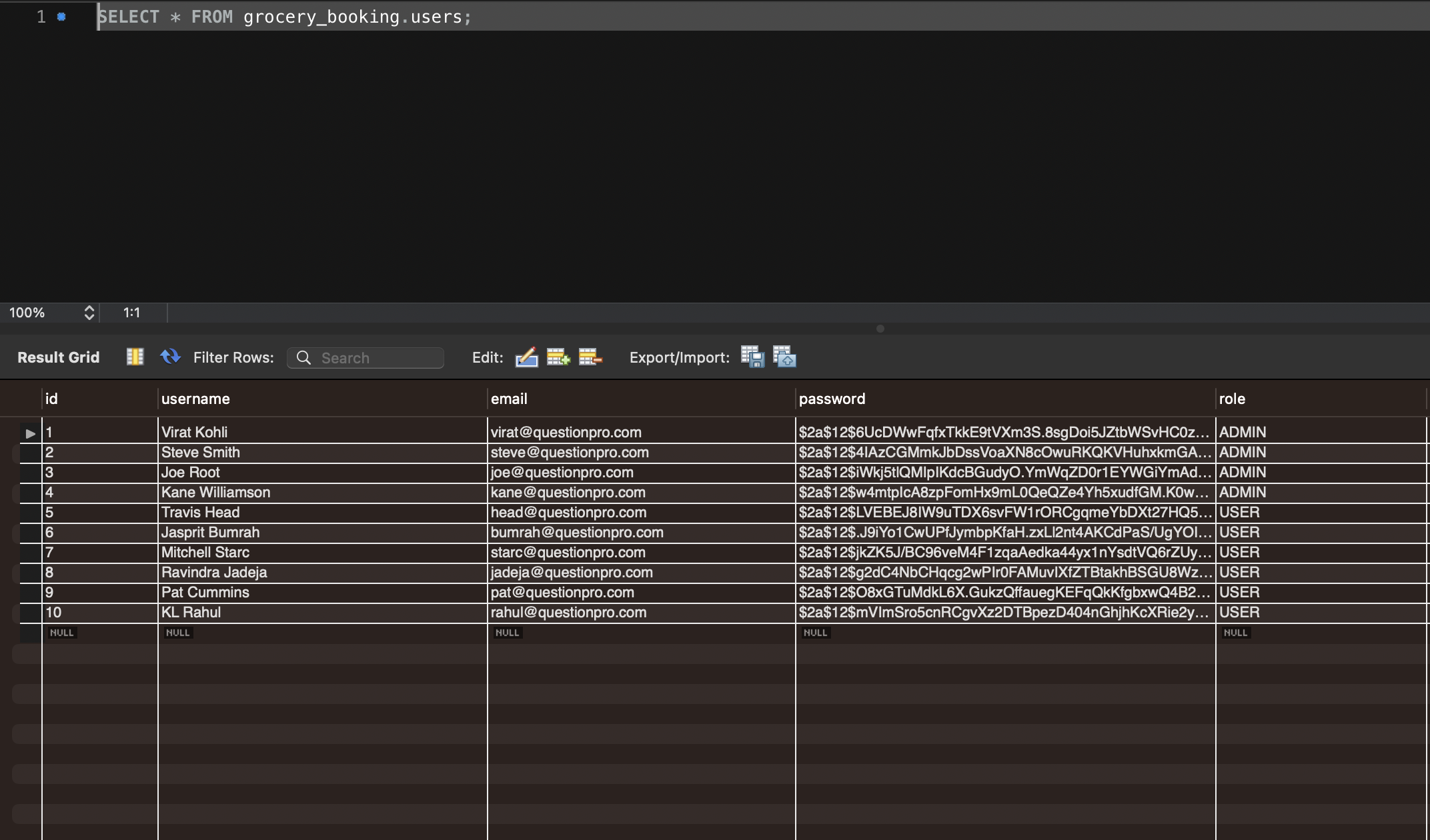
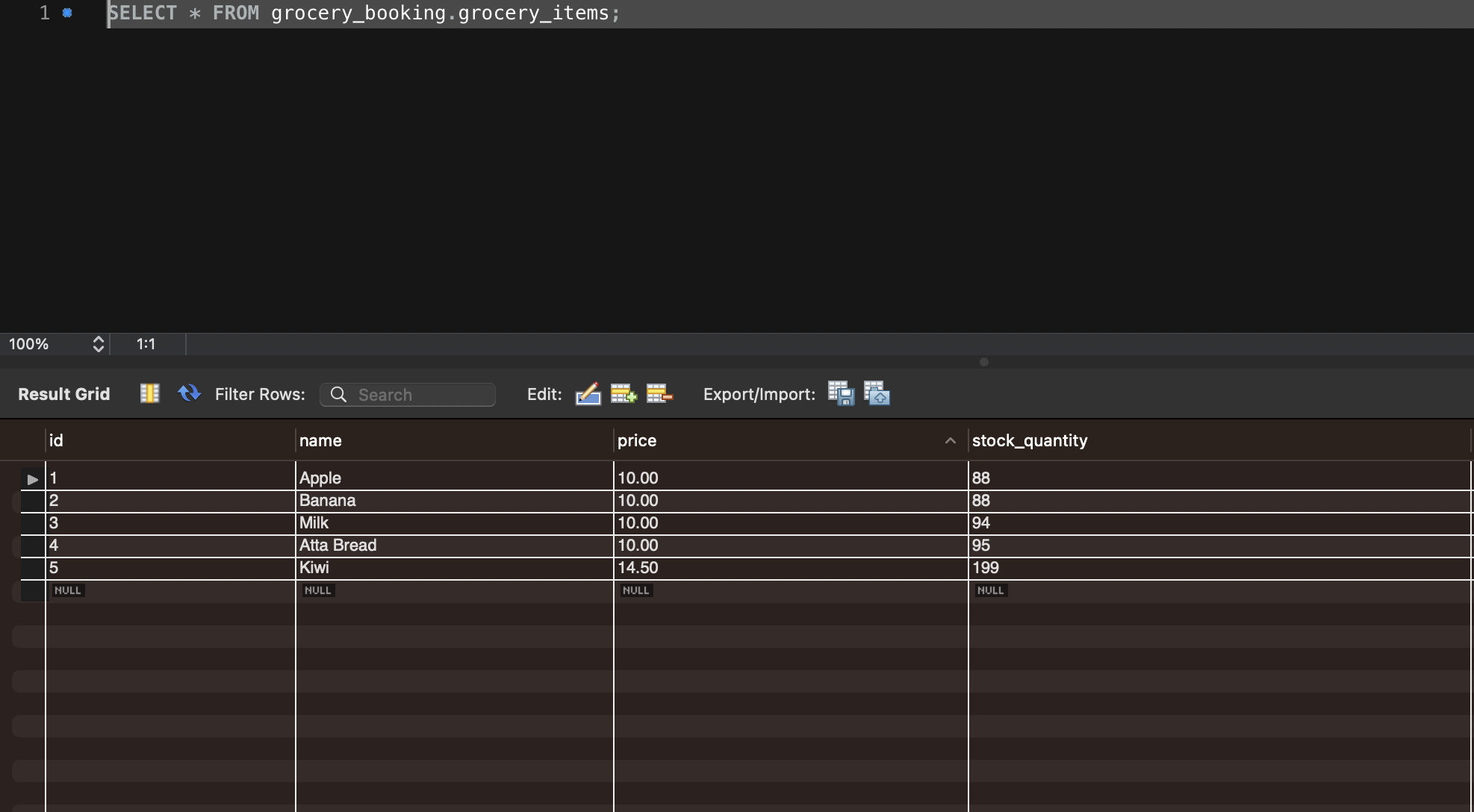
