**Зміст**

Зміст 2

Анотація 3

Вступ 4

1. Постановка задачі 5
2. Діаграми потоків даних 6

2.1. DFD-0 6

2.2. DFD-1 7

2.3. DFD-2 9

2.4. Матриця подій 10

2.5. Бізнес план 11

2.6. ER-діаграма (логічна) 11

2.7. ER-діаграма (фізична) 12

3. Керівництво користувача 13

Текст програми 26

**Анотація**

В даній роботі розроблена та описана інформаційна система «Аеропорт. Управління роботами».

Мета створення цього проекту – оптимізувати та автоматизувати роботу менеджерів аеропорту, а також полегшити їх взаємодію з робітниками аеропорту.

Програмне забезпечення (ПЗ) має інтуїтивно зрозумілий та простий інтерфейс і призначене для персоналу, який має елементарні навички роботи з комп’ютером та офісним програмним забезпеченням.

**Вступ**

На сьогоднішній день створення автоматизованих інформаційних систем є однією з домінуючих сфер розвитку нового інформаційного суспільства. У будь-якій сфері людської діяльності необхідна інформаційна система.

Мета інформатизації та комп’ютеризації – підвищити ефективність роботи. Але просте обладнання робочого місця „розумною” технікою не завжди призводить до підвищення робочих можливостей. Часто з’являється необхідність мати ефективну інформаційну систему, яка розроблена спеціально під бізнес-правила конкретного підприємства, розрахована на виконання не тільки загальних функцій, а й функцій специфічних для даного закладу. Завданням даної курсової роботи було створення такої системи для менеджерів аеропорту.

1. Постановка задачі

Реалізована в цьому проекті автоматизована система обробки даних призначена для використання менеджерами при організації роботи в аеропорті.

Створення програми має на меті полегшити роботу менеджерів технічного відділу, автоматизувати їхню взаємодію з працівниками свого відділу та іншими відділами.

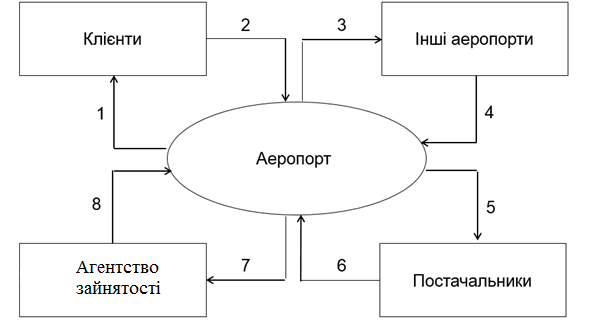
В даному курсовому проекті були реалізовані основні функції інформаційної системи, яка б сприяла оптимізації роботи менеджерів. Зокрема:

* Додавання неподільних робіт з відповідними необхідними вміннями до бази даних;
* Розбиття проблеми, задачі на неподільні роботи;
* Створення працівникам завдань на виконання;
* Створення запиту на працівника до відділу кадрів;

Для коректної роботи програми необхідне виконання відповідних вимог до технічних і програмних засобів, а саме:

* доступ до інтернету;
* процесор з архітектурою x86-64 (Intel з підтримкою EM64T, AMD з підтримкою AMD64);
* оперативна пам’ять 1024 Мб та вище;
* жорсткий диск 40Гб та вище­­;
* SVGA-відеокарта

1. **Діаграми потоків даних** 
   1. **DFD-0**



**Опис потоків даних на DFD-0**

1 – Квитки, інформація про розклад

2 – Бронювання, замовлення та оплата квитків

3 – Узгодження польотів від нашого аеропорту, можливість аварійної посадки на іншому аеропорту

4 – Узгодження польотів до нашого аеропорту, можливість прийняти аварійну посадку

5 – Замовлення матеріальної бази

6 – Виконання замовлення

7 – Вакансії

8 – Резюме, кандидати.

**2.2. DFD-1**

19

22

Інші аеропорти

Клієнти

Каси і довідкове бюро

23



20

21

18

17

16

Диспетчерська

Розклад

Літаки

15

14

13

7

6

Склад

Вакансії

Відділ обслуговування літаків

5

8

10

9

4

3

Відділ кадрів

Відділ матеріального забезпечення

12

Агентство зайнятості

Постачальники

1

2

11

**Опис потоків даних на DFD-1**

1 – Замовлення, гроші.

2 – Товари.

3 – Фіксування нових матеріалів\товарів на складі.

4 – Інформація про наявні матеріали на складі.

5 – Запит на необхідні для ремонту матеріали.

6 – Видача матеріалів.

7 – Запит на працівника з необхідними вміннями.

8 – Працівник.

9 – Інформація про необхідних працівників.

10 – Нові працівники.

11 – Вакансії.

12 – Кандидати.

13 – Інформація про всі літаки та їх технічний стан.

14 – Оновлення інформації про літаки та їх технічний стан.

15 – Інформація про доступні для посадки літаки.

16 – Формування розкладу польотів

17 – Можливість здійснити аварійну посадку

18 – Можливість прийняти аварійну посадку

19 – Узгодження польотів до нашого аеропорту

20 – Узгодження польотів від нашого аеропорту

21 – Розклад польотів

22 – Квитки, інформація про розклад

23 – Бронювання, замовлення та оплата квитків

­­­

**2.3. DFD-2**

Призначені роботи

Великі

задачі

Закінчення\зарахування виконаних завдань

Вакансії

Створення завдань для працівників

Завдання проектів

Планування роботи на проекті, розбиття на атомарні роботи.

