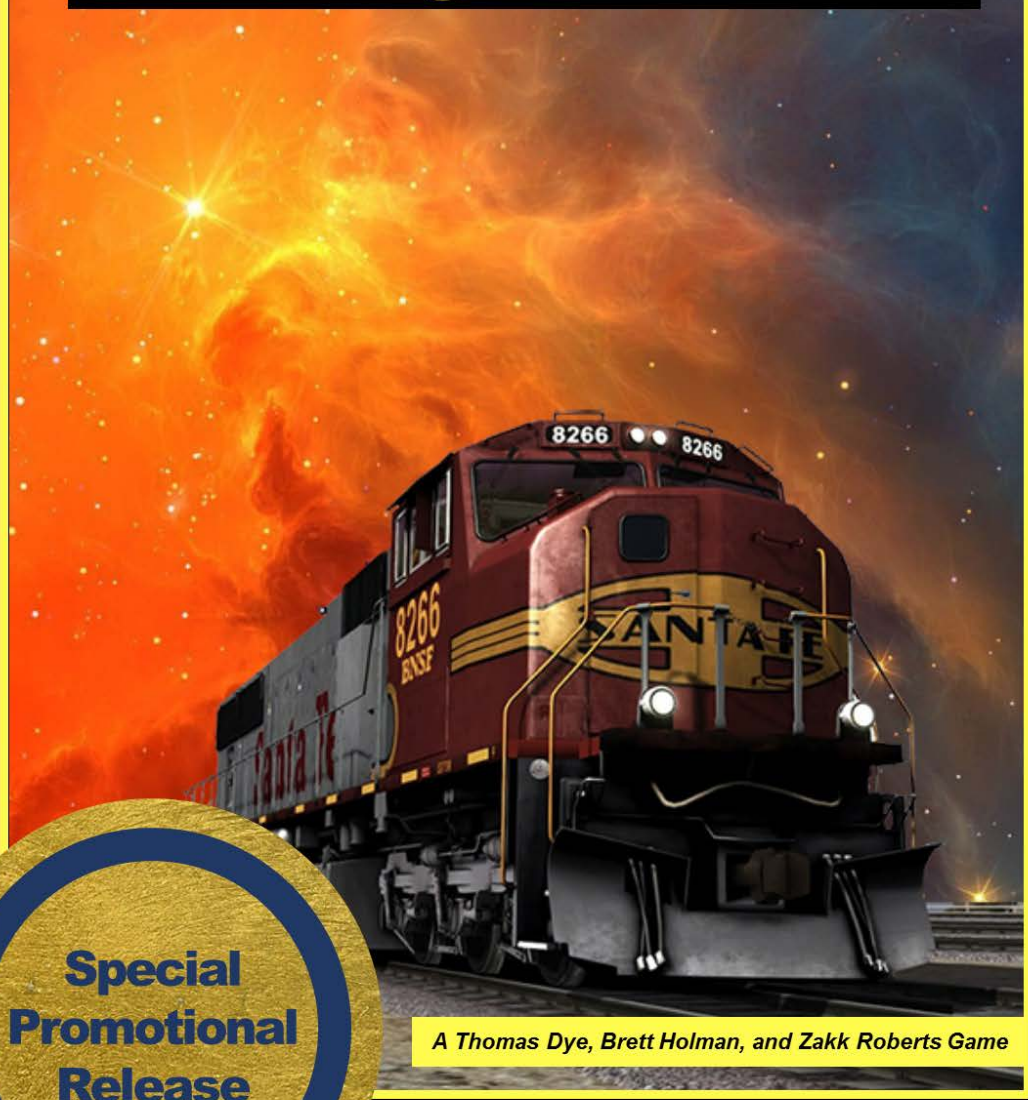
# FREIGHT COMMANDER

## The 2-D Freight Train Simulator




*A Thomas Dye, Brett Holman, and Zakk Roberts Game*

**Special Promotional Release**

Hip Replacements

*English Version contains: - Carte de reference et guide d'installation en Francais*

## INTRODUCTION

*The goal of Freight Commander is to simulate the railroad industry as depicted on model railroading layouts by dispatching and tracking freight rolling stock to and from modeled industries. Crew (players) receive dispatch orders as well as check in at various locations in via a smartphone application. Players are faced with a complex puzzle of how to move specific freight cars from the middle of a train into a railroad siding using only push and pull movements in a finite space, in a finite time. The frontend application is connected to a database tracking all train movements during a game session and issues dispatch orders to crew. The intent of this project is to construct that database.*

## TABLE OF CONTENTS

## ITERATION 3 CHANGE LOG

- Submitted TrainGame version 1.0 Release Candidate.
- Began TrainGame version 1.1.
- Merged test data into TrainGame1_1.sql
- Revised with 3NF pass:
    - Split Modules into Modules and ModulesAvailable
    - Split Industries into Industries, IndustriesAvailable, and IndustryActivities
    - Destroyed Industries trigger.
    - Created IndustryActivitiesTrigger.
    - Split IndustrySidings into IndustrySidings and IndustrySidingsAvailableLength
    - Edited IndustrySidings trigger for scope.
    - Created IndustriesAvailableLengthTrigger.
    - Split Trains into Trains and TrainLocations
    - Split Shipments into Shipments, ShipmentsPickedUp, and ShipmentsDelivered
- Added new entities to Entity Relationship Diagram.
- Fixed strong/weak entity relationships in Entity Relationship Diagram
- Added new relations to Relationship Model Diagram.
- Console application received total UI and functionality rewrite.
- Created LoadRollingStock stored procedure.
- Created UnloadRollingStock stored procedure.
- Wrote all manual TrainGame demonstration SQL queries in TrainGame1_1_Views.sql.
- Added Iteration 3 Reflection to deliverables.
- Added Iteration 3 SQL Statements to deliverables.
- Added Iteration 3 ChangeLog to deliverables.
- Created GetNextDestination function.
- Created ViewActiveModules view.
- Created ViewUserTrain view.
- Created ViewTrainConsist view.
- Created ViewModule view.
- Created single, comprehensive development manual from existing comments and documents.

# GAME OVERVIEW

Freight train operations are a great way to interact with a layout, treating it as a linear puzzle game. The goal of the game is to take trains to local industries to pick up and deliver goods. The challenge is getting a freight car out from the middle of your train into the siding ahead of it. How do you do this with only using push and pull movements? Now how do you do it in a finite space without blocking the main line with the rest of your train?

## PLAYER ROLES

- Dispatcher
  - Coordinator of layout staffing operations.
  - Assists Yard Masters with Car Card management.
- Yard Master
  - Assigns rolling stock to handle selected waybills.
  - Blocks together trains for Train Operators.
- Train Operator
  - Assigned a DCC throttle and transfers equipment or passengers from module to module.

## SHIPPING ORDERS, WAYBILLS, AND TRAINS

1. Shipping orders are received at a yard for a product delivery.
2. Waybills are used to associate freight cars to shipping orders.
3. Trains are blocked together in “consists” of freight cars.

   Note: These are all things that a game master might set up for a player. For regular players, they’re the ones that handle delivering the goods.

## FREIGHT CAR LIFECYCLE

- The 1st operator is assigned a locomotive and a train consisted with cars. An associated freight car is picked up from a local yard and is delivered to the waybill’s “load” point. When delivered, the relevant freight car remains in the module’s industry siding and the shipping order is considered to be “picked up”.
- The 2nd operator servicing an industry collects the car from the industry siding, adds the rolling stock to their train, and delivers it to the waybill’s “unload” point. The car card is again deposited in the module’s industry siding and the shipping order is considered to be “delivered”.
- The 3rd operator servicing an industry collects the car again, and this time delivers both card and “empty” rolling stock to the specified return yard. The car is then given to the yard master.

## SERVICING INDUSTRIES

The goal of the operator is to deliver all assigned rolling stock, return all empty cars to the assigned yards, and return the locomotive to its assigned yard. Operators can elect to pick up existing rolling stock at industries currently being serviced to deliver the cars to the next destination. If the loading track at an industry is full, the operator must collect all waiting rolling stock, but not to exceed the maximum train length.

## END OF GAME PARAMETERS

The operator has delivered all rolling stock in their assigned train, possesses no freight cars, and has returned to their assigned yard. Alternately, the operator has only “empty” rolling stock and has delivered it to the destination yard.

# DATABASE DESIGN AND RESULTS

The Freight Train Simulator database will handle three primary transaction types: Relation state, game session state, and player state.

- Relation State: This data defines game parameters and relationships, such as properties of rolling stock, product types, and industries.
- Game Session State: This data defines the "state of the world" for a game session. It includes active crews, available rolling stock, available industries, and industry occupancy.
- Player Session State: This data tracks player train length, active waybills (destination orders), and industry interaction.

In order to flesh as much of the design out as early as possible, and to best prevent major reworking later, we started forming our use cases and test data early on. The use cases were defined as thoroughly as possible before beginning on the database schema, and ultimately helped shape the relationships we created.

