mechanics need databases too

Prepared by Group 14: Heather Fillerup- Software Developer, Chris Nelson- Software Developer
CS 340-400: Spr 2020
Project- Step 4 Draft
May 15, 2020
http://flip3.engr.oregonstate.edu:5455/home

Feedback by TA

- Step 3 Final Feedback
    - Great job! Excellent quality so far.
    - Additional comments:
        - To make a 1:M relationship NULLABLE, you should be able to set a foreign_key value (in a 1:M relationship) to NULL at any time. Currently, this requirement is not met.
        - (Hint: this requirement is to force you to think about how you would make the proper LEFT JOINS on entities that have optional participation)
        - I noticed that you have DELETE buttons for all of your tables. You should take a second and reconsider this. Since some of your foreign_keys are REQUIRED (NOT NULL), what happens when you allow someone using your system to delete an entity that a foreign_key references?
            - For example, what would happen if you allowed someone to delete a "work_task"? Since it is required to be "not null" in work_orders, does it delete the whole work order? (Your answer to this question might be "yes", and that is fine. I just wanted you to think more about the implications of allowing certain DELETEs to happen based on your database structure, and future databases you might make.)

Feedback by the peer reviewer

- 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.
    - Honghao Li : Yes, I could find all of table inside of UI. But it is a little mess up of structure. I think it will be a good project if you finish all of the features. For now, you already contain all of the tables you have.
    - Paul Newling: Yes, although some of the tables are a bit buried
    - David Eaton: Yes
    - Joshua Sienkiewicz: Yes

- Does at least one SELECT utilize a search/filter with a dynamically populated list of properties?
    - Honghao Li: Yes, They have three search features in the part of cars, customers, and repair.
    - Paul Newling: Yes, Customers is searchable
    - David Eaton: Yes
    - Joshua Sienkiewicz: Yes
- 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.
    - Honghao Li: Yes, all three parts have the part of Add/Insert but placing in the individual model.
    - Paul Newling: Yes, I believe so
    - David Eaton: Not clear but can be assumed.
    - Joshua Sienkiewicz: Yes, the current page allows the user to add new Cars, Customers, and Repair orders, which all imply that an INSERT statement would be needed for each 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).
    - Honghao Li: I guess yes. Because there is only one table graph. So whenever something changed, the row of the table will be changed as well. And your M: M relationship is repair_order and work_tasks, I am a little confused probably for you haven't finished yet.
    - Paul Newling: This isn't made very clear, but it could be assumed so? Hopefully when the full site is up and running it will be a bit more clear that the insert will effect the M:M as well
    - David Eaton: Yes
    - Joshua Sienkiewicz: Yes
- 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.
    - Honghao Li: Yes, each row has a delete button so that it's really clear to modify the table.
    - Paul Newling: Yes
    - David Eaton: Yes
    - Joshua Sienkiewicz: Yes - delete implemented for the "Work Order" item
- 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?

    - Honghao Li: Yes, each row also has an update button for updating the data. but I think it belongs to the part of the edit.
    - Paul Newling: Each section appears to have an update associate with it
    - David Eaton: Yes
    - Joshua Sienkiewicz: Yes, also for work orders.
- 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.
    - Honghao Li: I don't think so. Because there is only one table to show the data. But probably they could make some features be NULL like "Current task" if the car has not been arranged to someone.
    - Paul Newling: If there is, the distinction isn't made clear that it is possible
    - David Eaton: Yes the instructions on the homepage states a car can be associated with zero or more repairs.
    - Joshua Sienkiewicz: Yes. - cars have an optional relationship with work orders
- Do you have any other suggestions for the team to help with their HTML UI?
    - Honghao Li: Yes, besides I said above about its structure, you could make clear building customers. I only see your "customer id" inside of the Update part.
    - And also, you should make your table clear or create more so that we could understand the relationship that you have
    - Paul Newling: Overall a good clean site, it might be beneficial to point to where the M:M and NULL-able sections of the project are with a note or something on the page. Overall though I think the work looks good
    - David Eaton: In my opinion the adaptable navigation menu is not as intuitive to use as a static navigation menu. I believe this is why Paul Newling stated the some of the tables were "buried" and Honghao Li stated the structure was a "mess". I believe making the simple switch from a dynamic menu to a static menu will resolve these types of conceptions about your site overall. Otherwise your site looked good.
    - Joshua Sienkiewicz: The navigation bar is not very intuitive. I believe this is due in part to the styling - using ">" in between nav items leads the user to think they are in a "drill down" menu. I think using a "|" (pipe) character to separate nav items would be more intuitive. Also, I think that the nav should be static, and maybe each of the car, customer, and repair order items can utilize a sub-nav, making it easier to get to each page from each other page rather than having to traverse multiple pages to get to other pages.
    - Does the overview describe what problem is to be solved by a website with DB back end?

