The Un*x File System

This project is a prototype for implementing Un*x File System using MySql database.We can term it as MySql based Un*x File system.

1. ER MODEL:

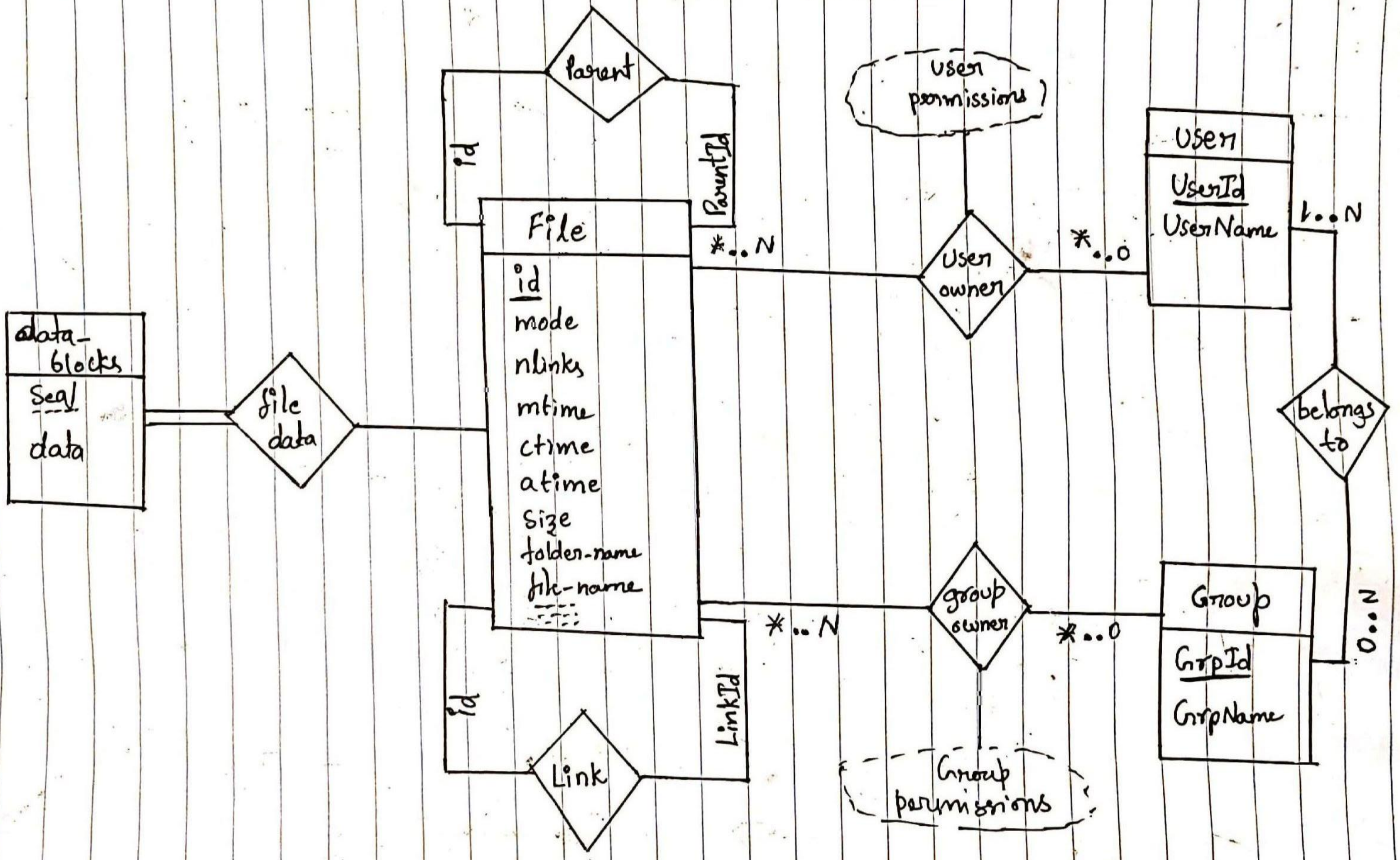
Below is the ER model for the un*x based filesystem.

Model Description:

