Railway Empire: Stored Procedure Documentation

## sp\_buyAsset

**Overview**

The sp\_buyAsset stored procedure facilitates the purchase of an asset in a Railway Project database. It takes a JSON input parameter, json\_input, containing essential information such as userId (the ID of the player buying the asset) and assetId (the ID of the asset to be purchased). The procedure validates the input JSON structure, checks the availability of funds and ownership, and updates the asset's owner accordingly.

**Parameters**

* json\_input (IN JSON): A JSON object containing the following properties:
  + userId (number): The ID of the player making the purchase.
  + assetId (number): The ID of the asset to be purchased.

**Example Usage**

CALL sp\_buyAsset('{"userId": 1, "assetId": 3}');

This example call attempts to purchase the asset with ID 3 for the player with ID 1. Ensure that the provided JSON input adheres to the specified schema:

{ "userId": 1, "assetId": 3 }

## sp\_changeNeeds

**Overview**

The **sp\_changeNeeds** stored procedure is designed to modify the needs and stockpiles associated with a specific asset in a Railway Project database. This stored procedure allows for adjustments in consumption levels based on the asset's level and the option to increase (**levelup = 1**) or decrease (**levelup = 0**) those levels. It operates on the Needs and Stockpiles tables, updating or inserting records accordingly.

**Parameters**

* **assetId** (IN INT): The ID of the asset for which needs and stockpiles will be modified.
* **assetLevel** (IN INT): The current level of the asset (1 to 5).
* **levelup** (IN INT): Flag indicating whether to increase (**1**) or decrease (**0**) the consumption levels.

**Example Usage**

CALL sp\_changeNeeds(1, 2, 1);

This example call increases the consumption levels for an asset with ID 1, currently at level 2.

## sp\_createAsset

**Overview**

The **sp\_createAsset** stored procedure facilitates the creation of a new asset in a Railway Project database based on the provided JSON input. It validates the input JSON structure, checks for the existence of an asset with the same name, and creates the asset accordingly. The response includes details of the created asset.

**Parameters**

* **json\_data** (IN JSON): A JSON object with properties:
  + **type** (string): The type of the asset (**TOWN** or **RURALBUSINESS**).
  + **name** (string): The name of the asset.
  + **position** (object): An object with X and Y coordinates of the asset's position.
    - **x** (number): The X-coordinate.
    - **y** (number): The Y-coordinate.
  + **worldId** (number): The ID of the world to which the asset belongs.

**Example Usage**

CALL sp\_createAsset('{"type": "TOWN", "name": "MyTown", "position": {"x": 10, "y": 20}, "worldId": 1}');

This call attempts to create a town named "MyTown" at position (10, 20) in the world with ID 1. Ensure that the provided JSON input adheres to the specified schema.

## sp\_createBusiness

**Overview**

The **sp\_createBusiness** stored procedure is designed to associate a specific type of business with a given asset in a Railway Project database. This procedure facilitates the creation of relationships in the **Makes** table, linking the asset with goods that the business produces. It supports various types of businesses such as 'RANCH,' 'FIELD,' 'FARM,' 'LUMBERYARD,' 'PLANTATION,' and 'MINE.'

**Parameters**

* **business** (IN VARCHAR(450)): The type of business to be associated with the asset.
* **assetId** (IN INT): The ID of the asset with which the business relationship will be established.

**Example Usage**

sqlCopy code

CALL sp\_createBusiness('FARM', 7);

This example call associates a 'FARM' business with the asset having ID 7. The procedure inserts records in the **Makes** table to represent the goods produced by the farm.

Please ensure that the provided business type is one of the supported types ('RANCH,' 'FIELD,' 'FARM,' 'LUMBERYARD,' 'PLANTATION,' 'MINE') and that the asset ID corresponds to an existing asset in the database.

## sp\_getAsset

**Overview**

The **sp\_getAsset** stored procedure retrieves information about a specific asset in a Railway Project database based on the provided asset ID. It validates the input JSON structure containing the asset ID, checks for the existence of the asset, and returns the asset details in a JSON response.

