**Complete Entity Relationship Diagram (ERD)**

This diagram shows all the tables and the relationships between them as defined in your code.

Code snippet

erDiagram

ApplicationUser ||--o{ HomeMember : "is member of"

ApplicationUser ||--o{ Contribution : "paid for"

ApplicationUser ||--o{ ContributionSplit : "owes"

Home ||--|{ HomeMember : "has"

Home ||--|{ Contribution : "has"

Home ||--|{ ShoppingListItem : "has"

Contribution ||--|{ ContributionSplit : "is split into"

Contribution ||--o{ ShoppingListItem : "purchased"

ApplicationUser {

string Id PK

string FullName

string Email

}

Home {

int Id PK

string Name

string InviteCode

}

HomeMember {

string UserId PK, FK

int HomeId PK, FK

string Role

}

Contribution {

int Id PK

int HomeId FK

string PaidById FK

string Description

decimal TotalAmount

}

ContributionSplit {

int ContributionId PK, FK

string UserId PK, FK

decimal AmountOwed

}

ShoppingListItem {

int Id PK

int HomeId FK

int ContributionId FK "nullable"

string Name

string Status

}

**How the Entities Interact: All Possible Flows**

Here’s how your data model handles every key user action, governed by the rules in your configuration files.

**1. User & Household Management Flows**

* **User Registration**: A new record is created in the AspNetUsers table (managed by Identity).
* **Creating a Home**:
  1. A new record is created in the **Homes** table with a unique InviteCode.
  2. A new record is created in the **HomeMembers** table, linking the creator's UserId to the new HomeId and setting their Role to "Admin".
* **Joining a Home**:
  1. The user provides an InviteCode. The system finds the corresponding Home.
  2. A new record is created in the **HomeMembers** table, linking the new user's UserId to the HomeId with the Role of "Member".
* **Removing a Member from a Home**:
  1. An Admin initiates the removal of another user from a Home.
  2. The system finds and deletes the corresponding record from the **HomeMembers** table.
  3. **Cascade Delete**: Because of the OnDelete(DeleteBehavior.Cascade) rule in ApplicationUserConfiguration and HomeConfiguration, deleting this HomeMember record also automatically deletes all **ContributionSplit** records associated with that user *for that specific home*, ensuring they are no longer part of future financial calculations.

**2. Core Financial Loop: Logging an Expense**

This is the most critical flow.

1. **Create Contribution**: A user logs a purchase (e.g., $50 for groceries). This creates **one** new record in the **Contribution** table, linking to the HomeId and the PaidById (the user who paid).
2. **Create Splits**: The system determines who the expense was for (e.g., 2 people). It creates **two** new records in the **ContributionSplit** table.
   * Each record is linked to the parent ContributionId.
   * Each record is linked to a UserId (one for each person involved).
   * The AmountOwed is set for each person (e.g., $25).
3. **Database Transaction**: These two operations (creating the contribution and its splits) must happen together in a single database transaction to ensure data consistency. If creating the splits fails, the contribution should not be saved.

**3. Shopping List Workflow**

1. **Add Item**: A user adds "Milk" to the list. This creates a new record in the **ShoppingListItem** table. The HomeId is set, but the ContributionId is null.
2. **Purchase Items**: A user buys "Milk" and other items in a single trip, creating a Contribution and its Splits as described above.
3. **Link Items**: As a final step, the system updates the "Milk" record in the **ShoppingListItem** table, setting its ContributionId to the ID of the newly created Contribution. This links the item to the financial transaction.

**4. Data Deletion & Integrity Flows**

The OnDelete rules you defined are critical for maintaining data integrity.

* **Deleting a Home**: This triggers a cascade delete. All associated **HomeMembers**, **Contributions**, and **ShoppingListItems** are automatically deleted from the database. This is a powerful but destructive action.
* **Deleting a Contribution**:
  1. The Contribution record is deleted.
  2. **Cascade Delete**: All associated **ContributionSplit** records are automatically deleted.
  3. **Set Null**: The OnDelete(DeleteBehavior.SetNull) rule on the PurchasedItems relationship means that any **ShoppingListItem** linked to this contribution is *not* deleted. Instead, its ContributionId is simply set back to null. This correctly un-links the item from the purchase without removing it from history.
* **Deleting a User**:
  1. A request is made to delete an ApplicationUser.
  2. **Restrict Rule**: The OnDelete(DeleteBehavior.Restrict) rule in ApplicationUserConfiguration is triggered. The database checks if this user's ID exists in the PaidById column of the Contributions table.
  3. **If they have paid for contributions**: The database will **reject and block** the deletion to prevent breaking the financial records. You must first reassign or handle their contributions before the user can be deleted.
  4. **If they have NOT paid for any contributions**: The deletion proceeds. The cascade delete rule will then automatically remove their associated HomeMemberships and OwedSplits.

**5. Data Calculation Flow (Read-Only)**

* **Calculating Balances**: To show "who owes who," the system performs a read-only calculation:
  1. For a given user, it sums the TotalAmount from all **Contributions** where PaidById matches their UserId.
  2. It then sums the AmountOwed from all **ContributionSplits** where UserId matches their UserId.
  3. The final balance is (Total Paid) - (Total Owed). This flow touches three tables (ApplicationUser, Contribution, ContributionSplit) but modifies no data.