- The **'File'** entity set is used to hold all the properties realted to file.In this entity 'id' can be used as the primary key to identify file entities uniquely. The attributes of this entity are id, mode(mode of the file or folder), nlinks(number of links to each file/folder),atime(accessed time),mtime(modified time), ctime(creation time),size(size of the file or folder), folder or file name and some other properties of each file and folder.
- Every file and folder has a parent associated with it.In order to represent this a **recursive relation 'parent'** is used.For every 'id' in 'File' entity set there is a 'parentId' associated with it.
- Some of the files in Un*x based files systems has links either soft or hard.In order to represent a link for a file, a **recursive relation 'Link'** is used.For the files,who has a link, a reference file id is stored in **'LinkId' .**
- In general only files have data or content in them and folders doesn't have any data.So to seperate this data attribute,I have created a weak entity which has attributes seq(sequence number of the data block) and data(BLOB to store file content).This weak entity is named **'data_blocks'**.
- In 'data_blocks' attribute **'seq'** acts as **discriminator**.So along with 'id' primary key of 'File' entity ,we can identify entities in this uniquely.So the primary key for 'data_blocks' is (File.id + data_blocks.seq).
- File system has many uses.A **'user'** entity is used to store all the users in the file system.This entity has attributes UserId, UserName .The primary key for this entity is 'UserId' which can be used to identify any user uniquely.
- Similar to many users ,there are many groups.A **'Group'** entity is used to represent all the groups with attributes GrpId,GrpName .'GrpId' can uniquely identify all the groups,so it can be used as primary key for this entity.
- Every user belongs to one or more groups ,so this participation is represented using "1..N' from user to group using a relation **'belongs to'**.This realtion can be uniquely identified using UserId and GrpId as primary key.
- All the files in file system has a single or multiple owners.This relation is represented as **'user owner'** from 'File' to 'user' entity with participation as **"*..N"**.(N can be treated as the max limit a system can allow).This realtion has attribute 'user permissions' which is a derived attribute.This realtion can be identified with (File.id + UserId) as primary key.
- Similarly every file belongs to a group.This relation is represent using **'group owner'** with same participation as above.This realtion has attribute 'group permissions' which is a derived attribute.This realtion can be identified with (File.id + GrpId) as primary key.
- 'data_blocks' entity is a weak entity, so should have a total participation in realtion 'file data' with 'File' entity.

* Used the standard notation given is lecture notes.
* Attached another file **'ER_Model file system.pdf'** for reference in case if the below ER model is unclear.

ER MODEL


parent
id
ParentId
User permissions
User
UserId
UserName
1..N
*..N
User owner
*..0
File
id
mode
nlinks
mtime
ctime
atime
size
folder-name
file-name
data-blocks
Seq
data
file data
belongs to
group owner
*..N
*..0
Group
GrpId
GrpName
0..N
id
LinkId
Link
Group permissions

2. RELATIONAL MODEL

The above ER model without any reduction can be convert to beow relational model :

- file(id,mode,nlinks,mtime,atime,ctime,size,folder_name,file_name,.....)
- data_blocks(seq,data)
- FileData(id,seq).
- user(UserId,UserName)
- grp(GrpId,GrpName)
- belongsTo(UserId,GrpId).
- UserOwner(id,UserId).
- GroupOwner(id,GrpId).
- Parent(id,parent_id).
- Link(id,LinkId).

There are lot of relations that can be combined to normalise the realtional model to reduce data redundency.

Reductions :

- belongsTo relation can be removed with an additional column in user realtion which referes to grp.
- Simlarly UserOwner and GroupOwner realtions can be combined with files using uid and gid attributes in file realtion.
- Parent and Link realtions can reduced by adding additional attributes to file.These attributes are LinkId and parent_id to be added to file reation.
- FileData realtion can be reduced using a attribute id in data_blocks which refers to file realtion.

Final realtional model after reduction and normalisation :

- **file(id,parent_id,mode,uid,gid,LinkIdnlinks,mtime,atime,ctime,size,folder_name,file_name,path,..)**
- **data_blocks(id,seq,data)**
- **grp(GrpId,GrpName)**
- **user(UserId,UserName,GrpId)**

***** For ease of access in the client program,I have divided the file table into two parts.I have split the tabel into “file” and the other is “properties”.Properties table has general attributes like mode, nlink, uid, gid, mtime, ctime, atime, size which are attributes for both files nad folders.

- file(id,parent_id,folder_name,file_name,LinkId,....)
- properties(id,mode,nlinks,uid,gid,mtime,atime,ctime,size,...)

SQL table creation satements along with primary and foreign keys:

*** This statements can be reffered from the attached external file ‘filesystem_tables.sql’ with project submission**

```
CREATE TABLE `data_blocks` (
        `id` bigint NOT NULL,
        `seq` int unsigned NOT NULL,
         `data` longblob,
        PRIMARY KEY (`id`,`seq`),
        CONSTRAINT `data_blocks_ibfk_1` FOREIGN KEY (`id`) REFERENCES `file` (`id`)
        ) ENGINE=InnoDB DEFAULT CHARSET=binary;

CREATE TABLE `file` (
        `id` bigint NOT NULL,
         `parent_id` bigint NOT NULL,
         `path` varchar(255) NOT NULL,
         `folder_name` varchar(255) NOT NULL,
         `file_name` varchar(255) NOT NULL,
        `LinkId` bigint DEFAULT NULL,
        UNIQUE KEY `name` (`id`,`parent_id`),
        KEY `path` (`path`),
        CONSTRAINT `file_ibfk_1` FOREIGN KEY (`LinkId`) REFERENCES `file` (`id`)
        );

CREATE TABLE `grp` (
        `GrpId` int unsigned NOT NULL,
        `GrpName` varchar(255) NOT NULL,
        PRIMARY KEY (`GrpId`)
        ) ;

CREATE TABLE `user` (
        `UserId` int unsigned NOT NULL,
        `UserName` varchar(255) NOT NULL,
        `GrpId` int unsigned NOT NULL,
        PRIMARY KEY (`UserId`),
        KEY `GrpId` (`GrpId`),
        CONSTRAINT `user_ibfk_1` FOREIGN KEY (`GrpId`) REFERENCES `grp` (`GrpId`)
        ) ;

CREATE TABLE `properties` (
        `id` bigint NOT NULL,
        `mode` int NOT NULL DEFAULT '0',
        `nlink` int DEFAULT NULL,
        `uid` int unsigned NOT NULL DEFAULT '0',
        `gid` int unsigned NOT NULL DEFAULT '0',
        `atime` int unsigned NOT NULL DEFAULT '0',
        `mtime` int unsigned NOT NULL DEFAULT '0',
        `ctime` int unsigned NOT NULL DEFAULT '0'
        `size` bigint NOT NULL DEFAULT '0',
        PRIMARY KEY (`id`),
        KEY `gid` (`gid`),
        KEY `uid` (`uid`),
        CONSTRAINT `properties_ibfk_1` FOREIGN KEY (`id`) REFERENCES `file` (`id`),
        CONSTRAINT `properties_ibfk_2` FOREIGN KEY (`gid`) REFERENCES `grp` (`GrpId`),
        CONSTRAINT `properties_ibfk_3` FOREIGN KEY (`uid`) REFERENCES `user` (`UserId`)
        ) ;
```