Літаки

Створення проекту/великої задачі

Атомарні роботи

Персонал

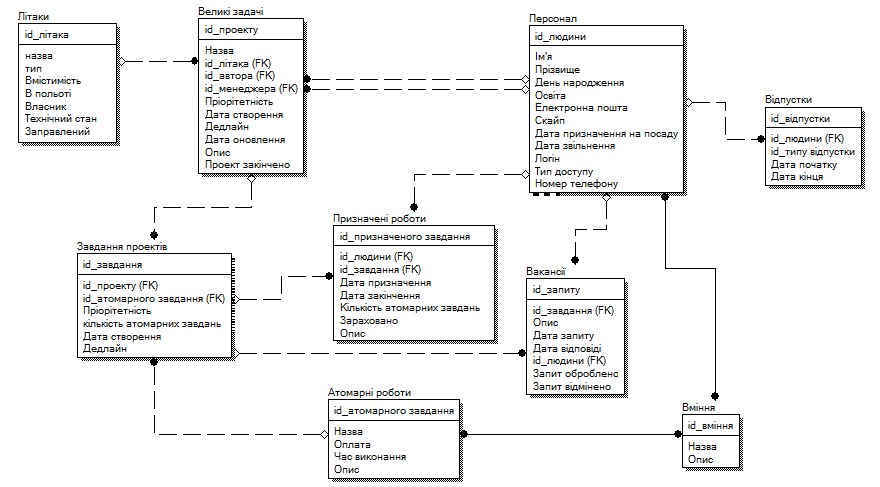
**2.4. Матриця подій**

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Опис події** | **Реакція системи на подію** | **Тип події** |
| **1.** | Створення проекту/великої задачі | Надати користувачу форму для створення проекту про закупку; перевірити правильність введення даних; додати задачу у БД Великі задачі. | N |
| **2.** | Розбиття задачі на завдання | Надати користувачу форму для створення нової задачі на основі атомарних робіт. Перевірити правильність введення даних. Якщо необхідної атомарної роботи нема, то вона додається до БД. Додати задачу у БД Завдання проектів. | N |
| **3.** | Призначення працівникам завдань | Надати користувачу форму для вибору працівника з відповідними вміннями і призначення йому завдання. Додати призначену задачу у БД Призначені роботи | N |
| **4.** | Зарахування виконаного завдання | Ввести необхідні зміни до БД Призначені завдання | N |
| **5.** | Звільнення робітника із завдання | Ввести необхідні зміни до БД Призначені завдання | N |
| **6.** | Створення запиту до відділу кадрів | Надати користувачу форму для створення запиту до відділу кадрів. Додати запис у БД Вакансії | N |
| **7.** | Створення нової атомарної роботи | Надати користувачу форму для створення атомарного завдання та вибору необхідних вмінь. Додати запис у БД Атомарні роботи | N |
| **8.** | Оновлення інформації про літак | Ввести необхідні зміни до БД Літаки | N |

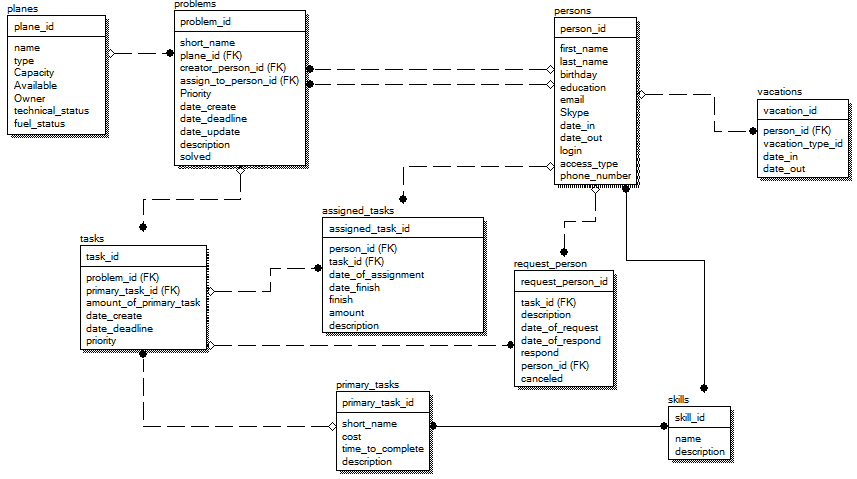
**2.5.** **Бізнес правила**

1. Відділ займається розподіленням робіт тільки між працівниками відділу обслуговування літаків.
2. Призначати роботу можна тільки працівникам, які мають необхідні для даної роботи вміння.
3. Призначати роботу можна тільки працівникам, які не у відпустці і не на лікарняному.
4. Відділ надсилає запит до відділу кадрів тільки у випадку, якщо людини з необхідними вміннями немає.
5. Працівнику оплачується тільки та робота, яка була виконана (зарахована менеджером).
6. Працівника можна звільнити тільки з того завдання, яке ще не було зарахованим менеджером.
7. У разі необхідності певних матеріалів для виконання роботи, працівник звертається до відділу матеріального забезпечення.

