Team 58 Project Proposal and Outline

Team 58: Alexander Wilson, Ethan Ruark
Project Name: Databar Events Planner
Project URL: http://flip3.engr.oregonstate.edu:2414/

HTML Feedback

Feedback from the peer reviews: Below is the feedback we received from our peer reviews.

Ethan Pierce provided the feedback:

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, each element is a part of the UI in some manner.

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

Yes each section has a search bar or some form of input for filtering.

- 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.

Yes there are inserts for each attribute in the form of input text boxes.

- 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, then we expect a way to INSERTing a new row to Order (e.g. orderID, customerID, date, total), and also to INSERT row(s) in the intersection table, e.g. OrderDetails (orderID, productID, qty, price and line_total).

Yes there are dropdown options.

- 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.

Yes each element has a delete button on the side.

- 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?

Beside the delete button there is an update option for each row.

- 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.

It's possible in some of the relationships, but I believe it needs to be implemented still.

- Do you have any other suggestions for the team to help with their HTML UI?

Overall I was really impressed with the UI. It looks great and as far as functionality goes it feels great.

Matthew Attebery provided the feedback:

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, each element is displayed throughout the site's UI.

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

Yes, each page's table has a sort function.

- 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.

Yes, each page's table has an insert feature.

- 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, then we expect a way to INSERTing a new row to Order (e.g. orderID, customerID, date, total), and also to INSERT row(s) in the intersection table, e.g. OrderDetails (orderID, productID, qty, price and line_total).

Yes, FKs are handled via dropdown selections. This would mean that they are linked in the backend of the webpage.

- 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.

Yes, each table displayed has a delete button for each row.

- 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, each table displayed has an update button for each row.

- 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.

The site does not show this directly, but an event does not need a full list of employees for the event.

- Do you have any other suggestions for the team to help with their HTML UI?

None that I can think f besides polishing the front page up to have neater images.

Andrew Eitel provided the feedback:

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, each element has a displayed UI system

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

Yes, each element has a sort function

- 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.

Yes, each element has an insert feature

- 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, then we expect a way to INSERTing a new row to Order (e.g. orderID, customerID, date, total), and also to INSERT row(s) in the intersection table, e.g. OrderDetails (orderID, productID, qty, price and line_total).

Yes, FK is implemented via dropdown options, assuming the backend will handle the rest.

- 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.

Yes, each row has an option to delete

- 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, each row has the option to update

- 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 suppose distributor for a designated product ID could be NULL if the bar is transitioning suppliers

- Do you have any other suggestions for the team to help with their HTML UI?

the ui is looking well put together and all features are implemented. building out the backend as the ui is shown will be great

Riley Jones provided the feedback:

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.

There is a Select for every table within the schema excluding the jobs table to show what jobs you have listed to choose from.

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

There is functionality to filter the employees page.

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.

The UI allows the ability to add entities to each table in the schema with their corresponding attributes. The only thing I didn't see was where new jobs could be created.

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, then we

expect a way to INSERTing a new row to Order (e.g. orderID, customerID, date, total), and also to INSERT row(s) in the intersection table, e.g. OrderDetails (orderID, productID, qty, price and line_total).

This is found within the relationship between Events and Employees as well as between Drinks and the inventory where the corresponding FK's can be inserted.

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 are several delete statements. I believe that you would be able to delete the M:M relationship between drinks and inventory by editing a drink and removing the inventory contents, but you might want to make this more clear. It appears that the events and employees relationship could operate in a similar fashion.

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?

You can edit the contents of multiple separate entities.

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 believe that an event could exist without employees assigned to it, or a drink without inventory product. If both of these are false then I believe one will need to be implemented to meet the requirements.

Do you have any other suggestions for the team to help with their HTML UI?

I really enjoyed reading through your proposal and looking through your UI, it is effective and intuitive. My only recommendation would be to make the M:M relationships more accessible in order to meet the project requirements of being able to delete these relationships and as I said ensuring that one of your M:M has a nullable side. Great work and good luck to both of you!

