trend\_category: This is a bridge table connecting trends and categories. You should consider creating a DAO and service methods to manage these relationships. You might have methods like:

associateTrendWithCategory(trendId, categoryId)

dissociateTrendFromCategory(trendId, categoryId)

getTrendsForCategory(categoryId)

getCategoriesForTrend(trendId)

The last two methods are already implemented in your CategoryDao. You might need similar ones in your TrendDao.

designer\_product: Another bridge table. This time between designers and products. Following a similar pattern:

associateDesignerWithProduct(designerId, productId)

dissociateDesignerFromProduct(designerId, productId)

getProductsForDesigner(designerId)

getDesignersForProduct(productId)

category\_popularity: If you intend to track the popularity of categories per season, you'd need methods like:

setCategoryPopularityForSeason(categoryId, season, score)

getCategoryPopularityForSeason(categoryId, season)

getAllCategoryPopularities(categoryId)

trend\_interactions: For tracking user interactions with trends. Some potential methods:

recordInteraction(userId, trendId, interactionType)

getUserInteractions(userId)

getTrendInteractions(trendId)

product\_popularity: For tracking product popularity in relation to trends.

setProductPopularityForTrend(productId, trendId, score)

getProductPopularityForTrend(productId, trendId)

getAllProductPopularities(productId)

trend\_popularity: Tracking the overall popularity of a trend.

setTrendPopularity(trendId, score)

getTrendPopularity(trendId)

You'll need respective DAO, Service, and potentially Controller layers for each of these entities or relationships, depending on your application's needs.

Moreover, consider the following:

DTOs: As discussed earlier, for API interactions, you might want to have DTOs (Data Transfer Objects) to decouple your database entities from the objects you send and receive over the API.

Validation: Add validation checks, especially for methods that associate or dissociate entities (to make sure the entities exist before the operation).

Logging: Add logging for crucial operations to help with debugging and monitoring.

Tests: Extend your tests to cover these new DAO and Service methods.

Transaction Management: If an operation involves multiple database updates, consider wrapping them in a transaction to ensure data integrity.