**Parameters**

* **json\_data** (IN JSON): A JSON object with the property:
  + **id** (number): The ID of the asset to retrieve information for.

**Example Usage**

CALL sp\_getAsset('{"id": 5}');

This example call retrieves information for the asset with ID 5. Ensure that the provided JSON input adheres to the specified schema.

## sp\_getAssets

**Overview**

The **sp\_getAssets** stored procedure retrieves information about all assets in a Railway Project database. It returns a JSON array containing details of each asset, including ID, name, type, population, level, stockpileMax, worldId, position, and owner ID. If no assets are found, a corresponding message is returned.

**Parameters**

* None

**Example Usage**

CALL sp\_getAssets();

This example call retrieves information for all assets in the database. The returned JSON array includes details for each asset.

## sp\_getAssetsStation

**Overview**

The **sp\_getAssetsStation** stored procedure retrieves information about the station associated with a specific asset in a Railway Project database. It takes the asset ID as input, checks for the existence of a station linked to the asset, and returns station details in a JSON response.

**Parameters**

* **json\_input** (IN JSON): A JSON object with the property:
  + **assetId** (number): The ID of the asset to retrieve station information for.

**Example Usage**

CALL sp\_getAssetsStation('{"assetId": 7}');

This example call retrieves information about the station associated with the asset with ID 7. Ensure that the provided JSON input adheres to the specified schema.

## sp\_getUserAssets

**Overview**

The **sp\_getUserAssets** stored procedure retrieves information about the assets owned by a specific user in a Railway Project database. It takes the user ID as input, checks for the existence of the user, and returns details of the user's assets in a JSON response.

**Parameters**

* **json\_data** (IN JSON): A JSON object with the property:
  + **id** (number): The ID of the user to retrieve asset information for.

**Example Usage**

CALL sp\_getUserAssets('{"id": 5}');

This example call retrieves information about the assets owned by the user with ID 5. Ensure that the provided JSON input adheres to the specified schema.

## sp\_generateGoods

**Overview**

The **sp\_generateGoods** stored procedure generates and updates stockpiles of goods for town assets in a Railway Project database based on their respective levels. It populates mail and passengers quantities according to predetermined values for each asset level.

**Parameters**

* None

**Example Usage**

CALL sp\_generateGoods();

This example call executes the procedure to generate and update stockpiles of goods for all town assets in the database, taking into account their respective levels. No parameters are required for this procedure.

## sp\_getGood

**Overview**

The **sp\_getGood** stored procedure retrieves information about a specific good in a Railway Project database. It takes the good ID as input, checks for the existence of the good, and returns details of the good in a JSON response.

**Parameters**

* **json\_data** (IN JSON): A JSON object with the property:
  + **id** (number): The ID of the good to retrieve information for.

**Example Usage**

CALL sp\_getGood('{"id": 3}');

This example call retrieves information about the good with ID 3. Ensure that the provided JSON input adheres to the specified schema.

## sp\_getGoods

**Overview**

The **sp\_getGoods** stored procedure retrieves information about all goods in a Railway Project database. It returns details of each good in a JSON array.

**Parameters**

This procedure does not require any input parameters.

**Example Usage**

CALL sp\_getGoods();

This example call retrieves information about all goods in the database. Ensure that you use the correct syntax when making the procedure call. The response will include a JSON array containing details of each good, such as ID and name.

## sp\_createIndustry

**Overview**

The **sp\_createIndustry** stored procedure creates a new industry in the Railway Project database. It checks various conditions based on the asset level and existing industries before allowing the creation. The procedure returns a JSON response indicating the success or failure of the operation.

**Parameters**

* **json\_data**: JSON object containing information about the industry to be created.
  + **"name"**: The name of the industry.
  + **"idAsset"**: The ID of the asset (town) where the industry is to be created.
  + **"type"**: The type of industry to be created, which must be one of the specified types.

