**Techical:**

* Web-server: NodeJS (NestJS + TypeScript)
* Database: PostgreSQL

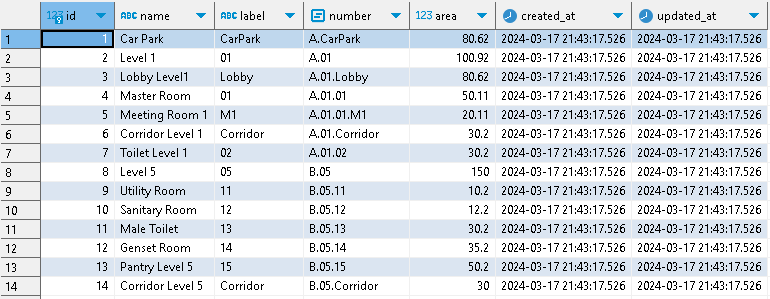
**Database Structure:**

**Table:**

location

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Type** | **Is Null** | **Constraint** | **Default** |
| id | serial4 | no | primary key |  |
| name | text | no |  |  |
| label | text | no |  |  |
| number | ltree | no | unique |  |
| area | float4 | no |  |  |
| created\_at | timestamp | no |  | current\_timestamp |
| updated\_at | timestamp | no |  | current\_timestamp |

**Sample Data:**



**Note:**

Each location comprises a name, label, number, and area. The label represents the code or short name of the location, and it is also a component within the location number. Location number utilizes the [Ltree](https://www.postgresql.org/docs/current/ltree.html) data type for representing labels of data stored in a hierarchical tree-like structure.

One challenge when using Ltree is integrating it with ORM libraries like TypeORM and Prisma for connecting to a web server. With TypeORM, there is support for [Tree Entities](https://typeorm.io/tree-entities), but during synchronization, the data is represented as foreign keys rather than a true hierarchical tree structure. On the other hand, it seems that Prisma does not support it directly. Due to the preference for storing data in the Ltree format, accessing or updating data through the web server will involve using raw SQL.

**Web server main work flow**

**API Create {POST}**

* Input:
  + Path: /api/v1/location
  + Body: {"name": "Lobby Level 2","number": "A-02-Lobby","area": 50.11}
* Output: success or failure or error
* Work flow:
  + Convert location number to Ltree format (from ‘-‘ to ‘.’)
  + Check the existence of the parent of this location based on the location number.
  + If the parent does not exist, return failure. If it exists, execute a raw SQL insert statement to add a new location.
  + If after executing the insert SQL that there are no additional rows added, indicating a conflict with the location number, return failure.

**API Update {PUT}**

* Input:
  + Path: /api/v1/location/:id
  + Body: {"name": "Lobby Level 2","number": "A-02-Lobby","area": 50.11}
* Output: success or failure or error
* Work flow:
  + Convert location number to Ltree format (from ‘-‘ to ‘.’)
  + Check the existence of the parent of this location based on the location number.
  + If the parent does not exist, return failure. If it exists, continue to check the existence of this location.
  + If this location already exists, return failure. If it does not exist yet, execute the SQL update statement.

**API Update {DELETE}**

* Input:
  + Path: /api/v1/location/:id
* Output: success or failure or error
* Work flow:
  + Check the existence of this location based on the id.
  + If this location does not exist, return failure. If it exists, execute the SQL delete statement for this location and all its child nodes.