        - Vinh Tran: Yes, the overview describes a problem with pen and paper that needs to be solved by a website with DB back end.
        - Kelley Neubauer: Yes, the overview describes an auto shop problem that can be solved by a DB back end. Great story and background!
        - Benjamin Mayinger: Yes, the idea is very practical and shows a real world problem that could be organized using a database backend. Furthermore, the overview is detailed and well written.
        - Sibai Lou: Yes
    - Does the overview list specific facts?
        - Vinh Tran: Yes. Some of them are creating a repair order, associating a car and a customer to the repair order.
        - Kelley Neubauer: The overview has some specifics, but it could be more detailed. How many mechanics will be accessing the system? How long does a car spend under repair? Is there only one location?
        - Benjamin Mayinger: The overview explains the relationships between the entities and the attributes of given entities in a concise way.
        - Sibai Lou: Yes
    - Are at least four entities described and does each one represent a single idea to be stored a s a list?
        - Vinh Tran Yes, 4 of them are customers, cars, repairs, and statuses. Each of them represent a single idea.
        - Kelley Neubauer: Yes, there are at least four entities described. customers, cars, repairs, statuses, and mechanics all represent unique ideas that could be stored as a list.
        - Benjamin Mayinger: The draft outlines five entities: customers, cars, repairs, statuses, and mechanics.
        - Sibai Lou: Yes
    - Does the outline of entity details describe the purpose of each, list attribute datatypes and constraints and describe relationships between entities? Does the outline clearly indicate which entities (tables) will be implemented and which team member is primarily assigned to the associated page(s)?
        - Vinh Tran: Yes, the outline of entity details describes the purpose of each, list attribute datatypes and constraints and describe relationships between entities. The outline clearly indicates which entities (tables) will be implemented and which team member is primarily assigned to the associated page(s)
        - Kelley Neubauer: Yes, the outline describes the purpose of each entity and lists datatypes and constraints. The relationships between entities are described.

        - Benjamin Mayinger: Each entity is well explained and so are the relationships between them. The ERD and Schema diagrams do a good job of showing this. The overview also shows which team members will be implementing which entities.
        - Sibai Lou: Yes
    - Are 1:M relationships correctly formulated? Is there at least one M:M relationship?
        - Vinh Tran: Yes, 1:M relationships are correctly formulated and there is at least one M:M relationship.
        - Kelley Neubauer: Yes, the 1:M relationships are correctly formulated. Yes, there is at least one M:M relationship.
        - Benjamin Mayinger: All the relationships seem to be in order.
        - Sibai Lou: Yes
    - Is there consistency in a) naming between overview and entity/attributes b) entities plural, attributes singular c) use of capitalization for naming?
        - Vinh Tran: Yes, there is consistency in areas as listed above.
        - Kelley Neubauer: Yes, there is consistency in naming conventions. I'd recommend that you make use of the convention to capitalize entity names.
        - Benjamin Mayinger: The document is actually really well formatted and holds its consistency throughout it.
        - Sibai Lou: Yes
    - Additional notes:
        - Kelley Neubauer: I am a little confused about why the mechanic works on statuses and not repairs. How does a mechanic work on "customer paid" or "customer picked up?" It would seem more natural to me for the relationship to be between mechanics and repairs. Maybe the statuses entity should have a different name? As is, it's a little confusing and could be clarified.
        - Kelley Neubauer: One little thing, check that the arrow for the repiairs_statuses FK, status_id, points to the correct attribute.
        - Benjamin Mayinger: Everything looks good to me! Well done!
        - Sibai Lou: It is a good project, and I think in statuses, the start and end date can be changed to how many days needed to repair the car. When I go to a auto shop, I perfer they show me how many days they took to repair my car.

