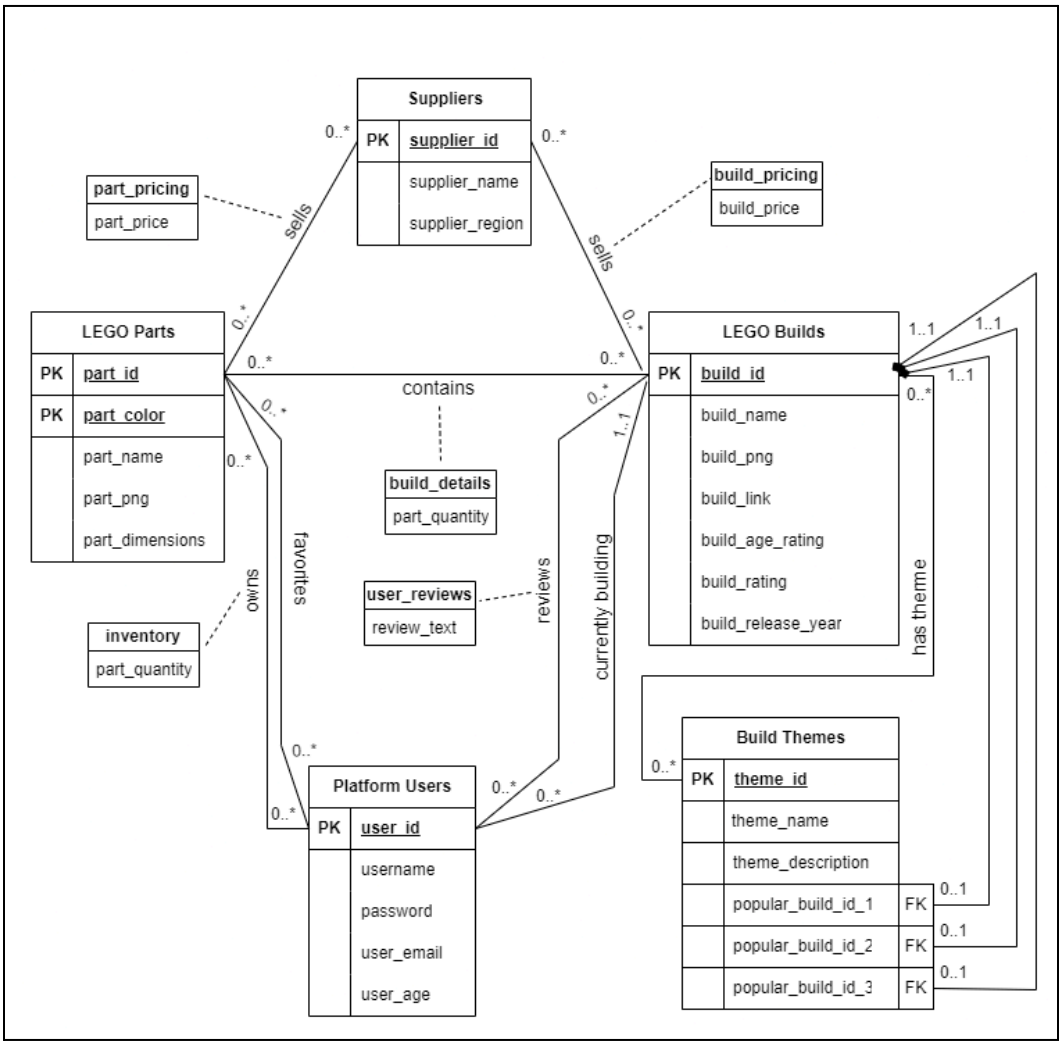
Conceptual and Logical Database Design

1. UML Diagram



2. Explanation
    a. Entities
        - LEGO Builds - The builds entity is important to have to keep track of the different builds that can be created by the users. This entity gives information about the build name, rating given by users, its release year, and the recommended ages for users to work on this build. A picture of the build is also presented to help users to understand what the final build might look like.
        - Suppliers - Suppliers is an entity because it is important to have the supplier's name and region that can be linked to the pricing options for

both builds and parts. Supplier region will be the geographical region where the supplier is based out of.