**2.6. ER – діаграма**

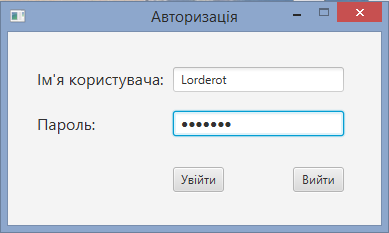
Логічний рівень

**2.7. ER – діаграма**

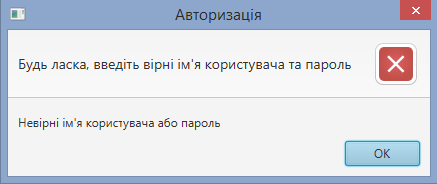
Фізичний рівень

1. **Керівництво користувача**

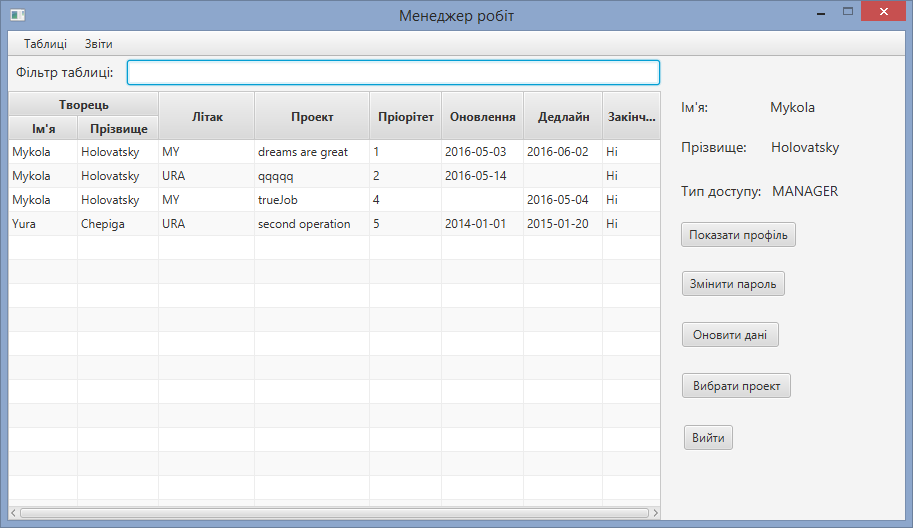
Рівень доступу до бази даних залежить від того, під яким логіном заходить користувач. Є 2 типи доступу: worker – для працівників, manager – для менеджерів. В даній роботі реалізовано інтерфейс тільки для типу доступу manager. При запуску програми з’являється вікно авторизації.



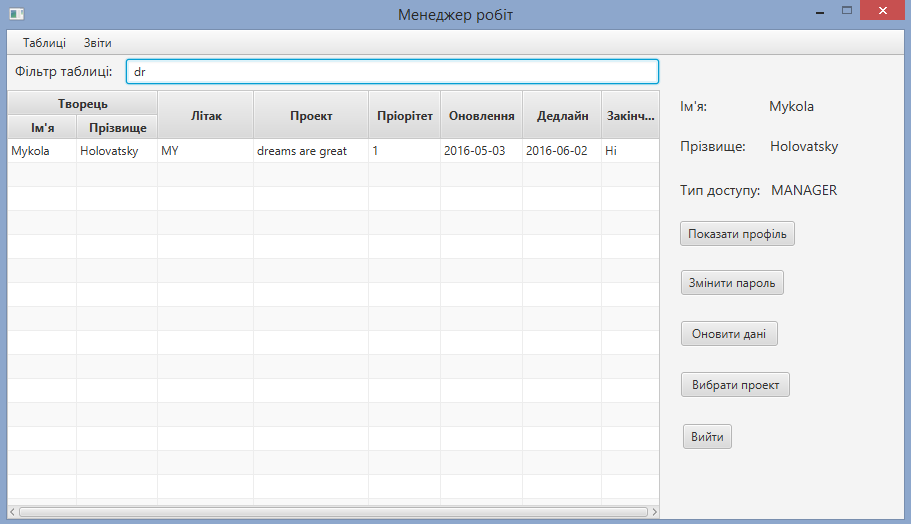
Тут користувач вводить свої логін та пароль. Якщо логін чи пароль введені невірно, то появляється вікно помилки, що повідомляє про це.



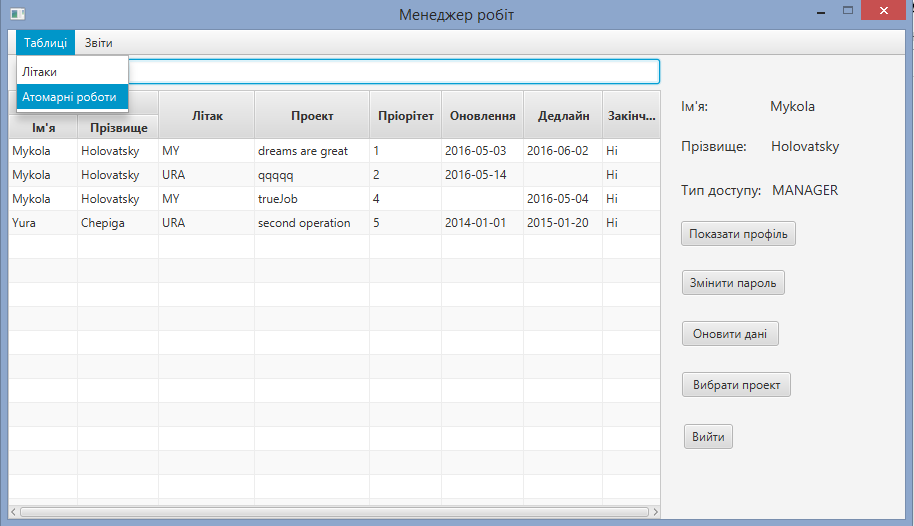
Після того, як менеджер введе вірно свої логін та пароль користувач потрапляє до свого кабінету, де відображаються тільки ті проекти, якими керує даний менеджер.

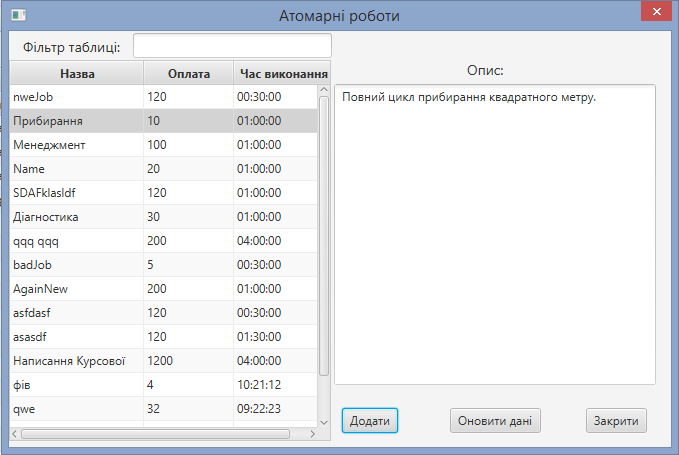


У полі «Фільтр таблиці» він може вводити ключові слова, щоб знайти потрібний проект, якщо таких багато.

