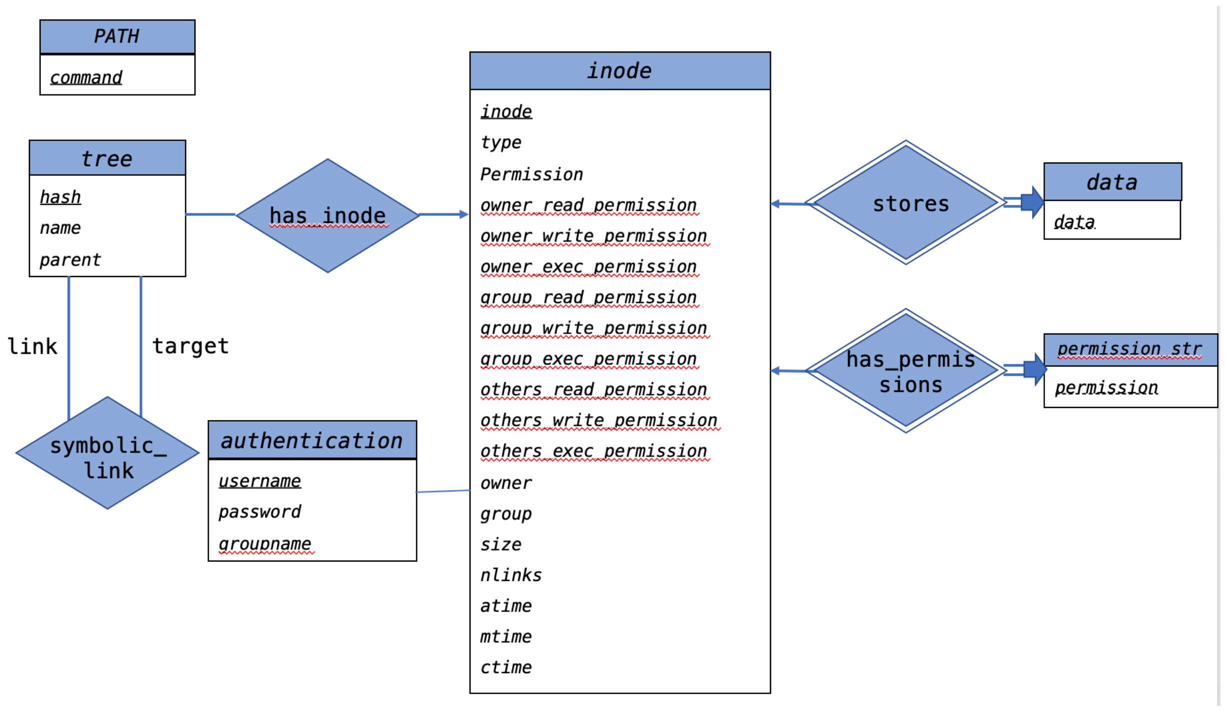
Entity-Relationship Model for the database:



Description of each entity (6 entities total):

PATH: represents available commands for use (ls, cd, find, grep)

- *command*: name of the command available

authentication: represents user information stored in file system

- *username*: name of the user (displayed in ls –l output)
- *password*: password of the user
- *groupname*: which group this user belongs to (displayed in ls –l output)

tree: represents hierarchical directory listing

- *hash*: full pathname of the file. *VARCHAR*
- *name*: name of the file (filename only). *VARCHAR*
- *parent*: hash value of the parent directory. *VARCHAR*
- *inode*: number of inode. *BIGINT*

inode: represents the actual inode

- *inode*: number of inode. *INT or BIGINT*

- *type*: type of the file, stored as the file mode bit. It uses '-' to represent file, 'd' to represent directory and 'l' to represent symbolic link. *CHAR(1)*
- *permissions*: the permission bits for the inode, e.g. *rwxrwxrwx, each column represents 1 bit of permission. These 9 permission bits can be grouped into 3 sets: owner, group and others. The bit uses 'r', 'w' and 'x' to represent read, write and execute, and uses '-' to represent no permission.*
    - *owner_read_permission:* owner's read permission bit. *CHAR(1)*
    - *owner_write_permission:* owner's write permission bit. *CHAR(1)*
    - *owner_exec_permission:* owner's execute permission bit. *CHAR(1)*
    - *group_read_permission:* group's read permission bit. *CHAR(1)*
    - *group_write_permission:* group's write permission bit. *CHAR(1)*
    - *group_exec_permission:* group's execute permission bit. *CHAR(1)*
    - *others_read_permission:* others' read permission bit. *CHAR(1)*
    - *others_write_permission:* others' write permission bit. *CHAR(1)*
    - *others_exec_permission:* others' execute permission bit. *CHAR(1)*
- *owner*: user name of the owner, references to authentication table *VARCHAR*
- *group*: group name of the owner, references to authentication table *VARCHAR*
- *size*: size of the data chunks in bytes. *BIGINT*
- *nlinks*: number of links to this inode. *INT*
- *atime, mtime, ctime*: Last accessed/modified/changed. *INT* (epoch time)

To reduce the effort needed on permission checking and to maintain atomicity, we split the mode string into ten fields in the inode table. But since we also need to display the full mode string in 'ls-l' output, we put the complete permission string in another table called permission_str (to maintain BCNF of inode).

permission_str: stores the mode string of an inode

- *permission*: mode string of an inode, in format "-rwxrwxrwx". *VARCHAR*

data: stores data blocks of an inode

- *data*: data block of corresponding inode. *BLOB*

Description of each relationship (4 relationships in total):

has_inode: tree and inode forms a one-to-many relationship called directory. To accommodate hard links, where multiple different full path (hash in tree) can have the same inode value, the cardinality of directory is one-to-many.

symbolic_links: To support symbolic link, tree and itself forms a symbolic_link relationship, where one of them is link, and the other one is target. Therefore, a symbolic link can be maintained by pointing the link to the target, and they have different inode values.

stores: inode and data forms a one-to-one stores relationship. Stores is a weak entity, because data cannot exist if the inode does not exist. For the sasme reason, data entity must fully participates in stores relationship.

has_permission: inode and permission_str forms a one-to-one has_permission relationship. Has_permission is a weak entity, because the permission string cannot exist if the inode does not exist. For the same reason, permission_str entity must fully participates in has_permission relationship.

To transform the ER design into relational database, we create 7 tables: 1 table per entity and 1 table for symbolic_links relationship. We merge stores into data, and merge has_permission into permission_str and merge has_inode into tree.