- LEGO Parts - The LEGO parts entity will give descriptions of the part that can be used to store all possible pieces across all possible builds. The attributes of this entity include: the main color of the part, a descriptive name for the part that conveys important information. A photo of the part to be displayed for the user. The parts dimensions to convey the part's compatibility in the build among other pieces.
- Platform Users - Platform Users is an entity to keep login information for each user separate from other data. This entity includes a unique identifier for each user. It also includes the user's username to be displayed on things like reviews and the password for login authentication. Additionally, we will store the user's email address and their age which can be used to automatically filter on the age-rating of the builds.
- Build Themes - Build theme is an entity to keep organizational information about attributes about themes such as name, description of what the theme includes, and top 3 builds that fall under that theme. This will be helpful for coming up with suggestions for users when they want to find what to potentially start building. The reason these are not just attributes in the Builds entity is because each build can have many themes and hence it makes sense for Themes to be an entity by itself.

b. Relationship Tables
    - Inventory - This stores the relationship attribute of the quantity of a specific part that a user owns. It is from the many to many relationship between the user_id and part_id.
    - Build_details - This stores the relationship attribute of the quantity of a specific part that a build needs. It is from the many to many relationship between the user_id and build_id.
    - User_reviews - This stores the relationship attribute of the user reviews that is left by users on different builds. It is from the many to many relationship between the user_id and build_id.
    - Build_pricing - This stores the relationship attribute of the selling price of different builds from different suppliers. It is from the many to many relationship between the supplier_id and build_id.
    - Part_pricing - This stores the relationship attribute of the selling price of different parts from different suppliers. It is from the many to many relationship between the supplier_id and part_id.

c. Cardinality

i. (sells)Parts to suppliers
    - This is a many to many relationship because each supplier_id can have many part_ids and each part_id can be sold by many suppliers.

ii. (sells)Suppliers to build
    - This is a many to many relationship because each supplier_id can have many build_ids and each build_id can be sold by many suppliers.

iii. Owns
    - This is a many to many relationship because each user_id can own many part_ids and each part_id can be owned by many user_ids because there can be more than one of the same part.

iv. Favorites
    - This is a many to many relationship because each user_id can have many part_ids favorited and each part_id can be favorited by many user_ids because there can be more than one of the same part.

v. Reviews
    - This is a many to many relationship because each user_id can have many build_ids reviews and each build_id can be reviewed by many user_ids.

vi. Currently building
    - This is a one to many relationship because each user_id can only have one build_id as their current build, but each build_id can be currently built by many user_ids.

vii. Contains
    - This is a many to many relationship because each part_id can be a part of many build_ids and each build_id can contain many build_ids.

viii. Has theme
    - This is a many to many relationship because each build_id can have many theme_ids and each theme_id can contain belong to build_ids.

3. UML Requirements
    a. Our five entities are Suppliers, LEGO Parts, LEGO Builds, Platform Users, and Build Themes
    b. We have all three types of relationships in our UML Diagram, here are few examples
        i. One-to-Many: We have a *currently building* relationship that is one to many because the same build can be built by many users but each user can only be currently building one specific build at a time.

ii. Many-to-Many: Most of our relationships are many-to-many. For example a build can *contain* many parts and the same part can be used in many builds.
iii. One-to-One: We have a foreign key constraint on popular_build_id_1/2/3 which creates a one-to-one mapping between the foreign key and the primary key build_id in the LEGO Builds entity

4. Normalization Proof

Platform Users:
Functional dependencies: user_id → username, password, user_email, user_age,

Left Side	Middle	Right Side	None
user_id,		username, password, user_email, user_age	

user_id is the primary key and also the determinant for username and password. Since user_id is a candidate key, this table already satisfies BCNF.