Actions based on the feedback from previous steps

- We thought we had made the 1:M relationship between customers and cars NULLABLE, however we made the changes to our database outline but forgot to make the changes in the ERD. So, the change has been made to the ERD.
- We will carefully consider the implementation of having a DELETE function for all tables. We plan to have node express logic to catch any MySQL errors and notify the user if they cannot delete a record. We are also deciding whether to implement ON DELETE CASCADE
- Redesigned navigation bar to include entities, and hovering over each entity provides a dropdown list to navigate to the Add and Update pages
- Each entity now has its own table on their main page that displays current records and includes buttons to add, update, delete and search.
- Each form for the entities now includes any foreign keys.
- Added more details to the project overview and better defined what the website will do and what the tracking display will look like.
- We are keeping our tables/entities lowercased, this makes it easier to code and view, especially in keeping with the format that the SQL keywords will be uppercase, having lowercase tables and columns makes it easier to read. Also we won't have to remember what is uppercase and what isn't.
- Kelley was confused by what we were trying to accomplish. A car comes into the shop for a repair, however a repair goes through various stages\tasks which are assigned to different mechanics and can take varying amounts of time to complete. A repair can have multiple mechanics working on it, e.g. one assigned to diagnose, a different one assigned to get customer approval, etc. This helped us realize that we needed to update our repairs table to repair_orders, which is the main tracking mechanism for the overall reason the car is in the shop, then we can add work orders (tracked in the composite table work_orders) that records the various tasks being done to the cars, the mechanic assigned to each task and start and end dates of the task.
- Fixed FK arrow pointing to wrong PK in schema
- Sibai mentioned tracking days needed to repair the car, we could calculate this based on the time between the repair_orders' start date and end date attributes. However, this would not provide meaningful data because we are not tracking the exact specific type of repair, we are only tracking the mechanic general tasks workflow. E.g. brake repair job is two hours, transmission replacement could be two weeks, 169 hours would be the average.

Upgrades to the Draft version

- Removed Parts table since we only need to implement one M:M relationship
- Removed cost, hours and rate attributes to focus on the tasks and mechanics for the repair order
- Changed the name of the status table to work_tasks for better clarification
- Changed the name of the repairs table to repair_orders for better clarification

- Changed repairs_statuses relationship table to work_orders and added it as a composite table for better clarification
- Made work_orders a composite table with attributes moved from work_tasks (mechanic_id, start_date and end_date). This was to satisfy the requirement that when we delete our M:M task and repair orders relationship record, we cannot delete any record from the repair_orders or work_tasks tables. We also gave work_orders a PK of id, instead of using the FKs as the PK.
- Updated the customers participation with cars, a customer can have 0 or more cars, this will allow a customer to be added to the database without requiring a car_id
- Updated the cars participation with repair orders, a car can have 0 or more repairs, this will allow a car to be added to the database without requiring a repair order
- Changed mechanics relationship with work_tasks (statuses). Mechanics has a M:M relationship with both repair_orders and work_tasks, which are both nullable.
- Removed parts_neeeded and current_status attributes from repair_order.
- Added pair programming to programming assignments because we want to work on everything together if possible.
- Changed year attribute to model_year and model to model_name, since year is an SQL keyword
- Removed address fields from customers and description field from cars, it will be concatenated from model_year, make and model_name for simplification

Project Outline

Mahinui auto shop, a single location, has seen record business in the last decade, repairing 50 or more cars on any given day. With more customers coming in by the day, keeping track of records has become a nightmare. The owner, Brad, has finally decided to upgrade his repair order workflow from pen and paper being passed between his 10 mechanics to a website database. Brad is looking to create a system for his mechanics to track the tasks involved with a car's repair, from diagnosis to customer pick up, and be able to view a display of the progress on the homepage. The website will allow users to:

