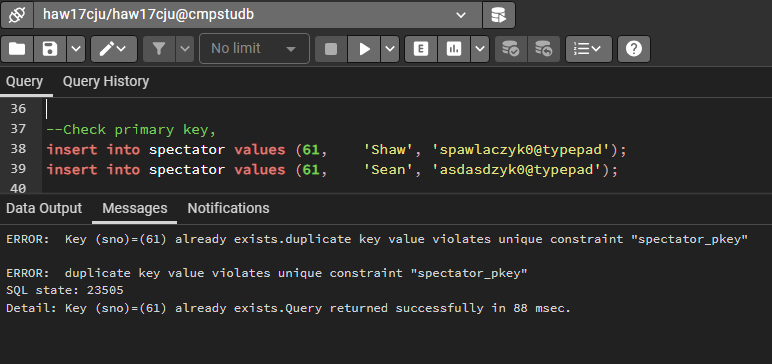
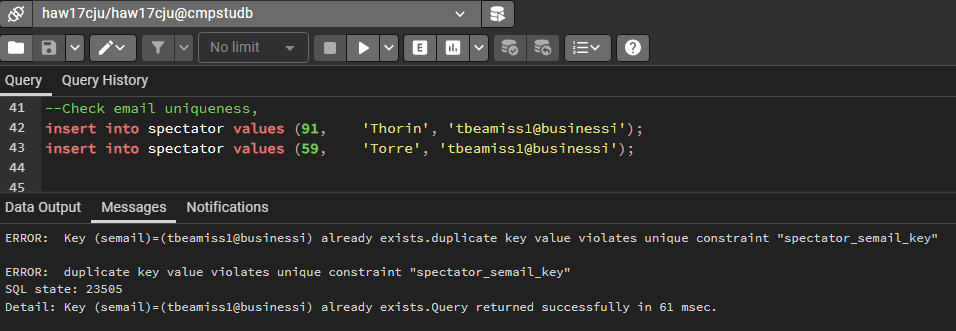
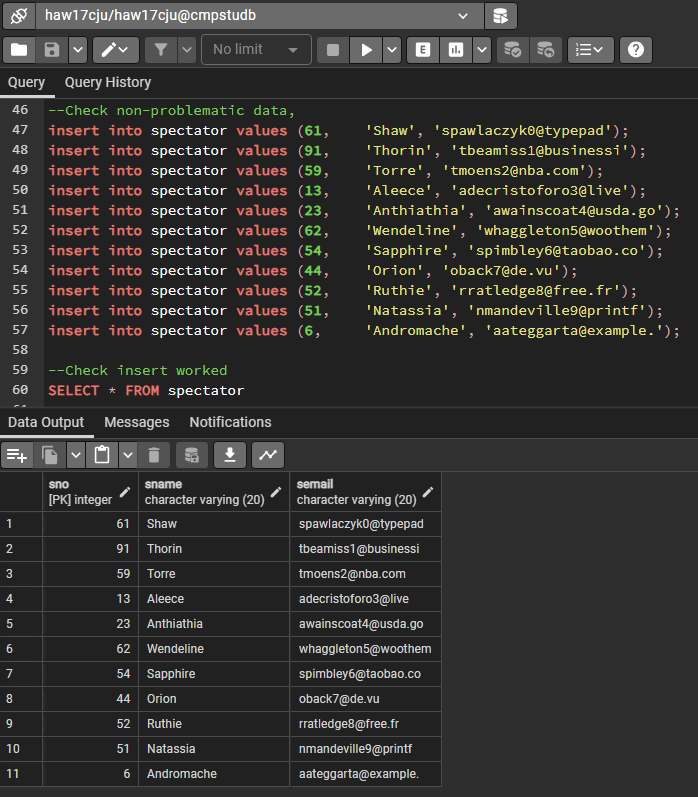
**Transactions of Interest:**

**Note:** this document was made while code was being edited so line numbers in submitted code may not match the line numbers in the screenshots (I’ve also neatened some of the layout so again there may be indents or comments that aren’t present in the screenshots), apologies for any inconvenience this causes.

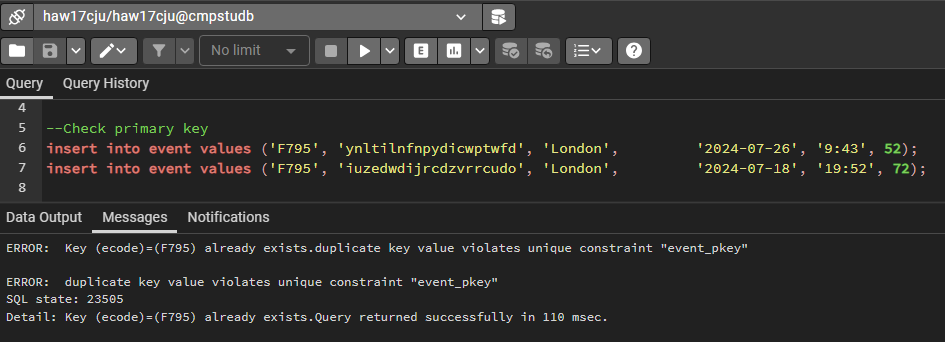
1. Insert a new spectator.



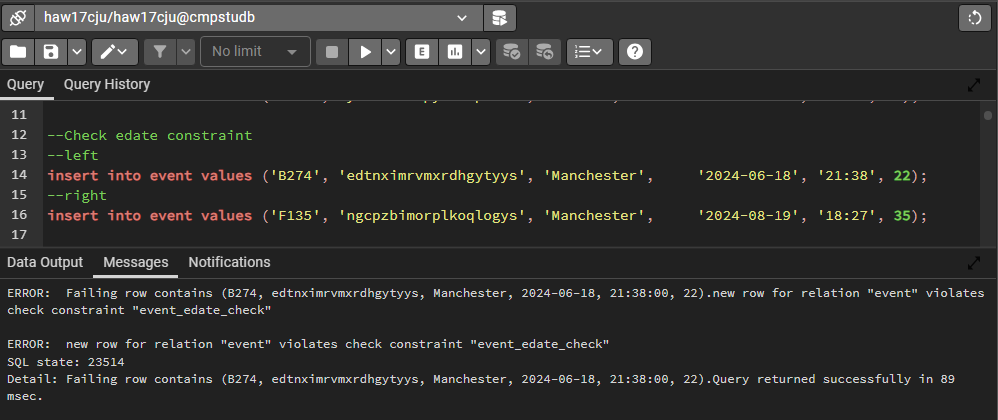




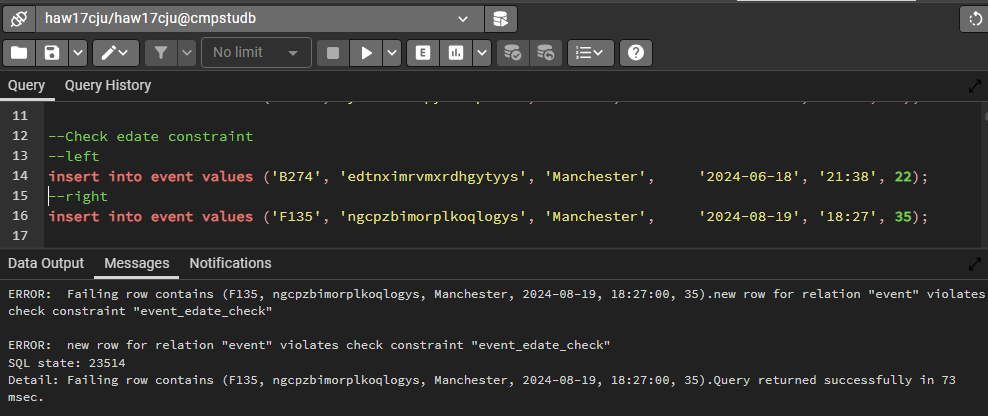
1. Insert a new event.