We categorized our use cases by milestone and by priority. The milestone structure helped us focus on the basic entities early before adding more detail later on, and the priorities ensured we knew what was important to complete for each milestone, versus what could be put on the backburner. Also, not all use cases had direct ties to actual tables, particularly those relating to a procedure, logic, or view.

## MILESTONE 1: LAYOUT, MODULES, INDUSTRIES, AND PHYSICAL MAPPINGS

Note: These settings cannot be modified if there are active game sessions.

### Module Use Case Analysis

#### *Create New Module*

- Priority: Basic
- Status: Available
- The user creates a new module and supplies a module name, module owner, and provides optional module type, optional module shape, and optional module description.
- The system creates a new module record.

#### *Remove Module*

- Priority: Performance
- The user selects a module.
- The system removes the module, if unused.

#### *Modify Module*

- Priority: Delighter
- The user selects a module and edits contents.
- The system updates the module record.

### Module Relations

#### *Modules*

- Defines properties of a physical space containing industries and railroad.
- Attributes:
    - ModuleName: Unique name for a module
    - ModuleOwner: Owner name for a module

    - ModuleType: Arbitrary description of module type
    - ModuleShape: Arbitrary description of module shape
    - Description: Other module notes
- Assumptions: ModuleType, ModuleShape, and Description are optional descriptive attributes that are also fully and non-transitively dependent on ModuleName.

### *ModulesAvailable*

- Defines the availability of a previously created module. A module is available if it is part of a layout, but can remain defined and not be available.
- Pre-conditions: A valid Modules entity must exist.
- Attributes:
    - ModuleName: References Modules ModuleName
    - IsAvailable: Allows a module to be disabled but not deleted
- Assumptions: IsAvailable is fully and non-transitively dependent on ModuleName.

## Main Line Use Case Analysis

### *Create Main Line*

- Priority: Basic
- Status: Available
- The user selects a module name and declares a contiguous main line.
- The system creates a new main line record associated with a module.
- Note: Main lines are contiguous if they span the entire module.

### *Remove Main Line*

- Priority: Performance
- The user selects a module name and a main line.
- The system removes the main line, if unused.

## MainLine Relations

### *MainLines*

- Defines a railroad track on a module that allows passage from at least one side of a module. A main line is contiguous if it allows through traffic to pass over the entirety of a module. Main lines are used by a routing engine and will indicate a path for a train to follow (version 2.0).
- Pre-conditions: A valid Modules entity must exist.
- Attributes:
    - LineName: Name for a module train track
    - OnModule: References Modules ModuleName
    - IsContiguous: Determines full transit authority
- Assumptions: IsContiguous is fully and non-transitively dependent on LineName

## Junction Use Case Analysis

### *Create Junction:*

- Priority: Basic
- Status: Available

- The user selects a module name and declares a junction between two main lines.
- The system creates a new junction record associated with two main lines on a module.
- Note:  Main line junctions are formed where two main lines converge or diverge.

### *Remove Junction*

- Priority:  Delighter
- The user selects a module name and a junction.
- The system removes the junction.

## Junction Relations

### *Junctions*

- Defines a connection point between two main lines on a module where trains may cross from one main line to another.  Physically, junctions are represented by turnouts, and are currently non-directional in nature.  Junctions are used by a routing engine and will indicate a path for a train to follow (version 2.0).
- Pre-conditions:  A valid Modules entity and two MainLines entities must exist.
- Input Constraint:  FromLine <> ToLine
- Attributes:
    - JunctionID:  Arbitrary identifier
    - OnModule:  References Modules ModuleName
    - FromLine References MainLines LineName
    - ToLine References MainLines LineName

## Industry Use Case Analysis

### *Create New Industry*

- Priority:  Basic
- Status:  Available
- The user creates a new industry and supplies a parent module ID, industry name, main line, activity level, a track siding number, and track siding length.
- The system creates a new industry record and assigns it to a module record.

### *Remove Industry*

- Priority:  Performance
- The user selects an industry.
- The system removes the industry.

### *Modify Industry*

- Priority:  Delighter
- The user selects an industry and edits contents.
- The system updates the industry record.

## Industry Relations

### *Industries*

- Defines a business, accessible by rail, producing or consuming goods, and transports those goods via rolling stock cars.
- Pre-conditions:  A valid Modules entity and a MainLines entity must exist.

- Attributes:
    - IndustryName: Name for an industry on a module
    - OnModule: References Modules ModuleName
    - OnMainLine: References MainLines LineName

### *IndustriesAvailable*

- Defines the availability of an existing industry to be used in creating new shipping orders. Industries have the ability to be disabled during a game session to prevent new shipping orders from being created.
- Pre-conditions: A valid Industries entity must exist.
- Attributes:
    - IndustryName: References Industries IndustryName
    - IsAvailable: Allows an industry to be disabled but not deleted
- Assumptions: IsAvailable is fully and non-transitively dependent on IndustryName.

### *IndustryActivities*

- Defines the overall frequency of an existing industry to be considered for new shipping orders.
- Pre-conditions: A valid Industries entity must exist.
- Input Constraint: ActivityLevel = {1, 2, 3} where 1 is lowest and 3 is highest.
- Attributes:
    - IndustryName: References Industries IndustryName
    - ActivityLevel: Setting to control automatic shipping order matching frequency
- Assumptions: ActivityLevel is fully and non-transitively dependent on IndustryName.

### *IndustrySidings*

- Defines which track siding an industry uses for rolling stock cars. An industry must have at least one track siding to be accessable.
- Pre-conditions: A valid Industries entity must exist.
- Input Constraints: SidingLength > 0
- Attributes:
    - ForIndustry: References Industries IndustryName
    - SidingNumber: Arbitrary siding ID
    - SidingLength: Arbitrary siding length, in feet
- Assumptions: SidingLength is fully and non-transitively dependent on ForIndustry and SidingNumber.

### *SidingsAvailableLength*

- Declares the remaining length available for an industry siding when occupied by rolling stock cars. AvailableLength is used to determine if industries should be considered for new shipping orders. If AvailableLength is less than the length of an incoming rolling stock car, the siding is considered to be "full" and should not be considered for new shipping orders.
- Pre-conditions: A valid Industries must exist with at least one declared IndustrySidings entity.
- Input Constraint: AvailableLength > 0, AvailableLength <= SidingLength
- Attributes:
    - ForIndustry: References Industries IndustryName
    - SidingNumber: References IndustrySidings SidingNumber
    - AvailableLength: Stored current value for available siding length remaining

- Assumptions: AvailableLength is fully and non-transitively dependent on ForIndustry and SidingNumber.

### *SidingAssignments*

- Defines preferences for specific product types going to only specific industry sidings. An industry siding will accept all product types for an industry by default. Siding assignments will restrict a siding to only authorized product types. Siding assignments should not be used unless at least two industry sidings exist.
- Pre-conditions: A valid Industries entity must exist more than one declared IndustrySidings entities and at least one ProductTypes entity.
- Input Constraint: The count for total industry sidings at an industry must be at least 2.
- Attributes:
    - ForIndustry: References Industries IndustryName
    - SidingNumber: References IndustrySidings SidingNumber
    - ForProductType: References ProductTypes ProductTypeName

## Yard Use Case Analysis

### *Create New Yard*

- Priority: Basic
- Status: Available
- The user creates a new yard and supplied a parent module ID, yard name, and main line.
- The system creates a new yard record and assigns it to a module record.

### *Remove Yard*

- Priority: Performance
- The user selects a yard.
- The system removes the yard.

### *Modify Yard*

- Priority: Delighter
- The user selects a yard and edits contents.
- The system updates the yard record.

## Yard Relations

### *Yards*

- Defines a collection of tracks used to store trains and rolling stock when not actively being used by crew in a game session. Yards are the typical origination point of empty rolling stock cars and the eventual destination for empty rolling stock cars after shipments have been completed.
- Pre-conditions: A valid Modules entity and a MainLines entity must exist.
- Attributes:
    - YardName: Name for a train yard on a module
    - OnModule: References Modules ModuleName
    - OnMainLine: References MainLines LineName

## Region Use Case Analysis

### *Create Region*

- Priority: Basic
- Status: Available
- The user creates a new region and supplies a region name and optional description.
- The system creates a new region record.

### *Remove Region*

- Priority: Version 2.0

### *Modify Region*

- Priority: Version 2.0

### *Add Module to Region*

- Priority: Version 2.0

### *Remove Module from Region*

- Priority: Version 2.0

### *Edit Module Map*

- Priority: Version 2.0
- Note: A module mapping order must be contiguous in a region (group of associated modules) and allows for module insertions and removals.

## Region Relations

### *Regions*

- Defines groupings of modules and assigns a common name for the purpose of layout maps. Regions are used by a routing engine to indicate a path for a train to follow (version 2.0). They also provide information to the shipping order generator to allow for physical layout spacing between load and unload points (version 2.0).
- Pre-conditions: A valid Modules entity must exist.
- Attributes:
    - RegionName: Name for a collection of modules
    - Module: References Modules ModuleName

# MILESTONE 2: LOGICAL MAPPINGS

Note: These settings cannot be modified if there are active game sessions.