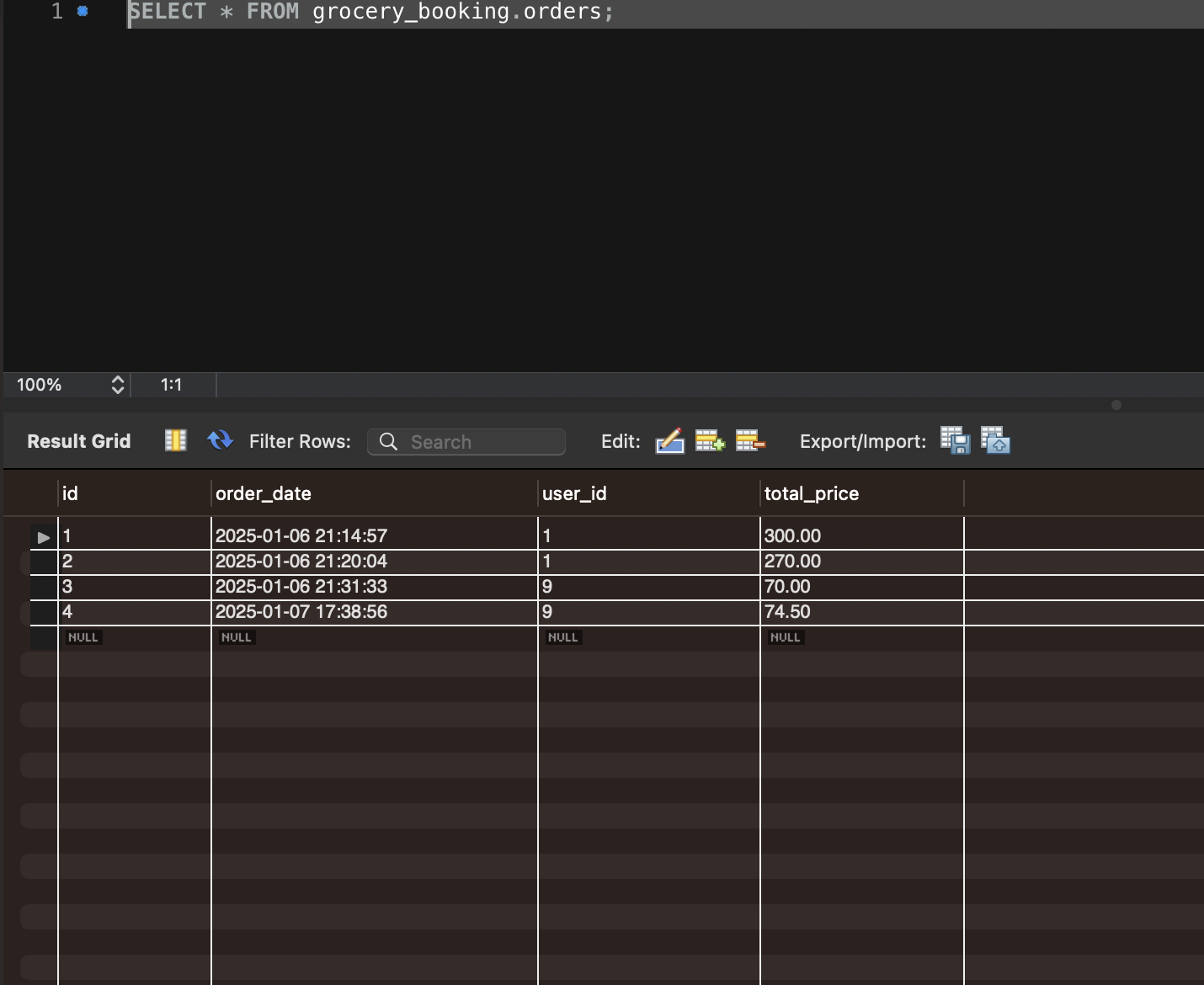
