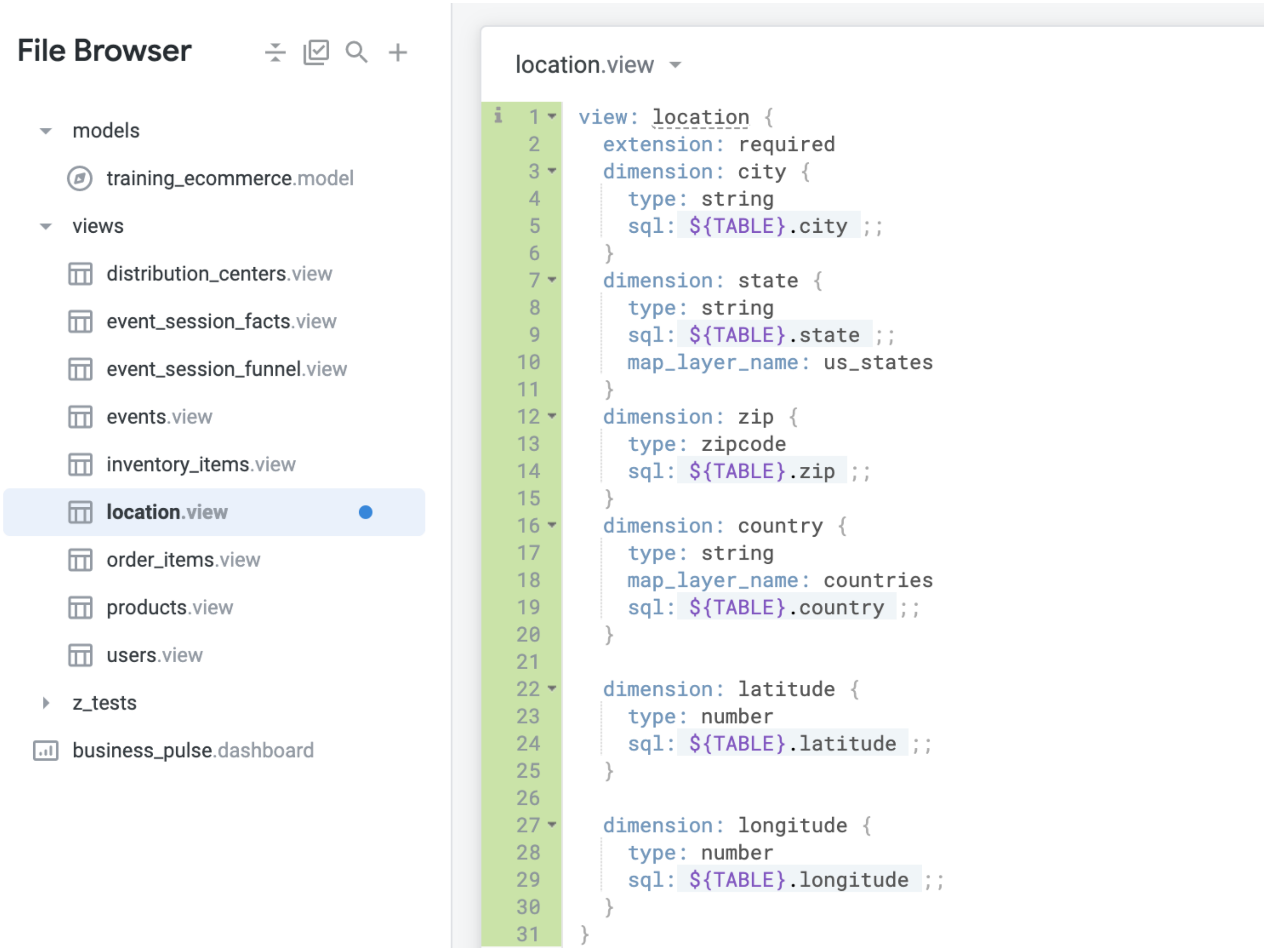
view: location { extension: required dimension: city { type: string sql: ${TABLE}.city ;; } dimension: state { type: string sql: ${TABLE}.state ;; map_layer_name: us_states } dimension: zip { type: zipcode sql: ${TABLE}.zip ;; } dimension: country { type: string map_layer_name: countries sql: ${TABLE}.country ;; } dimension: latitude { type: number sql: ${TABLE}.latitude ;; } dimension: longitude { type: number sql: ${TABLE}.longitude ;; } }

This view file contains the definitions for dimensions that you want to reuse in other views: **city**, **country**, **latitude**, **longitude**, **state**, and **zip**.

Notice line 2 (`extension: required`), which means that this view *cannot be joined to other views*, and thus will not be visible to users. To use this view, you must integrate it into another view using the `extends` parameter, which you do in the next section.

Also notice that, unlike with other views, you do not need to include the parameter `sql_table_name` in the view definition to identify which table to use for the data. Instead, this view will use the table specified in the view that will be extended in the next section.