## Rolling Stock Type Use Case Analysis

### *Create New Rolling Stock Type*

- Priority: Basic
- Status: Available
- The user creates a new rolling stock type supplying a rolling stock type name and rolling stock length.
- The system creates a new rolling stock type record.

### *Remove Rolling Stock Type*

- Priority: Performance
- The user selects a rolling stock type.
- The system removes the rolling stock type, if unused.

## Rolling Stock Type Relations

### *RollingStockTypes*

- Define a rolling stock car by name and length.
- Input Constraint: CarLength > 0
- Attributes:
    - CarTypeName: Name of a type of rolling stock car
    - CarLength: Arbitrary average length used for computing available space on sidings
- Assumptions: CarLength is fully and non-transitively dependent on CarTypeName.

## Product Type Use Case Analysis

### *Create New Product Type*

- Priority: Basic
- Status: Available
- The user creates a new product type, supplies the product name, and assigns a rolling stock type for transport.
- The system creates a new product type record and assigns it to a rolling stock type.

### *Remove Product Type*

- Priority: Performance
- The user selects a product type.
- The system removes the product type, if unused. The rolling stock type is unaffected if removed.

### *Create New Industry Product Association*

- Priority: Basic
- Status: Available
- The user selects an industry and selects a product type that they ship, declaring if it is something produced or consumed.
- The system records the association between the industry and the product type, denoting if it produced (load point) or consumed (unload point).

### *Remove Industry Product Association*

- Priority: Performance
- The user selects an industry and product type to remove.
- The system removes the association. Product types are unaffected.

## Product Type Relations

### *ProductTypes*

- Define a product type and which rolling stock car type it is carried by.
- Pre-conditions: A valid RollingStockTypes entity must exist.
- Attributes:

- ProductTypeName: Name of a product to be carried by a car
- OnRollingStockType: References RollingStockTypes CarTypeName

### *IndustryProducts*

- Defines which product types an industry produces or consumes. An industry product entity must be created for each product type served. If this entity exists and IsProducer is FALSE, then the industry is a consumer for that product type.
- Pre-conditions: A valid Industries entity and a ProductTypes entity must exist.
- Attributes:
  - ForIndustry: References Industries IndustryName
  - UsingProductType: References ProductTypes ProductTypeName
  - IsProducer: Determines if an industry produces or consumes a product
- Assumptions: IsProducer is fully and non-transitively dependent on ForIndustry.

## MILESTONE 3: GAME SESSION AND LOGIC

## Game Session Use Case Analysis

### *Start Game Session*

- Priority: Version 2.0
- The user requests to open a new game session and specifies a maximum train length (in rolling stock cars).
- The system records that a game is currently in progress and sets the applicable rule.

### *End Game Session*

- Priority: Version 2.0
- The user requests to end the game session.
- The system un-assigns and removes all active crews, trains, and rolling stock in service.

## Shipping Orders Use Case Analysis

### *Create Shipping Order*

- Priority: None
- Status: Refactor into other use cases
- The user indicates origination location.
- The system triggers a check on the game state to identify industries that have products that can be carried by that rolling stock car type that need to be serviced. A product is selected and load, unload, and reclassification (return empty to yard) instructions are returned back to the user. A full industry occupancy at the time of check will disqualify a particular industry as a load point. A reported activity level will cause an industry to be chosen as a destination more often unless industry occupancy is reported to be full. Load and unload destinations should be in different regions with a specified minimum module separation unless not possible due to layout configuration. Unload regions should be applied sequentially when more than two regions exist. Product types that do not have applicable producers and consumers in the layout should load or unload “off layout” and be delivered to yards for transfer.

### *Create Delivery Request*

- Priority: Version 2.0
- Note: Assign a train to pick up rolling stock at a specific industry.

### *Create Shipment*

- Priority: Basic
- Status: Manual entry only
- The user supplies a request for service at an industry to load products on rolling stock and deliver them to another industry.
- The system creates a record for the shipment request and records the time.
- Note: This can call the Generate Waybill use case in version 2.0.

### *Cancel Shipment*

- Priority: Performance
- The user selects a shipping ID.
- The system deletes the shipping record, if no activity has taken place, and it has not been assigned to rolling stock.

### *Update Shipment*

- Priority: Basic
- Status: Available
- The user selects a shipping ID and supplies a status update for initial pickup or delivery.
- The system updates the shipping record and records the time.

## Shipping Order Relations

### *Shipments*

- Declares a shipping order of a product type for pickup at one industry and delivery to another industry. Shipping orders are created on demand when rolling stock cars are added to a game session (version 2.0).
- Pre-conditions: A valid ProductTypes entity and two Industries entities must exist. For each industry, one IndustrySidings entity must exist.
- Input Constraint: FromIndustry <> ToIndustry
- Attributes:
    - ShipmentID: Arbitrary unique ID.
    - ProductType: References ProductTypes ProductTypeName
    - FromIndustry: References Industries IndustryName
    - FromSiding: References IndustrySiding SidingNumber
    - ToIndustry: References Industries IndustryName
    - ToSiding: References IndustrySiding SidingNumber
    - TimeCreated: Time stamp
- Assumptions: TimeCreated is fully and non-transitively dependent on ShipmentID.

### *ShipmentsPickedUp*

- Declares if a shipping order has been picked up from an Industry producing a certain product type. If this entity exists, the product has been loaded onto a rolling stock car.
- Pre-conditions: A valid Shipments entity must exist.

- Attributes:
  - ShipmentID:  References Shipments ShipmentID
  - TimePickedUp:  Time stamp
- Assumptions:  TimePickedUp is fully and non-transitively dependent on ShipmentID.

### *ShipmentsDelivered*

- Declares if a shipping order has been delivered to an Industry consuming a certain product type. If this entity exists, the product has been unloaded from a rolling stock car.
- Pre-conditions:  A valid Shipments entity must exist.
- Attributes:
  - ShipmentID:  References Shipments ShipmentID
  - TimeDelivered:  Time stamp
- Assumptions:  TimeDelivered is fully and non-transitively dependent on ShipmentID.

## MILESTONE 4:  ROLLING STOCK, TRAINS, AND CREW

Note:  An active game session must exist for these functions.

### Rolling Stock Car Use Case Analysis

#### *Create New Rolling Stock*

- Priority:  Basic
- Status:  Available
- The user selects a physical rolling stock car for use and supplies a car ID, identifies car type, provides an optional description, and specifies a location (industry or yard) origination.
- The system creates an active car's record.

#### *Remove Rolling Stock*

- Priority:  Performance
- The user selects a rolling stock car by ID.
- The system deletes the car record if not assigned to a train and no active shipping waybill.

#### *Modify Rolling Stock*

- Priority:  Performance
- The user selects a rolling stock car ID and supplies changes.
- The system modifies the car's record.

### Rolling Stock Car Relations

#### *RollingStockCars*

- Represents a physical train car used by players in a game session.
- Pre-conditions:  A valid RollingStockCarTypes entity must exist.
- Attributes:
  - CarID:  User defined unique ID for freight cars
  - CarType:  References RollingStockTypes CarTypeName
  - Description:  Other car notes

### *RollingStockAtYards*

- Declares the identies of rolling stock cars currently at a specific train yard. Rolling stock cars not consisted to a train or reported at an industry must report at a yard.
- Pre-conditions: A valid RollingStockCars entity and Yards entity must exist.
- Attributes:
    - CarID: References RollingStockCars CarID
    - AtYard: References Yards YardName
    - TimeArrived: Time stamp
- Assumptions: TimeArrived is fully and non-transitively dependent on CarID.

### *RollingStockAtIndustries*

- Declares the identities of rolling stock cars current at a specific industry. Rolling stock cars not consisted to a train or reported at a yard must report at an industry.
- Pre-conditions: A valid RollingStockCars entity and Industries entity must exist.
- Attributes:
    - CarID: References RollingStockCars CarID
    - AtIndustry: References Industries IndustryName
    - TimeArrived: Time stamp
- Assumptions: TimeArrived is fully and non-transitively dependent on CarID.

## Waybill Use Case Analysis

### *Add Waybill to Rolling Stock*

- Priority: Basic
- Status: Available
- The user selects a rolling stock car ID and requests assignment of shipping orders.
- The system updates the rolling stock car's record to associate a shipping order to specify car loading and unloading destinations, then assigns a reclassification (return empty to yard) destination.

### *Remove Waybill from Rolling Stock*

- Priority: Performance
- Status: Available
- The user selects a rolling stock car ID to cancel this shipment.
- The system removes the shipping assignments from the rolling stock car if the shipping order indicates loading has not taken place or the shipping order is completed.

### *Transfer Waybill to Rolling Stock*

- Priority: Delighter
- The user selects a rolling stock car ID to transfer a waybill from, and a rolling stock car ID to transfer the waybill to.
- The system calls the Add Waybill to Rolling Stock use case (without triggering a new shipment) on the new car ID, then removes the shipping assignments from the old car ID.

### *Generate Waybills*

- Priority: Version 2.0
- The user supplies a module ID with a yard containing rolling stock without active waybills.

