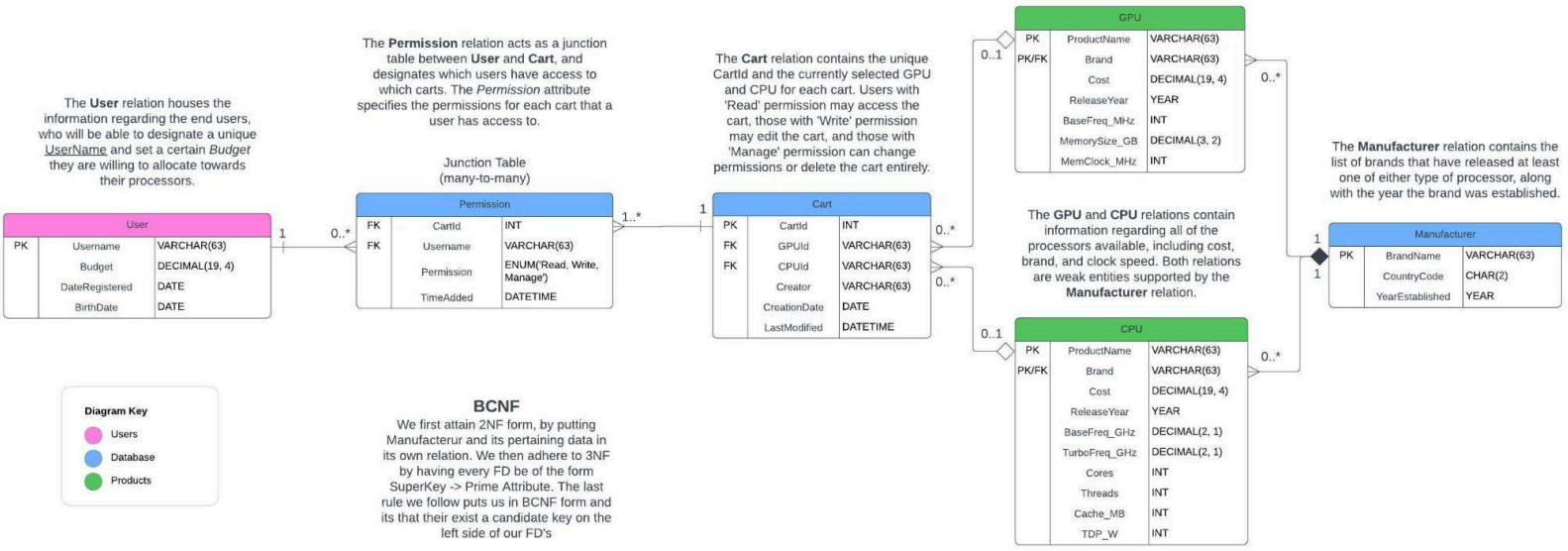
PYP Database Design

Details for Relational Schema and Implementation



[Link to the LucidChart]

Assumptions

From left to right, the first relation of our schema is the **Users** relation, which contains the *Budget* and unique UserName of each end user. This relation is connected to the **Cart** relation via a many-to-many relationship facilitated by the **Permission** junction table. Sharing carts and providing recommendations to other users can lead to many helpful insights regarding what purchases to make, so maintaining a many-to-many relationship between users and carts is crucial. By using a junction table, we plan to enable users to share and discuss their selections with peers, allowing others to edit and manage their choices as necessary. While each user can have access to as many carts as they would like, each cart must be associated with at least one user in order to ensure that the cart relation is not filled with unused carts. If no users are associated with a cart, it will be automatically deleted. Additionally, the cart must be associated with *exactly* one user with ‘Manage’ permission. If the user with ‘Manage’ permission decides to remove the cart from their account, they will have to

designate the ‘Manage’ permission to a user already associated with the cart or delete the cart entirely. This is to ensure that at least one user has the ability to add or remove users from the cart, and that at most one user has the ability to edit permissions in order to avoid discrepancies such as one manager attempting to remove another manager.

The **Cart** relation designates the processor selections and unique CartId for each cart. We chose to limit each cart to one CPU and one GPU in order to keep selections straightforward and to focus on the parts that most heavily affect performance. The access permissions of each user are determined by the **Permission** relation. Users with ‘Read’ permission will be able to see what selections have been made in a cart and be able to view the cart’s notes, but will not be able to make any changes to the cart. Users with ‘Write’ permission will be able to both read and edit the cart selections and notes, however they will be unable to change any of the cart’s permissions. Users with ‘Manage’ permission will be able to both read and edit the cart’s contents, as well as invite other users and manage their permissions. Additionally, users with the ‘Manage’ permission will be able to pass the ‘Manage’ permission to another user or opt to delete the cart entirely.

The rest of the relations pertain to market information. The **GPU** relation uses a unique Natural Key ProductName and Brand to distinguish different GPUs and contains information such as base frequency (GHz), memory size (GB), and clock speed (MHz). The **CPU** relation also uses a unique Natural Key ProductName and Brand to identify each CPU and contains information such as base frequency (GHz), overclock frequency (GHz), and the number of cores and threads. The relations that connect processors to carts will be one-to-many, as each processor can be a member of many carts, but carts can only hold one of each type of processor. We also include a relation that lists **Manufacturers** in order to maintain a record of which manufacturers have products within the database. Each processor must have exactly one designated manufacturer, and each manufacturer must have at least one of either type of processor.