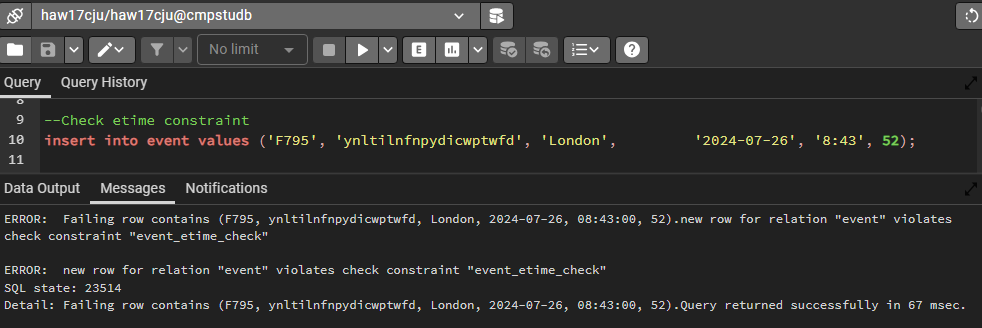


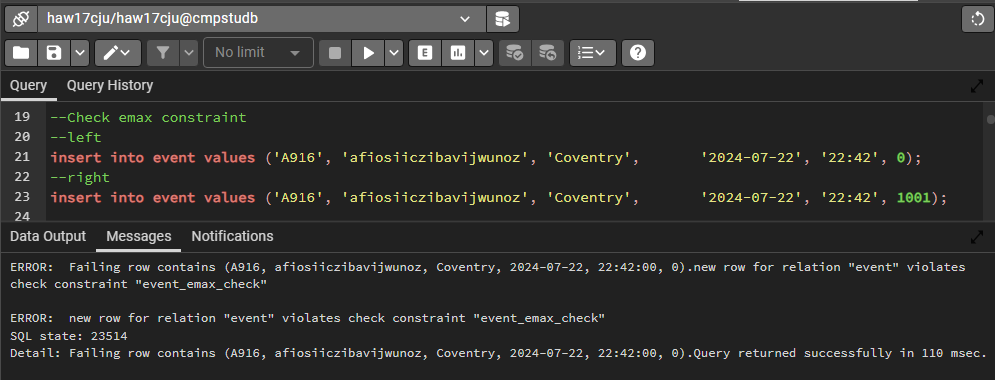
Left (<)



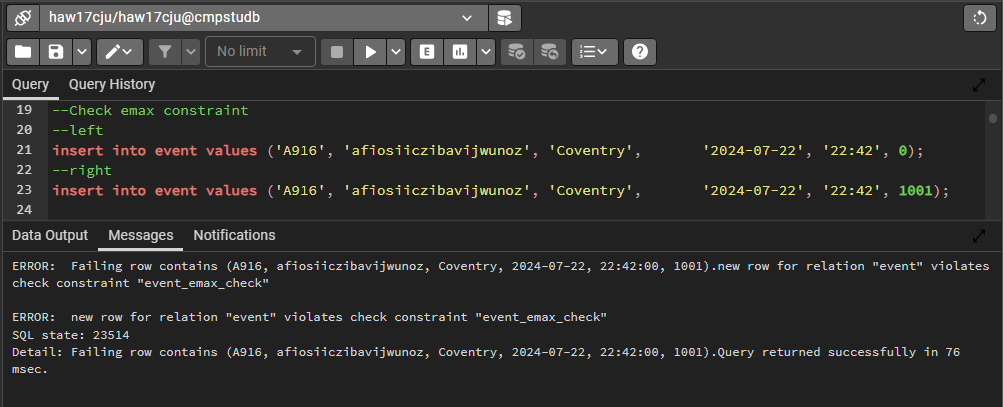
Right (>)



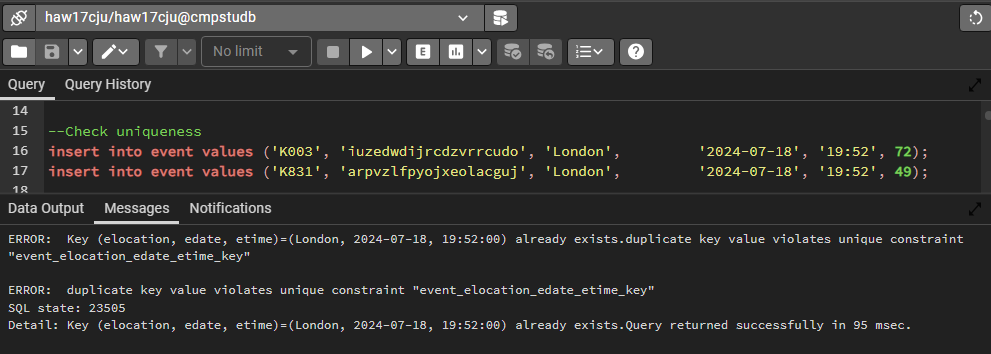


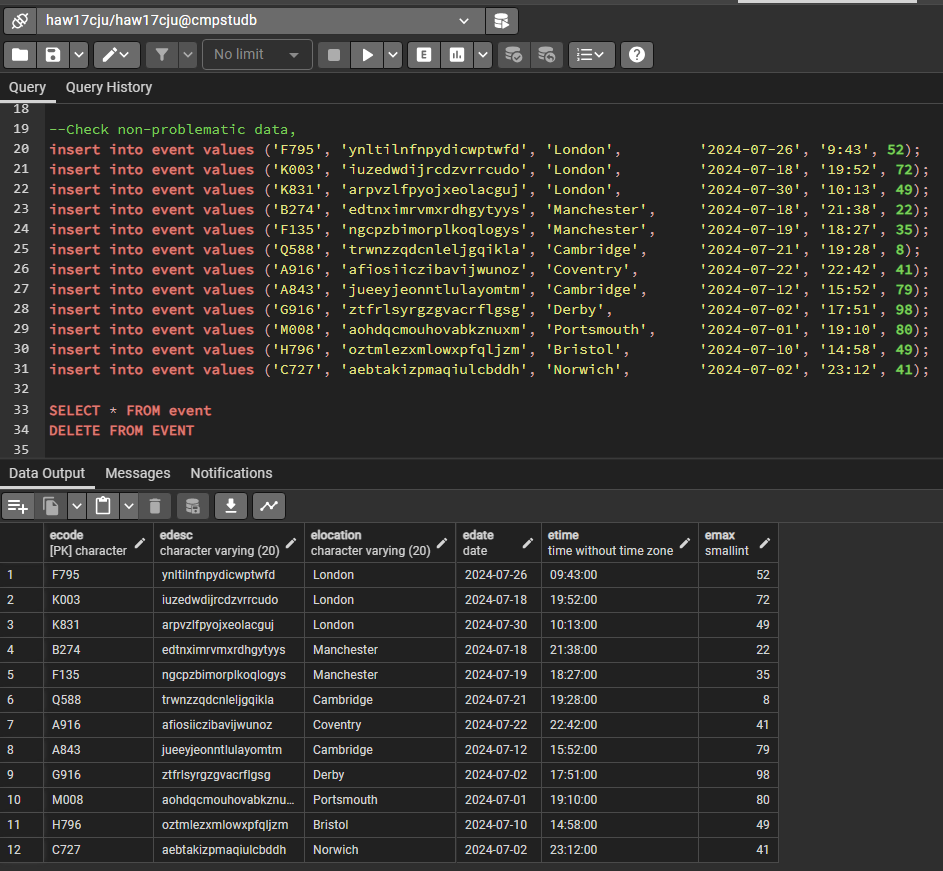
Left (<)