Actions

Actions taken based on peer review: Overall the feedback we received from our reviewers was pretty positive. The only suggestion we received was making sure our M:M relationship was more accessible and that one has a nullable side. We did not change anything on the UI portion of our project, but will definitely consider this advice when building our database. We also spent some time modifying the CSS portion of our project to make it look nicer and enhance the navigability.

Project Proposal Feedback

Feedback from the peer reviews: We were fortunate enough to receive four peer reviews, many of which hit on the same points. Below we have taken the pertinent points from their feedback.

Edward Lada provided the following notes:

- "**Entities**: There are four entities, but only **three** relationships described (Employees: Events: Drinks: [...] Inventory. Specifications ask for four relationships - this could be solved by changing job_title to an entity rather than a baked-in attribute of the Employee."
- "**Relationship formulation.** For a M:M relationship between drinks and inventory, I'd recommend representing that relationship with a separate intersection table that links a drink_name to as many product_IDs as needed, rather than storing four separate product_IDs as FK's in the Drink entity."
- "**Naming consistency:** The only note I have here is that sometimes the terms Employee and Worker are used interchangeably - I would pick one and stick with it."

Chaim Wald provided the following notes:

- **Entities:** "They do all represent a single idea. However, for the *Events* entity, maybe you can place all the staff, such as *bartender, barback etc…* into another table, have an attribute in *Events* called *workers* for example, and that would link to the table with those job positions."
- "Additionally, under *Inventory* you have an attribute called *distributor*. Maybe you can add a table to have many distributors and track more of your inventory cost."
- **Naming Consistency: "**In *Employees* relationship you say "worker" entity, I believe you meant 'Employees'. In many of the relationship descriptions, mentions of references to other entities should be capitalized, as per your outline. And in *Events* the attribute 'entity_iD' I believe you intended 'entity_ID'."

Carlyn Coccaro provided the following notes:

- "I did notice that your Employee table mentions a drop-down menu of employee types, in my opinion, that needs to be in its own entity (i.e. Employee_Type). I would do the same with your Inventory table with your drop-down of category choices (i.e. Inventory_Type)."

Kyle Brogdon provided the following notes:

- **Overview: "**While it does list a few specific facts such as event size, it could list a few other important pieces of information like amount of staff per shift, amount of staff per night, etc which would help cage the reader to [...] the scope of the problem more."
- **Entities: "**Yes there are four entities, but I think you may be missing a relationship in addition to having a little convoluted setup of employees and events. My recommendation would be to consider taking out job_title and multiple attributes from events (bartender through manager) and maybe putting those in an entity called Staff that has its own relationships."
- **Naming Consistency: "**For the most part, there appears to be a few times where switching back and forth between employee and worker can get a little confusing"

Note: In addition to the feedback quoted above, Kyle also noted that when we had originally posted our Overview and Outline, we had accidentally left out our ERD and Schema. We have chosen to omit this point, as we noticed our mistake and added those after his review had already been submitted.

Actions taken based on peer review: Based on the feedback that we received from our peers, we have realized that we were indeed one relationship short. We chose to take care of two birds with one stone, and also opted to split off Employee's "job_title" into its own entity, Jobs. This will not only allow Databar to standardize the rate of pay for each employee working a given job (thus eliminating the need to include "hourly_rate" as an attribute of the Employee entity), but will also create the fourth relationship that was previously missing from our design. The Primary Key of this new entity (job_code), then became a Foreign Key in the Employees relation.

In the course of making these changes, we have also decided to remove the specific job titles from the Events' attributes, and instead use Employee's Primary Key (employee_ID) as a Foreign Key in the newly named "employee_x" attributes to allow Databar to have flexible staffing for events that may require two bartenders, two security guards as well as combinations of all job types as needed. Lastly, we have gone through and standardized our use of "Employees" versus "workers" throughout our outline, as well as a few other typos.

Actions not taken: While a couple of our peers suggested creating a intersection table for the Drinks and Inventory entities, we decided to not pursue this route, as it seemed a bit redundant, as this would essentially just be duplicating the data already in the Drinks entity, with the exception of the "price" attribute.

The other suggestion that we chose not to follow was that of creating a separate "Distributors" entity. Our reasoning for this choice came primarily from our real-world experience working in bars, where it isn't very common to work with more than two-to-three distributors due to liquor laws and the typical distribution of market share.