5. Click **Save Changes**, and then click **Validate LookML**.
   No LookML errors were found, and your file should resemble the following:



```
view: location {
  extension: required
  dimension: city {
    type: string
    sql: ${TABLE}.city ;;
  }
  dimension: state {
    type: string
    sql: ${TABLE}.state ;;
    map_layer_name: us_states
  }
  dimension: zip {
    type: zipcode
    sql: ${TABLE}.zip ;;
  }
  dimension: country {
    type: string
    map_layer_name: countries
    sql: ${TABLE}.country ;;
  }

  dimension: latitude {
    type: number
    sql: ${TABLE}.latitude ;;
  }

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

Add extends

1. Open the **users.view** file.
2. On a new line at the top of the file (line 1), add the following code, which indicates that the users view is being extended using the location view:

include: location.view

3. On line 3 above `sql_table_name`, add the following code:

extends: [location] **Note:** Because the extends are added, the `sql_table_name` parameter identifies which table to use as the data source for both the existing objects in the file and the objects that are added from geography via the extend.

4. Remove the existing dimension definitions for: `city`, `country`, `latitude`, `longitude`, `state`, and `zip` (this is the existing order in the file). Instead of being explicitly defined in the **users.view** file, these dimensions are integrated via the extend from **location.view.**
5. Click **Save Changes**, and then click **Validate LookML**.

6. Open the **event.view** file.

7. On a new line at the top of the file (line 1), add the following code:

include: location.view

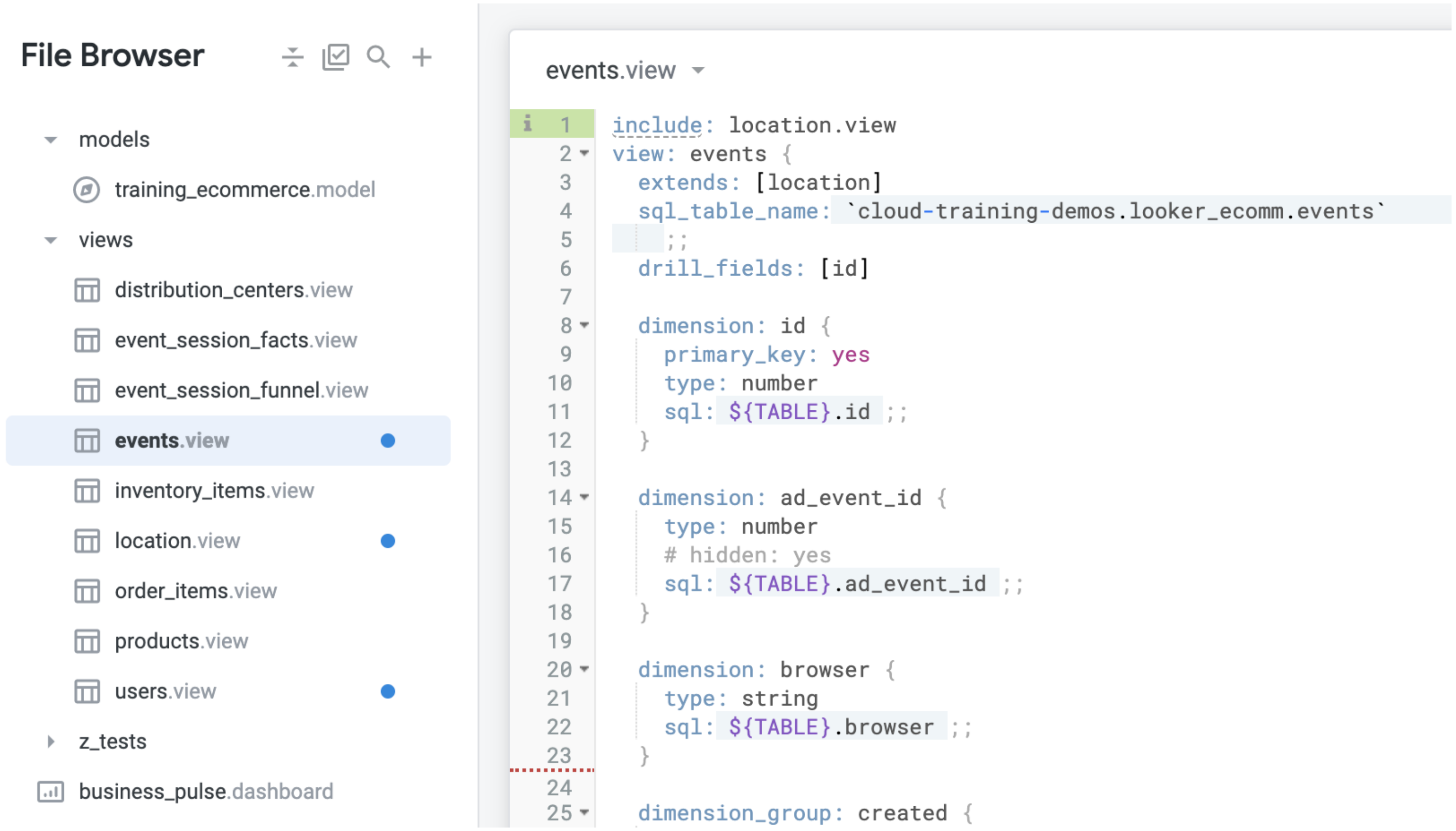
8. On line 3 above `sql_table_name`, add the following code:

extends: [location]

9. As you did with the users view, remove the existing dimension definitions for: `city`, `country`, `latitude`, `longitude`, `state`, and `zip`.

10. Click **Save Changes**, and then click **Validate LookML**.

Your file should now resemble the following:



```
include: location.view
view: events {
  extends: [location]
  sql_table_name: `cloud-training-demos.looker_ecomm.events`
    ;;
  drill_fields: [id]

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

  dimension: ad_event_id {
    type: number
    # hidden: yes
    sql: ${TABLE}.ad_event_id ;;
  }