Right (>)



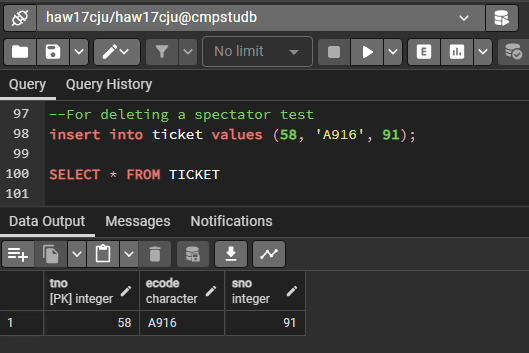
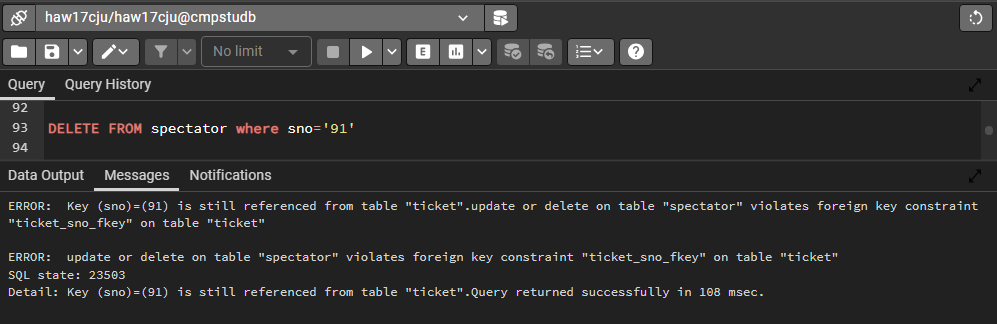
This comes from the assumption that one location can only host one event at a time (if this is not the case, we would exclude the constraint),

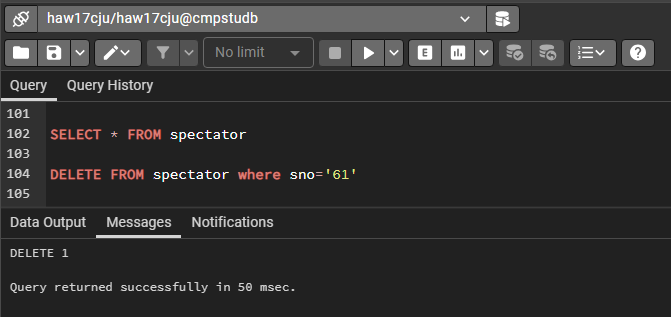




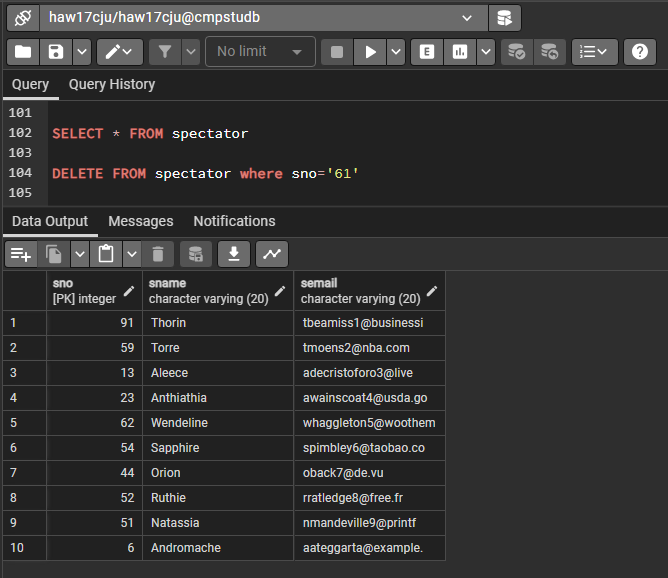
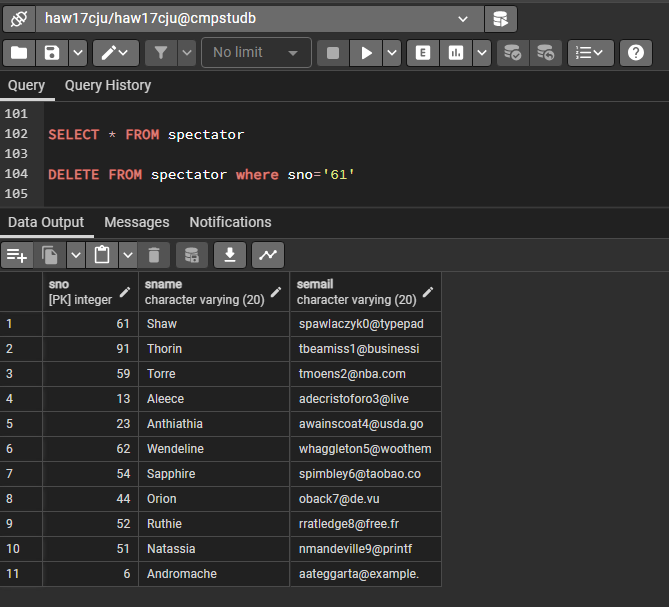
1. Delete a spectator. The spectator must not have any valid (i.e. not cancelled) tickets before it can be deleted.

We give a spectator a ticket, then try to delete them.

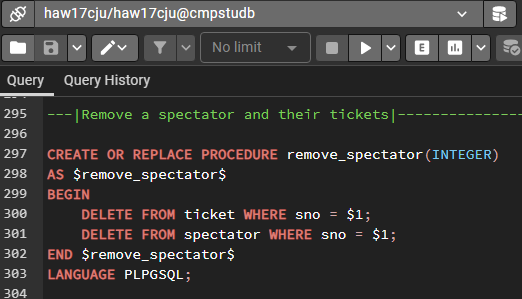
 

Now delete a spectator without a ticket. 

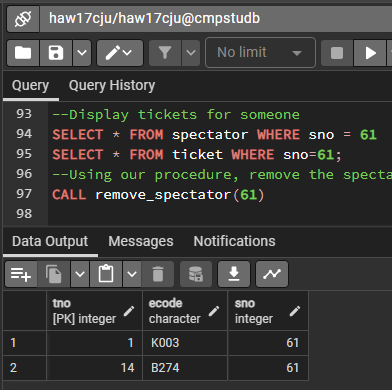
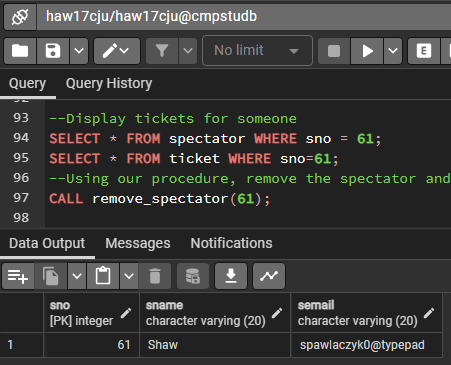
Before (left), After (right).