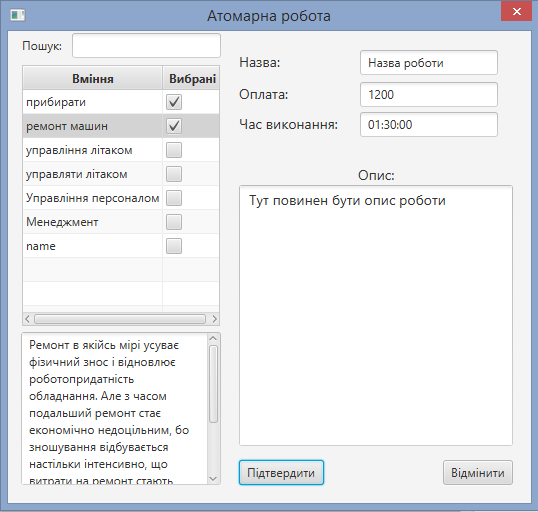
Опишемо основний функціонал програми:

**Створити атомарну роботу:**

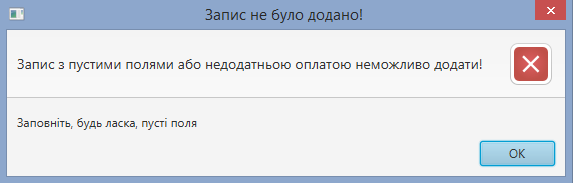
Натисніть на пункт меню «Таблиці» і виберіть таблицю «Атомарні роботи»

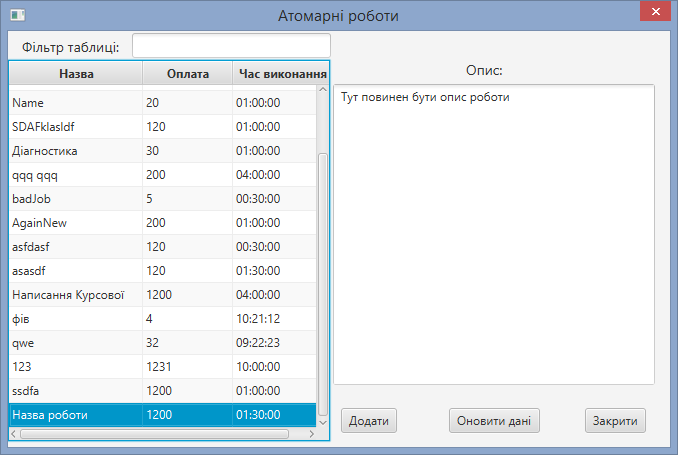
З’явиться таблиця з переліком атомарних робіт. Натисніть на кнопку «Додати»

З’явиться форма для створення атомарної роботи. Введіть назву роботи, оплату (число більше 0) та час виконання у вказаному форматі. Також потрібно зробити опис роботи у відповідному полі та вибрати необхідні вміння для виконання роботи, поставивши галочку, де потрібно. Після цього натисніть кнопку підтвердити.

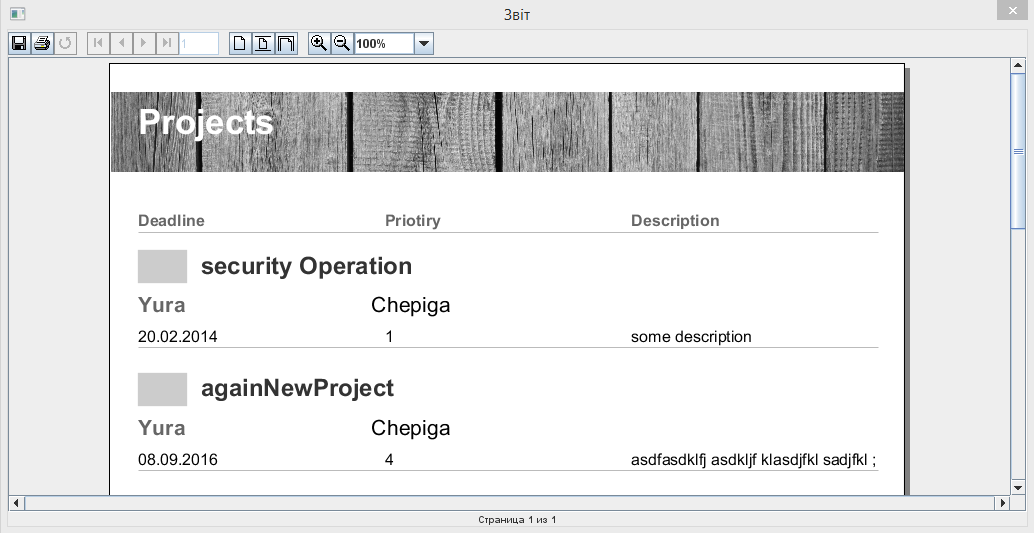


Якщо хоча б одне з полів буде пустим або щось буде некоректно заданим, то з’явиться вікно відповідної помилки, наприклад таке:

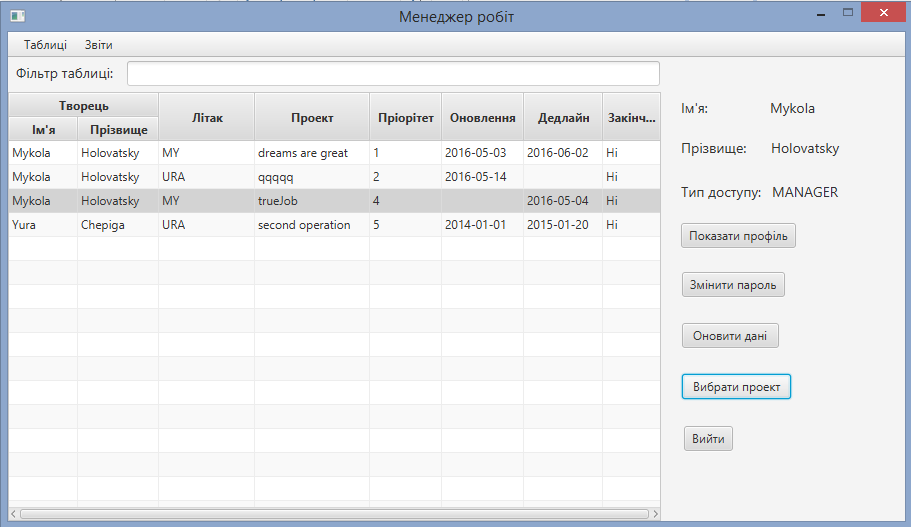
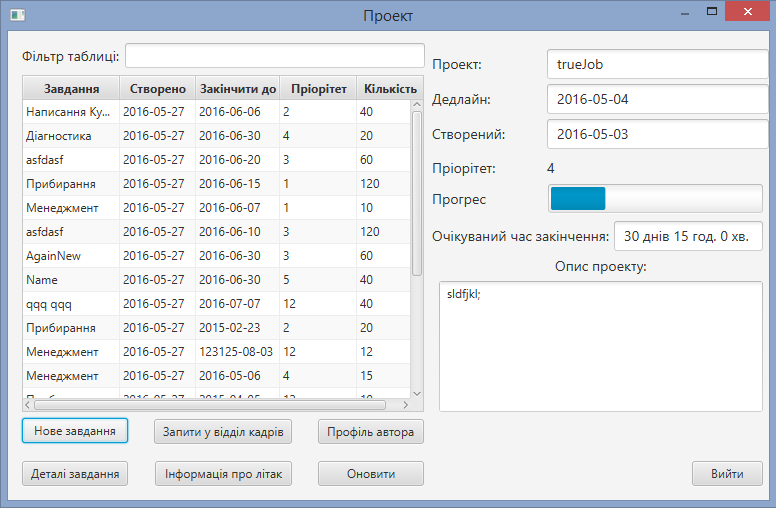