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

  dimension_group: created {
```

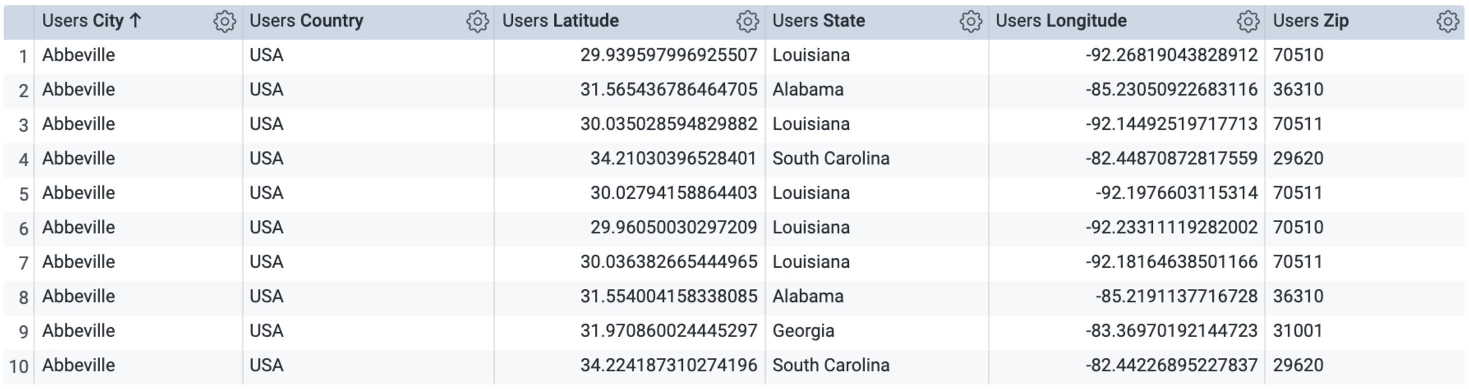
Test the extended view for Users and Events in the Order Items Explore

1. Navigate to the Explore page for **Order Items**.

2. From the **Users** view, select the **City**, **Country**, **Latitude**, **Longitude**, **State**, and **Zip** dimensions.

3. Click **Run**.

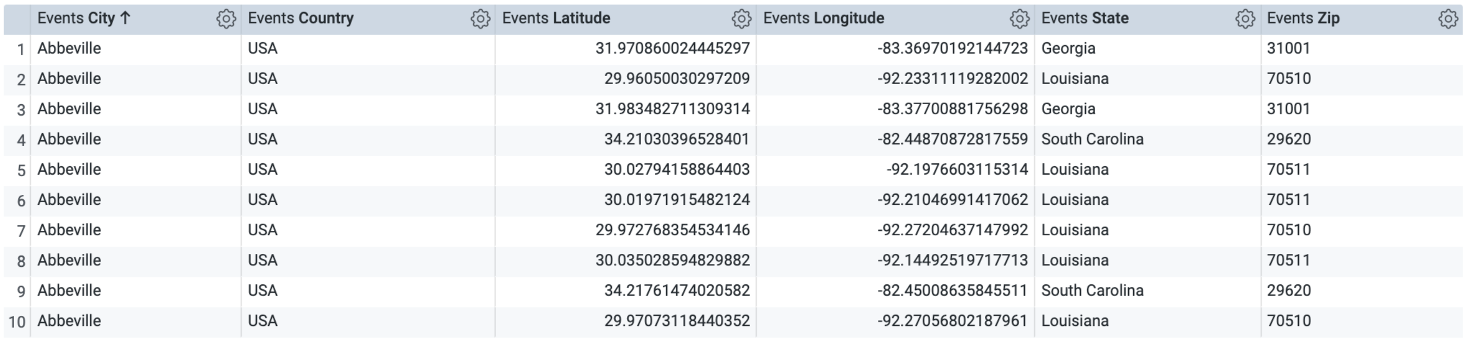
Even though you removed the definitions for these dimensions (city, country, latitude, longitude, state and zip) from the **users.view** file, you can see and use them because they were added to the **users.view** file using an extend from the **location.view** file!



	Users City ↑	Users Country	Users Latitude	Users State	Users Longitude	Users Zip
1	Abbeville	USA	29.939597996925507	Louisiana	-92.26819043828912	70510
2	Abbeville	USA	31.565436786464705	Alabama	-85.23050922683116	36310
3	Abbeville	USA	30.035028594829882	Louisiana	-92.14492519717713	70511
4	Abbeville	USA	34.21030396528401	South Carolina	-82.44870872817559	29620
5	Abbeville	USA	30.02794158864403	Louisiana	-92.1976603115314	70511
6	Abbeville	USA	29.96050030297209	Louisiana	-92.23311119282002	70510
7	Abbeville	USA	30.036382665444965	Louisiana	-92.18164638501166	70511
8	Abbeville	USA	31.554004158338085	Alabama	-85.2191137716728	36310
9	Abbeville	USA	31.970860024445297	Georgia	-83.36970192144723	31001
10	Abbeville	USA	34.224187310274196	South Carolina	-82.44226895227837	29620

4. Navigate to the **Events Explore**.

5. From the **Events** view, select the **City**, **Country**, **Latitude**, **Longitude**, **State**, and **Zip** dimensions.



	Events City ↑	Events Country	Events Latitude	Events Longitude	Events State	Events Zip
1	Abbeville	USA	31.970860024445297	-83.36970192144723	Georgia	31001
2	Abbeville	USA	29.96050030297209	-92.23311119282002	Louisiana	70510
3	Abbeville	USA	31.983482711309314	-83.37700881756298	Georgia	31001
4	Abbeville	USA	34.21030396528401	-82.44870872817559	South Carolina	29620
5	Abbeville	USA	30.02794158864403	-92.1976603115314	Louisiana	70511
6	Abbeville	USA	30.01971915482124	-92.21046991417062	Louisiana	70511
7	Abbeville	USA	29.972768354534146	-92.27204637147992	Louisiana	70510