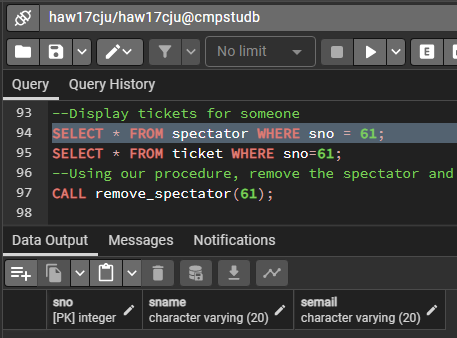
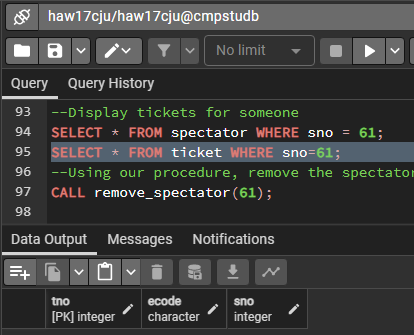
Now just for ease of use lets create a procedure to delete a spectator and all their tickets (this is not required for the transaction, but I thought I’d do it),



Before,

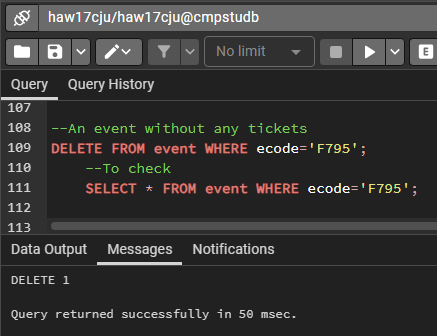
 

After,

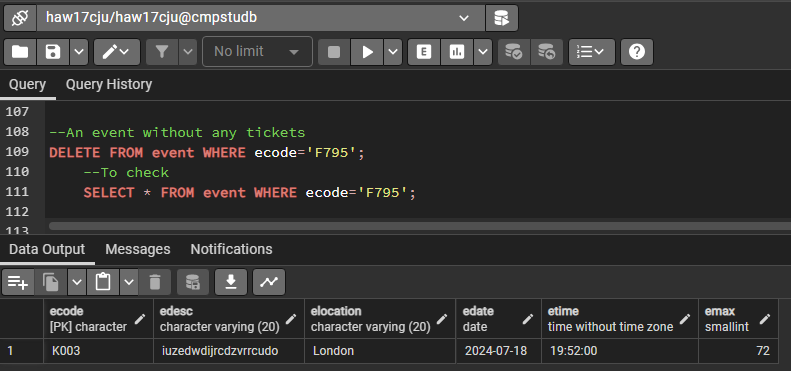
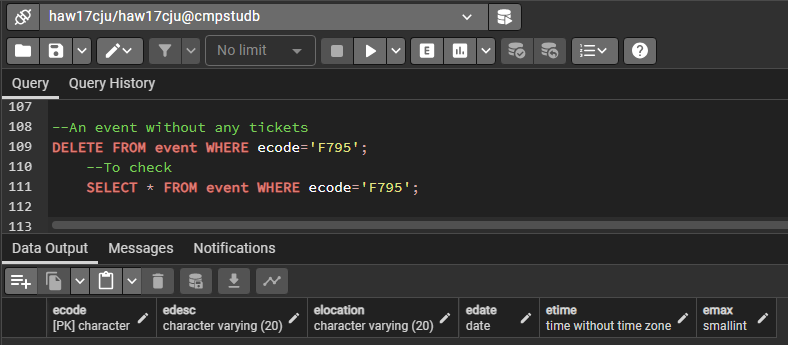


1. Delete an event. All the tickets for the event must be cancelled before an event can be deleted.

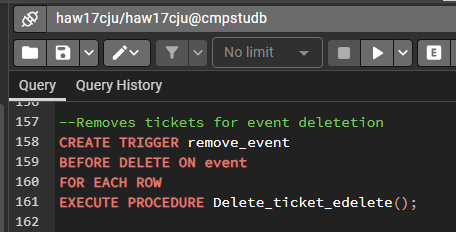
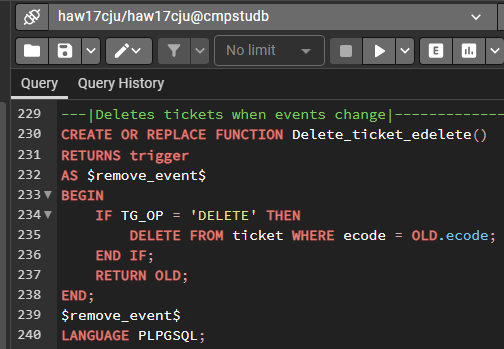
Deleting an event with no associated tickets is easy,

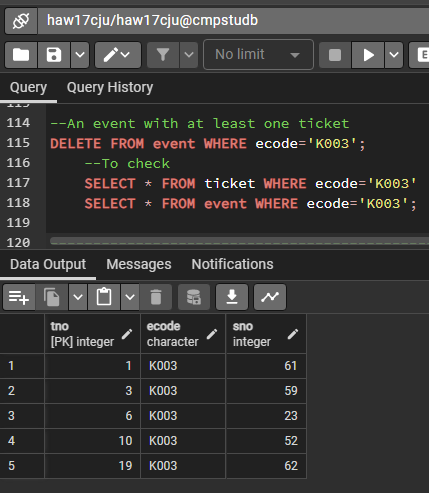
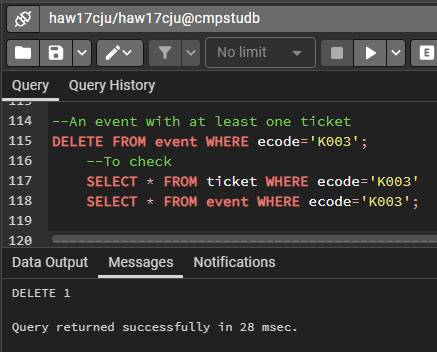


Before (left), After (right).

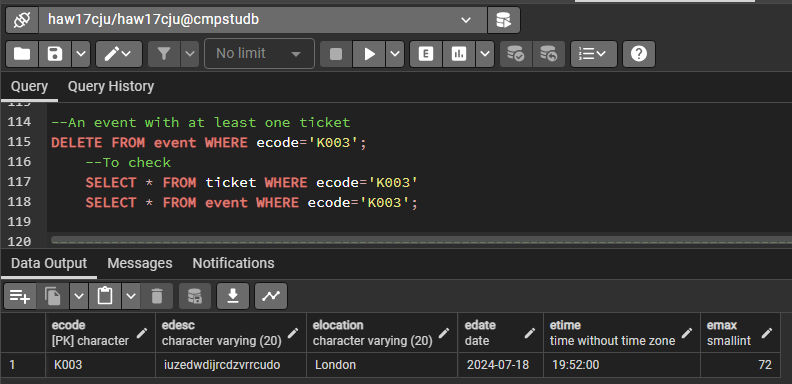
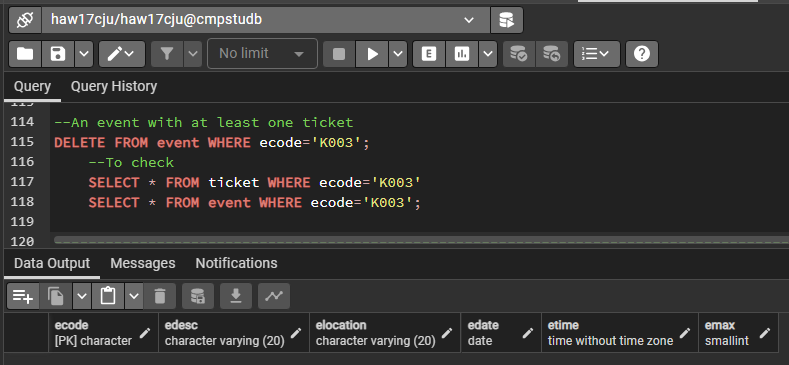
 