**Example Usage**

CALL sp\_createIndustry('{"name": "Bakery1", "idAsset": 2, "type": "BAKERY"}');

This example call creates a bakery named "Bakery1" in the town with ID 2. Ensure that you use the correct syntax when making the procedure call. The response will include a JSON object indicating the success or failure of the operation, along with details of the created industry if successful.

## sp\_getIndustries

**Overview**

The **sp\_getIndustries** stored procedure retrieves information about all industries in the Railway Project database. It returns a JSON response containing details about each industry, including its ID, name, warehouse capacity, and the ID of the associated asset.

**Parameters**

This procedure has no input parameters.

**Example Usage**

CALL sp\_getIndustries();

This example call retrieves information about all industries in the database. The response will include a JSON object indicating the success or failure of the operation, along with details of the industries if successful.

## sp\_getIndustry

**Overview**

The **sp\_getIndustry** stored procedure retrieves information about a specific industry in the Railway Project database based on the provided industry ID. It returns a JSON response containing details about the industry, including its ID, name, warehouse capacity, and the ID of the associated asset.

**Parameters**

* **json\_data** (IN): A JSON object containing the following property:
  + **id** (number, required): The ID of the industry to retrieve.

**Example Usage**

CALL sp\_getIndustry('{"id": 123}');

This example call retrieves information about the industry with ID 123. The response will include a JSON object indicating the success or failure of the operation, along with details of the industry if successful.

## **sp\_updateIndustryDaily**

**Overview**

The **sp\_updateIndustryDaily** stored procedure updates the daily status of industries in the Railway Project database. It processes each industry, consumes goods based on the industry type, produces new goods, and updates the warehouse capacity. Additionally, it calculates the change in warehouse capacity and adds funds to the owner of the associated asset based on the capacity change.

**Parameters**

This stored procedure does not have any parameters.

**Example Usage**

CALL sp\_updateIndustryDaily();

This example call updates the daily status of industries, including goods consumption, production, and warehouse capacity adjustments. The stored procedure iterates through each industry, performs the necessary updates, and adds funds to the asset owner based on the capacity change.

## sp\_getPlayer

**Overview**

The **sp\_getPlayer** stored procedure retrieves information about a player from the Railway Project database. It takes a JSON object as input containing the user ID and returns relevant details such as the username, associated world ID, and available funds for the specified player.

**Parameters**

* **json\_data** (JSON): A JSON object containing the following properties:
  + **userId** (number, required): The unique identifier of the player whose information is to be retrieved.

**Example Usage**

CALL sp\_getPlayer('{"userId": 123}');

This example call retrieves information about the player with the user ID 123. The stored procedure checks if the user exists, and if so, returns a JSON object containing details about the player, including username, world ID, and available funds.

## sp\_getPlayerIndustries

**Overview**

The **sp\_getPlayerIndustries** stored procedure retrieves information about industries owned by a specific player in the Railway Project database. It takes a JSON object as input containing the user ID and returns details about the player's assets along with associated industries, including information about each industry's ID, name, warehouse capacity, type, produced good name, cost, and upkeep cost.

**Parameters**

* **inputJSON** (JSON): A JSON object containing the following properties:
  + **userId** (number, required): The unique identifier of the player for whom industries are to be retrieved.

**Example Usage**

CALL sp\_getPlayerIndustries('{"userId": 123}');

This example call retrieves information about industries owned by the player with the user ID 123. The stored procedure checks if the user exists, and if so, returns a JSON object containing details about the player's assets and associated industries. If no industries are found, the response code is set to 404 with an appropriate message.

## sp\_getPlayerNeeds

**Overview**

The **sp\_getPlayerNeeds** stored procedure retrieves information about the needs of assets owned by a specific player in the Railway Project database. It takes a JSON object as input containing the user ID and returns details about the player's town assets along with their associated needs, including information about each asset's ID, name, and the goods it consumes.

**Parameters**

* **inputJSON** (JSON): A JSON object containing the following properties:
  + **userId** (number, required): The unique identifier of the player for whom asset needs are to be retrieved.