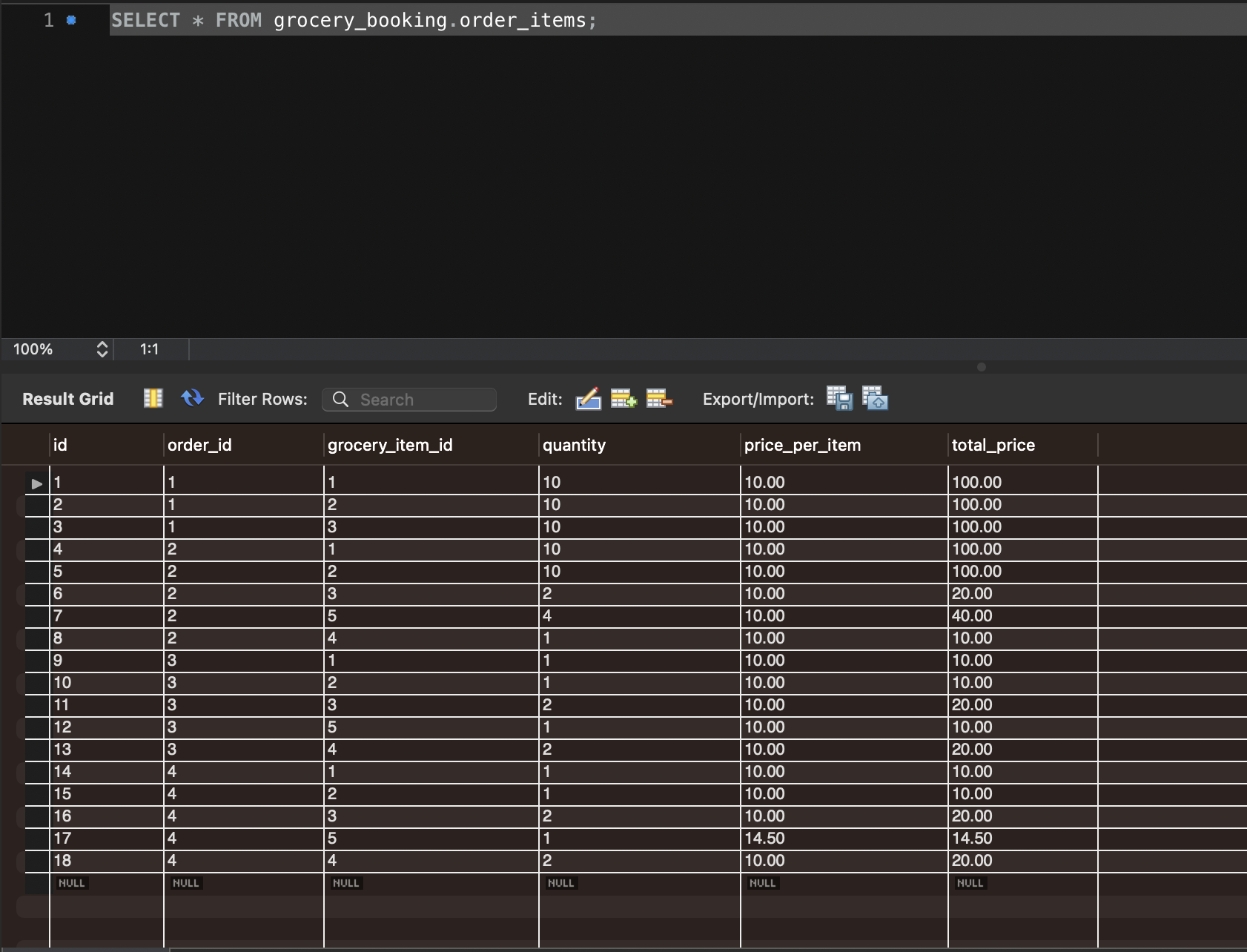
**Database Design**