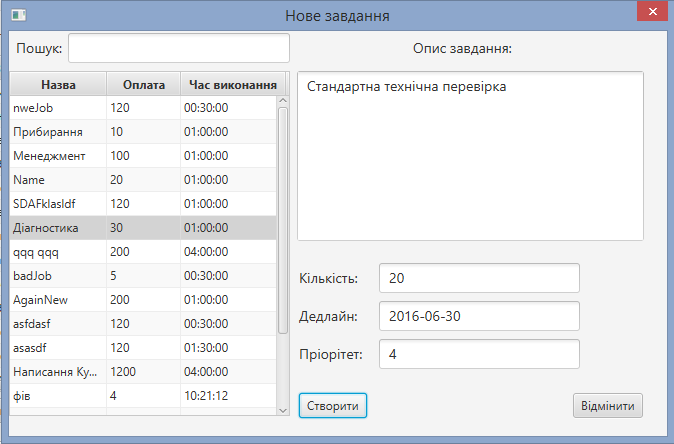
Якщо запис був успішно доданим, то з’явиться попередня форма з доданим в таблиці записом.

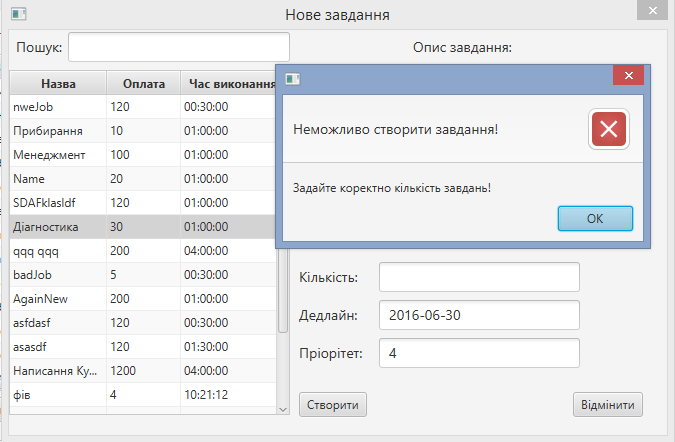
**Формування звіту:**

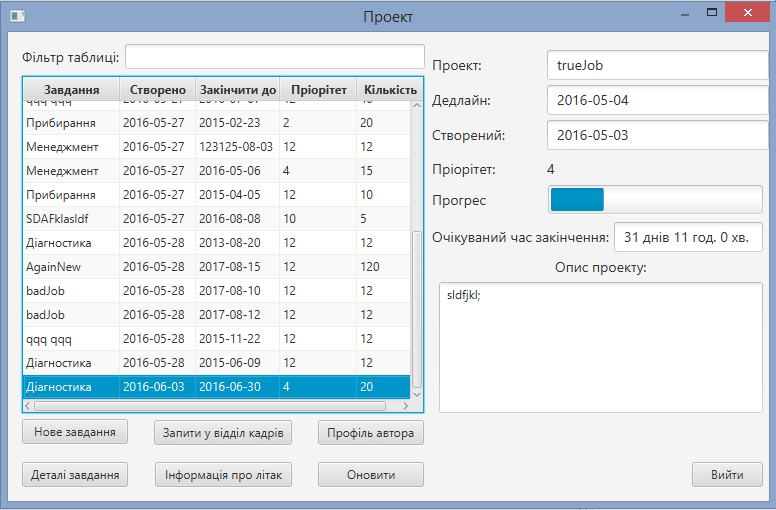
 Виберіть пункт меню «звіти» та натисніть на таблицю, по якій необхідно згенерувати звіт. Відкриється вікно зі сформованим звітом, який можна зберегти у потрібно форматі (наприклад, .pdf), а також роздрукувати.

**Планування роботи на проекті або розбиття проекту на завдання:**

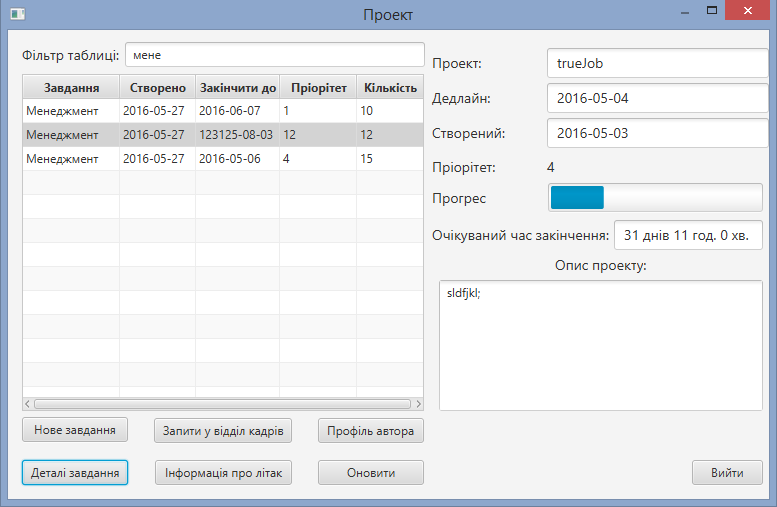
Виберіть необхідний для роботи проект у таблиці та натисніть кнопку «Вибрати проект». З’явиться форма з інформацією про проект. У таблиці буде перелік завдань, які необхідно виконати для закінчення проекту. Справа від таблиці знаходиться інформація про проект. Для розбиття проекту на завдання, потрібно створювати необхідні для виконання проекту завдання, натиснувши на клавішу «Нове завдання».