- The system generates waybills for each rolling stock car identified. Preference of loading locations is by region, by main line, determined by industry activity level, where possible. Multiple cars should go to the same industry if their activity level is high and they're not full, but limit to 50% capacity.

## Waybill Relations

### *Waybills*

- Declares the association of a shipping order and a specific rolling stock car. ReturnToYard determines the location empty rolling stock will be sent to after the shipping order is complete.
- Pre-conditions: At least one valid entity for RollingStockCars, Shipments, and Yards must exist.
- Attributes:
    - OnCar: References RollingStockCars CarID
    - UsingShipmentID: References Shipments ShipmentID
    - ReturnToYard: References Yards YardName

## Crew Use Case Analysis

### *Create New Crew*

- Priority: Basic
- Status: Available
- The user creates a new player supplying a crew name.
- The system creates a new crew record.

### *Remove Crew*

- Priority: Performance
- Status: Available
- The user selects a crew to remove.
- The system removes the crew if not currently assigned to a train.

## Crew Relations

### *Crews*

- Represents a player identity in a game session.
- Attributes:
    - CrewName: Name of player
    - Description: Other player notes

## Train Use Case Analysis

### *Create New Train*

- Priority: Basic
- Status: Available
- The user indicates that a new train is to be created, supplies a locomotive number, DCC address, and location (module) origination.
- The system records that a new train exists, the identifying information, location, and generates a train number. The train is unassigned to a crew and contains no rolling stock.

### *Remove Train*

- Status: Basic
- Status: Available
- The user selects a train and indicates that a train is no longer needed.
- The system deletes the active train record and removes rolling stock from that train, if any exist.
- Note: Existing trains signify that there is an active game session.

### *Modify Train*

- Priority: Performance
- Status: Available
- The user selects a train and updates information.
- The system updates the train record.

### *Add Rolling Stock to Train*

- Priority: Basic
- Status: Available
- The user selects a train and chooses physically available rolling stock to add to the train. For each car, the user supplies the rolling stock car ID.
- The system associates the rolling stock car ID to the train and records the time. For each car, that car is consisted to a train. Location is specified to be in the train's consist.

### *Remove Rolling Stock from Train*

- Priority: Basic
- Status: Available
- The user selects a train and a rolling stock car ID to remove.
- The system removes that rolling stock car from that train and records the time. Location is specified to be at a yard or industry.

### *Generate Consist*

- Priority: Version 2.0
- The user supplies an origination location and indicates that a new train consist is to be automatically generated.
- The system creates a new train with the Add New Train use case. For each available rolling stock in that module's yard or local industries that rolling stock car is consisted to that train (until maximum train length is reached). Highest priority for rolling stock on a module goes to cars with waybills delivering to the same region along the same main line. Delivery requests can be added for industries if industry occupancy is full and there is no other scheduled industry interaction.

## Train Relations

### *Trains*

- Represents a physical locomotive (or locomotive group) for a player to control in a game session. TimeCreated is for reference only and should not be updated.
- Pre-conditions: A Modules entity must exist for player origination.
- Attributes:
    - TrainNumber: Arbitrary unique ID
    - LeadPower: Number board identifier for lead locomotive

    - DCCAddress: Digital Command Controller ID number
    - TimeCreated: Time stamp
- Assumptions: LeadPower, DCCAddress, and TimeCreated are fully and non-transitively dependent on TrainNumber.

### *TrainLocations*

- Represents the location of a train on a layout. TimeUpdated will apply the current timestamp automatically with any activity.
- Pre-conditions: A valid Trains entity and Modules entity must exist.
- Attributes:
    - TrainNumber: References Trains TrainNumber
    - OnModule: References Modules ModuleName
    - TimeUpdated: Time stamp for last action
- Assumptions: OnModule and TimeUpdated is fully and non-transitively dependent on TrainNumber.

### *ConsistedCars*

- Represents the association of individual rolling stock cars attached to trains.
- Pre-conditions: A valid Trains entity must exist, and for each car added, a RollingStockCars entity must exist.
- Attributes:
    - OnTrain: References Trains TrainNumber
    - UsingCar: References RollingStockCars CarID
    - TimeAdded: Time stamp
- Assumptions: TimeAdded is fully and non-transitively dependent on OnTrain and UsingCar.

## Crew Use Case Analysis

### *Add Crew to Train*

- Priority: Basic
- Status: Available
- The user selects an existing train to crew.
- The system assigns that train to that crew and records the time.

### *Remove Crew from Train*

- Priority: Performance
- Status: Available
- The user selects a crew and a train.
- The system removes the association of that crew from that train and records the time.

## Crew Relations

### *TrainCrews*

- Declares the association of a crew with a train.
- Pre-conditions: A valid Trains entity and valid Crews entity must exist.
- Attributes:
    - OnTrain: References Trains TrainNumber
    - WithCrew: References Crew CrewName

    - TimeJoined: Time stamp
- Assumptions: WithCrew and TimeJoined is fully and non-transitively dependent on OnTrain.

# MILESTONE 5: GAMEPLAY

## Industry Servicing Use Case Analysis

### *Check-In at Module*

- Priority: Basic
- Status: Available
- The user reports that his or her train has arrived or is doing work at a module.
- The system records the crew's current location and records the time.

### *Deliver Rolling Stock*

- Priority: Basic
- Status: Available
- The user reports that his or her train is servicing an industry and indicates that a rolling stock car has been dropped off.
- The system calls the Check-In at Module use case, then the Remove Rolling Stock from Train use case. The industry occupancy is updated to show that a rolling stock car of the type's length is occupying space on an industry siding and that space is no longer available until the car is removed.

### *Receive Rolling Stock*

- Priority: Basic
- Status: Available
- The user reports that his or her train is servicing an industry and indicates that a rolling stock car has been picked up.
- The system calls the Check-In at Module use case, then the Add Rolling Stock to Train use case. The industry occupancy is updated to show that a rolling stock car of the type's length has been removed from the industry siding and that length is now available.

## Player Interaction Use Case Analysis

### *Display List of Modules*

- Priority: Basic
- Status: Available
- The user requests to see modules in the layout.
- The system displays modules that have been added.

### *Display Module*

- Priority: Basic
- Status: Available
- The user requests to see detailed information about a module.
- The system displays module information, including main lines, junctions, and industries on that module.

### *Display Industry*

- Priority: Performance
- Status: Partial support enabled
- The user requests to see detailed information about an industry.
- The system displays industry information, including all product types produced and consumed, and current occupancy.

### *Display User Train*

- Priority: Basic
- Status: Available
- The user selects a train.
- The system displays the crew, locomotive, DCC number, and for each rolling stock car: car ID, car type, loaded product if exists, and car next destination.

### *Display Active Trains*

- Priority: Performance
- Status: Available
- The user requests to view all active trains in an open game session.
- The system reports the train number, locomotive number, DCC number, crew name, last reported location, and time since last report for each train.

### *Display Rolling Stock Car*

- Priority: Performance
- Status: Available
- The user selects a train and rolling stock car ID.
- The system displays the car's ID, car type, description, waybill, and steps complete in the waybill lifecycle.

### *Display Rolling Stock at Location*

- Priority: Version 2.0
- Note: Shows rolling stock on modules and waybills.

### *Display Map*

- Priority: Version 2.0
- The user supplies a region and requests to see a breakdown of industries by main line.
- The system displays a module list, ordered by physical connection.

### *Display Crew History*

- Priority: Delighter
- The user requests to view work performed by a crew during an open game session.
- The system reports a crew assignments by train number. For each train, all rolling stock assignments, module check-ins, and industry movements, and associated timestamps are displayed.

### *Display Industry Shipping Orders*

- Priority: Delighter
- The user supplies an industry ID.

- The system reports all previous (closed) and current shipping orders for that industry. Open shipping orders display rolling stock scheduled to load or deliver products to that industry.

## NORMAL FORM COMPLIANCE

Our design, as of this Iteration 3, is verified to be in both 2nd and 3rd normal forms as described in comments and assumptions located in the TrainGame1_1.sql file. We had to make some changes to our design in order to achieve this:

- Split **Modules** table into *Modules* and *ModulesAvailable*.
- Split **Industries** table into *Industries*, *IndustriesAvailable*, and *IndustryActivities*.
- Split **IndustrySidings** table into *IndustrySidings* and *SidingsAvailableLength*.
- Split **Trains** table into *Trains* and *TrainLocations*.
- Split **Shipments** table into *Shipments*, *ShipmentsPickedUp*, and *ShipmentsDelivered*.

The overarching theme of these modifications was abstracting and separating as many dependencies as possible into their own tables, without introducing any unnecessary redundancy.

As an example, prior to achieving 3NF compliance, our **Shipments** table was a one-stop shop for all details related to a shipment, including what the shipment carried, whether it was picked up (and if so, where), and whether it was delivered (and if so, where). That design did not make optimum use of space, as some rows would not hold values for certain attributes (like pickup and delivery details, if they represented a shipment that hadn't been picked up or delivered yet), while other rows would.