Other Updates: Originally “drink_name” was the Primary Key for the Drinks entity. We decided that this could lead to a slippery slope as that could open the door to spelling errors, case-sensitivity errors, and the fact that drinks often have very simila r names based on their ingredients (“Vodka-soda” vs. “Vodka-cran”). To address this we added a new attribute “menu_item” to be used as the Primary Key instead.

Another small correction that we made is that none of our reviewers seemed to notice that though we stated that each drink could have up to five ingredients in our Overview, in our Outline we only had four ingredient attributes. So we added the fifth for consistency’s sake.

Lastly, we during the process of creating out HTML draft, it occurred to us that an operator of our database would likely find the need to search drinks by the primary liquor (whiskey-drinks versus vodka drinks) so we updated the inventory categories from liquor to specific types of liquor in order to facilitate such a search.

Overview: Databar is a bar with a large event space that can be rented out by its customers. In 2021, Databar hosted over 200 events. As their venue is becoming increasingly popular they are realizing the need to establish a database to help them organize as they grow. This database will be used to plan upcoming events, including the staffing and inventory needs associated with each event. Databar has the capacity to hold only one event per night. Each event needs to be appropriately staffed, including several employees (up to five) of assorted job titles. While events will have access to the full bar, they will also include a featured drink special, which will be selected from the Drinks list. The drink special is available at a discounted price for the duration of the event. Drinks can have a maximum of five ingredients, but must have at least one (single liquor specials are allowed). Using this database, Databar will have confidence that it will never overbook its event space, events are fully staffed, and the appropriate ingredients for each event’s drink special are readily available.

Outline:

- **Employees**: The purpose of this entity is to track the employees at Databar. It can (and will) be used by management to staff upcoming events.
    - employee_ID: int, unique, auto_increment, not NULL, PK
    - first_name: varchar, not NULL
    - last_name: varchar, not NULL
    - telephone: varchar, not NULL
    - job_code: int, not NULL, FK (job_code)
    - start_date: date, not NULL
    - Relationship: Employee entity will have a many to one relationship with events entity. Each event will require employees with different job titles to operate properly. The employee_ID will be the FK inside of the event entity. Employees will also have a one to one relationship with jobs. The job_title will be the FK inside of jobs.
- **Jobs**
    - job_code: int, unique, auto_increment, not NULL, PK

    - job_title: char, unique, not NULL
    - hourly_rate: decimal(18, 2), Not NULL
    - Relationship: The Jobs entity will have a many to one relationship with employees. Each employee will have one job title and there will be multiple employees with the same job title.

