Team: Savage and Average
Team Members: Mallory Huston and Sory Diagouraga
Date: Monday, July 25th, 2022

Project Step 3 Final Version: Design HTML Interface

URL to an index.html page:

https://web.engr.oregonstate.edu/~diagours/cs340/index.html

Feedback by the peer reviewers

Owen Williamson

Does the UI utilize a SELECT for every table in the schema? **In other words, data from each table in the schema should be displayed on the UI. Note: it is generally not acceptable for just a single query to join all tables and displays them.**

Yes, data from each table is present and the data is not all combined into one table. I would recommend using an inner join function for your foreign keys so that instead of seeing a number in the, for example concert_ID column of tickets you can see the concert name instead, which would make the program more user friendly

Does at least one SELECT utilize a search/filter with a dynamically populated list of properties?

Yes it does, but again I would suggest doing a little more with the inner joins like I mentioned above.

Does the UI implement an INSERT for every table in the schema? **In other words, there should be UI input fields that correspond to each table and attribute in that table.**

I see the ability to add artists on the home page, but I do not see a UI for any other inserts. I do not think that the UI is functional at this point.

Does each INSERT also add the corresponding FK attributes, including at least one M:M relationship? **In other words if there is a M:M relationship between Orders and Products, INSERTing a new Order (e.g. orderID, customerID, date, total), should also INSERT row(s) in the intersection table, e.g. OrderDetails (orderID, productID, qty, price and line_total).**

Given that the UI insert is not working I cannot verify that this is working, but it looks like in the UI it will be possible to add foreign keys.

Is there at least one DELETE and does at least one DELETE remove things from a M:M relationship? **In other words, if an order is deleted from the Orders table, it should also delete the corresponding rows from the OrderDetails table, BUT it should not delete any Products or Customers.**

The UI is present to delete an artist, but it is not yet functional.

Is there at least one UPDATE for any one entity? **In other words, in the case of Products, can productName, listPrice, qtyOnHand, e.g. be updated for a single ProductID record?**

Again, the UI for an artist update is present, but it is not yet funcitonal.

Is at least one relationship NULLable? **In other words, there should be at least one optional relationship, e.g. having an Employee might be optional for any Order. Thus it should be feasible to edit an Order and change the value of Employee to be empty.**

Yes, there are multiple optional relationships.

Do you have any other suggestions for the team to help with their HTML UI? For example using AS aliases to replace obscure column names such as fname with First Name.

I would recommend using artist and employee names to identify them in your select queries instead of their id to make it more readable for the user. It also looks like you are still working on your UI, but I would recommend having the artist insert interface, and all of them for that matter, in their respective webpages below the table, instead of on the main page.

Adam Pruitt

Great job group 24! I like your design. It seems functional and you guys did great work!

- ***Does the UI utilize a SELECT for every table in the schema?*** **In other words, data from each table in the schema should be displayed on the UI. Note: it is generally not acceptable for just a single query to join all tables and displays them.**

The UI has links to all the tables except for the two intersection tables. You could always do the suggestion in the project guide which says “One exception is oftentimes it works better for the UX to have a single web page for a Many-to-Many relationship between 2 tables (so the user

doesn't have to view two pages to complete a transaction in both tables). So in that case if you had 4 entities that were implemented as 5 tables, with 1 many-to-many relationship between 2 of those tables, and the 2 tables in that m:m were managed on a single web page, then we expect 4 web pages in the project."

- ***Does at least one SELECT utilize a search/filter with a dynamically populated list of properties?***

Yes, there is at least one select that has an inner join. This would allow the UI to have a search/filter.

- ***Does the UI implement an INSERT for every table in the schema?*** **In other words, there should be UI input fields that correspond to each table and attribute in that table.**

No, at this point it does not have an insert for each table. There are SELECT queries for each of the tables though.

- ***Does each INSERT also add the corresponding FK attributes, including at least one M:M relationship?*** **In other words if there is a M:M relationship between Orders and Products, INSERTing a new Order (e.g. orderID, customerID, date, total), should also INSERT row(s) in the intersection table, e.g. OrderDetails (orderID, productID, qty, price and line_total).**

I don't see any FK attributes because I don't see any inserts into any of the intersection tables. All of the inserts seem correct but I am unsure about the FK attributes.