1. Add Customer
2. Add Car
3. Add repair order to a car
4. Add work orders to repair order
    a. Add work task to work order
        i. Diagnosis, customer approval, order parts, repair, test drive and finally contact customer
    b. Add Mechanic to work order
5. Add end date to work order to complete
6. Add new work order
7. View on the website homepage the following display of all the cars currently being repaired at the shop and the current task being performed. **This display is not intended to be part of the grading requirements, but more to show the usefulness of the database.**

Mahinui Auto Shop Dashboard				
Customer Name	**Car Description**	**Current Task**	**Start Date**	**Mechanic**
Jason Bateman	2015 Honda Accord	Diagnosis	03/1/2020	Johnny
Charlize Theron	2014 Toyota Civic	Customer Approval	3/2/2020	Ben
Ryan Reynolds	2011 Honda Ridgeline	Order Parts	3/3/2020	Cameron
Scarlet Johansen	2009 Toyota Front Runner	Repair	3/4/2020	Peter
Jeff Bridges	2014 Fiat 500	Test Drive	3/5/2020	Frank
Tyler Perry	2017 Kia Soul	Contact Customer	3/6/2020	Brian
Brandon Fraser	2019 Chevrolet Corvette	Diagnosis	3/6/2020	Ben

Programming Implementation and Assignments

For this project we will be implementing pair programming when possible. The goal is for the code in this project to be done together and we will switch back and forth between who is actively programming and who is giving feedback and checking for errors. This will allow us to learn from each other and help ensure that the code is of good quality.

Database Outline

customers: records details about the customers who own the cars being repaired (Heather)

- id: INT, AUTO_INCREMENT, UNIQUE, NOT NULL, PK
- f_name: VARCHAR, NOT NULL
- l_name: VARCHAR, NOT NULL
- contact_no: VARCHAR, NOT NULL
- email_address: VARCHAR, NOT NULL
- relationship: a 1:M relationship between customers and cars is implemented with customer_id as a FK inside of cars, where a customer can have 0 to many cars, and a car can only have one customer.

cars: records details about the car being repaired (Chris)

- id: INT, AUTO_INCREMENT, UNIQUE, NOT NULL, PK
- customer_id: INT, FK
- license_plate: VARCHAR, NOT NULL
- make: VARCHAR, NOT NULL
- model_name: VARCHAR, NOT NULL
- model_year: YEAR, NOT NULL
- relationship: a 1:M relationship between cars and repair_orders is implemented with car_id as a FK inside of repair_orders, were a car can have 0 or more repair orders and a repair order can have only one car ; a 1:M relationship between customers and cars is implemented with customer_id as a FK inside of cars, where a car requires zero or one customer and a customer can have 0 or more cars

repair_orders: records details about the repair order being done on a car (Heather and Chris)

- id: INT, AUTO_INCREMENT, UNIQUE, NOT NULL, PK
- car_id: INT, FK NOT NULL
- date_received: DATE
- date_completed: DATE
- relationship: a M:M relationship between repair_orders and work_tasks and a M:M relationship between repair_orders and mechanics are both implemented with a composite table work_orders; a 1:M relationship between cars and repair_orders is implemented with car_id as a FK inside of repair_orders, where a repair order must have only 1 car, but a car can have 0 or more repairs.

work_tasks: records the types of tasks that can be added to repair orders, these tasks are associated to repair orders through work orders (Heather)

- id: INT, AUTO_INCREMENT, UNIQUE, NOT NULL, PK
- name: VARCHAR, NOT NULL (Diagnosis, Customer Approval, Order Parts, Repair, Test Drive, Contact Customer)
- relationship: a M:M relationship between repair_orders and work_tasks and a M:M relationship between mechanics and work_tasks are both implemented with a composite table work_orders