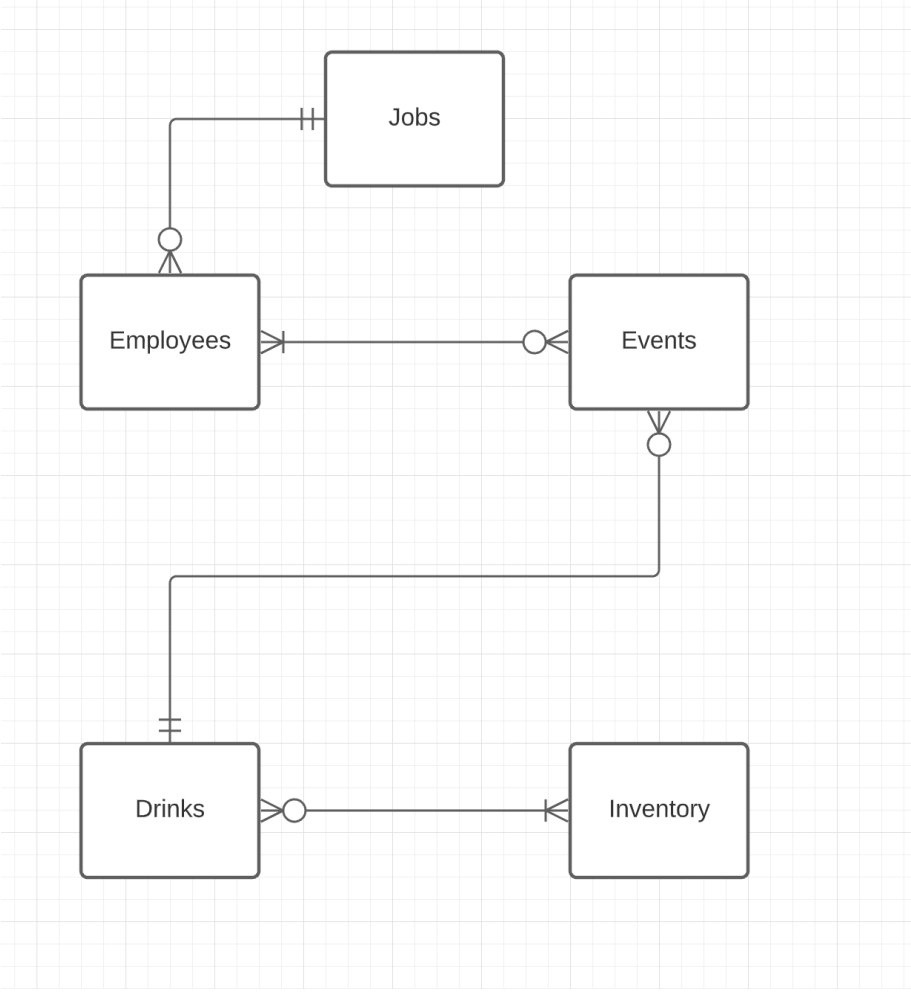
- **Events**: The purpose of this entity is to track upcoming events at Databar. It will have attributes such as employee IDs for the staff that will be running the event and the drink special for the night.
    - event_ID: int, unique, auto_increment, not NULL, PK
    - event_name: varchar, not NULL
    - event_date: date, not NULL
    - employee_1: int, FK (employee_ID), can be NULL, exactly one
    - employee_2: int, FK (employee_ID), can be NULL, exactly one
    - employee_3: int: FK (employee_ID), can be NULL, exactly one
    - employee_4: int, FK (employee_ID), can be NULL, exactly one
    - employee_5: int, FK (employee_ID), can be NULL, exactly one
    - guest_count: int, not NULL, exactly one
    - menu_item: int, not NULL, FK (menu_item)
    - Relationship: The events entity will have a one to many relationship with the employees entity. The events entity will also have a one to one relationship with the drink entity.

- **Drinks** The purpose of this entity is to track event drink special options and their recipes.
    - menu_item : int, unique, auto_increment, not NULL, PK
    - drink_name : varchar, unique, not NULL
    - ingredient_1: int, not NULL, FK (product_ID)
    - ingredient_2: int, can be NULL, FK (product_ID)
    - ingredient_3: int, can be NULL, FK (product_ID)
    - ingredient_4: int, can be NULL, FK (product_ID)
    - ingredient_5: int, can be NULL, FK (product_ID)
    - price: decimal(18,2), not NULL
    - Relationship: a 1:M relationship between Drinks and Events is implemented with drink_name as a FK inside of Events. A M:M relationship also exists between Drinks and Inventory, as implemented by the Inventory product_key utilized as a FK within Drinks.