- ***Is there at least one DELETE and does at least one DELETE remove things from a M:M relationship?*** **In other words, if an order is deleted from the Orders table, it should also delete the corresponding rows from the OrderDetails table, BUT it should not delete any Products or Customers.**

There is at least one DELETE for use with Tickets. I do not see a DELETE to remove items from a M:M relationship.

- ***Is there at least one UPDATE for any one entity?*** **In other words, in the case of Products, can productName, listPrice, qtyOnHand, e.g. be updated for a single ProductID record?**

Yes, there are two updates right now for Fans and Employees. There is also an update in the UI for Artists.

- ***Is at least one relationship NULLable?*** **In other words, there should be at least one optional relationship, e.g. having an Employee might be optional for any Order. Thus it should be feasible to edit an Order and change the value of Employee to be empty.**

I am not 100% sure about this question but in the DDL_clearwaterArena file there are some table attributes that are DEFAULT NULL. For example in the table Fans there are multiple that are set as DEFAULT NULL.

- ***Do you have any other suggestions for the team to help with their HTML UI? For example using AS aliases to replace obscure column names such as fname with First Name.***

I don't have any other suggestions. I did like Beckett's suggestions for tickets where instead of having the ID you have the concert name or something like that.

Beckett Lewis

Nice job on this guys! Here are my responses to the questions.

- *Does the UI utilize a SELECT for every table in the schema?* In other words, data from each table in the schema should be displayed on the UI. Note: it is generally not acceptable for just a single query to join all tables and displays them.

Yes, it looks like there each table in the schema has a view in the UI.

- *Does at least one SELECT utilize a search/filter with a dynamically populated list of properties?*

Yes there are multiple tables that utilize inner joins to populate the columns in the DML file. II would recommend maybe even going a little farther with the joins, like on your Tickets UI showing the name of the concert or artist performing with the name of the fan instead of just showing the ID numbers.

- *Does the UI implement an INSERT for every table in the schema?* In other words, there should be UI input fields that correspond to each table and attribute in that table.

It doesn't look like it has been added to the HTML files but there are SQL statements in the DML file for inserting into the table.

- *Does each INSERT also add the corresponding FK attributes, including at least one M:M relationship?* In other words if there is a M:M relationship between Orders and Products, INSERTing a new Order (e.g. orderID, customerID, date, total), should also INSERT

row(s) in the intersection table, e.g. OrderDetails (orderID, productID, qty, price and line_total).

It looks like each insert is correctly formulated to reference foreign keys. I don't think there are any auto generated items in the intersection table form it though.

- *Is there at least one DELETE and does at least one DELETE remove things from a M:M relationship?* In other words, if an order is deleted from the Orders table, it should also delete the corresponding rows from the OrderDetails table, BUT it should not delete any Products or Customers.

I think there is the ability to delete tickets but not from the M:M intersection table at this point.

- *Is there at least one UPDATE for any one entity?* In other words, in the case of Products, can productName, listPrice, qtyOnHand, e.g. be updated for a single ProductID record?

Yes, the Fans and Employees can be updated in the DML and there looks to be that option in the UI.

- *Is at least one relationship NULLable?* In other words, there should be at least one optional relationship, e.g. having an Employee might be optional for any Order. Thus it should be feasible to edit an Order and change the value of Employee to be empty.

I don't see any option for NULLable relationships at this point.

- *Do you have any other suggestions for the team to help with their HTML UI? For example using AS aliases to replace obscure column names such as fname with First Name.*

I think one suggestion I might have is to put the forms for adding/updating/ or deleting inside of the Artist html file instead of artist so it's easier to match up who I want to change or delete. Also, I mentioned in another question but for tickets it might be useful to see more information about who the fan that has the tickets is and maybe the artist(s) that the concert is for.

Paul Reyneveld

Does the UI utilize a SELECT for every table in the schema? In other words, data from each table in the schema should be displayed on the UI. Note: it is generally not acceptable for just a single query to join all tables and displays them.

The UI does have links to dummy tables for every entity in the diagram except for the intersection tables.

Does at least one SELECT utilize a search/filter with a dynamically populated list of properties?

If by search/filter the question is referring to a select statement with an inner join that is parameterized, then yes. The DML has multiple such queries.