Lego Parts:
Functional dependencies: part_id, part_color → part_name, part_png, part_dimensions

Left Side	Middle	Right Side	None
part_id, part_color		part_name, part_png, part_dimensions	

The composite key (part_id, part_color) is the primary key and determines all the other attributes.The table is already in BCNF because the determinant (part_id, part_color) is a candidate key.

Lego Builds:
Functional dependencies: build_id → build_name, build_png, build_link, build_age_rating, build_rating, build_release_year.

Left Side	Middle	Right Side	None
build_id		build_name, build_png, build_link, build_age_rating, build_rating, build_release_year	

build_id is the primary key and determines all the other attributes.This table is already in BCNF because the determinant build_id is a candidate key.

Build Themes:

Functional dependencies: theme_id → theme_name, theme_description, popular_build_id1, popular_build_id2, popular_build_id3

Left Side	Middle	Right Side	None
theme_id		theme_name, theme_description, popular_build_id1, popular_build_id2, popular_build_id3.	

theme_id is the primary key and determines all other attributes.This table is already in BCNF because theme_id is a candidate key.

Suppliers:
Functional dependencies: supplier_id → supplier_name, supplier_region

Left Side	Middle	Right Side	None
supplier_id		supplier_name, supplier_region	

supplier_id is the primary key and determines all other attributes. This table is already in BCNF because supplier_id is a candidate key.

As for each functional dependency, the primary key is able to define the other attributes for the entity it is in, the primary key of each entity is the minimal super key and thus all functional dependencies comply with BCNF. The database is already normalized.

5. Logical Design (Relational Schema)
    a. Entities

Platform Users (
    user_id: VARCHAR(100) [PK],
    username: VARCHAR(100),
    password: VARCHAR(100),
    user_email: VARCHAR(100,
    user_age: INTEGER,
)

Lego Parts (
    part_id: VARCHAR(100) [PK],
    part_color: VARCHAR(100)[PK],
    part_name: VARCHAR(100),
    part_png: VARCHAR(255),
    part_dimensions: VARCHAR(255),
)

Lego Builds (
build_id: VARCHAR(100)[PK],
build_name: VARCHAR(100),
build_png: VARCHAR(100),
build_link: VARCHAR(255),
build_age_rating: INTEGER,
build_rating: DECIMAL(4, 2),
build_release_year: VARCHAR(10)
)

Build Themes (
theme_id: VARCHAR(100)[PK],
theme_name: VARCHAR(100),
theme_description: VARCHAR(1000),
popular_build_id1: VARCHAR(100)[FK to Lego Builds.build_id]
popular_build_id2: VARCHAR(100)[FK to Lego Builds.build_id]
popular_build_id3: VARCHAR(100)[FK to Lego Builds.build_id]
)

Supplier Id (
supplier_id: VARCHAR(100)[PK],
supplier_name: VARCHAR(100),
supplier_region: VARCHAR(100)
)

b. Many-to-many Relationships

Inventory (
user_id: VARCHAR(100) [PK] [FK to Platform Users.user_id],
part_id: VARCHAR(100) [PK] [FK to Lego Parts.part_id],
Part_quantity: INTEGER,
)

Build Details (
user_id: VARCHAR(100) [PK] [FK to Platform Users.user_id],
build_id: VARCHAR(100) [PK] [FK to Lego Builds.build_id],
Part_quantity: INTEGER,
)

User Reviews (
    user_id: VARCHAR(100) [PK] [FK to Platform Users.user_id],
    build_id: VARCHAR(100) [PK] [FK to Lego Builds.build_id],
    review_text: VARCHAR(1000),
)

Build Pricing (
    supplier_id: VARCHAR(100) [PK] [FK to Suppliers.supplier_id],
    build_id: VARCHAR(100) [PK] [FK to Lego Builds.build_id],
    build_price: DECIMAL(6, 2),
)

Part Pricing (
    supplier_id: VARCHAR(100) [PK] [FK to Suppliers.supplier_id],
    part_id: VARCHAR(100) [PK] [FK to Lego Parts.part_id],
    part_price: DECIMAL(6, 2),
)