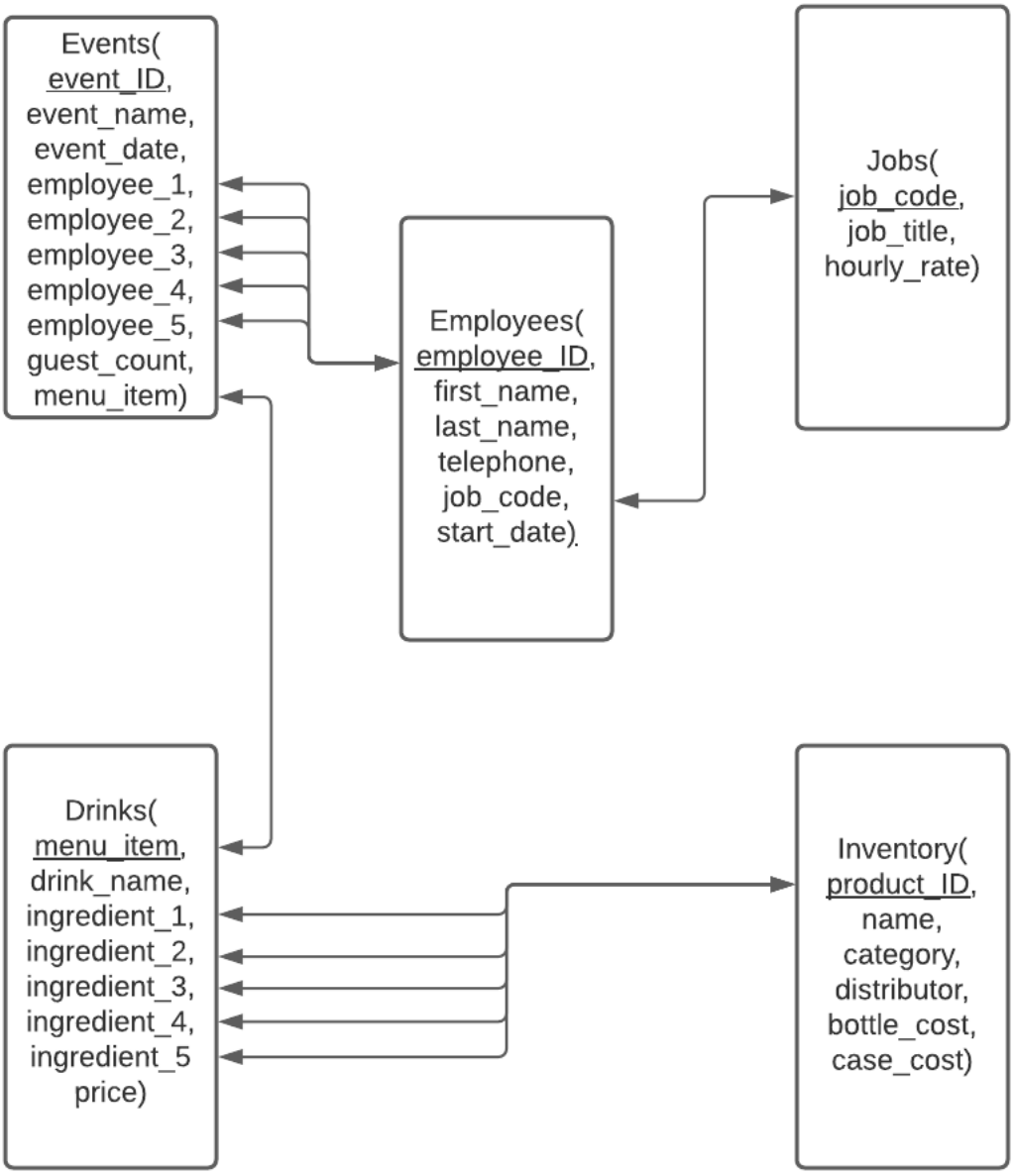
- **Inventory**: The purpose of this entity is to track the ingredients inventory needed for event drink specials.
    - product_ID: int, unique, auto-increment, not NULL, PK
    - name: varchar, not NULL
    - category: char, not NULL (the following category choices will be implemented via dropdown menu when inputting new items)
        - Whiskey/Bourbon

        - Tequila/Mezcal
        - Vodka
        - Gin
        - Rum
        - Beer
        - Wine
        - Mixer
        - Misc.
    - bottle_cost: decimal(18, 2), can be NULL
    - case_cost: decimal(18, 2), can be NULL
    - distributor: varchar, not NULL
    - Relationship: a M:M relationship between Inventory is implemented with product_id as a FK inside of Drinks.

Entity Relationship Diagram



Schema


Events(
event_ID,
event_name,
event_date,
employee_1,
employee_2,
employee_3,
employee_4,
employee_5,
guest_count,
menu_item)
Employees(
employee_ID,
first_name,
last_name,
telephone,
job_code,
start_date)
Jobs(
job_code,
job_title,
hourly_rate)
Drinks(
menu_item,
drink_name,
ingredient_1,
ingredient_2,
ingredient_3,
ingredient_4,
ingredient_5
price)
Inventory(
product_ID,
name,
category,
distributor,
bottle_cost,
case_cost)