Does the UI implement an INSERT for every table in the schema? In other words, there should be UI input fields that correspond to each table and attribute in that table.

No. At this point the UI hasn't been fully fleshed out to have all of that functionality. The UI provides potential functionality for adding, updating, and deleting artists.

Does each INSERT also add the corresponding FK attributes, including at least one M:M relationship? In other words if there is a M:M relationship between Orders and Products, INSERTing a new Order (e.g. orderID, customerID, date, total), should also INSERT row(s) in the intersection table, e.g. OrderDetails (orderID, productID, qty, price and line_total).

The DML provides queries for INSERTs into all non-interesection tables. I don't see functionality for auto-generated intersection tables at this time.

Is there at least one DELETE and does at least one DELETE remove things from a M:M relationship? In other words, if an order is deleted from the Orders table, it should also delete the corresponding rows from the OrderDetails table, BUT it should not delete any Products or Customers.

There is a DELETE operation in the DML for tickets, but I don't see any other DELETEs listed in the DML.

Is there at least one UPDATE for any one entity? In other words, in the case of Products, can productName, listPrice, qtyOnHand, e.g. be updated for a single ProductID record?

There are multiple UPDATEs listed in the DML. There are UPDATEs for both the Fans and the Employees tables.

Is at least one relationship NULLable? In other words, there should be at least one optional relationship, e.g. having an Employee might be optional for any Order. Thus it should be feasible to edit an Order and change the value of Employee to be empty.

I don't see any NULLable options, but in all honesty, I didn't realize this was a project requirement and didn't include that in my own project.

Do you have any other suggestions for the team to help with their HTML UI? For example using AS aliases to replace obscure column names such as fname with First Name.

Nothing comes to mind. Obviously, this is a front end for an administrator so aesthetics really shouldn't be much of a consideration. I assume the team will add more forms for other DB operations, but otherwise, it seems fine to me.

Fixes based on Feedback from Previous Steps

Actions Taken:

Overview: We immediately changed the typo of "Clearwater Arena" to "Clearwater Casino" in the overview to avoid any further confusion. We then took out "promotions" and "genres" in the overview since we did not cover that whatsoever since it would make this project way too complicated.
Employees and Artists: We also took out any relationship between the Employees and Artists because it was not necessary and would be far too complicated. Plus, it is not how real life works and whichever employees would be helping out the artists are the ones who are on staff that night. There cannot be more than one show per night either.
numberOfTickets: We also put in "numberOfTickets" under the Concerts entity so that Employees can keep track of how many people show up and that the venue never goes over max capacity. artistID: We then made the artistID entity the same as the other entities that are unique, not NULL, auto_increment, etc.
Fans: We decided to rename the entity "Customers" to "Fans" because that type of demographic makes more sense for a concert arena anyway and it's less confusing.
MySQL Cascade: We had to input CASCADE operations between two tables that were dependent on each other. We put ON DELETE CASCADE for the constraint `fk_Tickets_Concerts1`, ON UPDATE CASCADE for `fk_Tickets_Fans1`, and ON DELETE CASCADE for `fk_Fans_Concerts1`. We put in ON DELETE CASCADE for all the foreign keys in the intersection tables as well.
Concerts_concertID: We decided to remove "Concerts_concertID" and all other similar names from the intersection tables and other entities to avoid any further confusion.
Fans & Concerts: We decided to remove the relationship between Fans and Concerts because Tickets is already an intersection table in between those two entities.

AUTO_INCREMENT: Earlier, we forgot to make artistID, concertID, fanID, ticketID, and employeeID all AUTO_INCREMENT, so we fixed that. We also updated the website to no longer have any of the IDs be filled out manually, since their values are all automatically generated.
Show Details: We detailed the Artists.name performing and Employees working for the Concerts table, specifically the concertDate. We put in Artists.name, Fans.firstName, and Fans.lastName as well under the Tickets table. All of that went into the DML file.
Update & Delete Operations: We added an update option for the Artists as well as delete options for both Employees and Artists from Concerts in the DML file.

Actions Not Taken:

Genres and Promotions: We did not follow the other suggestions of putting in genres and promotions since it would overcomplicate our database.
Employee role: We also want to make it clear that all Employees can only have one role each, not multiple.
Artistscol and Artistscol1: We did delete “Artistscol” and “Artistscol 1” since they did not make much sense anyway.
concertID: Additionally, we want to make it clear that the relationship between Concerts and Fans is implemented with “concertID” while Artists and Concerts has an implementation table.