**Example Usage**

CALL sp\_getPlayerNeeds('{"userId": 123}');

This example call retrieves information about the needs of town assets owned by the player with the user ID 123. The stored procedure checks if the user exists, and if so, returns a JSON object containing details about each town asset and its associated needs (consumed goods). If no needs are found, the response code is set to 404 with an appropriate message.

## sp\_getPlayerRailways

**Overview**

The **sp\_getPlayerRailways** stored procedure retrieves information about stations and connected railways owned by a specific player in the Railway Project database. It takes a JSON object as input containing the user ID and returns details about each station owned by the player, including information about connected railways and their connected stations.

**Parameters**

* **inputJSON** (JSON): A JSON object containing the following properties:
  + **userId** (number, required): The unique identifier of the player for whom railway information is to be retrieved.

**Example Usage**

CALL sp\_getPlayerRailways('{"userId": 123}');

This example call retrieves information about stations and connected railways owned by the player with the user ID 123. The stored procedure checks if the user exists, and if so, returns a JSON object containing details about each station, connected railways, and connected stations. If no stations are found, the response code is set to 404 with an appropriate message.

## sp\_getPlayers

**Overview**

The **sp\_getPlayers** stored procedure retrieves information about all players in the Railway Project database. It returns a JSON array containing details about each player, including user ID, username, and world ID.

**Parameters**

This stored procedure has no input parameters.

**Example Usage**

CALL sp\_getPlayers();

This example call retrieves information about all players in the database. The stored procedure returns a JSON object with a status code, a message indicating the success or failure of the operation, and an array of player data. If no players are found, the response code is set to 404 with an appropriate message.

## sp\_getPlayerStockpiles

**Overview**

The **sp\_getPlayerStockpiles** stored procedure retrieves information about the stockpiles of goods for a specific player in the Railway Project database. It takes a JSON input parameter containing the user ID (**userId**) and returns a JSON object with details about the assets owned by the player, including the stockpiles of goods within each asset.

**Parameters**

* **userId** (required): The user ID for which the stockpile information is to be retrieved.

**Example Usage**

CALL sp\_getPlayerStockpiles('{"userId": 123}');

This example call retrieves information about the stockpiles of goods for the player with the user ID 123. The stored procedure returns a JSON object with a status code, a message indicating the success or failure of the operation, and an array of asset data, each containing information about the asset's stockpiles. If no stockpiles are found, the response code is set to 404 with an appropriate message.

## sp\_getPlayersTrains

**Overview**

The **sp\_getPlayersTrains** stored procedure retrieves information about the trains owned by a specific player in the Railway Project database. It takes a JSON input parameter containing the user ID (**userId**) and returns a JSON object with details about each train owned by the player. The response includes information such as the train's ID, name, cost, operational cost, traveled distance, starting asset, destination asset, and the goods it transports.

**Parameters**

* **userId** (required): The user ID for which the train information is to be retrieved.

**Example Usage**

CALL sp\_getPlayersTrains('{"userId": 123}');

This example call retrieves information about the trains owned by the player with the user ID 123. The stored procedure returns a JSON object with a status code, a message indicating the success or failure of the operation, and an array of train data, each containing detailed information about a specific train. If no trains are found or the user does not own a town, the response code is set to 404 with an appropriate message.

## sp\_getSalt

**Overview**

The **sp\_getSalt** stored procedure retrieves the salt associated with a specific user's password in the Railway Project database. It takes a username (**p\_username**) as an input parameter and returns the corresponding salt value.

**Parameters**

* **p\_username** (required): The username for which the salt is to be retrieved.

**Output**

The stored procedure outputs the salt value associated with the provided username.

**Example Usage**

CALL sp\_getSalt('example\_user');

This example call retrieves the salt value for the user with the username 'example\_user'. The stored procedure returns the salt value as a result. If the username is not found, the result will be empty.