Solving this involved "flattening" the tables, splitting them up based on their different functional dependencies. In the new, improved design, **Shipments** will only hold the *common* details of any shipment (e.g. what it carries), while the **ShipmentsPickedUp** and **ShipmentsDelivered** tables independently hold sets of records describing pickup & delivery events. Each row now always holds a complete set of data for what it represents, and no fields ever need to be empty or NULL.

## PROJECT EVALUATION

This project has taken quite a bit of combined effort overall, as you might expect for a set of interactions and relationships this complex. We ultimately built 27 different tables, seven triggers to enforce various constraints on them, a SQL function for a commonly used task, two stored procedures, and five user views for future client interaction.

Fortunately, having spent the time to work out a good design early on, bringing it to completion has not necessitated any major reworking. No drastic changes were required at any point, and achieving things like 2NF and 3NF has taken only minor refinements, here and there, as we built.

At this point, our design is well-structured and complete, in terms of core functionality. It meets the needs of the scenario and game data, and does so in a logical and efficient manner. Of course, in the larger context, there is more we could add, but what we have feels solid. There is likely not much we would do differently a second time.

If we had another week, we would continue to expand on additional functionalities for gameplay, which were outlined in our use-case list (as “delighter” features). We knew from the outset that it was unlikely we’d complete all of these before submission, but our aim was to design the schema cleanly enough that these stretch goals could be accomplished later, without requiring any major modifications to what we’d already built. It does feel like we’ve achieved that.

## TESTING

While the entity relationship diagram contains notes and the relationship model diagram has a clear list of all relations, attributes, primary keys, foreign keys (and their associated constraints), non-referential constraints and assumptions reside as comments in the TrainGame1_1.sql file.

The package that the Freight Commander game is being built with is MySQL and is being tested on MySQL 5.6.27. The contents of TrainGame1_1.sql has been imported to that server, in that order.

The static test data in TrainGame1_1.sql is modeled after a collection of actual model railroad modules, depicted Figure 1. For the purpose of simulating gameplay, a small layout is constructed. The purpose of the test data is to construct several modules, identify properties of industries on those modules, and allow players to build trains and deliver products via railcar to those industries. Please see the TrainGame1_1.sql file comments for additional details.

For version 1.1 (this release), the algorithm for shipping orders is as follows:

For every industry, industries have product types that are produced or consumed. Each product type is associated with a specific type of rail car. Shipping orders are generated to pick up a “produced” product at one industry and deliver to another “consumed” industry. Every “produced” industry interacts with every other matching “consumed” industry by product type, except itself. Shipping orders only exist for types of rail cars in use for the current game session.

Additionally, 12 rail cars for three players have been created to test the gameplay mechanic. For each player, a train is assigned. For each train, rail cars are added. For each rail car, a shipment order is created, assigned to a train via a waybill, and the car is added to the train. Finally, players are assigned to trains as crew. From this test data, players will be able to deliver rail cars to industries on the layout.

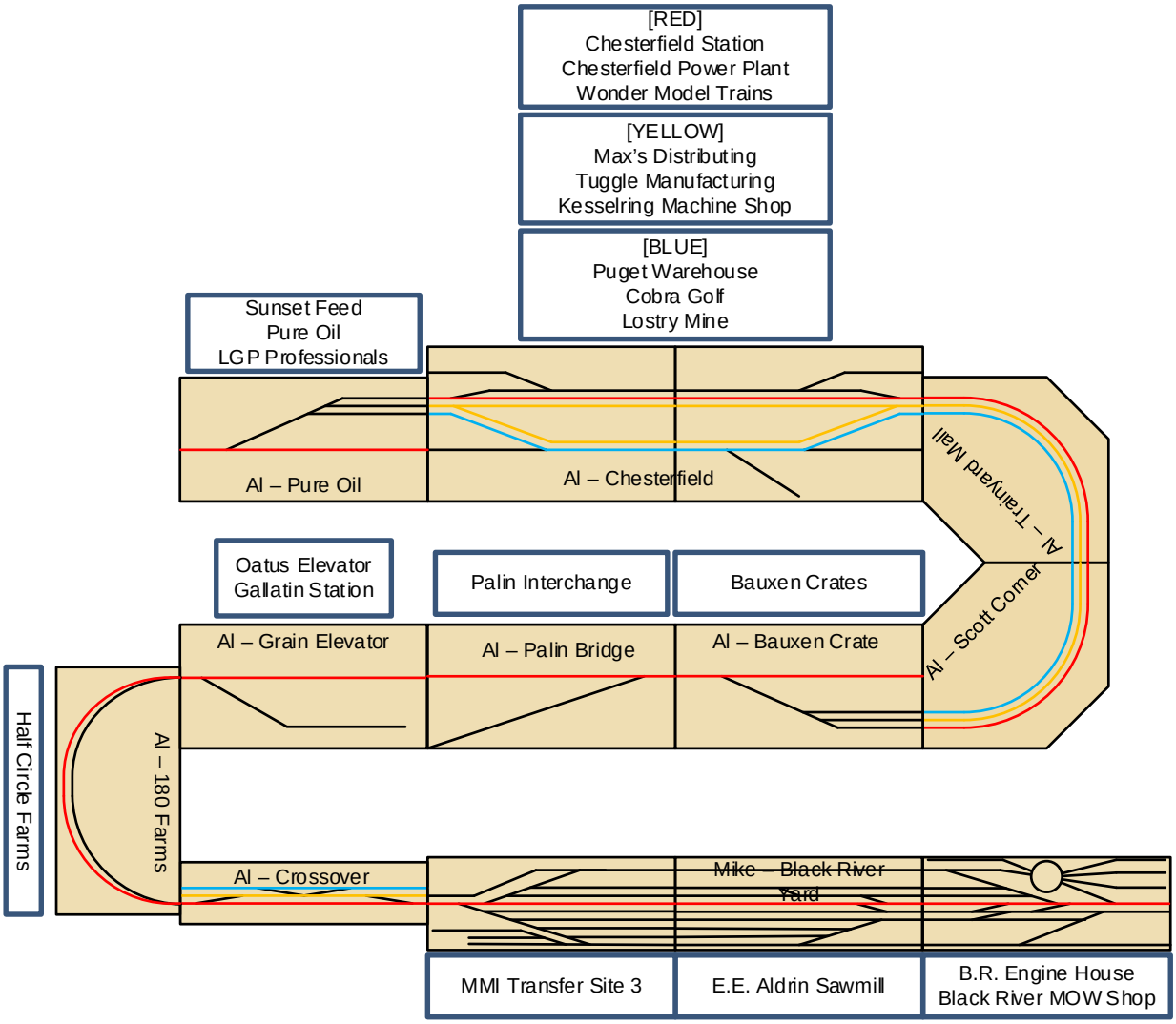



Figure 1: Test Layout

This test data creates a small model railroad layout, train cars, and players. Data insertion is handled in a sequence order that is either required by the database or encountered through normal program interaction. This data, as it is, will only be used for development testing. Normal game operation allows for the application or interface to supply data.

## SECTION A: INITIALIZE GAME PARAMETERS

Expected Insertion Order:

- Define all RollingStockTypes.
- Define all ProductTypes.
- Define rolling stock car types
- Define product type assignments

Note: For each category of rolling stock in use, declare a car type and car length.

Note: For each category of rolling stock car type, multiple product types can be carried. Declare all product types and rolling stock car associations.

## SECTION B: BUILD LAYOUT

Expected Insertion Order (for each Module):

- Declare Modules
- Declare MainLines for Modules
- Declare Junctions on MainLines
- Declare Yards on MainLines
- Declare Industries on MainLines
- Declare IndustriesAvailable for Industries
- Declare IndustryActivities for Industries
- Declare IndustryProducts for Industries
- Declare IndustrySidings for Industries
- Declare SidingsAvailableLength for IndustrySidings
- Declare SidingAssignments for IndustrySidings

Note: Data declared in this section is likely to remain persistent and will be used across multiple game sessions.

## SECTION C: INITIALIZE GAME SESSION

Expected Insertion Order:

- Declare all ModulesAvailable for Modules.
- Declare all Regions for Modules.
- Activate available modules for gameplay.
- Add modules into specific regions on the map.

Note: This section contains non-persistent data which typically only exists for a single game session. A game session is considered active when trains, rolling stock cars, and crews are in existence. To start a game, select which modules are present from the library of previously defined modules and declare which region they are to be in to determine layout shape.

## SECTION D: GAME SESSION

Expected Insertion Order:

- Declare Trains
- Declare TrainLocations for Trains
- Declare RollingStockCars
- Declare Shipments
- Declare Waybills for Shipments on RollingStockCars
- Declare ConsistedCars for Trains using RollingStockCars
- Declare Crews
- Declare TrainCrews for Crews on Trains

Note: A locomotive is selected to declare a new train will exist, originating on a particular module. Available rolling stock for trains are identified and must be assigned to an originating yard or industry. Shipment orders are created on demand based off of current layout and industry

parameters. For each rolling stock car, waybills must be associated with a shipping order before that car is consisted to a train. For each train, crews are declared and then assigned to a train. The train and player are then considered to be ready for game play.