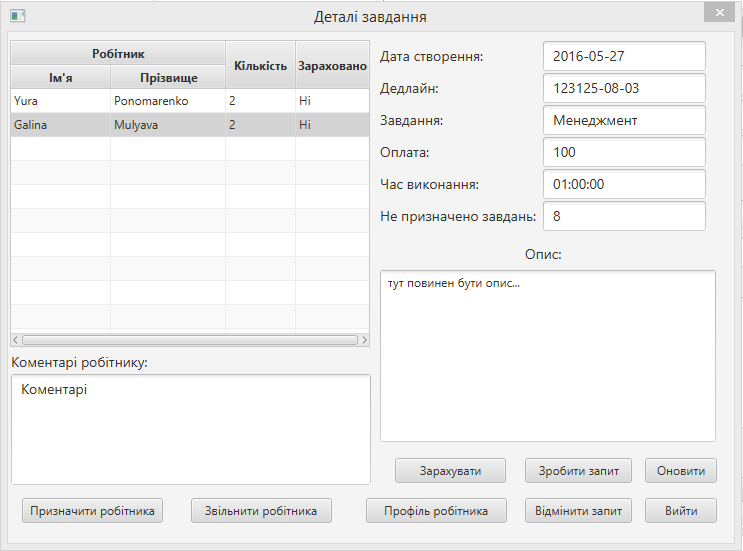
З’явиться вікно з переліком атомарних робіт. Виберіть необхідну атомарну роботу. Введіть кількість атомарних робіт, з якої складається ваше завдання. Введіть в поле Дедлайн дату, до якої завдання повинно бути виконано. У поле пріоритет введіть додатне число, яке буде відображати терміновість даного завдання у порівнянні з іншими. Натисніть кнопку «Створити».

Якщо якесь з полів буде некоректно заданим або не буде вибрана атомарна робота, то програма повідомить вам про це у вигляді інформаційного вікна про помилку.

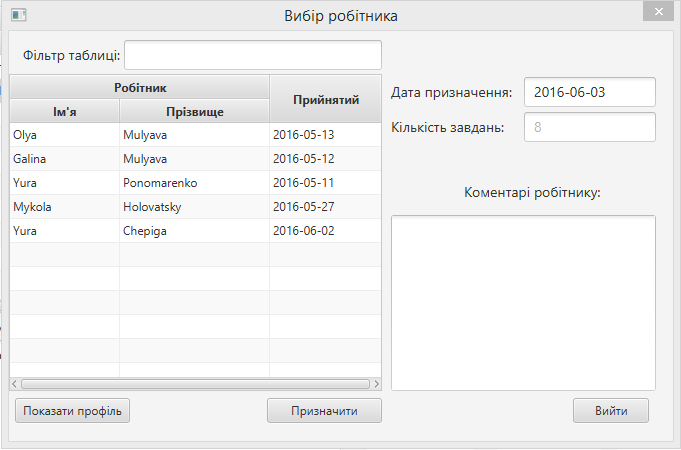
Якщо все коректно було введено і запис був успішно доданим, то з’явиться попередня форма з доданим до таблиці записом та оновленими даними про проект.

**Призначити завдання:**

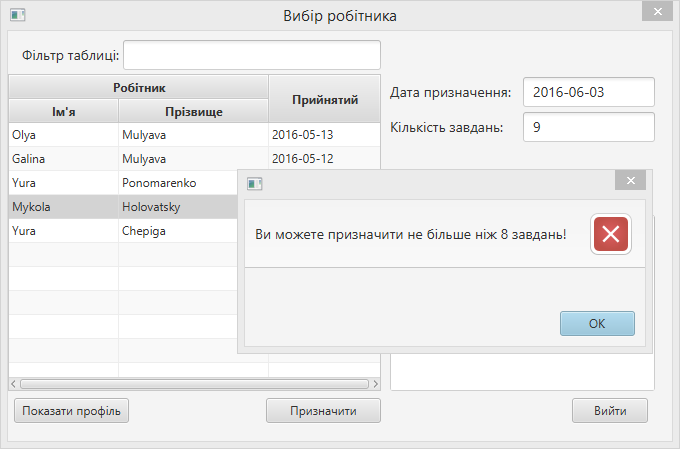
Виберіть потрібне завдання і натисніть кнопку «Деталі завдання».

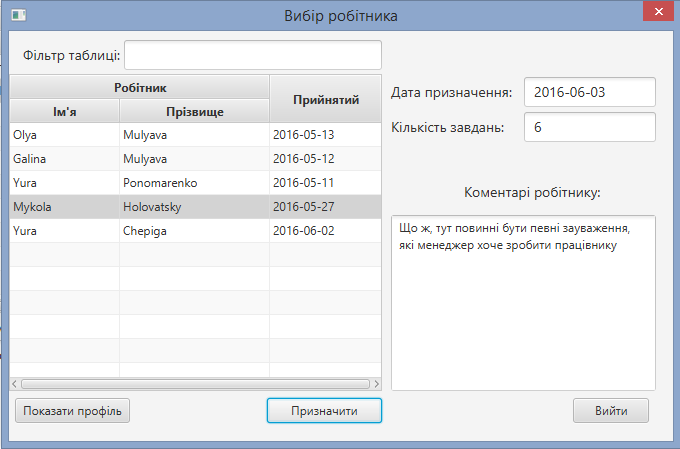
Відкриється нове вікно з деталями завдання. У таблиці будуть вказані люди, яким вже призначена певна частина даного завдання.