## sp\_Login

**Overview**

The **sp\_Login** stored procedure handles user login functionality in the Railway Project database. It takes a JSON input parameter containing the **username**, **password**, and **token**. The stored procedure verifies the provided credentials and returns a response based on the success or failure of the login attempt.

**Parameters**

* **json\_data** (required): JSON object with the following properties:
  + **username** (string): The username of the user attempting to log in.
  + **password** (string): The password provided by the user for authentication.
  + **token** (string): The authentication token associated with the user.

**Output**

The stored procedure returns a JSON object with the following structure:

* **status\_code** (integer): The HTTP status code indicating the result of the login attempt.
* **message** (string): A message providing additional details about the login result.
* **result** (object): An object containing user information and token if the login is successful.

**Example Usage**

CALL sp\_Login('{"username": "example\_user", "password": "example\_password", "token": "example\_token"}');

This example call attempts to log in a user with the specified username, password, and token. The stored procedure returns a JSON object with the login result. If the login is successful, the response includes user information and the provided token. If unsuccessful, it provides an appropriate error message and status code.

## sp\_Register

**Overview**

The **sp\_Register** stored procedure handles user registration functionality in the Railway Project database. It takes a JSON input parameter containing the **username**, **password**, and **salt** for the new user. The stored procedure registers the user, assigns the user to a world, and returns a response based on the success or failure of the registration.

**Parameters**

* **json\_data** (required): JSON object with the following properties:
  + **username** (string): The username of the new user.
  + **password** (string): The password for the new user.
  + **salt** (string): The salt used in password hashing for the new user.

**Example Usage**

CALL sp\_Register('{"username": "new\_user", "password": "new\_password", "salt": "new\_salt"}');

This example call attempts to register a new user with the specified username, password, and salt. The stored procedure performs the registration process and provides a response indicating the success or failure of the registration. If unsuccessful, it provides an appropriate error message and status code

## sp\_createRailway

**Overview**

The **sp\_createRailway** stored procedure handles the creation of a new railway between two stations in the Railway Project database. It takes a JSON input parameter containing the **station1Id**, **station2Id**, and **userId**. The stored procedure verifies the existence of the stations, checks the maximum limit of railways per station, calculates the distance between stations, creates a new railway, and connects it to the specified stations. It also deducts funds from the user's account for the railway creation.

**Parameters**

* **json\_data** (required): JSON object with the following properties:
  + **station1Id** (number): The ID of the first station.
  + **station2Id** (number): The ID of the second station.
  + **userId** (number): The ID of the user initiating the railway creation.

**Example Usage**

CALL sp\_createRailway('{"station1Id": 1, "station2Id": 2, "userId": 123}');

This example call attempts to create a new railway between the stations with IDs 1 and 2, initiated by the user with ID 123. The stored procedure performs the necessary validations, creates the railway, connects it to the stations, deducts funds, and provides a response indicating the success or failure of the railway creation.

## sp\_getRailway

**Overview**

The **sp\_getRailway** stored procedure retrieves information about a railway in the Railway Project database based on the provided railway ID. It takes a JSON input parameter containing the **id** property, representing the ID of the railway to be retrieved. The procedure checks the existence of the specified railway ID and returns the associated data if found.

**Parameters**

* **json\_data** (required): JSON object with the following properties:
  + **id** (number): The ID of the railway to be retrieved.

**Example Usage**

CALL sp\_getRailway('{"id": 123}');

This example call attempts to retrieve information about the railway with ID 123. The stored procedure checks the existence of the specified railway ID and returns the associated data if found.

## sp\_getRailways

**Overview**

The **sp\_getRailways** stored procedure retrieves information about all railways in the Railway Project database. It does not require any input parameters. The procedure returns a JSON array containing details about each railway, including its ID and distance.

**Parameters**

This stored procedure does not require any input parameters.

**Example Usage**

CALL sp\_getRailways();

This example call retrieves information about all railways in the database. The stored procedure returns a JSON array containing details about each railway, including its ID and distance.