Now to delete an event with at least one ticket we’ll need a trigger and a function (we could also use a procedure, but I prefer triggers because it’s automatic),

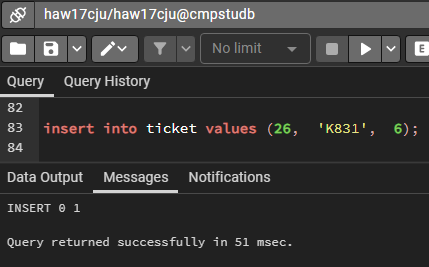
 

Before (left), After (right).

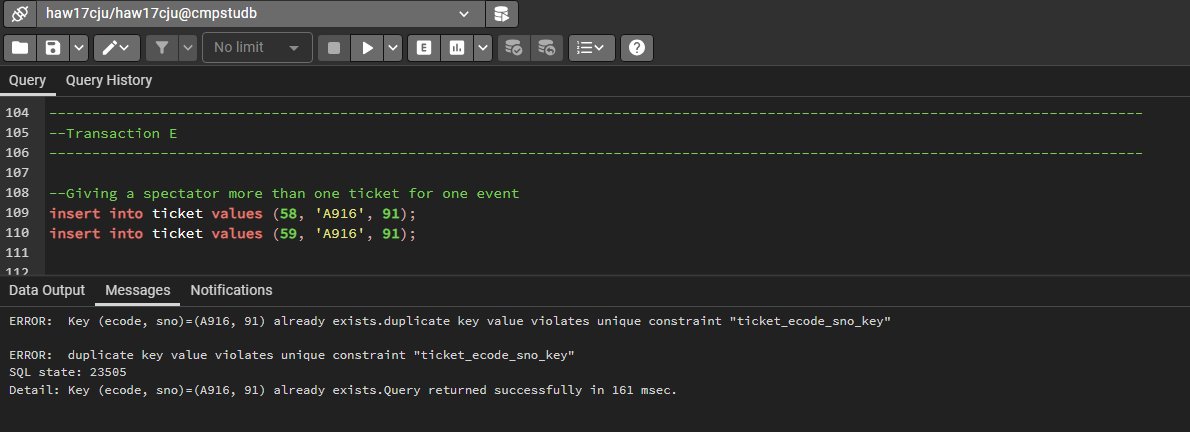
 

1. Issue a ticket for an event. A spectator may have only one ticket for a given event.

Successful ticket issue shown above. (but for completeness sake we’ll input one from the python file)

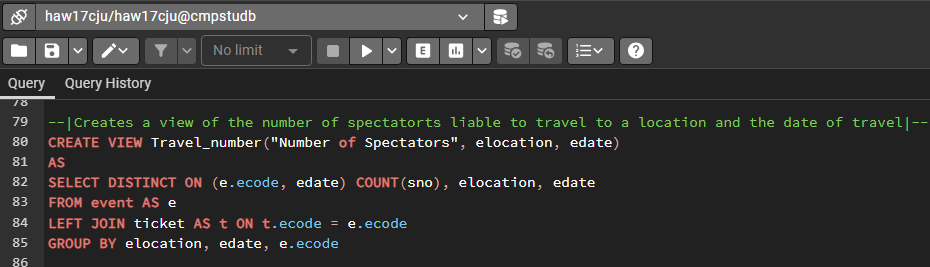


attempt to issue a spectator more than one ticket for the one event,



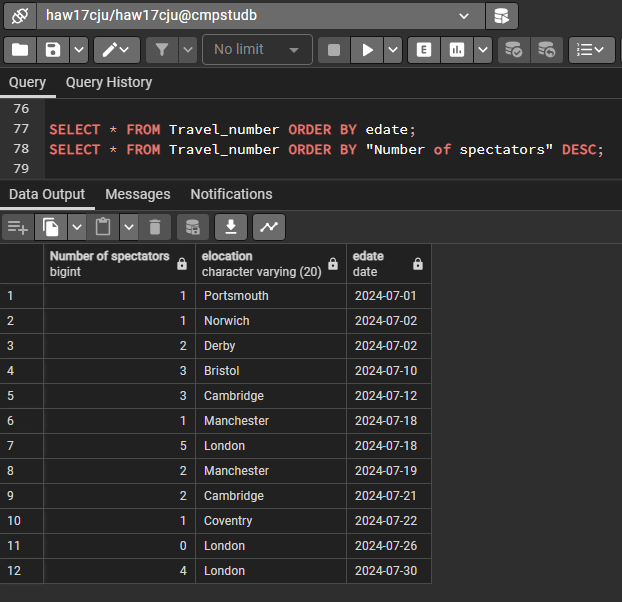
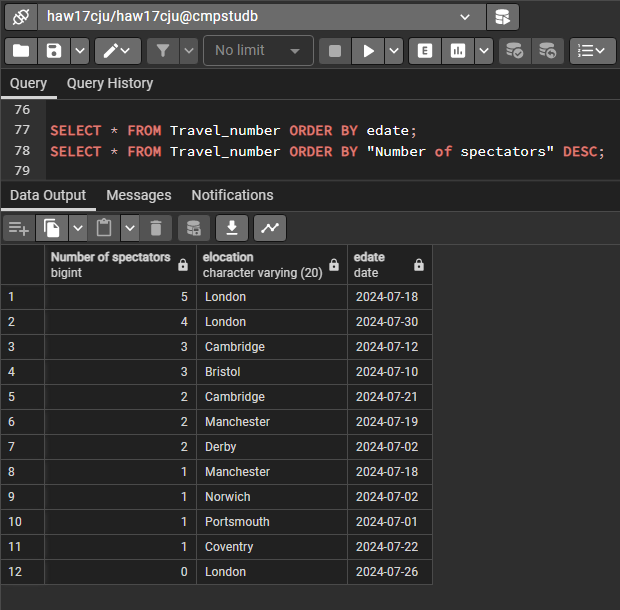
1. Produce a report showing the total number of spectators liable to travel to a location. The table should show the total number of spectators that could travel to a location on each date an event is held at a location.