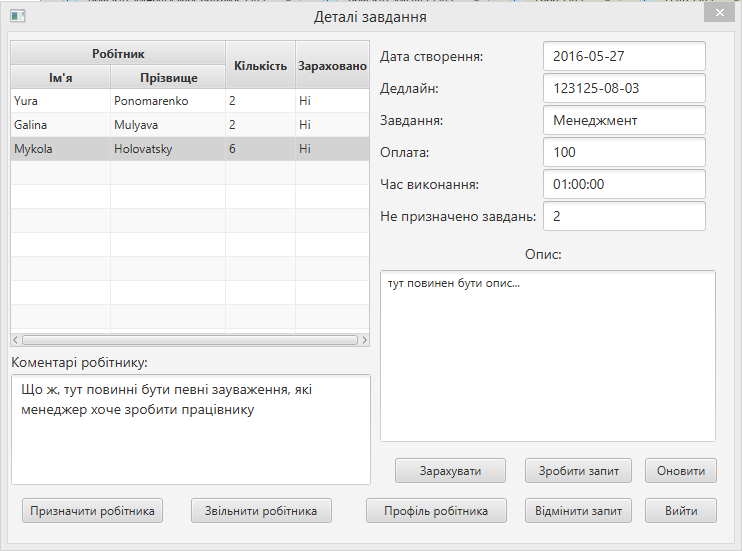
Натисніть кнопку «Призначити робітника».

Відкриється форма, де в таблиці будуть вказані доступні для виконання завдання робітники з необхідними вміннями. Справа від таблиці знаходиться поле «Кількість завдань», куди потрібно ввести кількість атомарних робіт, з яких складається завдання, призначене робітнику.

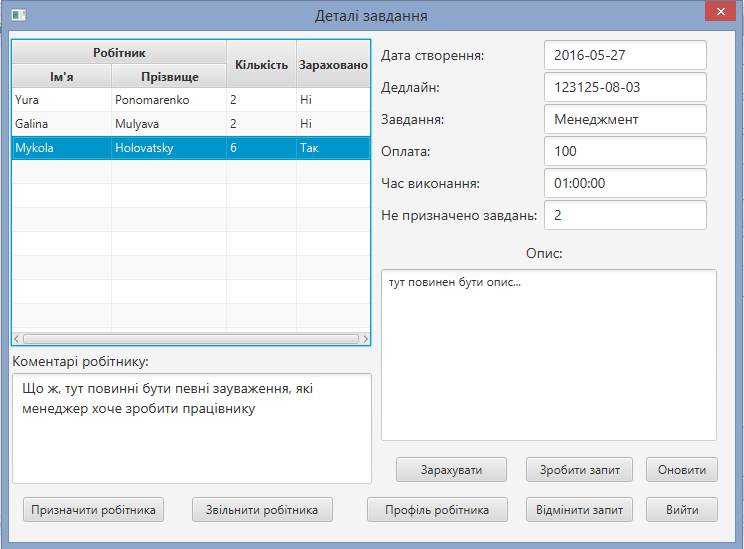
Ця кількість повинна не перевищувати ту кількість атомарних робіт, що залишилось виконати для завершення завдання проекту. Інакше програма повідомить про це таким інформаційним вікном:



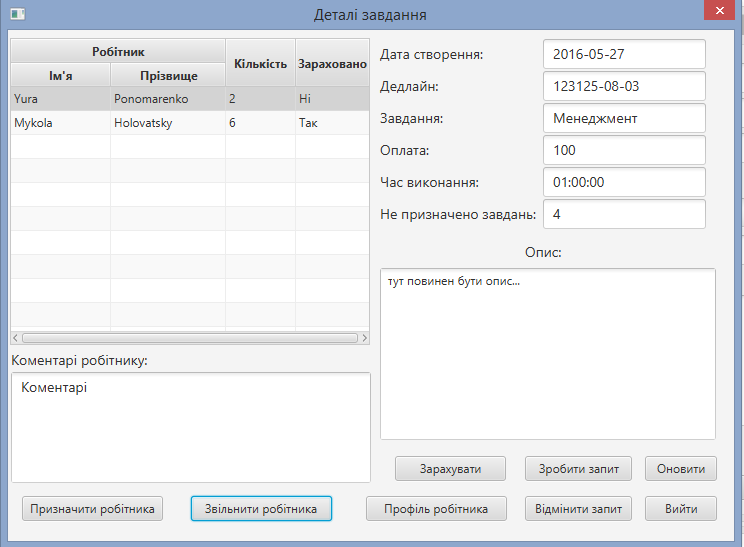
Нижче можна ввести певні коментарі та зауваження для працівника. Потім натисніть кнопку «Призначити»

У разі коректного введення даних, запис буде успішно доданим і перед користувачем з’явиться попередня форма з оновленими даними:

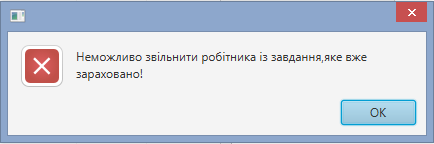
**Зарахування роботи менеджером:**

 Щоб зарахувати роботи, виберіть працівника і натисніть кнопку «Зарахувати». В таблиці у вибраного запису значення поля «Зараховано» зміниться на «Так».

**Звільнення працівника із завдання:**

Щоб звільнити працівника із завдання, виберіть потрібного працівника і натисніть кнопку «Звільнити робітника». При успішному звільненні працівника, відповідний запис зникне з таблиці, а дані будуть оновлені, зокрема збільшиться кількість непризначених завдань, що відображається справа від таблиці.

