# Relational Database Schema

## Entities:

1. Credentials (
    netID**:** VARCHAR(20)[PK],
    password: VARCHAR(20),
    permission: VARCHAR(20)
)
    - Credentials are a core part of our application, as we need a way to verify whether a particular individual is allowed either to reserve a particular item. Thus, credentials are an entity because there are two important relationships that stem from it, the capacity to reserve an ITEM for a particular user, and the capacity to determine the MAJOR information (if present) of a user, which would later be used to determine if the individual is eligible to reserve under a major restriction (e.g. a philosophy major may not be allowed to borrow a VR headset restricted to only CS majors).

2. Major (
    majorID: INT [PK],
    majorName: VARCHAR(20),
    majorAbrv: VARCHAR(20),
    deptName: VARCHAR(20)
)
    - Assumptions: Majors has to be a separate entity, as we are assuming that students can have multiple degrees/majors. If we were to have Majors and Departments as an attribute of Credentials, we would have to have multiple records of the same student with different majors. Separating Majors as an entity and creating a HAS relationship allows us to reduce redundancy in the credentials table.

3. Inventory (
    itemID: INT [PK],
    ItemName: VARCHAR(50),
    Availability: BOOL,
    Condition: INT,
    LocationID:[FK to Facilities.LocationID],
    Duration: INT
)
    - Assumptions:

- Inventories are best as a separate entity, considering that items are the main thing we want to keep track of, considering it is an ‘inventory management app’. This is because items need to serve four capabilities: the capacity to reserve a particular item, the ability to restrict items based on major(s), the ability to view information about reservable computers, and the ability to designate what facility this item is stored in. As a result, inventory is a necessary entity that helps map four key relationships, representative of the four capabilities needed for the inventory management app. ‘Inventory’ standalones as an entity as they can exist independently of other entities or relations.

4. Computers (
    itemID: INT [PK, FK to Inventory.itemID],
    OS: VARCHAR(10),
    numMonitors: INT,
    Availability: BOOL,
    Condition: INT,
)

- Assumptions: Computers are items that have unique properties such as OS and monitor setup, so we feel it is better to set it up as a separate entity. If merged with the inventory entity, there will be a lot of wasted storage space due to many items not containing these unique properties for computers.

5. Facilities (
    LocationID: INT [PK],
    BldgName: VARCHAR(20),
    Floor/Section: VARCHAR(20),
    Longitude: FLOAT,
    Latitude: FLOAT,
    MapURL: VARCHAR(200)
)

- Assumptions: We think this deserves to be its own entity rather than just an attribute since each facility has its own unique attributes. And one facility can be home to many items and it would be very redundant to have all the data for the same facility repeated for every item.

---

**Relations (With Associated Relational Schema):**

1. Reserve (
    reservationID: INT [PK],
    netId: VARCHAR(20) [PK, FK to Credentials.netID],
    itemID: VARCHAR(20) [PK, FK to Inventory.itemID],
    startTime: VARCHAR(20),
    returnTime: VARCHAR(20),
    deadLine: VARCHAR(20)
)
    - Assumptions: This relation is a many to many relationship because we want to allow students to reserve multiple items and have items be reserved more than once (not simultaneously). We also need to include a reservationID to act as our primary key since we also want to allow a student to reserve the same item more than once and store each instance. The other attributes like startTime, returnTime, and deadLine are needed for functionalities we want to implement for the app.

2. MajorRestrictions(
    itemID: VARCHAR(20) [PK, FK to Inventory.itemID]
    majorID: INT [PK, FK to Major.majorID]
)
    - Assumptions: One item can have multiple restrictions, so we need this table to model a many to many relation.

3. Has(
    NetId: VARCHAR(20) [PK, FK to Credentials.netID],
    MajorId: INT [PK, FK to Major.majorID]
)
    - Assumptions: Because one student can have multiple majors, we need a HAS table to model this many to many relation

**Relations (Without Associated Relational Schema)**

4. Store
    - Assumption: One facility can store 0 to many items, while each item can only be stored at one facility.

5. Is
    - Assumption: One item in the inventory can either be or not be a computer, while a computer is always an item.