Create a view based on the requirements (The time of day the event is held at is not considered),



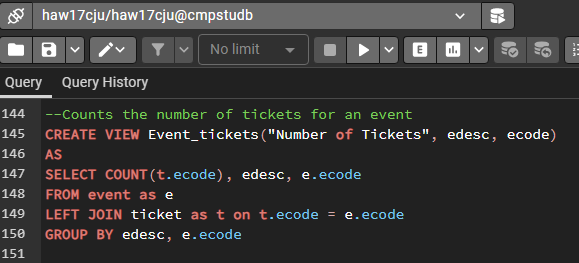
Call the view with an order by clause,

*Left (ordered by edate), Right (ordered by number of tickets)*

****

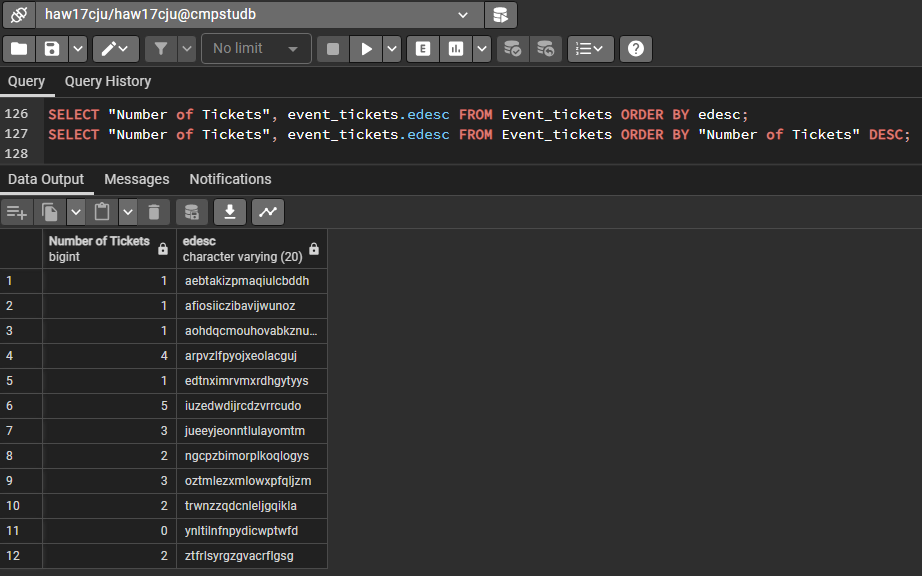
1. Produce a report showing the total number of tickets issued for each event. Present the data in event description sequence. *Under the assumption issued here means valid and not any ticket produced.*

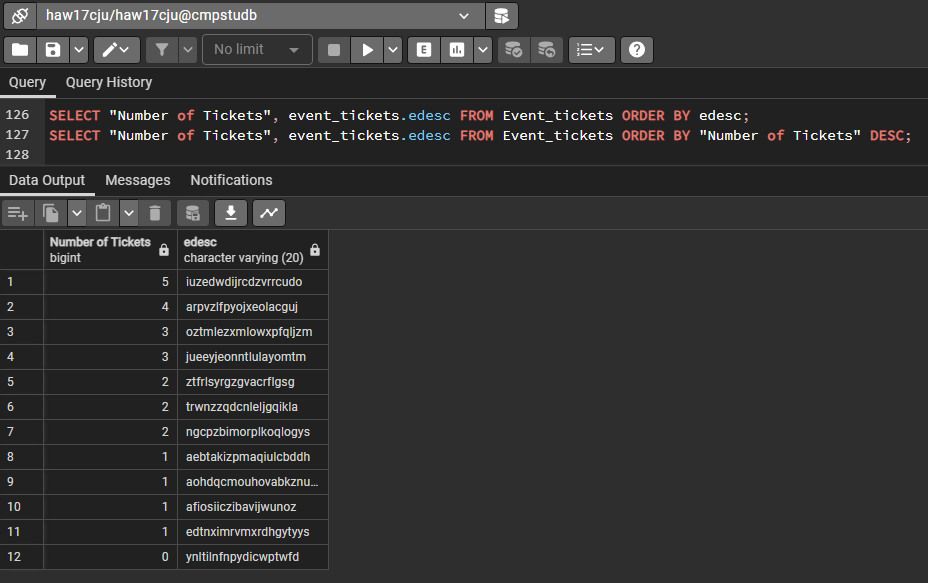
Create a view based on the requirements (we include ecode for ease of use),



Call the view with an order by clause (call only columns asked for),

*First (ordered by edesc), Last (ordered by count of tickets)*

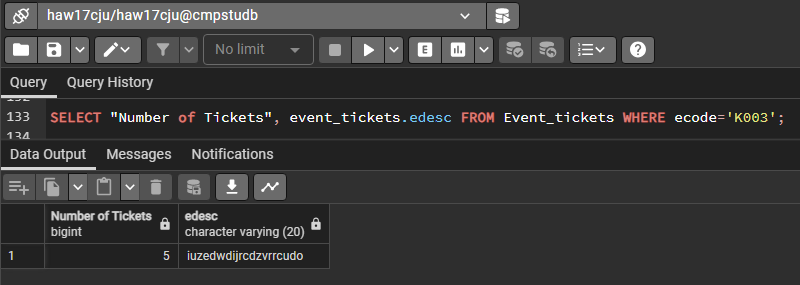




If we’re not operating under this assumption. That it’s just valid tickets but rather any ticket made for an event including the cancelled ones. We would have to union a search through both tables ticket and cancel (after join each with spectator separately). Then we must count and group by for that so that we don’t have separate rows for cancelled and valid tickets. In other words, it becomes quite complicated.

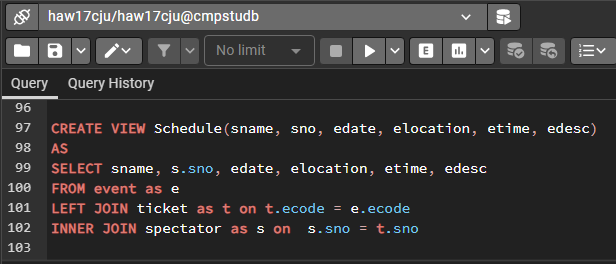
1. As task G but only for a given event which is specified by the event code.

We use the view from G which we can sort by ecode because we’ve included it. Then use where to decide the ecode we want. *Remove order by since we only have one item.*

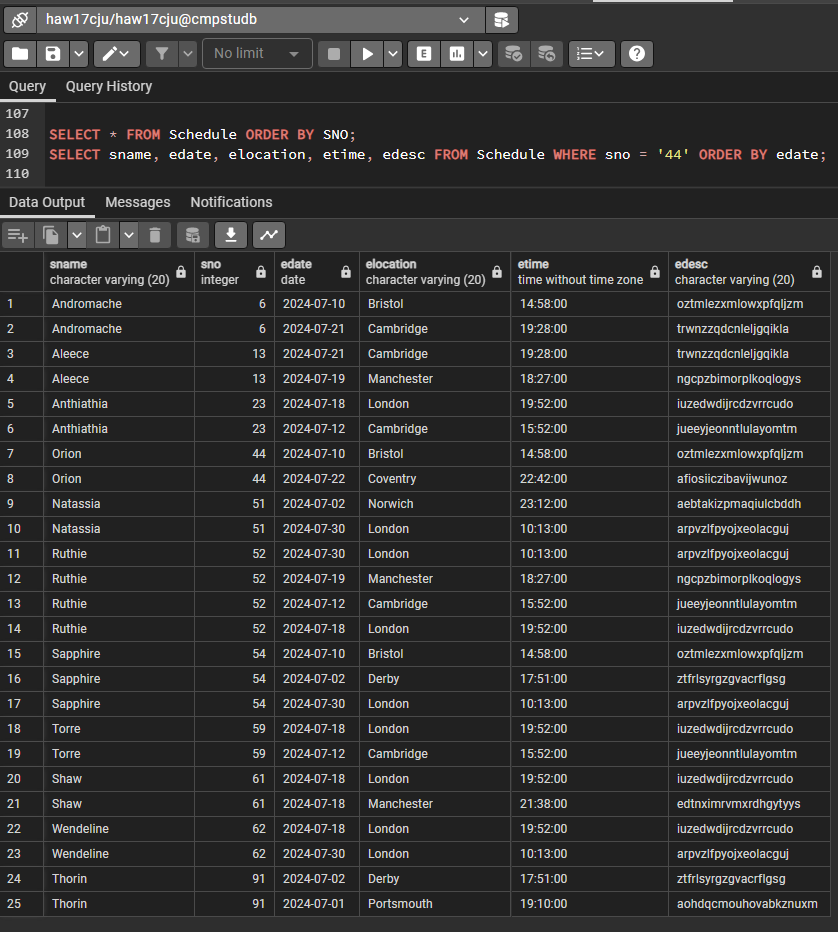


1. Produce a report showing the schedule for a given spectator. The spectator is specified by his/her spectator number. The schedule should contain the spectator's name and the date, location, time and event description of each event for which the spectator has been issued a ticket.