## SQL STATEMENTS

For testing, we manually generated a wide range of test data covering all possible use cases. The test data was based on real-world gameplay, collected by Thomas, who has extensive experience with railroad simulation games like these. Using his experience as a guide, we were able to ensure correct behavior each step of the way.

Input validation through triggers was manually tested (but not included in the final test data) to ensure that acceptable values were accepted by the database, and values that were "out of range" were not. This testing methodology was also applied to logic tests.

To perform continuous integration testing, a self-contained test that goes through all supported steps of gameplay in TrainGame version 1.1 is supplied in the file TrainGame1_1_Demo.sql. This demo performs the following tests:

- Create a demo test train, including a player crew, a new train, and two cars
- Add a crew to a train to verify associative relations
- Display all active modules
- Display the player's train
- Display the train's consist, which computes each car's destination by referencing shipping orders
- Move the player train from one module to another
- Display detailed module information at location
- Load rolling stock cars at an industry
- Unload rolling stock cars at another industry
- Adding and removing cars from the player train
- Displaying industry information, including siding assignments for different product types
- Returning rolling stock to the yard
- Removing crew from a train

This demo test is self-contained and can be run on the live database multiple times without modifying other test data used for the console application's gameplay.

## Create Demo Test Train

| Create Demo Test Train | |
|---|---|
| INSERT INTO Trains VALUES (4, 1234, 1234, DEFAULT);<br>INSERT INTO TrainLocations VALUES (4, 'Black River Yard', DEFAULT);<br><br>INSERT INTO RollingStockCars VALUES ('XA', 'Reefer', NULL);<br>INSERT INTO RollingStockAtYards VALUES ('XA', 'Black River Yard', DEFAULT);<br>INSERT INTO Shipments VALUES (DEFAULT, 'Dairy', 'Half Circle Farms', 1, 'MMI Transfer Site 3', 1, DEFAULT);<br>INSERT INTO Waybills VALUES ('XA', LAST_INSERT_ID(), 'Black River Yard');<br>INSERT INTO ConsistedCars VALUES (4, 'XA', DEFAULT);<br>DELETE FROM RollingStockAtYards WHERE CarID = 'XA';<br><br>INSERT INTO RollingStockCars VALUES ('XB', 'Tank Car', NULL);<br>INSERT INTO RollingStockAtYards VALUES ('XB', 'Black River Yard', DEFAULT);<br>INSERT INTO Shipments VALUES (DEFAULT, 'Gasses', 'LGP Professionals', 4, 'Palin Interchange', 1, DEFAULT);<br>INSERT INTO Waybills VALUES ('XB', LAST_INSERT_ID(), 'Black River Yard');<br>INSERT INTO ConsistedCars VALUES (4, 'XB', DEFAULT);<br>DELETE FROM RollingStockAtYards WHERE CarID = 'XB';<br><br>INSERT INTO Crews VALUES ('Demo Player', NULL); | Creates a #4 train, two cars in a yard, and a new player. Shipping orders are created for the two cars. Waybills associate the shipping order to a car. The cars are removed from a yard and attached (consisted) to a train. |

Figure 2: Create Demo Test Train

## Add Crew to Train

| Add Crew To Train | |
|---|---|
| SET @player = 'Demo Player';<br>SET @playerTrain = 4;<br>INSERT INTO TrainCrews VALUES (@playerTrain, @player, DEFAULT); | The user selects an existing train to crew. The system assigns that train to that crew and records the time. |

Figure 3: Add Crew to Train

## Display List of Modules

| Display List of Modules | |
|---|---|
| SELECT *<br>FROM Modules<br>WHERE ModuleName<br>IN (SELECT ModuleName<br>FROM ModulesAvailable<br>WHERE IsAvailable = TRUE); | The user requests to see modules in the layout. The system displays modules that have been added and are marked as active. |
| Output:<br>ModuleName \| ModuleOwner \| ModuleType \| ModuleShape \| Description<br>180 Farms \| Al Lowe \| oNeTrak \| 180 Corner \| NULL<br>Bauxen Crate \| Al Lowe \| Transition \| Straight \| Access to all lines available<br>Black River Yard \| Mike Donnelly \| oNeTrak \| 3-Straight \| Contains the Black River Yard<br>Chesterfield \| Al Lowe \| Ntrak \| 2-Straight \| No crossovers<br>Crossover \| Al Lowe \| Ntrak \| Straight \| Access to all main lines available<br>Grain Elevator \| Al Lowe \| oNeTrak \| Straight \| NULL<br>Palin Bridge \| Al Lowe \| oNeTrak \| Straight \| NULL<br>Pure Oil \| Al Lowe \| Transition \| Straight \| Access to all main lines available.<br>Scott Corner \| Al Lowe \| Ntrak \| Corner \| NULL<br>Trainyard Mall \| Al Lowe \| Ntrak \| Corner \| NULL<br>NULL \| NULL \| NULL \| NULL \| NULL | |

Figure 4: Display List of Modules

## Display User Train

**Display User Train**

```
SET @playerTrain = 4;
SELECT t.TrainNumber, t.LeadPower, t.DCCAddress, c.WithCrew, l.onModule, l.TimeUpdated
 FROM Trains t, TrainCrews c, TrainLocations l
 WHERE t.TrainNumber = c.OnTrain
  AND t.TrainNumber = l.TrainNumber
      AND t.TrainNumber = @playerTrain;
```

The user selects a train to view. The system displays the train number, locomotive number, digital command controller (DCC) address, which crew is associated with the train, the module the train is currently residing on, and when the train's last movement was.

Output:

| TrainNumber | LeadPower | DCCAddress | WithCrew | onModule | TimeUpdated |
|---|---|---|---|---|---|
| 4 | 1234 | 1234 | Demo Player | Black River Yard | 2015-12-05 15:45:56 |

Figure 5: Display User Train

## Display Train Consist

**Display Train Consist**

```
SET @playerTrain = 4;
SELECT r.CarID, r.CarType, s.ProductType,
 (CASE
  WHEN s.ShipmentID IN (SELECT ShipmentID
   FROM ShipmentsPickedUp
   WHERE ShipmentID = s.ShipmentID)
    AND s.ShipmentID IN (SELECT ShipmentID
     FROM ShipmentsDelivered
     WHERE ShipmentID = s.ShipmentID)
  THEN (SELECT @nextDestination := (SELECT ReturnToYard
     FROM Waybills
     WHERE UsingShipmentID = s.ShipmentID))
  WHEN s.ShipmentID NOT IN (SELECT ShipmentID
   FROM ShipmentsPickedUp
   WHERE ShipmentID = s.ShipmentID)
  THEN (SELECT @nextDestination := (SELECT FromIndustry
     FROM Shipments
     WHERE ShipmentID = s.ShipmentID))
  WHEN s.ShipmentID IN (SELECT ShipmentID
   FROM ShipmentsPickedUp
   WHERE ShipmentID = s.ShipmentID)
  THEN (SELECT @nextDestination := (SELECT ToIndustry
     FROM Shipments
     WHERE ShipmentID = s.ShipmentID))
 END) AS NextDestination,
```

The user selects a train to view waybill information. The system displays the train's consist (attached cars). For each car, it displays: car ID, car type, loaded product if exists, and car next destination.

```
IF(@nextDestination IN (SELECT IndustryName
  FROM Industries
  WHERE IndustryName = @nextDestination),
 (SELECT OnModule
  FROM Industries
  WHERE IndustryName = @nextDestination),
 (SELECT OnModule
  FROM Yards
  WHERE YardName = @nextDestination)) AS Module
FROM RollingStockCars r, Waybills w, Shipments s
WHERE r.CarID = w.OnCar
 AND w.UsingShipmentID = s.ShipmentID
 AND OnCar IN (SELECT UsingCar
  FROM ConsistedCars
  WHERE OnTrain = @playerTrain);
```

Output:

| CarID | CarType | ProductType | NextDestination | Module |
|---|---|---|---|---|
| XA | Reefer | Dairy | Half Circle Farms | 180 Farms |
| XB | Tank Car | Gasses | LGP Professionals | Pure Oil |

Figure 6: Display Train Consist

## Check-In at Module (Move Train)

**Check-In at Module (Move Train)**

```
SET @playerModule = '180 Farms';
SET @playerTrain = 4;
UPDATE TrainLocations SET OnModule = @playerModule WHERE TrainNumber = @playerTrain;
```

The user reports that his or her train has arrived or is doing work at a module. The system records the train's current location and records the time.