## sp\_createStation

**Overview**

The **sp\_createStation** stored procedure creates a new station in the Railway Project database. It takes input parameters in the form of a JSON object, including the station name (**name**) and the asset ID (**assetId**) to which the station is associated. The procedure also deducts the cost of creating a station from the user's funds.

**Parameters**

* **json\_data** (JSON): A JSON object containing the following properties:
  + **name** (string): The name of the new station.
  + **assetId** (number): The ID of the asset (town) to which the station is associated.

**Example Usage**

CALL sp\_createStation('{"name": "New Station", "assetId": 123}');

This example call creates a new station named "New Station" associated with the asset (town) having the ID 123. The stored procedure returns a JSON object containing details about the created station if successful.

## sp\_getStation

**Overview**

The **sp\_getStation** stored procedure retrieves information about a specific station in the Railway Project database. It takes an input parameter in the form of a JSON object, which includes the **id** property representing the ID of the station to be retrieved.

**Parameters**

* **json\_data** (JSON): A JSON object containing the following properties:
  + **id** (number): The ID of the station to be retrieved.

**Example Usage**

CALL sp\_getStation('{"id": 123}');

This example call retrieves information about the station with the ID 123. The stored procedure returns a JSON object containing details about the station if it exists.

## sp\_getStationByName

**Overview**

The **sp\_getStationByName** stored procedure retrieves information about a station in the Railway Project database based on its name. It takes an input parameter in the form of a JSON object, which includes the **station\_name** property representing the name of the station to be retrieved.

**Parameters**

* **json\_data** (JSON): A JSON object containing the following properties:
  + **station\_name** (string): The name of the station to be retrieved.

**Example Usage**

CALL sp\_getStationByName('{"station\_name": "Central Station"}');

This example call retrieves information about the station named "Central Station." The stored procedure returns a JSON object containing details about the station if it exists.

## sp\_getStations

**Overview**

The **sp\_getStations** stored procedure retrieves information about all stations in the Railway Project database. It returns a JSON array containing details about each station, including its ID, name, cost, operation cost, and associated asset ID.

**Parameters**

This stored procedure does not require any parameters.

**Example Usage**

CALL sp\_getStations();

This example call retrieves information about all stations in the database. The stored procedure returns a JSON array containing details about each station.

## sp\_payStations

**Overview**

The **sp\_payStations** stored procedure iterates through all players in the Railway Project database and charges them based on the number of stations they own. For each player, the procedure calculates the total cost of all stations associated with their owned assets (towns) and deducts the corresponding funds.

**Parameters**

This stored procedure does not require any parameters.

**Example Usage**

CALL sp\_payStations();

This example call executes the stored procedure, which calculates the total cost of all stations owned by each player and deducts the corresponding funds.

## sp\_createTrain

**Overview**

The **sp\_createTrain** stored procedure is designed to create a new train in the Railway Project database. It also randomly assigns loads to the train's wagons based on the needs of the destination town. The procedure deducts the corresponding funds from the player's account.

**Parameters**

* **name** (VARCHAR): The name of the train.
* **idRailway** (INT): The ID of the railway to which the train belongs.
* **idAsset\_Starts** (INT): The ID of the asset (town) where the train starts its journey.
* **idAsset\_Destines** (INT): The ID of the asset (town) where the train's journey ends.
* **willReturnWithGoods** (BOOLEAN): A flag indicating whether the train will return with goods.

**Example Usage**

CALL sp\_createTrain('Express 001', 1, 100, 200, true);

This example call creates a new train named 'Express 001' on the railway with ID 1, starting from the town with ID 100 and ending at the town with ID 200. The train will return with goods.

## sp\_deleteTrain

**Overview**

The **sp\_deleteTrain** stored procedure is designed to delete a train and its associated wagons from the Railway Project database. The procedure also refunds the player with funds upon successful deletion.

**Parameters**

* **userId** (INT): The ID of the player who owns the train.
* **trainId** (INT): The ID of the train to be deleted.