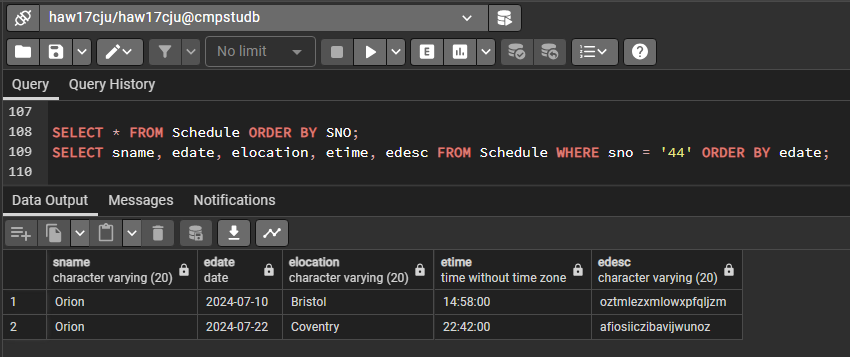
Under the assumption issued here means valid and not any ticket produced. We create a view based on the requirements,

****

Call the view with a spectator number to check it works,



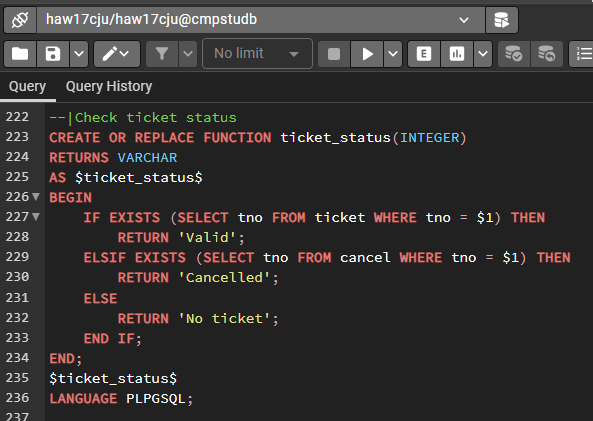
Call the view with spectator number specified (ordered by date for ease of use, sno not asked for so not included in selection),



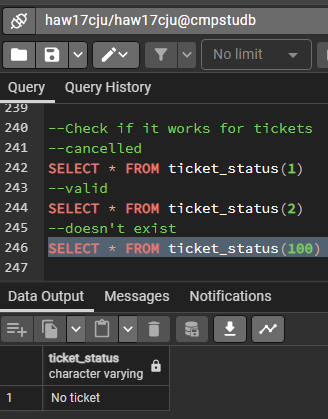
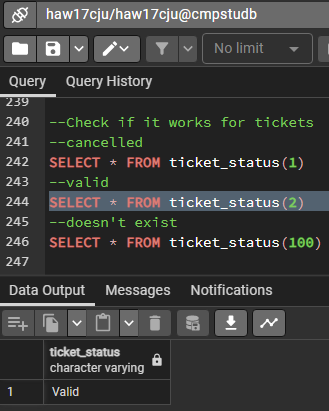
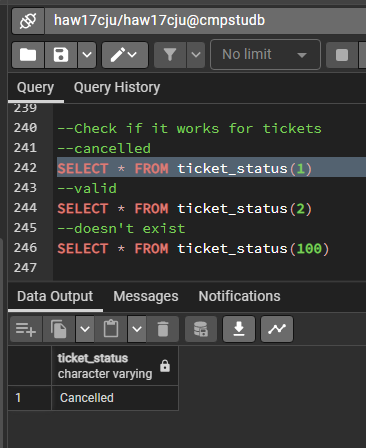
1. Given a specific ticket reference number, display the name of the spectator and the event code for the ticket and indicate if the ticket is valid or is cancelled.

There’s a simple solution to this problem but it involves assuming that any ticket number provided not in the table ticket is cancelled since this is an inelegant solution, I opted for a more complicated one.

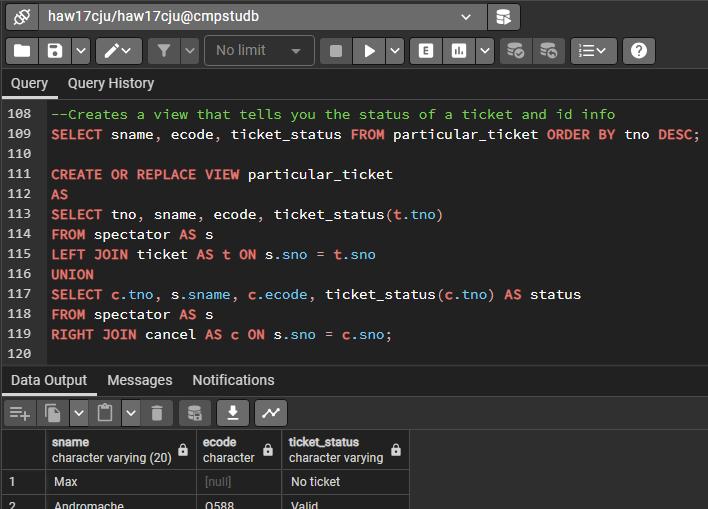
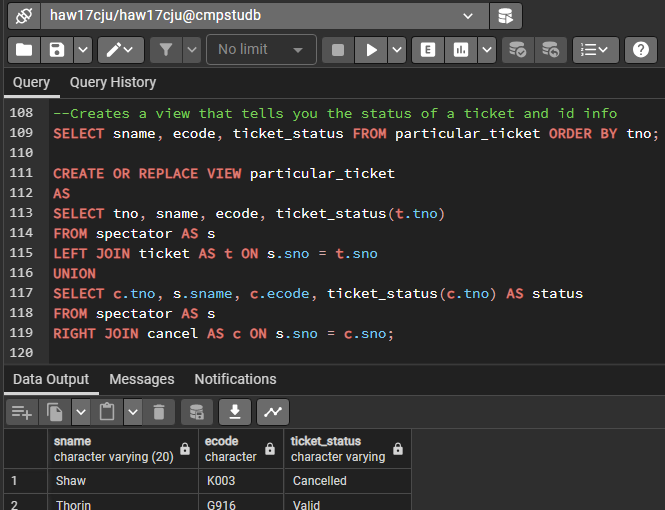
To start we’ll make a function that given a tno tells us whether the ticket is valid, cancelled or doesn’t exist (we don’t necessarily need the doesn’t exist part however it’s more informative).