Fixes based on Feedback from Previous Steps

Actions Taken:

Overview: We immediately changed the typo of “Clearwater Arena” to “Clearwater Casino” in the overview to avoid any further confusion. We then took out “promotions” and “genres” in the overview since we did not cover that whatsoever since it would make this project way too complicated.
Employees and Artists: We also took out any relationship between the Employees and Artists because it was not necessary and would be far too complicated. Plus, it is not how real life works and whichever employees would be helping out the artists are the ones who are on staff that night. There cannot be more than one show per night either.
numberOfTickets: We also put in the entity “numberOfTickets” under Concerts so Employees can keep track of how many people show up and that the venue never goes over max capacity.
artistID: We then made the artistID entity the same as the other entities that are unique, not NULL, auto_increment, etc.
Fans: We decided to rename the entity “Customers” to “Fans” because that type of demographic makes more sense for a concert arena anyway and it’s less confusing.
MySQL Cascade: We had to input CASCADE operations between two tables that were dependent on each other. We put ON DELETE CASCADE for the constraint `fk_Tickets_Concerts1`, ON UPDATE CASCADE for `fk_Tickets_Fans1`, and ON DELETE CASCADE for `fk_Fans_Concerts1`. We put in ON DELETE CASCADE for all the foreign keys in the intersection tables as well.
Concerts_concertID: We decided to remove “Concerts_concertID” and all other similar names from the intersection tables and other entities to avoid any further confusion.

Actions Not Taken:

Genres and Promotions: We did not follow the other suggestions of putting in genres and promotions since it would over complicate our database.
Employee role: We also want to make it clear that all Employees can only have one role each, not multiple.
Artistscol and Artistscol1: We did delete “Artistscol” and “Artistscol 1” since they did not make much sense anyway.
concertID: Additionally, we want to make it clear that the relationship between Concerts and Fans is implemented with “concertID” while Artists and Concerts has an implementation table.

Actions Taken:

Overview: We immediately changed the typo of “Clearwater Arena” to “Clearwater Casino” in the overview to avoid any further confusion. We then took out “promotions” and “genres” in the overview since we did not cover that whatsoever since it would make this project way too complicated.
Employees and Artists: We also took out any relationship between the Employees and Artists because it was not necessary and would be far too complicated. Plus, it is not how real life works and whichever employees would be helping out the artists are the ones who are on staff that night. There cannot be more than one show per night either.
numberOfTickets: We also put in the entity “numberOfTickets” under Concerts so Employees can keep track of how many people show up and that the venue never goes over max capacity.
artistID: We then made the artistID entity the same as the other entities that are unique, not NULL, auto_increment, etc.
Fans: We decided to rename the entity “Customers” to “Fans” because that type of demographic makes more sense for a concert arena anyway and it’s less confusing.
MySQL Cascade: We had to input CASCADE operations between two tables that were dependent on each other. We put ON DELETE CASCADE for the constraint `fk_Tickets_Concerts1`, ON UPDATE CASCADE for `fk_Tickets_Fans1`, and ON DELETE CASCADE for `fk_Fans_Concerts1`. We put in ON DELETE CASCADE for all the foreign keys in the intersection tables as well.
Concerts_concertID: We decided to remove “Concerts_concertID” and all other similar names from the intersection tables and other entities to avoid any further confusion.

Actions Not Taken:

Genres and Promotions: We did not follow the other suggestions of putting in genres and promotions since it would over complicate our database.
Employee role: We also want to make it clear that all Employees can only have one role each, not multiple.
Artistscol and Artistscol1: We did delete “Artistscol” and “Artistscol 1” since they did not make much sense anyway.
concertID: Additionally, we want to make it clear that the relationship between Concerts and Fans is implemented with “concertID” while Artists and Concerts has an implementation table.

Upgrades to the Draft version

Project Outline

Clearwater Arena is a stadium near multiple small towns in the state of Wisconsin that provide world-class entertainment through their venues. They provide 30 concerts per month from different artists around the country. It requires a database to store their ticket sales, artists, fan information, and price per ticket. The venue has 5,000 seats and the Employees have to make sure that it never goes over maximum capacity. There are currently 25 employees at Clearwater Arena who provide different services to fans such as bartenders, customer service reps, cashiers, and security. A database driven website will record all of the sales of Tickets for Fans, Artists, and Employees.