# FDs and 3NF Normalization

Why 3NF over BCNF:

- We chose 3NF since it was lossless and we did not want to lose any functional dependencies as some may not be as apparent.

Credentials (netID, password, permission, majorID)

- Functional dependencies:
  - netID -> password, permission, majorID
- Minimal basis:
  - netID -> password, permission, majorID (the minimal basis is the same as the FDs because there's no redundancies)
- **3NF = (netId, password, permission, majorID)**
  - The set that obtains all attributes in the closure is **(netId, password, permission, majorID)**. Every attribute can provide itself and the set contains all the attributes, so its closure will also contain all the attributes.

Major (majorID, majorName, majorAbrv, deptName)

- Functional dependencies:
  - majorName -> majorAbrv
  - majorID -> majorName, majorAbrv, deptName
- Minimal basis:
  - majorName -> majorAbrv
  - majorID -> majorName, majorAbrv, deptName (the minimal basis is the same as the FDs because there's no redundancies)
- **3NF = (majorName, majorAbrv); (majorID, majorName, majorAbrv, deptName)**
  - The set that obtains all  attributes in the closure is **(majorID, majorName, majorAbrv, deptName).** Every attribute can provide itself and the set contains all the attributes, so its closure will also contain all the attributes.

Inventory (itemID, availability, condition, locationID)

- Functional dependencies:
  - itemID -> availability, condition, locationID
- Minimal basis:

    - itemID -> availability, condition, locationID (the minimal basis is the same as the FDs because there's no redundancies)
- **3NF = (itemID, availability, condition, locationID)**
    - The set that obtains all attributes in the closure is **(itemID, availability, condition, locationID).** Every attribute can provide itself and the set contains all the attributes, so its closure will also contain all the attributes.

Computers (itemID, os, numMonitors, availability, condition)

- Functional dependencies:
    - itemID -> os, numMonitors, availability, condition
- Minimal basis:
    - itemID -> os, numMonitors, availability, condition (the minimal basis is the same as the FDs because there's no redundancies)
- **3NF = (itemID, os, numMonitors, availability, condition)**
    - The set that obtains all attributes in the closure is **(itemID, os, numMonitors, availability, condition).** Every attribute can provide itself and the set contains all the attributes, so its closure will also contain all the attributes.

Facilities (locationID, bldgName, floor/section, longitude, latitude, mapURL)

- Functional dependencies:
    - bldgName -> (longitude, latitude)
    - (latitude, longitude) -> bldgName
    - locationID -> bldgName, floor/section, longitude, latitude, mapURL
- Minimal basis:
    - bldgName -> (longitude, latitude)
    - (latitude, longitude) -> bldgName
    - locationID -> bldgName, floor/section, longitude, latitude, mapURL (the minimal basis is the same as the FDs because there's no redundancies)
- **3NF = (bldgName, longitude, latitude); (locationID, bldgName, floor/section, longitude, latitude, mapURL)**
    - The set that obtains all attributes in the closure is **(locationID, bldgName, floor/section, longitude, latitude, mapURL).** Every attribute can provide itself and the set contains all the attributes, so its closure will also contain all the attributes.

Reservations (reservationID, netId, startTime, returnTime, deadLine, itemID)

- Functional dependencies:
    - startTime -> deadLine

    - reservationID -> netId, startTime, returnTime, deadLine, itemID
- Minimal basis:
    - startTime -> deadLine
    - reservationID -> netId, startTime, returnTime, deadLine, itemID (the minimal basis is the same as the FDs because there's no redundancies)
- **3NF = (startTime, deadLine); (reservationID, netId, startTime, returnTime, deadLine, itemID)**
    - The set that obtains all  attributes in the closure is **(reservationID, netId, startTime, returnTime, deadLine, itemID).** Every attribute can provide itself and the set contains all the attributes, so its closure will also contain all the attributes.