8	Abbeville	USA	30.035028594829882	-92.14492519717713	Louisiana	70511
9	Abbeville	USA	34.21761474020582	-82.45008635845511	South Carolina	29620
10	Abbeville	USA	29.97073118440352	-92.27056802187961	Louisiana	70510

Again, even though you removed the definitions for these dimensions from the **events.view** file, you can see and use them because they were added to the **events.view** file using an extend.

6. Navigate back to the **events.view** file in the Looker IDE.

Commit changes and deploy to production

1. Click **Validate LookML** and then click **Commit Changes & Push**.

2. Add a commit message and click **Commit**.

3. Lastly, click **Deploy to Production**.

Click *Check my progress* to verify the objective. Extend a view to add columns from another view

Task 2. Extend an Explore to add joins from another Explore

Instead of copying/pasting the same joins across multiple Explores in a model file, you can create one base Explore that contains the most commonly used joins across your Explores. Then you can use extends to reuse that base Explore to define and customize other Explores defined in the model file.

A common business use case for this is creating one core Explore that can be used to create other Explores for specific user groups/teams within your organization.

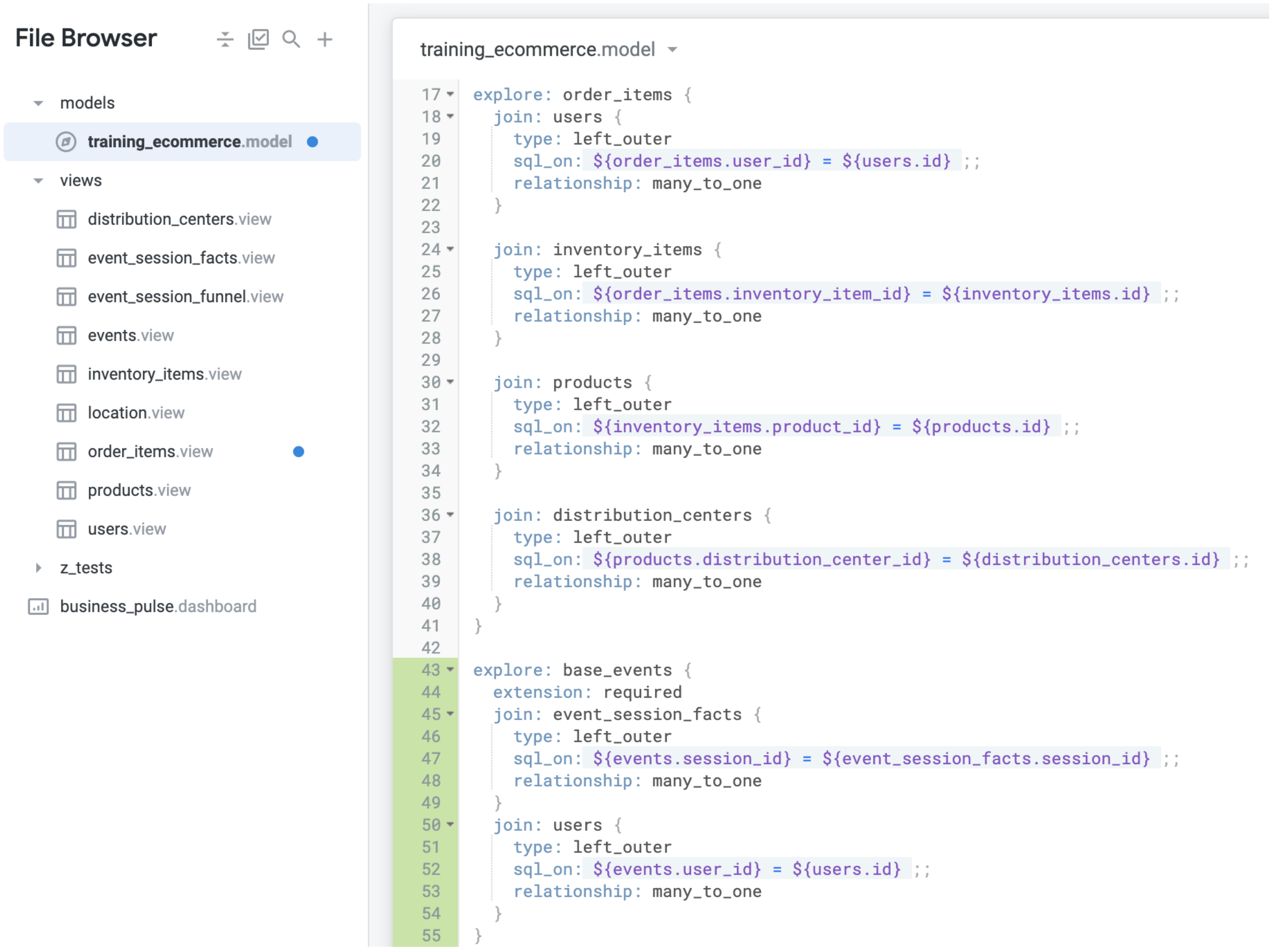
In this section, you create one base Explore that contains joins for all views that are needed by multiple business teams, and then use extends to reuse and customize that base Explore for multiple teams.

1. Navigate to the **training_ecommerce.model** file.

2. After the `order_items` Explore definition (around line 43), create a new base Explore called `base_events`, using the following code:

explore: base_events { extension: required join: event_session_facts { type: left_outer sql_on: ${events.session_id} = ${event_session_facts.session_id} ;; relationship: many_to_one } join: users { type: left_outer sql_on: ${events.user_id} = ${users.id} ;; relationship: many_to_one } }

Notice again the line for `extension: required`, which means that this Explore is not visible to users. Your file should resemble the following:



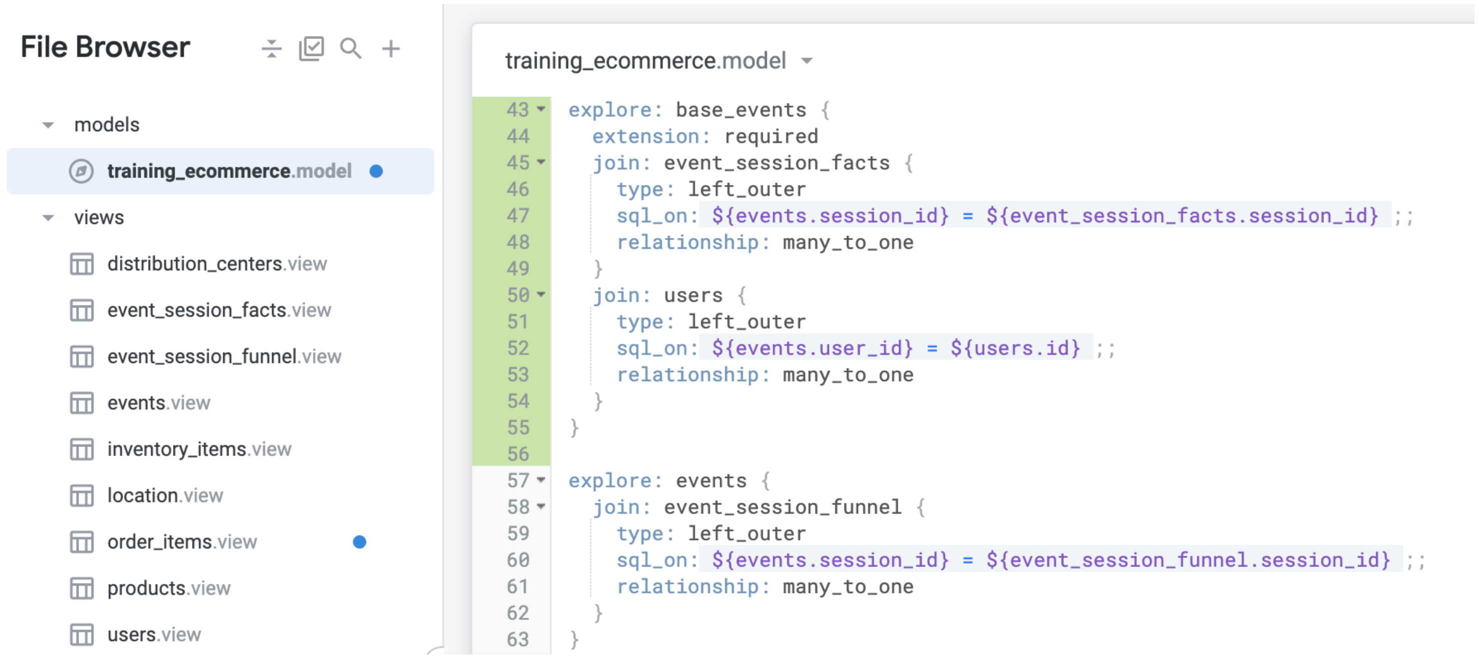
training_ecommerce.model