**Example Usage**

CALL sp\_deleteTrain('userId': 123, 'trainId': 456);

This example call deletes the train with ID 456 owned by the player with ID 123.

## sp\_demandTrain

**Overview**

The **sp\_demandTrain** stored procedure is designed to create a new train and wagons for transporting goods from one town to another on a specified railway. This procedure simulates the demand for goods in a town and the transportation of those goods via a train.

**Parameters**

* **assetFromId** (INT): The ID of the town where the goods originate.
* **assetToId** (INT): The ID of the town where the goods are to be delivered.
* **railwayId** (INT): The ID of the railway connecting the two towns.
* **goodId** (INT): The ID of the type of goods to be transported.
* **amount** (INT): The quantity of goods to be transported.

**Example Usage**

CALL sp\_demandTrain('assetFromId': 1, 'assetToId': 2, 'railwayId': 3, 'goodId': 4, 'amount': 100);

This example call creates a new train with wagons to transport 100 units of goods from town 1 to town 2 on railway 3.

## sp\_fillTrain

**Overview**

The **sp\_fillTrain** stored procedure is designed to simulate the loading of wagons for a given train with goods and passengers. It randomly selects goods from the destination town's needs or production and assigns them to the wagons of the specified train. Additionally, it randomly loads mail or passengers to fill any remaining empty wagons.

**Parameters**

* **input\_idAsset\_Starts** (INT): The ID of the town where the train starts its journey.
* **input\_idAsset\_Destines** (INT): The ID of the town where the train is destined to arrive.
* **input\_train\_id** (INT): The ID of the train for which wagons are being filled.

**Example Usage**

CALL sp\_fillTrain(1, 2, 3);

This example call fills the wagons of the train with ID 3, simulating the loading of goods and passengers for a journey from town 1 to town 2.

## sp\_getTrain

**Overview**

The **sp\_getTrain** stored procedure retrieves information about a train based on the provided train ID. It checks the validity of the JSON input, verifies the existence of the specified train ID, and returns details about the train if it exists.

**Parameters:**

* **Id** (INT): The ID of the train to be retrieved.

**Example Usage**

sp\_getTrain('{"id": 1}');

This example call retrieves information about the train with ID 1.

## sp\_getTrains

**Overview**

The **sp\_getTrains** stored procedure retrieves information about all trains in the system. It returns a list of trains, including details such as train ID, name, cost, operational cost, and associated railway ID.

**Parameters**

* None

**Example Usage**

CALL sp\_getTrains();

This example call retrieves information about all trains in the system.

## sp\_moveTrain

**Overview**

The **sp\_moveTrain** stored procedure simulates the movement of trains within the railway system. It updates the traveled distance of each train and triggers the arrival event when a train reaches its destination or returns to its starting point.

**Parameters**

* None

**Example Usage**

CALL sp\_moveTrain();

This example call simulates the movement of trains within the system.

## sp\_trainArrived

**Overview**

The **sp\_trainArrived** stored procedure handles the arrival event for trains in the railway system. It calculates and rewards the owner based on the goods transported by the train and updates the stockpile at the destination asset. Additionally, if the train is configured to return with goods, it triggers the refilling of the train for the return journey.

**Parameters**

* **trainId** (IN): The unique identifier of the train that has arrived.
* **ownerID** (IN): The owner of the train.
* **days\_needed** (IN): The number of days the train took to travel.
* **receiverID** (IN): The unique identifier of the asset where the train has arrived.

**Example Usage**

CALL sp\_trainArrived(1, 101, 5, 102);

This example call simulates the arrival of a train with the specified parameters

## sp\_getWorld

**Overview**

The **sp\_getWorld** stored procedure retrieves information about a specific world in the railway system based on the provided world ID. It returns details such as the world's unique identifier (**id**) and the date of its creation (**creationDate**).

**Parameters:**

* **Id** (INT): The ID of the world to be retrieved.

**Example Usage**

CALL sp\_getWorld('{"id": 1}');

This example call retrieves information about the world with an ID of 1.