Keys and indexes:

Primary Keys:

- file -> id,parent_id
- data_blocks -> id,seq
- properties-> id
- user -> UserId
- grp -> GrpId

Foreign keys:

- file -> LinkId reference file(id)
- data_blocks -> id references file(id)
- properties -> id references file(id)
- properties -> uid references user(UserId)
- properties -> gid references grp(GrpId)
- user -> GrpId references grp(GrpId)

Index :

- An additional index on attribute file(path) which is accessed quite often
- Addition indexes on properties(uid) and properties(gid) to speed up join operations.

3. POPULATING DATABASE

- The database is populated with Unix file system files along with properties.This database file system is loaded with around 220k (file + folders).
- os.stat was used for each and every file and folder to capture all details of the files and folders.
- os.stat was used to check if the files are softlinks,in such case stored the reference file along with the softlink to the file.
- Content of the file is read and stored in database 'data_blocks' table for all the files.
- Executable files are also stored in the database just like the general file contents.
- All the folders are recursively traversed using the script and respective details and contents of the files and folders are stored in tables.

*** Python script "data_dump.py" is attached along with the project submission which can be used to load entire unix filesystem to the database.Instructions for running the file are mentioned in README.md**

4. PROTOTYPE CLIENT

*** A video demo of the client prototype has been submitted with file name "656_final_presentation.mp4".**

Following client utilites has been created on this database and can be given as inputs to the scripts:

- **pwd :** pwd commad can be used to give the present working directory.
- **cd :** change directory command has been implemented in many ways just like the terminal cd command.Some of the ways that can be utilised with examples are 'cd ..', 'cd ./home' , ' cd home', 'cd home/usr' , 'cd ./home/usr' . All the above ways can be used to traverse through a directory.
- **ls :** list command has be implemented in two ways.A general 'ls' query will give all the files and folders in the pwd.It gives folders in blue color and files in white color.**'ls -l'** command can be used just like the terminal command to get all the details about files and folders.Those deatils include type i.e. either a file or a folder or a link,user ,group and others permission bits,user owner,group of file/folder , its size , number of links,modified time and at last the name of file or folder.
- **find :** as per the requirments find command accepts the directory and partial name of the file to be found. This command finds all the file names which has partial file name in their file names.The result will be 'ls -l" i.e. all the details of file along with the directory in which they are.
- **grep :** this command accepts the pattern to be searched and the partial file name. This command gives results of the line number in the file where the pattern was found and will even print the line for us. This finds the pattern in all the files which has partial file name in their file name. As there will be results from multiple files, the results will mention from which file the pattern was matched.
- **echo $PATH :** this command can be used to check the PATH varaibles of the system.
- **exciutables :** We can execute any executables which are in the path which is the PATH varaible just by giving file name directly. Example,to check the version of mysql ,we can give the command "mysql -V". These executables are stored in database.No calling them ,their content is downloaded to a file and then executed as instructed. To execute a exicutable in pwd we can use the command ./"exicutable-name" just like in general unix based system.

To make this database file system more dynamic ,the following commands has been implemented:

- **touch :** This command is implemented to create a new file in the pwd. Example : "touch a.txt". This creates a new file in the database along with all properties like other files.
- **mkdir :** This command is implemented to create new directory in pwd. Example : "mkdir 656".This directories are traversable.We can traverse to the directory after creating it. Example "cd 656". We can even create files in the directory just like the general mkdir command. This will create a new directory in database with all the properties realated to it.
- **rm file_name :** This command can be used to delete specific file. Example : "rm a.txt" will delete the file form the database based file system.

- **rm -r dir_name :** This command can be used to delete the a directory along with all the files and folders inside it. Example : “rm -r 656” ,this will delete al the files and folders in 656 directory along with folder 656 from the database based file system.

*** The python script written for this is added along with the project submission with file name “bash.py”.**
*** Instrctions to run the file are mentioned in README.md**
*** This project was done individually not in a group.**