1. **Tables**
   1. **USERS** | Will maintain USER or ADMIN in single Table
      1. Single Table, Why?
         1. Reduces redundancy as ADMIN or USER will have common attributes.
         2. Can distinguish based on the ROLE attribute
         3. Implement Authentication & Authorization
         4. Hence, with all necessary attributes ROLES field will be added in the USERS Table
   2. **GROCERY\_ITEMS** | To store Grocery items
   3. **ORDERS** | Order Details
   4. **ORDER\_ITEMS** | Details of Items in each Order
2. **Entities**
   1. **USERS  
      [id,username,email,password,role]  
        
      **

* 1. **GROCERY\_ITEMS   
     [id,name,price,stock\_quantity]  
       
       
     **
  2. **ORDERS  
     [id,order\_date,user\_id,total\_price]  
       
     **
  3. **ORDER\_ITEMS [id,order\_id,grocery\_item\_id,quantity,price\_per\_item,total\_price]  
       
     **

1. **Relationships**
   1. A **USER(s)** can have multiple **ORDERS**
   2. **ORDERS** can contain multiple **ORDER\_ITEMS**
   3. **ORDER\_ITEMS** can have single/same **GROCERY\_ITEMS**

**Project Structure**

1. **API Documentation for Grocery App**

**AdminController**

Base URL: /admin

**1. Add Grocery Item**

**POST** /grocery-items

**Description:** Adds a new grocery item to the inventory.

**Request Body:**

{

"name": "string",

"price": "number",

"quantity": "number"

}

**Response:**

Grocery Item added successfully

**Response Status:** 201 Created

**2. Get All Grocery Items**

**GET** /grocery-items

**Description:** Retrieves a list of all grocery items.

**Response:**

[

{

"id": "number",

"name": "string",

"price": "number",

"quantity": "number"

}

]

**Response Status:** 200 OK

**3. Delete Grocery Item**

**DELETE** /grocery-items/{id}

**Description:** Deletes a grocery item by its ID.

**Path Parameter:**

* id: The ID of the grocery item to delete.