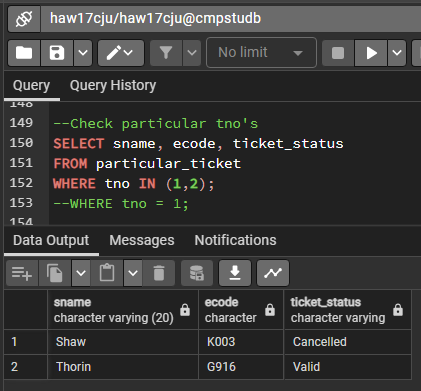
Just to check this works, test it with the three cases,



Now we’ve made this function we can make a view, this is where it get complicated because we have to check both cancel and ticket for tno. We can’t just easily do this with joins, it makes the most sense to use a union of each check (and since tno is assumed to be unique we don’t need to worry about contradictory rows).

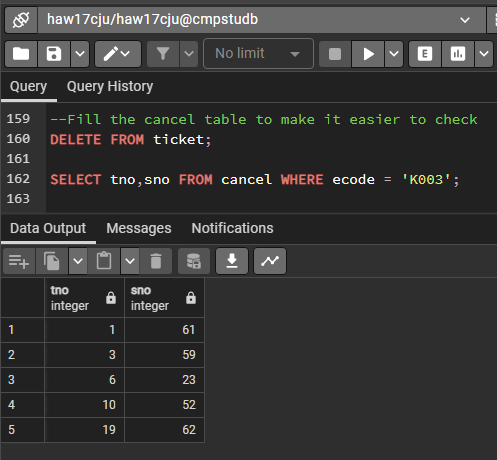


Now we can select from this view any given tno we would like,

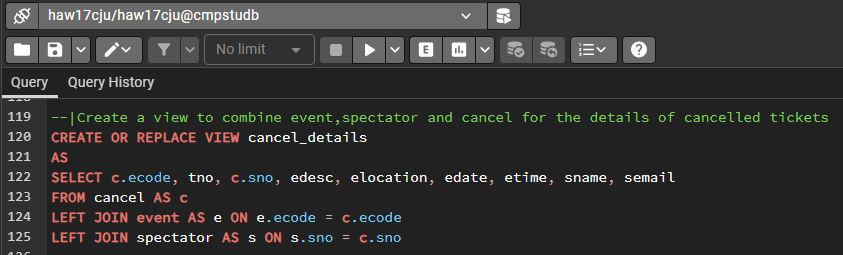


1. View the details of all cancelled tickets for a specific event.

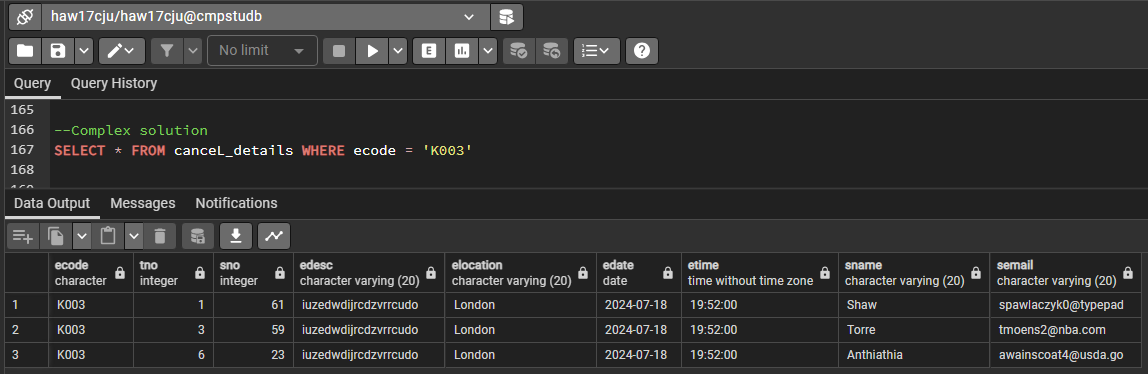
A very simple interpretation of this question is answered by a single select statement,



But let’s assume we want the spectator and event details to accompany the ticket, we’ll create a view to do this.



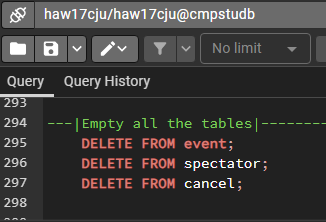
Then using a select,



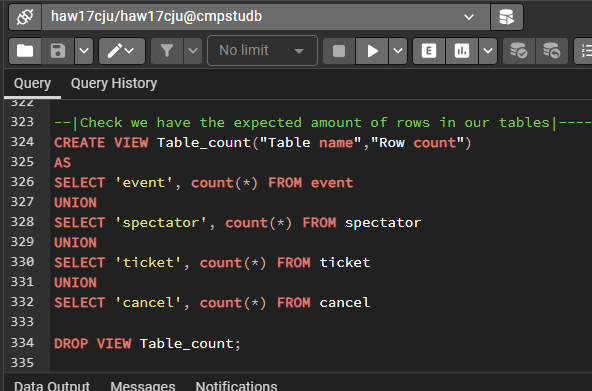
I included ecode here despite the fact we know it for the sake of being able to sanity check, but you can get rid of it by modifying the select.

1. Delete the contents of the database tables.

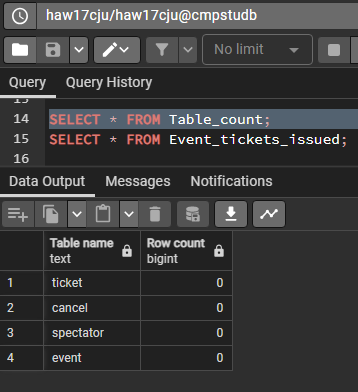
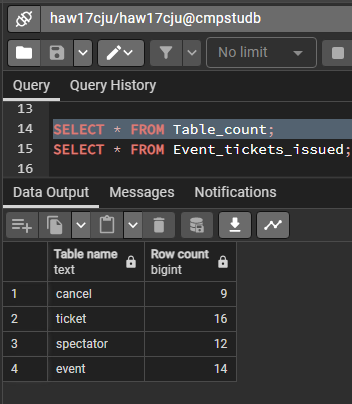
A very simple solution is just to delete the tables in an order that doesn’t cause problems for our constraints or trigger functions (i.e. ordered a specific way, for example if you put delete cancel first it would end up filling up with all the deleted tickets). Also because when an event is delete all associated tickets are deleted we don’t need to include a delete for the ticket table (since every ticket is tied to an event).



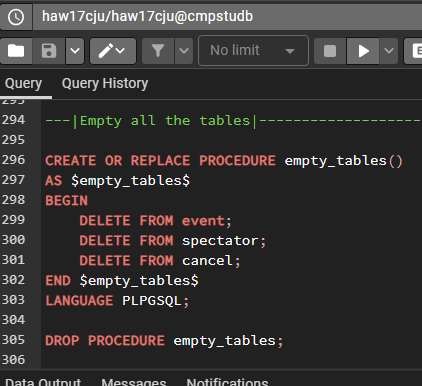
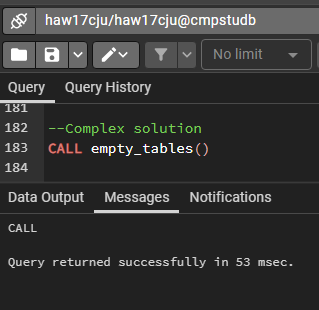
To help display the effect of this quickly we’ll a table count view,



Before (left), After (right)



Instead, we can create a procedure that does all four one after the other so we can then just call the procedure (procedure not function because it doesn’t need to return a value).

Before (left), After (right)

