Ria Desai, Julie Lima, Nitya Sunkad, Ojaswee Chaudhary

Project Track 1 Stage 2

Writeup and Logical Schema

Logical (Relational) Schema

Users
(UserId: INT NOT NULL [**PK**],
Username: VARCHAR(30) NOT NULL,
Password: VARCHAR(30) NOT NULL,
FirstName: VARCHAR(255) NOT NULL,
LastName: VARCHAR(255) NOT NULL,
Email: VARCHAR(255) NOT NULL
)

Products
(ProductId: INT NOT NULL [**PK**],
ProductName: VARCHAR(255) NOT NULL,
ProductURL: VARCHAR(255),
Price: REAL NOT NULL,
LikeCount: INT,
BrandId: INT NOT NULL [**FK** to Brands.BrandId]
)

Reviews
(ReviewId: INT NOT NULL [**PK**],
Rating: REAL,
Text : VARCHAR(1000),
ProductId: INT NOT NULL [**FK** to Products.ProductId]
)

Brands
(BrandId: INT NOT NULL [**PK**],
BrandName: VARCHAR(30) NOT NULL,
BrandRating: REAL
)

Tags
(TagId: INT NOT NULL [**PK**],
TagName: VARCHAR(64) NOT NULL,

```
Standing: INT NOT NULL
)

BagItems
(UserId: INT NOT NULL [PK, FK to Users.UserId],
ProductId: INT NOT NULL [PK, FK to Products.ProductId],
DateAdded: DATE
)

ProductTags
(ProductId: INT NOT NULL [PK, FK to Products.ProductId],
TagId: INT NOT NULL [PK, FK to ProductTags.TagId]
)
```

Assumptions

Explain your assumptions for each entity and relationship in your model. Discuss why you've modeled something as an entity rather than an attribute of another entity. Describe the cardinality of relationships, like why a student is linked to only one advisor. These assumptions might come from customer requirements or application constraints. Please clarify them.

Users (entity):

- Assumptions: Each user represents a unique individual using the system, identified by a UserId.
- Justification: Users encompass various attributes such as username, password, first name, last name, and email, which collectively define a distinct entity in the system. These attributes are essential for user authentication, identification, and personalization.

Products (entity):

- Assumptions: Each product represents a unique beauty item in the system, identified by a ProductId.
- Justification: Products have various attributes like name, URL, price, like count, and brand, which collectively define a distinct entity. Modeling products as an entity allows for efficient management, tracking, and manipulation of product-related data.
- Cardinality: Products has a many-to-one relationship with Brands (since a specific product can only be produced by one brand) and a one-to-many relationship with Reviews (since a specific product can have many different reviews written about it).

BagItems (relationship):

- Assumptions: BagItems represents the many-to-many relationship between Users and Products, modeled as a table with a foreign key to both UserId and ProductId. Each row in this table represents a specific product that is in a specific user's makeup bag, along with an attribute for the date the product was added to the makeup bag.
- Justification: There needs to be a many-to-many relationship between Users and Products because each user can have various different products in their makeup bag, and each product can be in several different users' makeup bags.

Tags (entity):

- Assumptions: Each tag represents a unique descriptive label or category assigned to products.
- Justification: Tags have attributes such as name and standing (the "weight" assigned to the tag), and they are not inherently an attribute of a product. Modeling tags as an entity allows them to be used for organizing and categorizing products based on common attributes or characteristics.

ProductTags (relationship):

- Assumptions: ProductTags represents the many-to-many relationship between Products and Tags, modeled as a table with a foreign key to both ProductId and TagId. Each row in this table represents a specific tag that is assigned to a specific product.
- Justification: There needs to be a many-to-many relationship between Products and Tags because each product can have various different tags, and each tag can be assigned to several products.

Reviews (entity):

- Assumptions: Each review represents a unique user-generated feedback and/or rating for a specific product.
- Justification: Reviews possess attributes such as rating and text, which are unique to each review and are not attributes of either users or products alone. Modeling reviews as a separate entity allows for easier tracking and querying of review data.
- Cardinality: Reviews has a many-to-one relationship with Products (since a specific review can only be referring to a single product).

Brands (entity):

- Assumptions: Each brand represents a unique company producing beauty products.
- Justification: Brands have attributes like name and rating, which are distinct from product attributes. Modeling brands as an entity allows for a clear relationship between various different products that are created by the same brand, while keeping brand-specific attributes abstracted.
- Cardinality: Brands has a one-to-many relationship with Products (since a specific brand can produce many different products).

Normalization Process

Normalize your database. Apply BCNF or 3NF to your schema or show that your schema adheres to one of these normal forms. **Describe why you choose to use BCNF vs 3NF**

Functional Dependencies:
UserId -> Username, Password, FirstName, LastName, Email
ProductId -> ProductName, ProductURL, Price, LikeCount, BrandId
ReviewId -> Rating, Text, ProductId
BrandId -> BrandName, BrandRating
TagId -> TagName, Standing
(UserId, ProductId) -> DateAdded
ProductId -> TagId

Our database is normalized according to the functional dependencies listed above. This schema adheres to 3NF. We chose 3NF over BCNF because, in the context of our database and the existing dependencies, creating stricter constraints and normalizing to BCNF would not significantly reduce the redundancy or update anomalies. Currently most of the tables have their attributes solely dependent on the primary key, and there are no attributes that cause redundancy. With the current schema structure, we have optimized it for queries and operations that the database needs to support, such as retrieving all products under a certain brand name, or fetching all the reviews for a product. Normalizing further to BCNF may make these queries more complex and over-complicated due to the addition of more tables. The specific structure and relationships defined in this schema allow for 3NF to be sufficient, as this form still minimizes redundancy and makes for more streamlined lookups than applying BCNF would.

We can show that this schema adheres to 3NF by obtaining a minimal basis for the functional dependencies. We start by making the right hand side of each dependency a singleton:

UserId -> Username
UserId -> Password
UserId -> FirstName
UserId -> LastName
UserId -> Email
ProductId -> ProductName
ProductId -> ProductURL
ProductId -> Price
ProductId -> LikeCount
ProductId -> BrandId
ReviewId -> Rating
ReviewId -> Text
ReviewId -> ProductId
BrandId -> BrandName

BrandId -> BrandRating
TagId -> TagName
TagId -> Standing
(UserId, ProductId) -> DateAdded
ProductId -> TagId

In this minimal basis, there are no extraneous attributes. DateAdded cannot be inferred from ProductId alone, thus ProductId is not extraneous in the (UserId,ProductId) tuple. There are also no redundancies. Thus, our minimal basis is the same as the original form. With this, we have proved that our schema is in 3NF because in the minimal basis we have obtained, the non-prime attributes are only functionally dependent on the superkeys and nothing else.