**Response:**

Grocery Item deleted successfully

**Response Status:** 200 OK

**4. Update Grocery Item**

**PUT** /grocery-items/{id}

**Description:** Updates the details of a grocery item.

**Path Parameter:**

* id: The ID of the grocery item to update.

**Request Body:**

{

"name": "string",

"price": "number",

"quantity": "number"

}

**Response:**

Grocery Item updated successfully

**Response Status:** 200 OK

**5. Update Grocery Stock**

**PATCH** /grocery-items/{id}/stock

**Description:** Updates the stock quantity of a grocery item.

**Path Parameter:**

* id: The ID of the grocery item to update.

**Request Body:**

{

"stockQuantity": "number"

}

**Response:**

Stock updated successfully

**Response Status:** 200 OK

**OpenEndPointsController**

**1. Register User**

**POST** /register

**Description:** Registers a new user.

**Request Body:**

{

"username": "string",

"password": "string"

}

**Response:**

{

"id": "number",

"username": "string"

}

**Response Status:** 201 Created

**2. User Login**

**POST** /login

**Description:** Logs in an existing user.

**Request Body:**

{

"username": "string",

"password": "string"

}

**Response:**

Login successful  
  
**Note**: ***This generates a Token which can be pass in the AUTHORIZATION HEADER for subsequent requests.***

**Response Status:** 200 OK

**UserController**

Base URL: /users

**1. Get All Grocery Items**

**GET** /grocery-items

**Description:** Retrieves a list of all grocery items available to users.

**Response:**

[

{

"id": "number",

"name": "string",

"price": "number",

"quantity": "number"

}

]

**Response Status:** 200 OK

**2. Place Order**

**POST** /orders

**Description:** Places a new order for grocery items.

**Request Parameters:**

* userId: The ID of the user placing the order.

**Request Body:**

[

{

"itemId": "number",

"quantity": "number"

}

]

**Response:**

Order placed successfully

**Response Status:** 201 Created

1. **JWT Config**

**Security Configuration Documentation for Grocery App**

**SecurityConfig**

**Overview:**

The SecurityConfig class is responsible for setting up the security configurations of the Grocery App using Spring Security. It defines the security filter chain, authentication provider, and authentication manager.

**1. Security Filter Chain**

**Method:** securityFilterChain(HttpSecurity http)

**Description:** Configures HTTP security settings, such as CSRF protection, authorization rules, session management, and JWT filter integration.

**Configuration Details:**

* **CSRF:** Disabled.
* **Authorization Rules:**
  + /register and /login are publicly accessible.
  + All other endpoints require authentication.
* **Session Management:** Stateless session policy is used.
* **JWT Filter:** The JwtFilter is added before UsernamePasswordAuthenticationFilter.

**2. Authentication Provider**

**Method:** authenticationProvider()

**Description:** Configures the DaoAuthenticationProvider to use a custom UserDetailsService and password encoder.

**Password Encoder:** BCryptPasswordEncoder with a strength of 12.

**3. Authentication Manager**

**Method:** authenticationManager(AuthenticationConfiguration authenticationConfiguration)

**Description:** Provides the AuthenticationManager to handle authentication requests.

**JwtFilter**

**Overview:**

The JwtFilter class extends OncePerRequestFilter and is responsible for intercepting incoming requests to validate JWT tokens.

**doFilterInternal() Method:**

**Description:** This method intercepts each request to:

1. Extract the JWT token from the Authorization header.
2. Validate the token.
3. Set the authentication context if the token is valid.

**Steps:**

* Extract the Authorization header.
* Check if the header starts with "Bearer ".
* Extract the username from the token using JWTService.
* Load user details and validate the token.
* Set the authentication context using SecurityContextHolder.

**JWTService**

**Overview:**

The JWTService class handles the generation, extraction, and validation of JWT tokens.

**1. Token Generation**

**Method:** generateToken(String username)

**Description:** Generates a JWT token with the given username and a 1-hour expiration time.

**Note**: ***This generates a Token which can be pass in the AUTHORIZATION HEADER for subsequent requests.***

**2. Extract Username**

**Method:** extractUsername(String token)

**Description:** Extracts the username from the provided JWT token.

**3. Token Validation**

**Method:** validateToken(String token, UserDetails userDetails)

**Description:** Validates if the provided JWT token is valid and matches the user details.

**4. Check Token Expiration**

**Method:** isTokenExpired(String token)

**Description:** Checks if the JWT token has expired.