**Tables:**

1. **Products**
   * id (INT, PRIMARY KEY, AUTO\_INCREMENT)
   * name (VARCHAR(255), NOT NULL)
   * is\_completed (BOOLEAN, DEFAULT FALSE)
   * created\_at (TIMESTAMP, DEFAULT CURRENT\_TIMESTAMP)
   * updated\_at (TIMESTAMP, DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP)
2. **Phases**
   * id (INT, PRIMARY KEY, AUTO\_INCREMENT)
   * product\_id (INT, NOT NULL, FOREIGN KEY referencing Products.id)
   * name (VARCHAR(255), NOT NULL)
   * is\_completed (BOOLEAN, DEFAULT FALSE)
   * created\_at (TIMESTAMP, DEFAULT CURRENT\_TIMESTAMP)
   * updated\_at (TIMESTAMP, DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP)
3. **Tasks**
   * id (INT, PRIMARY KEY, AUTO\_INCREMENT)
   * phase\_id (INT, NOT NULL, FOREIGN KEY referencing Phases.id)
   * name (VARCHAR(255), NOT NULL)
   * estimated\_completion\_time (DATETIME)
   * needs\_approval (BOOLEAN, DEFAULT FALSE)
   * is\_approved (BOOLEAN, DEFAULT FALSE)
   * approval\_requested (BOOLEAN, DEFAULT FALSE)
   * approved\_by\_user\_id (INT, NULL, FOREIGN KEY referencing a Users table - see below)
   * completed\_at (TIMESTAMP, NULL)
   * created\_at (TIMESTAMP, DEFAULT CURRENT\_TIMESTAMP)
   * updated\_at (TIMESTAMP, DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP)
4. **Documents**
   * id (INT, PRIMARY KEY, AUTO\_INCREMENT)
   * task\_id (INT, NOT NULL, FOREIGN KEY referencing Tasks.id)
   * file\_name (VARCHAR(255), NOT NULL)
   * file\_path (VARCHAR(255), NOT NULL)
   * uploaded\_by\_user\_id (INT, NOT NULL, FOREIGN KEY referencing a Users table - see below)
   * uploaded\_at (TIMESTAMP, DEFAULT CURRENT\_TIMESTAMP)
5. **Roles**
   * id (INT, PRIMARY KEY, AUTO\_INCREMENT)
   * name (VARCHAR(50), UNIQUE, NOT NULL)
6. **Permissions**
   * id (INT, PRIMARY KEY, AUTO\_INCREMENT)
   * name (VARCHAR(100), UNIQUE, NOT NULL, e.g., 'create\_product', 'delete\_task', 'approve\_task', 'upload\_finance\_document')
7. **RolePermissions** (Many-to-Many relationship between Roles and Permissions)
   * role\_id (INT, NOT NULL, FOREIGN KEY referencing Roles.id)
   * permission\_id (INT, NOT NULL, FOREIGN KEY referencing Permissions.id)
   * PRIMARY KEY (role\_id, permission\_id)
8. **Users** (You'll likely need a user table for authentication and associating actions)
   * id (INT, PRIMARY KEY, AUTO\_INCREMENT)
   * username (VARCHAR(50), UNIQUE, NOT NULL)
   * password (VARCHAR(255), NOT NULL) // Store hashed passwords
   * email (VARCHAR(100), UNIQUE)
   * role\_id (INT, NOT NULL, FOREIGN KEY referencing Roles.id)
   * created\_at (TIMESTAMP, DEFAULT CURRENT\_TIMESTAMP)
   * updated\_at (TIMESTAMP, DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP)

**Relationships Explained:**

* A Product can have multiple Phases (one-to-many).
* A Phase belongs to one Product (many-to-one).
* A Phase can have multiple Tasks (one-to-many).
* A Task belongs to one Phase (many-to-one).
* A Task can have multiple Documents attached (one-to-many).
* A Document belongs to one Task (many-to-one).
* A User has one Role (many-to-one).
* A Role can have multiple Permissions (many-to-many through RolePermissions).
* A Task can be approved by one User (many-to-one).
* A Document is uploaded by one User (many-to-one).

**How this structure supports your requirements:**

* **Products, Phases, Tasks Hierarchy:** The product\_id and phase\_id foreign keys establish the parent-child relationships.
* **Permissions Management:** The Roles, Permissions, and RolePermissions tables allow you to define roles (e.g., admin, finance) and assign specific permissions to them. You can then check these permissions in your backend logic.
* **Task Attributes:** The Tasks table includes fields for estimated\_completion\_time, needs\_approval, is\_approved, and approval\_requested. The approved\_by\_user\_id links the approval action to a specific user.
* **Document Upload:** The Documents table stores information about uploaded files and links them to specific tasks.
* **Phase and Product Completion:** The is\_completed flags in the Products and Phases tables can be used to track their completion status. You'll need backend logic to update these based on the completion of their children (tasks for phases, phases for products).
* **User Association:** The Users table and foreign keys in Tasks and Documents allow you to track which user performed certain actions.

**Further Considerations:**

* **Indexing:** You'll want to add indexes to foreign key columns (e.g., product\_id in Phases, phase\_id in Tasks) to optimize database queries.
* **Data Types:** I've used common data types. You might need to adjust them based on the specific database you choose.
* **Auditing:** If you need to track changes over time, you might consider adding audit logs or using database features for auditing.
* **Soft Deletes:** For data retention and recovery, consider adding a deleted\_at timestamp to your tables instead of permanently deleting records.
* **Relationships in Code:** Your backend code will use these relationships to query and manage your data (e.g., fetching all phases for a given product).

This database structure provides a solid foundation for your application. You can adapt and expand it as your requirements evolve. Remember to choose a database system that suits your needs and integrate it with your Next.js backend.

Sources and related content