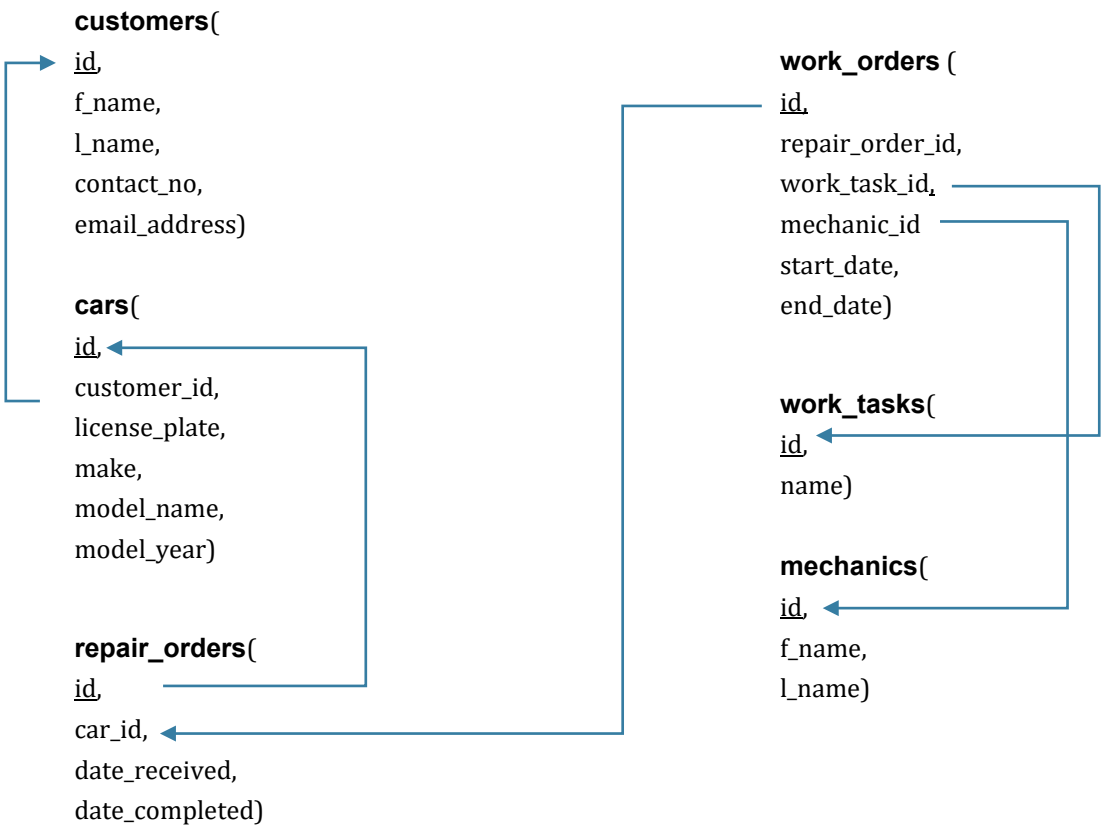
work_orders: composite table that records the tasks that have been added to the repair_orders and also tracks the mechanic responsible for the work order (Heather and Chris)