Figure 7: Check-In at Module

## Display Module

| Display Module | |
|---|---|
| SET @playerModule = '180 Farms';<br>SELECT *<br>FROM Industries<br>WHERE IndustryName IN (SELECT IndustryName<br>FROM IndustriesAvailable<br>WHERE IsAvailable = TRUE)<br>AND OnModule = @playerModule; | The user requests to see industries on a module. The system displays active industries on that module and which main line they are on. |
| Output:<br>IndustryName \| OnModule \| OnMainLine<br>Half Circle Farms \| 180 Farms \| Red<br>NULL \| NULL \| NULL | |

Figure 8: Display Module

## Load Rolling Stock (formerly Deliver Rolling Stock)

| Load Rolling Stock (formerly Deliver Rolling Stock) | |
|---|---|
| SET @playerIndustry = 'Half Circle Farms';<br>SET @playerCar = 'XA';<br>CALL LoadRollingStock(@playerIndustry, @playerCar); | The user reports that his or her train is servicing an industry and indicates that a rolling stock car has been dropped off for loading. The system ensures the product type carried matches a product type produced by the industry and verifies that the shipping order has not previously been recorded as being picked up. The car is removed from the train and added to the industry siding. The shipping order is updated to record that the load has been picked up. |

Figure 9: Load Rolling Stock

## Display Rolling Stock at Industry

| Display Rolling Stock At Industry | |
|---|---|
| SET @playerIndustry = 'Half Circle Farms';<br>SELECT c.CarID, c.CarType, i.AtIndustry, i.TimeArrived<br>FROM RollingStockCars c<br>JOIN RollingStockAtIndustries i ON c.CarID = i.CarID<br>WHERE i.AtIndustry = @playerIndustry; | The user requests to see rolling stock servicing an industry. The system displays, for each car: car ID, car type, and arrival time. |
| Output:<br>CarID \| CarType \| AtIndustry \| TimeArrived<br>XA \| Reefer \| Half Circle Farms \| 2015-12-05 15:46:25 | |

Figure 10: Display Rolling Stock at Industry

## Receive Rolling Stock

| Receive Rolling Stock | |
|---|---|
| SET @playerCar = 'XA';<br>SET @playerTrain = 4;<br>DELETE FROM RollingStockAtIndustries WHERE CarID = @playerCar;<br>INSERT INTO ConsistedCars VALUES (@playerTrain, @playerCar, DEFAULT); | The user selects a car servicing an industry to add to a train. The system removes the car from the industry siding, consists the car to a train, and records the time. |

Figure 11: Receive Rolling Stock

Player calls Check-In at Module again to move to the 'Black River Yard' module.

## Display Industry

| Display Industry | |
|---|---|
| SET @playerIndustry = 'MMI Transfer Site 3';<br>SELECT s.*, p.UsingProductType<br>FROM IndustrySidings s<br>JOIN IndustryProducts p ON s.ForIndustry = p.ForIndustry<br>WHERE p.UsingProductType NOT IN (SELECT ForProductType<br>FROM SidingAssignments<br>WHERE ForIndustry = @playerIndustry)<br>AND s.SidingNumber NOT IN (SELECT SidingNumber<br>FROM SidingAssignments<br>WHERE ForIndustry = @playerIndustry)<br>AND s.ForIndustry = @playerIndustry<br>UNION SELECT s.*, p.UsingProductType<br>FROM IndustrySidings s<br>JOIN IndustryProducts p ON s.ForIndustry = p.ForIndustry<br>WHERE p.UsingProductType IN (SELECT ForProductType<br>FROM SidingAssignments<br>WHERE ForIndustry = @playerIndustry)<br>AND s.SidingNumber IN (SELECT SidingNumber<br>FROM SidingAssignments<br>WHERE ForIndustry = @playerIndustry)<br>AND s.ForIndustry = @playerIndustry; | The user requests to see detailed information about an industry. The system displays industry siding information and identifies which product types can be delivered to specific sidings, if siding assignments are defined. |
| Output: | |

| ForIndustry | SidingNumber | SidingLength | UsingProductType |
|---|---|---|---|
| MMI Transfer Site 3 | 1 | 100 | Dairy |
| MMI Transfer Site 3 | 1 | 100 | Meats |
| MMI Transfer Site 3 | 1 | 100 | Produce |
| MMI Transfer Site 3 | 2 | 100 | Dairy |
| MMI Transfer Site 3 | 2 | 100 | Meats |
| MMI Transfer Site 3 | 2 | 100 | Produce |
| MMI Transfer Site 3 | 3 | 150 | General Merchandise |
| MMI Transfer Site 3 | 3 | 150 | Manufactured Foods |

Figure 12: Display Industry

### Unload Rolling Stock

| Unload Rolling Stock (formerly Deliver Rolling Stock) | |
|---|---|
| SET @playerIndustry = 'MMI Transfer Site 3';<br>SET @playerCar = 'XA';<br>CALL UnloadRollingStock(@playerIndustry, @playerCar); | The user reports that his or her train is servicing an industry and indicates that a rolling stock car has been dropped off for unloading. The system ensures the product type carried matches a product type consumed by the industry and verifies that the shipping order has not previously been recorded as being delivered. The car is removed from the train and added to the industry siding. The shipping order is updated to record that the load has been delivered. |

Figure 13: Unload Rolling Stock

Player calls Receive Rolling Stock, returning car 'XA' to train 4.

### Return Rolling Stock to Yard

| Return Rolling Stock to Yard | |
|---|---|
| SET @playerTrain = 4;<br>SET @playerCar = 'XA';<br>SET @playerYard = 'Black River Yard';<br>DELETE FROM ConsistedCars WHERE OnTrain = @playerTrain AND UsingCar = @playerCar;<br>INSERT INTO RollingStockAtYards VALUES (@playerCar, @playerYard, DEFAULT);<br>DELETE FROM Waybills WHERE OnCar = @playerCar; | The user selects an empty car from a train to return to a yard to be reclassified into other trains. The system removes the car from the player's train. The car is assigned to the yard and the waybill is destroyed. |

Figure 14: Return Rolling Stock to Yard

### Remove Crew from Train

| Remove Crew From Train | |
|---|---|
| SET @playerTrain = 4;<br>DELETE FROM TrainCrews WHERE OnTrain = @playerTrain; | The user is finished with a game session. The system disassociates a crew from a train. |

Figure 15: Remove Crew from Train

## APPLICATION

The console application for this game is fairly simple, using the Entity Framework provided by .NET. It makes a connection to the database if the correct connection string is defined and provided by the user during program compilation. The connection string is used to define the location of the MySQL server, port, schema, username, and password. There is no way to change it during runtime, but since this application was designed for a simple demonstration of database connectivity and interactivity, this is something that can be changed in future versions.

## INSTALLATION INSTRUCTIONS

These instructions contains basic steps to compile the demonstration presentation console client for TrainGame v1.1.

Use the Database Presentation.sln file to open the solution for this Visual Studio project.

Ensure that Visual Studio 2013 EntityFramework is installed and up-to-date. You can do this by drilling down from the Visual Studio menu bar on TOOLS->NuGet Package Manager->Manage NuGet Packages for Solution... Update EntityFramework if necessary for the project.

You will need to connect a database to this application in DBHandler.cs.

Supply the appropriate server, port, schema, userName, and password strings for connection to your locally hosted MySQL server.

From the Visual Studio menu bar: BUILD->Clean Solution.

From the Visual Studio menu bar: BUILD->Build Solution.

The compiled application executable located at bin/Debug/Database Presentation.exe

## DEMONSTRATION PRESENTATION.EXE

When the application is launched, you are presented with a console that asks your name then asks for you to select a train that you want to have control of. The list of trains that you will be presented with is all of the trains that have been defined, and technically if there were multiple instances of this game running at the same time (i.e. multiple players) all players could control the same train.

Once you've set up your player and train you will be presented with a list of 10 items that you can choose. The first 4 items are designed to show information about the game and your train. The last options are actually designed to let you play the game. As options are chosen the game will walk you through the steps that are required to complete that requested action.

# APPENDICES

In order presented:

- RM Diagram (Iteration2)
- ER Diagram (Iteration2)
- RM Diagram (Iteration3)
- ER Diagram (Iteration3)

CSS 475 - Project Iteration 2
Brett Holman, Thomas Dye, Zakk Roberts

Modules

| ModuleName | ModuleOwner | isAvaliable | ModuleType | ModuleShape | Description |
|---|---|---|---|---|---|

Regions

| RegionName | Module |
|---|---|

MainLines

| LineName | OnModule | IsContiguous |
|---|---|---|

Junctions

| JunctionID | FromLine | ToLine | OnModule |
|---|---|---|---|

Industries

| IndustryName | OnMainLine | OnModule | IsAvaliable | ActivityLevel |
|---|---|---|---|---|

Yards

| YardName | OnMainLine | OnModule |
|---|---|---|

RollingStockTypes

| CarTypeName | CarLength |
|---|---|

ProductTypes

| ProductTypeName | OnRollingStockType |
|---|---|

IndustryProducts

| ForIndustry | UsingProductType | IsProducer |
|---|---|---|

IndustrySidings

| ForIndustry | SidingNumber | SidingLegnth | AvaliableLength |
|---|---|---|---|

SidingAssignments

| ForIndustry | SidingNumber | ForProductType |
|---|---|---|

Shipments

| ShipmentID | ProductType | FromIndustry | FromSiding | ToIndustry | ToSiding | TimeCreated | TimePickedUp | TimeDelivered |
|---|---|---|---|---|---|---|---|---|

RollingStockCars

| CarID | CarType | Description |
|---|---|---|

RollingStockAtIndustries

| CarID | AtIndustry | TimeArrived |
|---|---|---|

RollingStockAtYards

| CarID | AtYard | TimeArrived |
|---|---|---|

Waybills

| OnCar | UsingShipmentID | ReturnToYard |
|---|---|---|

Crews

| CrewName | Description |
|---|---|

Trains

| TrainNumber | LeadPower | DCCAddress | OnModule | TimeCreated | TimeUpdated |
|---|---|---|---|---|---|

ConsistedCars

| OnTrain | UsingCar | TimeAdded |
|---|---|---|

TrainCrews

| OnTrain | WithCrew | TimeJoined |
|---|---|---|

CSS 475 Project Iteration 2

Brett Holman, Thomas Dye, Zakk Roberts

Note:

The ER diagram is grouped into collections of similar entities.