Database Outline

Object Entities:

- Fans
    - fanID: int, auto_increment, unique, not NULL, PK
    - email: varchar
    - firstName: varchar
    - lastName: varchar
    - phone: varchar
    - address: varchar
    - city: varchar
    - state: varchar
    - zipCode: varchar
    - Relationships:
        - A 1:M relationship between Fans and Tickets is implemented with fan_ID as a foreign key inside of Tickets. One fan may buy a ticket and a ticket can be purchased by multiple fans.

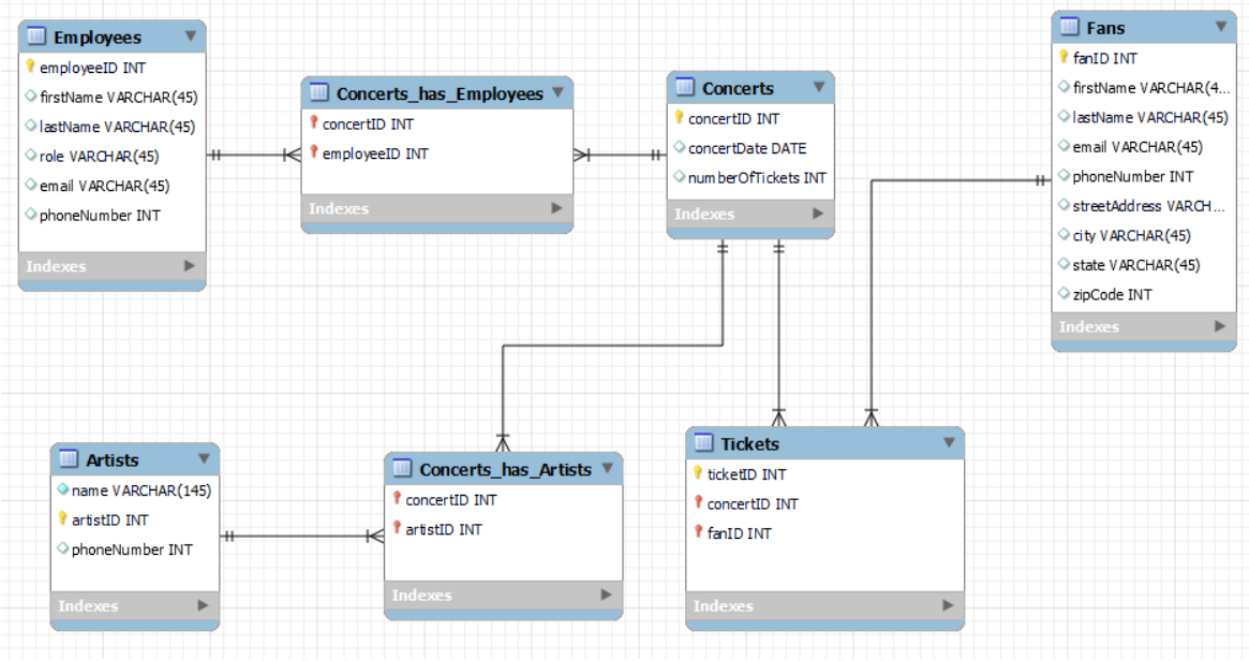
- Employees
    - employeeID: int, auto_increment, unique, not NULL, PK
    - role: varchar
    - firstName: varchar
    - lastName: varchar
    - email: varchar
    - phoneNumber: int
    - Relationships:
        - A M:M relationship between Employees and Concerts is implemented with an intersection table, since one Employee may be responsible for

multiple Concerts, and one Concert may be served by multiple Employees.

- Concerts
    - concertID: int, auto_increment, unique, not NULL, PK
    - concertDate: date
    - numberOfTickets: int
    - Relationships:
        - A 1:M relationship exists between Concerts and Fans. It's implemented with concertID as a foreign key inside of Fans. One Fan may only attend one Concert and a Concert may have many Fans.
        - A M:M relationship exists between Concerts and Artists. A concert can have many artists and an artist can have multiple concerts.
- Artists
    - artistID: int, auto_increment, unique, not NULL, PK
    - name: varchar, not NULL
    - phoneNumber: int
    - Relationship:
        - A M:M relationship between Artists and Fans is implemented with an intersection table, since one Artist may perform at multiple concerts and a Concert may have multiple Artists performing.