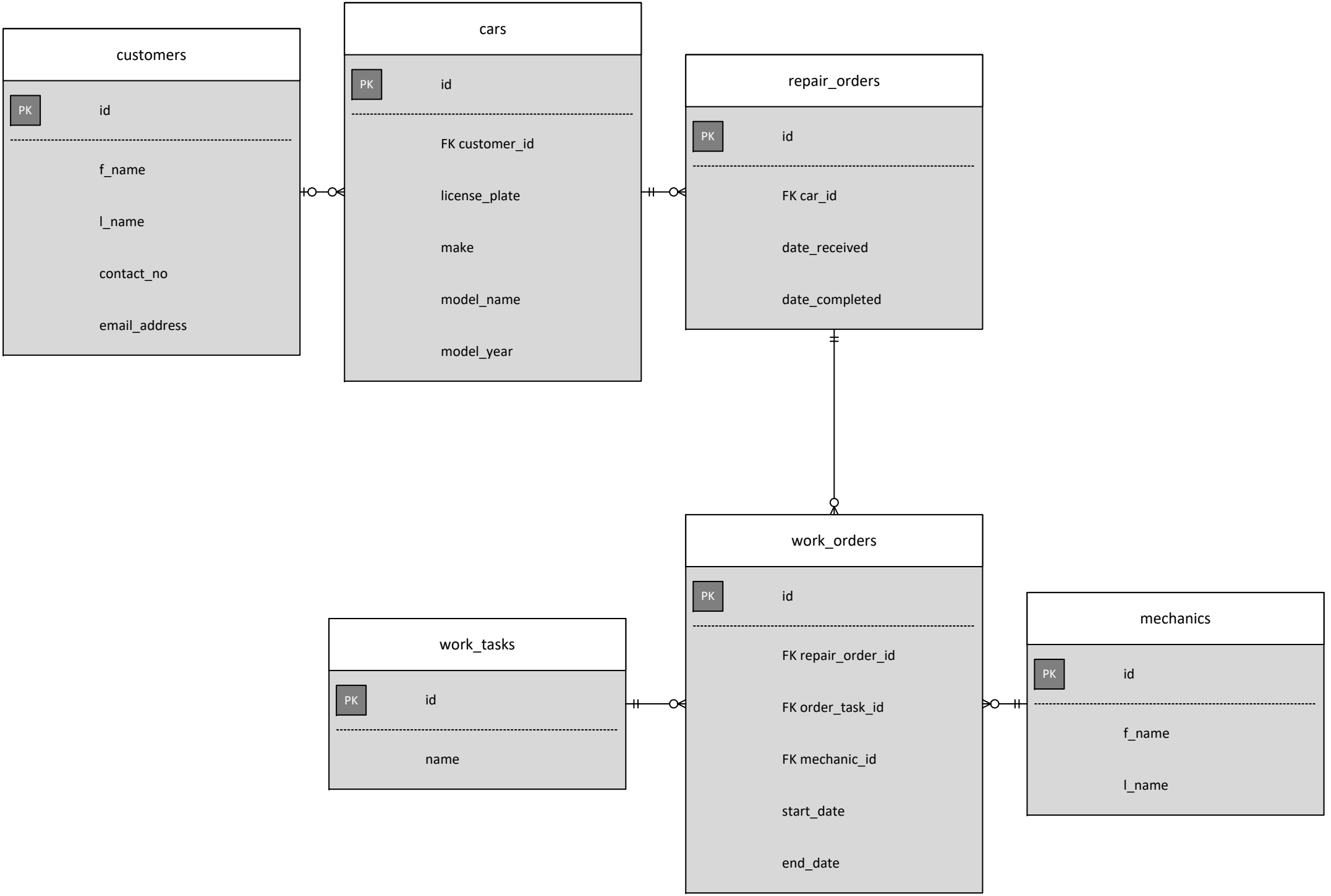
- id: INT, AUTO_INCREMENT, UNIQUE, NOT NULL, PK
- repair_order_id, NOT NULL FK
- order_task_id, NOT NULL FK
- mechanic_id: INT, NOT FK
- start_date: DATE
- end_date: DATE

mechanics: records details of the mechanic responsible for the work orders (Chris)

- id: INT, AUTO_INCREMENT, UNIQUE, NOT NULL, PK
- f_name: VARCHAR, NOT NULL
- l_name: VARCHAR, NOT NULL
- relationship: a 1:M relationship between mechanics and work order is implemented with mechanic_id as a FK inside of work_orders, where a mechanic can have 0 or more work_orders but a work order can only have one mechanic; a M:M relationship between repair_orders and mechanics and a M:M relationship between mechanics and work_tasks are both implemented with a composite table work_orders;

Schema


customers(
id,
f_name,
l_name,
contact_no,
email_address)
cars(
id,
customer_id,
license_plate,
make,
model_name,
model_year)
repair_orders(
id,
car_id,
date_received,
date_completed)
work_orders (
id,
repair_order_id,
work_task_id,
mechanic_id
start_date,
end_date)
work_tasks(
id,
name)
mechanics(
id,
f_name,
l_name)


customers
PK
id
f_name
l_name
contact_no
email_address
cars
PK
id
FK customer_id
license_plate
make
model_name
model_year
repair_orders
PK
id
FK car_id
date_received
date_completed
work_orders
PK
id
FK repair_order_id
FK order_task_id
FK mechanic_id
start_date
end_date
work_tasks
PK
id
name
mechanics
PK
id
f_name
l_name