## sp\_getWorldAssets

**Overview**

The **sp\_getWorldAssets** stored procedure retrieves information about assets within a specific world in the railway system based on the provided world ID. It returns details such as asset identifiers (**idAsset\_PK**), asset names, types, populations, levels, maximum stockpile capacities, positions, owner IDs, costs, and associated goods (for rural businesses).

**Parameters:**

* **worldId** (INT): The ID of the world of which assets to be retrieved.

{ "worldId": 1 }

**Example Usage**

CALL sp\_getWorldAssets('{"worldId": 1}');

This example call retrieves information about assets within the world with an ID of 1.

## sp\_getWorldAssets

**Overview**

The **sp\_getWorldAssets** stored procedure retrieves information about all worlds in the railway system. It returns details such as world identifiers (**idWorld\_PK**) and creation dates.

**Parameters**

* **inputJson** (IN): This parameter is not utilized within the stored procedure.

**Example Usage**

CALL sp\_getWorldAssets('{}');

This example call retrieves information about all worlds in the system.

## sp\_addFunds

**Overview**

The **sp\_addFunds** stored procedure is responsible for increasing the funds of a specific player in the railway system. It takes two input parameters: the **amount** to be added to the player's funds and the **playerID** identifying the player.

**Parameters**

* **amount** (IN): An integer representing the funds to be added to the player's account.
* **playerID** (IN): An integer representing the unique identifier of the player whose funds will be increased.

**Example Usage**

CALL sp\_addFunds(5000, 123);

This example call adds 5000 units of currency to the funds of the player with the ID 123.

## sp\_checkTokenExists

**Overview**

The **sp\_checkTokenExists** stored procedure checks whether a token of a specified type exists in the system. It takes one input parameter, **p\_token**, which is the token type to be checked.

**Parameters**

* **p\_token** (IN): A string representing the type of the token to be checked for existence.

**Example Usage**

CALL sp\_checkTokenExists('authentication');

This example call checks if a token of type 'authentication' exists in the system.

## sp\_deleteToken

**Overview**

The **sp\_deleteToken** stored procedure deletes a token from the system based on its type. It takes one input parameter, **token\_value**, which is the type of the token to be deleted.

**Parameters**

* **token\_value** (IN): A string representing the type of the token to be deleted.

**Example Usage**

CALL sp\_deleteToken('authentication');

This example call deletes a token of type 'authentication' from the system.

## sp\_deleteFunds

**Overview**

The **sp\_deleteFunds** stored procedure deducts a specified amount from the funds of a player identified by their **playerID**. It takes two input parameters - **amount** (the amount to be deducted) and **playerID** (the unique identifier of the player).

**Parameters**

* **amount** (IN): An integer representing the amount to be deducted from the player's funds.
* **playerID** (IN): An integer representing the unique identifier of the player.

**Example Usage**

CALL sp\_deleteFunds(500, 123);

This example call deducts 500 units of currency from the player with the unique identifier 123.

## sp\_updateAssetNeeds

**Overview**

The **sp\_updateAssetNeeds** stored procedure updates the population of an asset based on the satisfaction of its needs. It calculates the satisfaction percentage by comparing the available stockpile quantities with the consumption rates of required goods. Depending on the satisfaction percentage, it adjusts the population of the asset.

**Parameters**

* **assetId** (IN): An integer representing the unique identifier of the asset for which the needs are to be updated.

**Example Usage**

CALL sp\_updateAssetNeeds(1001);

This example call updates the needs and population of the asset with the unique identifier 1001.

## sp\_updateStockpile

**Overview**

The **sp\_updateStockpile** stored procedure iterates through all town assets (assets with the type 'TOWN') and calls the **sp\_updateAssetNeeds** procedure for each town. This procedure, in turn, updates the needs and population of each town based on the available stockpiles.

**Example Usage**

CALL sp\_updateStockpile();

This example call updates the stockpiles for all town assets by iterating through each town and calling the **sp\_updateAssetNeeds** procedure.