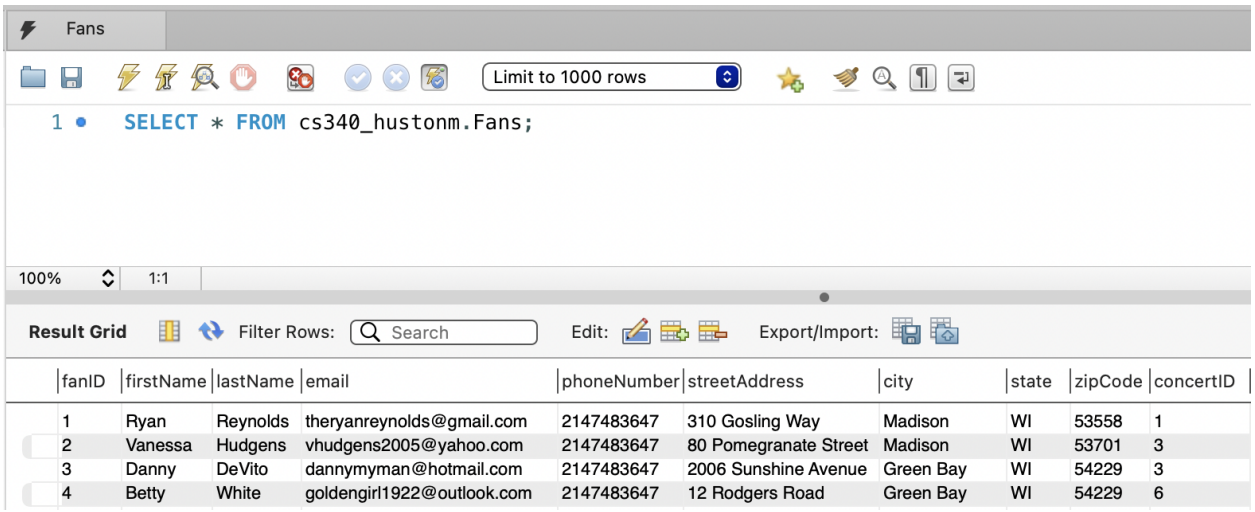
- Tickets
    - ticketID: int, auto_increment, unique, not NULL, PK
    - Relationships:
        - A 1:M relationship exists between Tickets and Fans. A ticket may only be bought by one fan, but a fan can buy multiple tickets.
        - A 1:M relationship exists between Tickets and Concerts. A ticket may only be for one concert, while a concert may have many tickets.

Entity Relationship Diagram / Schema



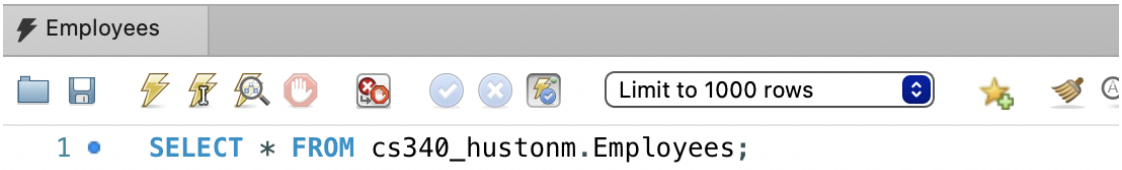
Sample Data

Fans:

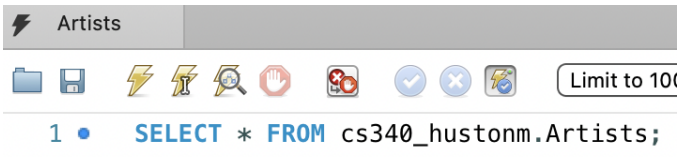


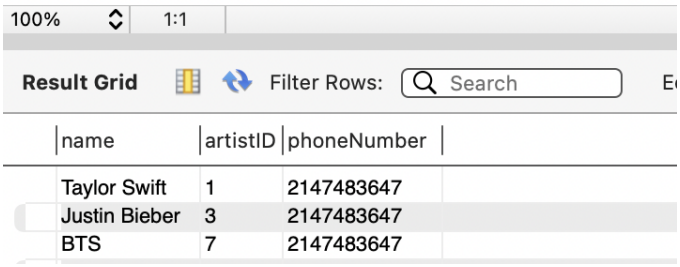
fanID	firstName	lastName	email	phoneNumber	streetAddress	city	state	zipCode	concertID
1	Ryan	Reynolds	theryanreynolds@gmail.com	2147483647	310 Gosling Way	Madison	WI	53558	1
2	Vanessa	Hudgens	vhudgens2005@yahoo.com	2147483647	80 Pomegranate Street	Madison	WI	53701	3
3	Danny	DeVito	dannymyman@hotmail.com	2147483647	2006 Sunshine Avenue	Green Bay	WI	54229	3
4	Betty	White	goldengirl1922@outlook.com	2147483647	12 Rodgers Road	Green Bay	WI	54229	6

Employees:

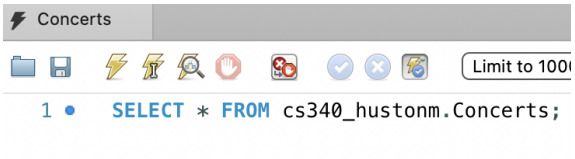


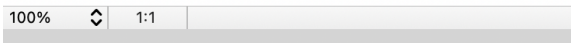
Artists:



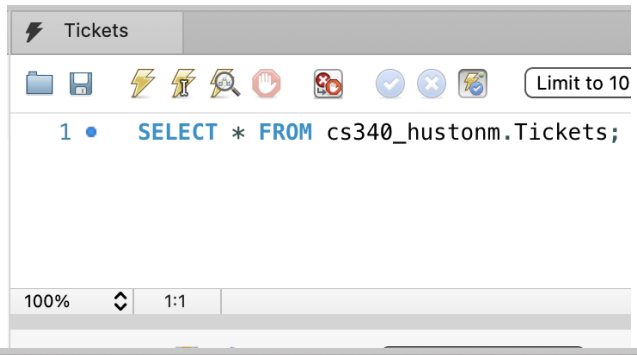


Concerts:

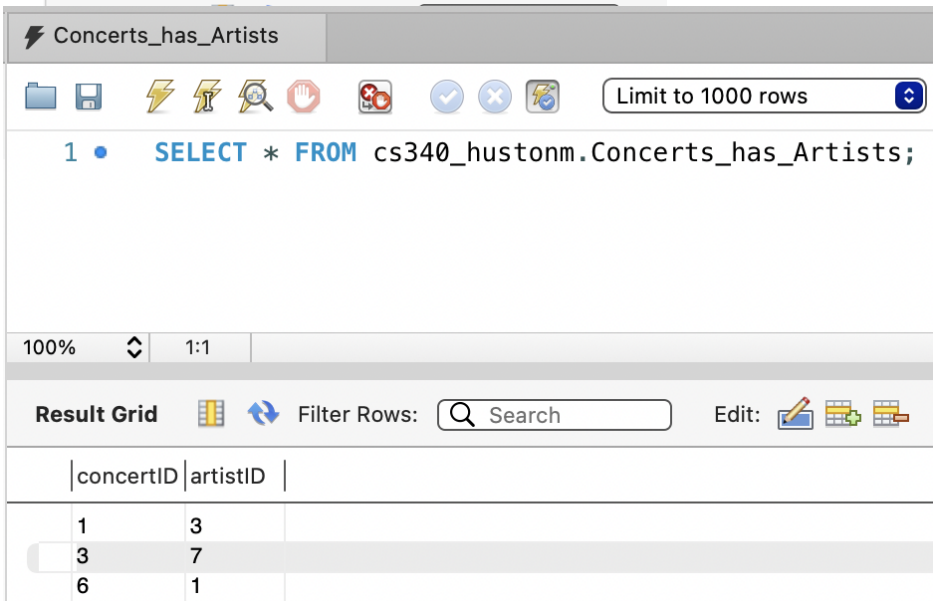




Tickets:



Concerts_has_Artists (Intersection Table):



Concerts_has_Employees (Intersection Table):