```
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
  }

  join: products {
    type: left_outer
    sql_on: ${inventory_items.product_id} = ${products.id} ;;
    relationship: many_to_one
  }

  join: distribution_centers {
    type: left_outer
    sql_on: ${products.distribution_center_id} = ${distribution_centers.id} ;;
    relationship: many_to_one
  }
}

explore: base_events {
  extension: required
  join: event_session_facts {
    type: left_outer
    sql_on: ${events.session_id} = ${event_session_facts.session_id} ;;
    relationship: many_to_one
  }
  join: users {
    type: left_outer
    sql_on: ${events.user_id} = ${users.id} ;;
    relationship: many_to_one
  }
}
```

Next, you'll modify the existing definition for the **events** Explore to extend it with the views from `base_events`.

3. From the **events** Explore, remove the existing joins for `event_session_facts` and `users`.

These joined views are integrated from the `base_events` Explore via code added in the next step. Notice that the join definition for `event_session_funnel` remains to customize this Explore for a particular set of users. Your file should resemble the following:

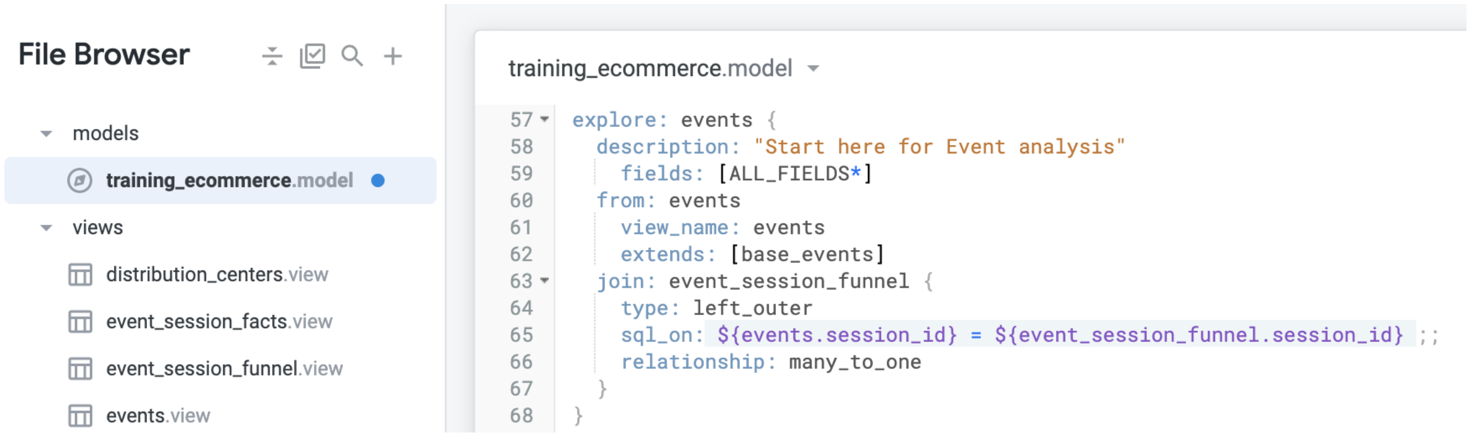


