Update the “Departments” application so it could store it’s data in the DB. You can choose Postgres or Mysql. What it should to be: for now any department or employee should be stored in the DB. On the application restart any data has to be readed from database. Any crud operation (create / update / delete) has to affect the data in the db. Don’t delete the code that responses for reading/writing in/from files.

**Requirements:**

* The application has to create tables if they aren’t exist
* The SQL exceptions shouldn’t be gently wrapped and “user friendly” message should be showed to user in the console (“The internal db error” something like this but not the stack trace for God sake)
* The application should validate data on crud operations. (For example “cr -e -a fdf” could rise an error if that value “dfd” will be passed to a column with type int. So, to prevent such kind of exceptions you have to check argument of crud operations are they really valid int).
* Restrictions for the table: column “name” for both tables should be max 25 symbols. Column age is a tinyint / short. Columns “language” and “methodology” varchar max 20 symbols.
* Create additional command “all” that prints grid with columns : Department Name, Employee Name, Employee Type, Employee Age. This grid contains all employees with their department names. Should be sorted by “Department Name” column
* Create additional command “search -e -a age\_to\_search -d department” that search for employee where:
  1. -e marks that we perform search by employees
  2. -a age of employee that has to be searched
  3. -d department name where we have to look for employees

When enter pushed we have to see a list with employees with columns:Employee Name, Employee Type, Department Name. Add this command to list of -help commands.

* Create additional command “top -d -t type\_of\_employee” that shows the department with the largest number of employees. Add this command to list of -help commands. Where:
  + -d marks that we perform filter by department
  + -t the type of employee that should be included in the filter. Allowed only “m” and “d” value. For example “top -d -t m” will show the department that has largest number of managers and “top -d -t d” will show the department that has largest number of developers.

You could check a lot of examples here [http://www.mkyong.com/tutorials/jdbc-tutorials](http://www.mkyong.com/tutorials/jdbc-tutorials/).

**Tip:**

I can propose to do something like this to have the opportunity to keep 2 DAO implementations: One of them can work with Files (IO) and other one can work with DB (jdbc). The code could look something like this:

interface DAO<T extends Entity> {

T getById();

void remove(int id);

void add(T employee);

}

interface EmployeeDAO extends DAO<Employee> {

List<Employee> getAll();

List<Employee> find(String departmentName, int age);

}

interface DepartmentDAO extends DAO<Department> {

List<Department> findTop(String employeeType);

}

class DepartmentDBDAO implements DepartmentDAO {

//realization logic here with JDBC

}

class DepartmentFileDAO implements DepartmentDAO {

//realization logic here with FILE

}

class EmployeeDBDao implements EmployeeDAO {

//realization logic here with JDBC

}

class EmployeeFileDao implements EmployeeDAO {

//realization logic here with FILE

}

//usage

//it reads from Database

EmployeeDAO employeeDAO = new EmployeeDBDao();

//it reads from the File

DepartmentDAO departmentDAO = new DepartmentFileDAO();