Fetch:
Relational Diagram:
The ER (Entity-Relationship) has the relationships between the entities: `USERS`, `RECEIPTS`, and `BRANDS`, 'CPG_BRANDS' along with a linking table `ITEMS`.

Here's an explanation of the mappings:

1. `USERS` to `RECEIPTS`:
   - This is a one-to-many relationship. This means that one user can have many receipts, but each receipt is associated with only one user. The `user_id` field in `RECEIPTS` is a foreign key (FK) that references the primary key (PK) `user_id` in the `USERS` table.

2. `RECEIPTS` to 'ITEMS`:
   - This is also a one-to-many relationship, where one receipt can contain many items, but each item is associated with only one receipt. The `receipt_id` field in `ITEMS` is a foreign key that references the primary key `receipt_id` in the `RECEIPTS` table.

3. `BRANDS` to `ITEMS`:
   - This is a many-to-many relationship. It indicates that brands can have many items on different receipts, and each receipt item can be associated with different brands. The relationship is established through the `barcode` field, which is used as a foreign key in `ITEMS` to reference the `barcode` in `BRANDS`.

4. 'CPG_BRANDS' to 'BRANDS':
  -One-to-many relationship , where a CPG (Consumer Packaged Goods) brand can have many individual brands associated with it. The cpg_id in Brands serves as an 'FK' back to CPG_BRANDS

5. 'BRANDS' to 'RECEIPTS':
   - This is a many-to-many relationship. The relationship is established through 'name' field in 'RECEIPTS' to reference 'name' in 'BRANDS'

6. 'USERS' to 'BRANDS':
   - This is a many-to-many relationship. The relationship is established through 'user_id' field, which is used as a foreign key in 'BRANDS'.

Given the relational diagram and the mappings between `USERS`, `RECEIPTS`, `BRANDS`, `CPG_BRANDS`, and the linking table `ITEMS`, the creation of the `ITEMS` table plays a crucial role in data normalization, especially due to the existence of the nested field `rewardsReceiptItemList` within `RECEIPTS`. Here are some points highlighting the importance and necessity of creating the `ITEMS` table for data normalization:

a. Elimination of Nested Fields: The `rewardsReceiptItemList` nested field within `RECEIPTS` can lead to complex and inefficient data structures. By transitioning this data into a separate `ITEMS` table, we significantly simplify the structure, making it easier to query, update, and maintain.

b. Improved Data Integrity: A separate `ITEMS` table allows for the enforcement of data integrity through the use of foreign keys. This ensures that items are always correctly associated with their corresponding receipts and brands, reducing the potential for data anomalies.

c. Enhanced Query Performance: Queries targeting specific items within receipts become more straightforward and efficient with a normalized `ITEMS` table. This structure eliminates the need for complex joins or nested queries to access item-level details, thereby improving overall database performance.

d. Scalability: As the volume of data grows, particularly with the addition of new items to receipts, a normalized `ITEMS` table ensures that the database can scale more effectively. It allows for easier addition of new items without affecting the performance of queries targeting the `RECEIPTS` table directly.

e. Flexibility for Analytics: With items normalized into their own table, it becomes easier to perform detailed analytics on item-level data, such as identifying trends in item purchases, analyzing item-level profitability, or understanding brand-item associations.

f. Simplification of Many-to-Many Relationships: The `ITEMS` table serves as a crucial linking mechanism in the many-to-many relationships between `BRANDS` and `RECEIPTS`, and potentially other many-to-many relationships. This simplifies the data model and makes it more manageable.

In conclusion, the creation of the `ITEMS` table not only addresses the challenges posed by the `rewardsReceiptItemList` nested field but also significantly enhances the overall quality and manageability of the database. This strategic move towards normalization supports efficient data management practices, catering to both current and future data needs.