```
explore: base_events {
  extension: required
  join: event_session_facts {
    type: left_outer
    sql_on: ${events.session_id} = ${event_session_facts.session_id} ;;
    relationship: many_to_one
  }
  join: users {
    type: left_outer
    sql_on: ${events.user_id} = ${users.id} ;;
    relationship: many_to_one
  }
}

explore: events {
  join: event_session_funnel {
    type: left_outer
    sql_on: ${events.session_id} = ${event_session_funnel.session_id} ;;
    relationship: many_to_one
  }
}
```

4. Under the first line of the **events** Explore definition, add the following code:

description: "Start here for Event analysis" fields: [ALL_FIELDS*] from: events view_name: events extends: [base_events] **Note:** The new lines provide a description for the Explore info button, identify which fields from which view file to include (all fields), and specify which Explore is being used to extend the **events** Explore.

Your final definition for the **events** Explore should resemble the following:



```
explore: events {
  description: "Start here for Event analysis"
    fields: [ALL_FIELDS*]
  from: events
    view_name: events
    extends: [base_events]
  join: event_session_funnel {
    type: left_outer
    sql_on: ${events.session_id} = ${event_session_funnel.session_id} ;;
    relationship: many_to_one
  }
}
```

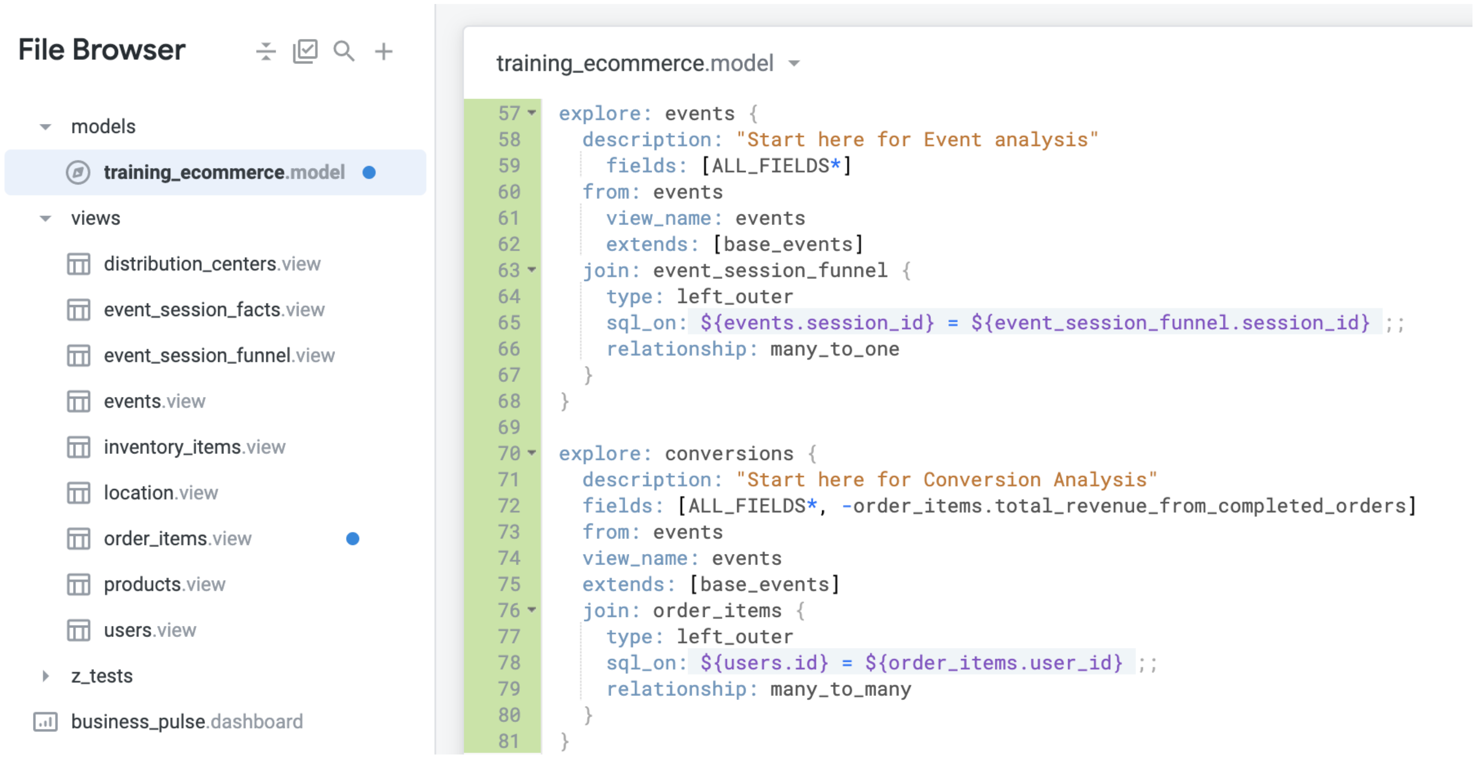