Normalization

For normalization of the database, we have opted to use BCNF decomposition, as it is stricter than 3NF and will therefore leave less room for error regarding redundancies and logical inconsistencies. The first step of our adherence to this form is following the 2NF form by separating the Manufacturing data into its own table. This is so there are no redundancies in the CPU or GPU relations. We then adhere to the additional rule that 3NF form requires which is that all FD's follow the format of: Super Key -> Prime Attribute. We display this below. The last rule will put us in BCNF form and it is that on the left side of our FD's there exists a set of attributes that are a candidate key, which again will be shown below. The format is Functional Dependencies, followed by the justification for each form.

Users(Username, Budget, DateRegistered, BirthDate)

- FD's: {Username} → {Budget, DateRegistered, BirthDate}
- 2NF: No redundancies
- 3NF: Username is a SuperKey, because it identifies all attributes of user, and that points to Prime Attributes
- BCNF: Username is on the left side and is a Candidate key

Permission(CartId, UserName, Permission, TimeAdded)

- FD's: { CartId, UserName } → {Permission, TimeAdded}
- 2NF: No redundancies
- 3NF: Since { CartId, UserName } uniquely identify all attributes its a superkey
- BCNF: Since there is no subset of { CartId, UserName } that uniquely identifies all attributes in the relation, it is the only candidate key and is on the left side of the FD

Cart(CartId, GPUId, CPUId, Creator, CreationDate, LastModified)

- FD's: { CartId } → { GPUId, CPUId, Creator, CreationDate, LastModified}
- 2NF: No redundancies
- 3NF: { CartId } → { GPUId, CPUId, Creator, CreationDate, LastModified} is a nontrivial functional dependency and CartId acts as a superkey for **Cart**.
- BCNF: The left side, CartId, is a candidate key for this relation

GPU(ProductName, Brand, Cost, ReleaseYear, BaseFreq_MHz, MemorySize_GB, MemClock_MHz)

- FD's: { ProductName, Brand} → { Cost, ReleaseYear, BaseFreq_MHz, MemorySize_GB, MemClock_MHz }
- 2NF: No redundancies by separating Manufacturer
- 3NF: { ProductName, Brand} → {Cost, ReleaseYear, BaseFreq_MHz, MemorySize_GB, MemClock_MHz } is a nontrivial functional dependency and ProductId acts as a superkey for **GPU**.
- BCNF: The left side, {ProductName, Brand}, is a candidate key for this relation

CPU(ProductName, Brand,Cost, ReleaseYear, BaseFreq_GHz, TurboFreq_GHz, Cores, Threads, Cache_MB, TDP_W)

- FD's: {ProductName, Brand} → {Cost, ReleaseYear, BaseFreq_GHz, TurboFreq_GHz, Cores, Threads, Cache_MB, TDP_W }
- 2NF: No redundancies by separating Manufacturer
- 3NF: { ProductId } → { ProductName, Cost, Brand, ReleaseYear, BaseFreq_GHz, TurboFreq_GHz, Cores, Threads, Cache_MB, TDP_W } is a nontrivial functional dependency and ProductId acts as a superkey for **CPU**.
- BCNF: The left side contains a candidate key

Manufacturer(BrandName, CountryCode, YearEstablished)

- FD's: {BrandName} → {CountryCode, YearEstablished}
- 2NF: No redundancies
- 3NF: BrandName is a superkey
- BCNF: The left side, BrandName, is a candidate key for this relation

Database Schema

```
User(
  UserName: VARCHAR(63) [PK],
  Budget: DECIMAL(19, 4),
  DateRegistered: DATE,
  BirthDate: DATE
)

Permission(
  CartId: INT [FK to Cart.CartId],
  UserName: VARCHAR(63) [FK to User.UserName],
  Permission: ENUM('Read', 'Write', 'Manage'),
  TimeAdded: DATETIME
)



Cart(
  CartId: INT [PK],
  GPUId: VARCHAR(63) [FK to GPU.ProductId],
  CPUId: VARCHAR(63) [FK to CPU.ProductId],
  Creator: VARCHAR(63),
  CreationDate: VARCHAR(63),
  LastModified: DATETIME
)

GPU(
  ProductName: VARCHAR(63) [PK],
  Brand: VARCHAR(63) [FK to Manufacturer.BrandName] [PK],
  Cost: DECIMAL(19, 4),
  ReleaseYear: YEAR,
  BaseFreq_MHz: INT,
  MemorySize_GB: DECIMAL(3,2),
  MemClock_MHz: INT
)
```

```
CPU(
  ProductName: VARCHAR(63) [PK],
  Brand: VARCHAR(63) [FK to Manufacturer.BrandName] [PK],
  Cost: DECIMAL(19, 4),
  ReleaseYear: YEAR,
  BaseFreq_GHz: DECIMAL(2, 1),
  TurboFreq_GHz: DECIMAL(2, 1),
  Cores: INT,
  Threads: INT,
  Cache_MB: INT,
  TDP_W: INT
)

Manufacturer(
  BrandName: VARCHAR(63) [PK],
  CountryCode: CHAR(2),
  YearEstablished: YEAR
)
```