RegionName — **Regions** — A module is a section of a layout containing track and buildings

Regions — N — Has — 1 — Modules

**Industries**: IndustryName, IsAvailable, ActivityLevel

**Modules**: ModuleName, ModuleOwner, isAvailable, Description, ModuleType

**Junctions**: JunctionID

Industries — N — contain — 1 — Modules

Modules — 1 — contain — N — Junctions

Most modules contain one or more industries.

Junctions provide access from one main line to another. Not all modules have junctions.

Modules — 1 — Has — N — **Yards**: YardName, ModuleShape

Industries — N — isOn — 1 — Yards

Yards — N — isOn — 1 — Modules

Modules — 1 — has — N — **MainLines**: LineName, IsContiguous

Junctions — N — fromLine — 1 — MainLines

Junctions — N — toLine — 1 — MainLines

A few modules contain train yards.

A contiguous main line is a track that spans an entire module.

This is the same entity

**Shipments**: ShipmentID, TimeCreated, TimePickedUp, TimeDelivered

Shipments — N — from — 1 — Industries

Shipments — N — to — 1 — Industries

Shipments — N — contain — 1 — ProductTypes (Duplicate)

Shipments — N — from — 1 — *IndustrySidings (Duplicate)*

Shipments — N — to — 1 — *IndustrySidings (Duplicate)*

**Industries** — 1 — has — N — **IndustrySidings**: SidingLength, AvailableLength

Industries — 1 — contain — N — **IndustryProducts**: IsProducer

Industries — 1 — for — N — **SidingAssignments**

IndustrySidings — 1 — have — N — SidingAssignments

Most industries have one siding. Some have multiple. If there are no siding preferences, they are shared. Other industries want specific products delivered to specific sidings.

IndustryProducts — N — have — 1 — ProductTypes

SidingAssignments — N — for — 1 — ProductTypes

**RollingStockTypes**: CarLength, CarTypeName

RollingStockTypes — 1 — Have — N — ProductTypes — ProductName

Shipments are generated based off of industry demand.

Only certain types of cars can carry certain products.

**RollingStockCars**: Description, CarID

RollingStockTypes — 1 — becomes — N — RollingStockCars

RollingStockCars — 1 — become — N — **ConsistedCars**

ConsistedCars — N — belongTo — 1 — **Trains**: TrainNumber, LeadPower, TimeUpdated, TimeCreated, DCCAddress

ConsistedCars — TimeAdded

Trains — N — onModule — 1 — **Module (Duplicate)**

**Crews**: CrewName, Description

Crews — 1 — contain — N — **TrainCrews**: TimeJoined

TrainCrews — N — have — 1 — Trains

A train and its rolling stock are considered a consist.

Waybills use existing shipment requests before being assigned to rolling stock.

RollingStockCars — 1 — have — N — **Waybills**

Waybills — N — usingShippingID — 1 — **Shipments (Duplicate)**

Waybills — N — returnTo — 1 — **Yards (Duplicate)**

RollingStockCars — 1 — assignedTo — N — **RollingStockAtYards**

RollingStockAtYards — N — at — 1 — Yards (Duplicate)

RollingStockAtYards — CarID, TimeArrived

RollingStockCars — 1 — assignedTo — N — **RollingStockAtIndustries**

RollingStockAtIndustries — N — at — 1 — **Industries (Duplicate)**

RollingStockAtIndustries — CarID, TimeArrived

Industry sidings can fill to capacity, so they are tracked.

CSS 475 - Project Iteration 3
Brett Holman, Thomas Dye, Zakk Roberts

Modules

| ModuleName | ModuleOwner | ModuleType | ModuleShape | Description |
|---|---|---|---|---|

ModulesAvailable

| ModuleName | isAvailable |
|---|---|

Regions

| RegionName | Module |
|---|---|

MainLines

| LineName | OnModule | IsContiguous |
|---|---|---|

Junctions

| JunctionID | FromLine | ToLine | OnModule |
|---|---|---|---|

Yards

| YardName | OnMainLine | OnModule |
|---|---|---|

Industries

| IndustryName | OnMainLine | OnModule |
|---|---|---|

IndustriesAvailable

| IndustryName | IsAvailable |
|---|---|

IndustryActivities

| IndustryName | ActivityLevel |
|---|---|

RollingStockTypes

| CarTypeName | CarLength |
|---|---|

ProductTypes

| ProductTypeName | OnRollingStockType |
|---|---|

IndustryProducts

| ForIndustry | UsingProductType | IsProducer |
|---|---|---|

IndustrySidings

| ForIndustry | SidingNumber | SidingLegnth |
|---|---|---|

SidingsAvailableLength

| ForIndustry | SidingNumber | AvailableLength |
|---|---|---|

SidingAssignments

| ForIndustry | SidingNumber | ForProductType |
|---|---|---|

Shipments

| ShipmentID | ProductType | FromIndustry | FromSiding | ToIndustry | ToSiding | TimeCreated |
|---|---|---|---|---|---|---|

ShipmentsPickedUp

| ShipmentID | TimePickedUp |
|---|---|

ShipmentsDelivered

| ShipmentID | TimeDelivered |
|---|---|

RollingStockCars

| CarID | CarType | Description |
|---|---|---|

RollingStockAtIndustries

| CarID | AtIndustry | TimeArrived |
|---|---|---|

RollingStockAtYards

| CarID | AtYard | TimeArrived |
|---|---|---|

Waybills

| OnCar | UsingShipmentID | ReturnToYard |
|---|---|---|

Crews

| CrewName | Description |
|---|---|

Trains

| TrainNumber | LeadPower | DCCAddress | TimeCreated |
|---|---|---|---|

TrainLocations

| TrainNumber | OnModule | TimeUpdated |
|---|---|---|

ConsistedCars

| OnTrain | UsingCar | TimeAdded |
|---|---|---|

TrainCrews

| OnTrain | WithCrew | TimeJoined |
|---|---|---|

# CSS 475 Project Iteration 3

Brett Holman, Thomas Dye, Zakk Roberts

Note:

The ER diagram is grouped into collections of similar entities.

Most modules contain one or more industries.

A module is a section of a layout containing track and buildings

Junctions provide access from one main line to another. Not all modules have junctions.

A few modules contain train yards.

A contiguous main line is a track that spans an entire module.

This is the same entity

Shipments are generated based off of industry demand.

Most industries have one siding. Some have multiple. If there are no siding preferences, they are shared. Other industries want specific products delivered to specific sidings.

Only certain types of cars can carry certain products.

A train and its rolling stock are considered a consist.

Waybills use existing shipment requests before being assigned to rolling stock.

Industry sidings can fill to capacity, so they are tracked.

IsAvailable
RegionName
**Regions**
**ModulesAvailable**
Has
is
IndustryName
ModuleName
ModuleOwner
Description
JunctionID
IsAvailable
**IndustriesAvailable**
is
Industries
contain
**Modules**
contain
Junctions
YardName
has
ModuleType
ActivityLevel
**IndustryActivities**
have
Yards
ModuleShape
fromLine
toLine
isOn
isOn
has
LineName
MainLines
IsContiguous
TimePickedUp
TimeDelivered
**ShipmentsPickedUp**
**ShipmentsDelivered**
are
are
TimeCreated
ShipmentID
Shipments
from
from
to
contain
to
**IndustrySidings (Duplicate)**
ProductTypes (Duplicate)
SidingLength
AvailableLength
Industries
has
**IndustrySidings**
has
**SidingsAvailableLength**
contain
for
have
**IndustryProducts**
**SidingAssignments**
IsProducer
have
for
CarLength
**RollingStockTypes**
have
ProductTypes
ProductName
CarTypeName
Description
CarID
becomes
RollingStockCars
become
**ConsistedCars**
belongTo
Trains
TimeAdded
have
assignedTo
assignedTo
**Waybills**
**RollingStockAtYards**
**RollingStockAtIndustries**
usingShippingID
returnTo
at
at
TimeArrived
TimeArrived
Shipments (Duplicate)
Yards (Duplicate)
Industries (Duplicate)
CrewName
Description
TimeJoined
**Crews**
contain
**TrainCrews**
have
DCCAddress
TimeCreated
TrainNumber
LeadPower
at
onModule
**TrainLocations**
**Modules (Duplicate)**
TimeUpdated