Note: The `from` and `view_name` are both pointing to the events view, so why include both? The `from` makes sure that you are using the original view called events (not an alias name for the view and not an extended one), and the `view_name` is the view file name, which could be an alias, etc.

5. Below the modified **events** Explore definition, to add a new Explore called **conversions**, use the following code:

explore: conversions { description: "Start here for Conversion Analysis" fields: [ALL_FIELDS*, -order_items.total_revenue_from_completed_orders] from: events view_name: events extends: [base_events] join: order_items { type: left_outer sql_on: ${users.id} = ${order_items.user_id} ;; relationship: many_to_many }
}

Lines 2-6 here provide a description for the Explore info button, identify which fields from which view file to include (all fields except the `total_revenue_from_completed_orders` measure in the order items view), and specify which Explore is being used to extend this Explore (i.e., the same **base_events** Explore that was used to extend the **events** Explore).

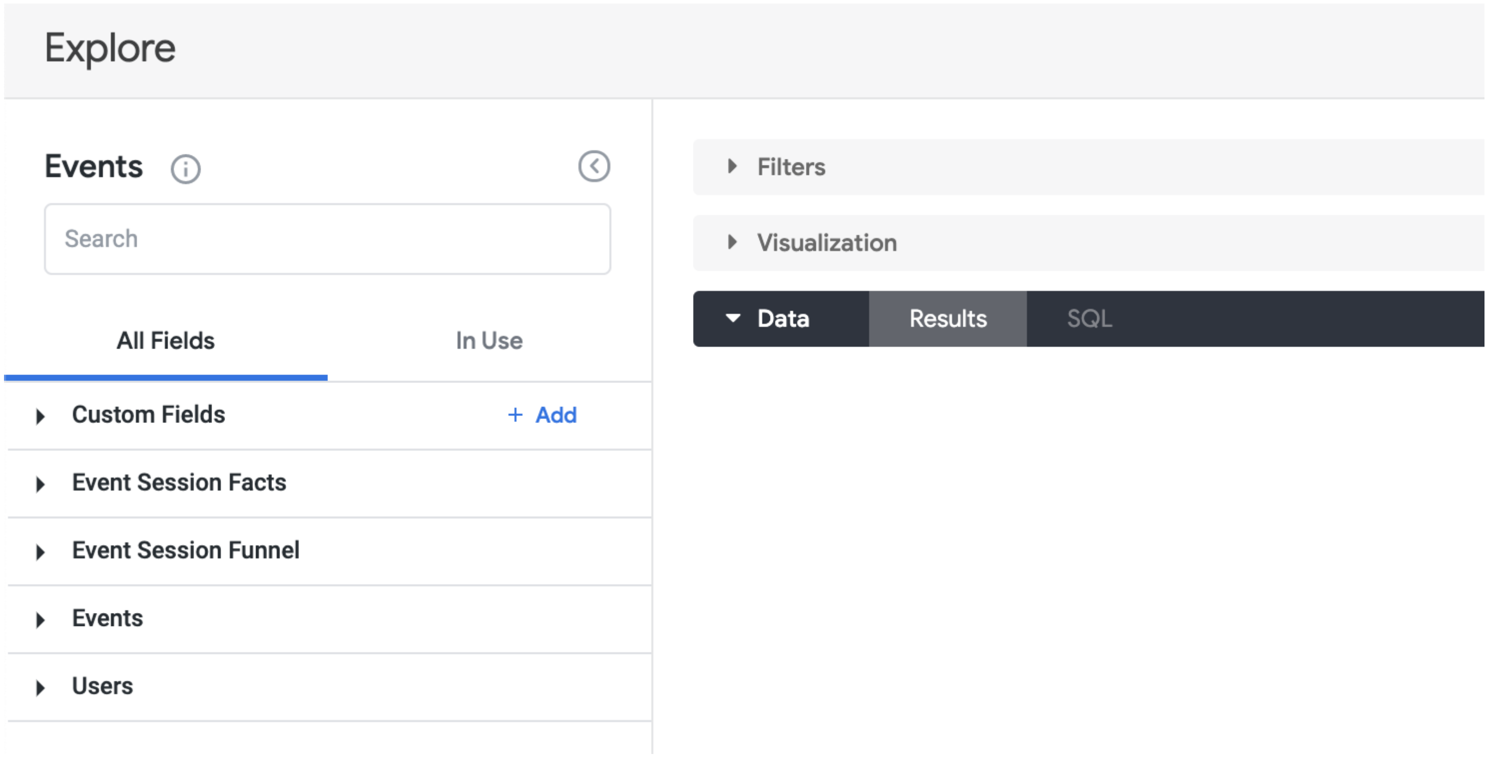
6. Click **Save Changes**, and then click **Validate LookML**.
   No LookML errors were found, and your file should resemble the following:



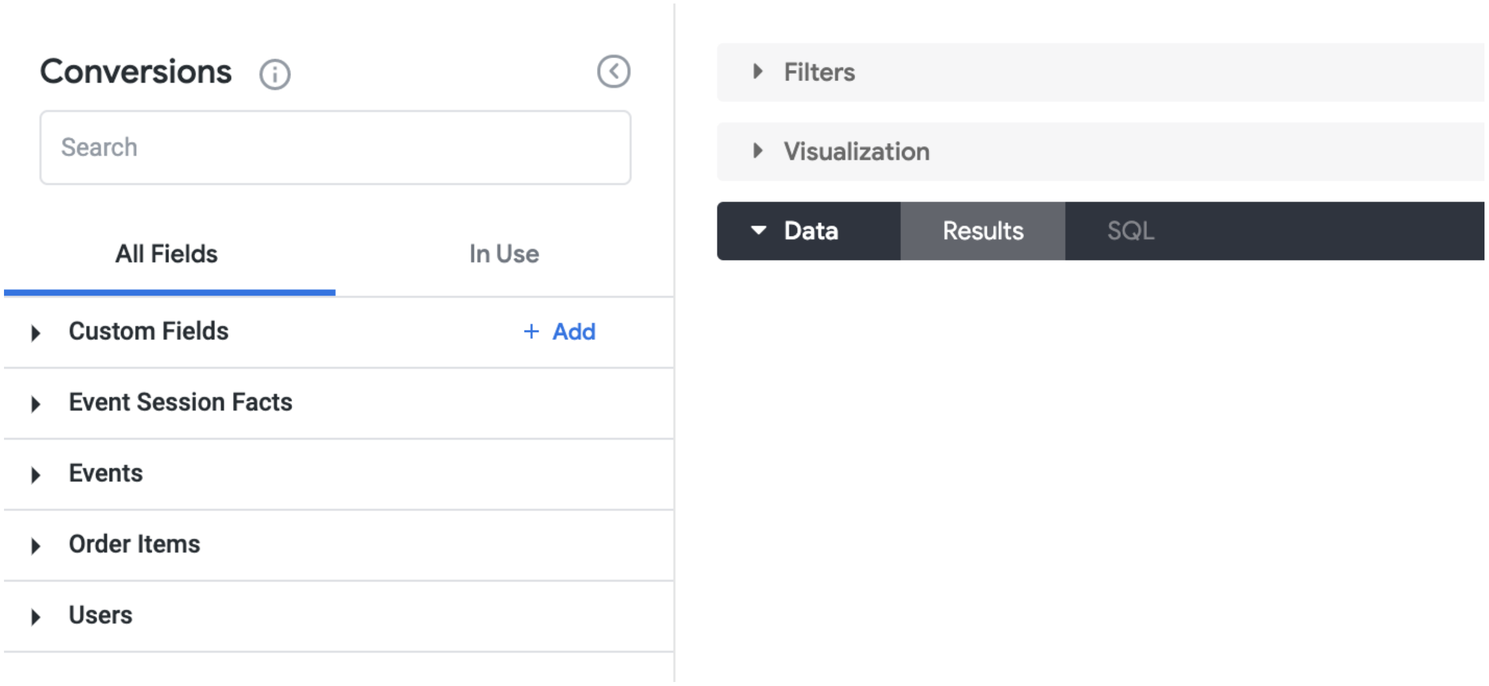
```
explore: events {
  description: "Start here for Event analysis"
    fields: [ALL_FIELDS*]
  from: events
    view_name: events
    extends: [base_events]
  join: event_session_funnel {
    type: left_outer
    sql_on: ${events.session_id} = ${event_session_funnel.session_id} ;;
    relationship: many_to_one
  }
}

explore: conversions {
  description: "Start here for Conversion Analysis"
  fields: [ALL_FIELDS*, -order_items.total_revenue_from_completed_orders]
  from: events
  view_name: events
  extends: [base_events]
  join: order_items {
    type: left_outer
    sql_on: ${users.id} = ${order_items.user_id} ;;
    relationship: many_to_many
  }
}
```

Now it's time to test your new Explores. Go to each Explore (**Events** and **Conversions**), and notice which views are included. Because the Explores share a core set of views but are customized with additional views, each one serves a different user audience.

7. Navigate to the **Events** Explore, which contains the views joined in the base Explore (**Events**, **Event Session Facts**, **Users**) plus the **Event Session Funnel** view.



8. To review the description, hold the pointer over **Information** (ⓘ) next to **Events**.
9. Navigate to the **Conversions** Explore, which contains the views joined in the base Explore (**Events**, **Event Session Facts**, **Users**) plus the **Order Items** view.



10. To review the description, hold the pointer over **Information** (ⓘ) next to **Conversions**.
11. Review the measures in the **Order Items** view; `total_revenue_from_completed_orders` is not listed.
12. Return to the **training_ecommerce.model** file in the Looker IDE.

Commit changes and deploy to production

1. Click **Validate LookML** and then click **Commit Changes & Push**.
2. Add a commit message and click **Commit**.
3. Lastly, click **Deploy to Production**.

Click *Check my progress* to verify the objective. Extend an Explore to add joins from another Explore

Congratulations!

In this lab, you created a new view containing location dimensions by extending the users and events view, created a base Explore that contained joins for multiple views, and used extends to reuse and customize the base Explore for multiple teams and users.

Finish your quest

This self-paced lab is part of the Advanced LookML Concepts in Looker and the Manage Data Models in Looker skill badge quest. A quest is a series of related labs that form a learning path. Completing a quest earns you a badge to recognize your achievement. You can make your badge (or badges) public and link to them in your online resume or social media account. Enroll in a quest and get immediate completion credit if you've taken this lab. See the Google Cloud Skills Boost catalog to see all available quests.

Next steps / learn more

- LookML quick reference
- LookML terms and concepts
- Looker Community
- Additional LookML basics

Google Cloud training and certification

...helps you make the most of Google Cloud technologies. Our classes include technical skills and best practices to help you get up to speed quickly and continue your learning journey. We offer fundamental to advanced level training, with on-demand, live, and virtual options to suit your busy schedule. Certifications help you validate and prove your skill and expertise in Google Cloud technologies.

Manual Last Updated September 06, 2023

Lab Last Tested September 06, 2023