Однак, робітника можна звільнити тільки з ще незарахованого завдання. Програма про це повідомить наступним інформаційним вікном:



**Текст програми**

Для розробки інформаційної системи було використано СУБД Postgres, яка дозволяє легко адмініструвати базу даних, зокрема створювати таблиці, тригери, процедури. Сама програма була реалізована на мові програмування – Java, що дозволяє користуватись даною програмою на будь-якій операційній системі.

***Користувач***

create group manager;

create group worker;

create user Lorderot in group manager password ‘12’;

create user Ura\_Mulyava in group worker password ‘12’;

create user Olya in group worker password ‘12’;

create user Galina in group worker password ‘12’;

GRANT SELECT ON ALL TABLES IN SCHEMA public TO manager;

GRANT SELECT ON assigned\_tasks TO worker;

GRANT SELECT ON primary\_tasks to worker;

GRANT SELECT ON persons to worker;

GRANT SELECT ON tasks to worker;

GRANT insert, update on assigned\_tasks to manager;

GRANT insert, update on problems to manager;

GRANT insert on con\_primary\_task\_skill to manager;

GRANT update on planes to manager;

GRANT insert on tasks to manager;

***Процедура оцінки часу до завершення завдання***

create or replace function get\_time\_to\_finish\_task(id\_of\_task integer)

returns bigint as

$$

declare

time bigint := 0;

time\_to\_finish bigint := 0;

finished\_amount\_of\_task integer := 0;

amount\_of\_task integer:=0;

primary\_id integer := 0;

begin

select amount\_of\_primary\_task into amount\_of\_task

from tasks where task\_id = id\_of\_task;

select sum(amount) into finished\_amount\_of\_task

from assigned\_tasks where task\_id = id\_of\_task AND (finish = true);

IF finished\_amount\_of\_task IS NULL then

finished\_amount\_of\_task := 0;

END IF;

select primary\_task\_id into primary\_id

from tasks where task\_id = id\_of\_task;

select time\_to\_complete into time

from primary\_tasks where primary\_task\_id = primary\_id;

time\_to\_finish = time \* (amount\_of\_task - finished\_amount\_of\_task);

IF time\_to\_finish IS NULL then

return 0;

END IF;

return time\_to\_finish;

end

$$ language plpgsql;

***Процедура оцінки часу до закінчення проекту***

create or replace function get\_time\_to\_finish\_project(project\_id integer)

returns bigint as

$$

declare

task RECORD;

acc\_time bigint := 0;

begin

for task IN select \* from tasks where problem\_id = project\_id

LOOP

acc\_time := acc\_time + get\_time\_to\_finish\_task(task.task\_id);

END LOOP;

return acc\_time;

end

$$ language plpgsql;

***Процедура оцінки прогресу виконання завдання***

create or replace function get\_task\_progress(id\_of\_task integer)

returns float AS

$$

declare

finished\_amount\_of\_tasks integer := 0;

amount\_of\_tasks integer := 0;

begin

select sum(amount) into finished\_amount\_of\_tasks

from assigned\_tasks where task\_id = id\_of\_task AND finish = true;

select amount\_of\_primary\_task into amount\_of\_tasks

from tasks where task\_id = id\_of\_task;

IF amount\_of\_tasks = 0 OR finished\_amount\_of\_tasks is NULL THEN

return 0.0;

END IF;

return finished\_amount\_of\_tasks::float / amount\_of\_tasks::float;

end

$$ language plpgsql;

***Процедура оцінки прогресу виконання проекту***

create or replace function get\_project\_progress(project\_id Integer)

returns float AS

$$

declare

task RECORD;

acc\_finished\_tasks float := 0.0;

acc\_amount\_of\_tasks float := 0.0;

finished\_amount\_of\_tasks integer := 0;

amount\_of\_tasks integer := 0;

begin

for task IN select \* from tasks where problem\_id = project\_id

LOOP

select sum(amount) into finished\_amount\_of\_tasks

from assigned\_tasks where task\_id = task.task\_id AND finish = true;

select amount\_of\_primary\_task into amount\_of\_tasks

from tasks where task\_id = task.task\_id;

IF finished\_amount\_of\_tasks is NULL THEN

finished\_amount\_of\_tasks := 0;

END IF;

acc\_amount\_of\_tasks := acc\_amount\_of\_tasks + amount\_of\_tasks;

acc\_finished\_tasks := acc\_finished\_tasks + finished\_amount\_of\_tasks;

END LOOP;

IF acc\_amount\_of\_tasks = 0 THEN

return 0;

END IF;

return (acc\_finished\_tasks::float / acc\_amount\_of\_tasks::float);

end

$$ language plpgsql;

***Процедура, яка повертає людей, які зараз працюють на проекті і кількість завдань, що вони виконують***

create or replace function getProjectWorkers(project\_id Integer)

returns table(first\_name varchar(100), last\_name varchar(100), number\_of\_tasks bigint) as

$$

begin

return QUERY select cw.first\_name as name, cw.last\_name as surname,

count(at.task\_id) as number\_of\_tasks

from current\_workers as cw

join assigned\_tasks as at on cw.person\_id = at.person\_id

where (at.date\_finish is null OR at.finish = true) AND

cw.person\_id IN (select person\_id from assigned\_tasks

where task\_id IN (select task\_id from tasks where problem\_id = project\_id))

group by name, surname

order by number\_of\_tasks desc, name asc;

end

$$ language plpgsql;

***Розріз даних(View) – список доступних людей***

create view current\_workers as select \* from persons where access\_type = 'worker' AND person\_id NOT IN (select person\_id from vacations where current\_date > date\_in AND current\_date < date\_out);

**Код курсової роботи:**

**package** DAO;  
  
**import** model.AssignedTask;  
**import** org.hibernate.Query;  
**import** org.hibernate.Session;  
**import** org.hibernate.Transaction;  
**import** util.HibernateUtil;  
**import** java.util.List;  
**public class** AssignedTaskDAO {  
 **public void** addAssignedTask(AssignedTask assignedTask) {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 String sqlQuery = **"insert into assigned\_tasks(person\_id, task\_id, "** +  
 **"date\_of\_assignment, finish, amount, description) "** +  
 **"values(:person\_id, :task\_id, :date\_of\_assignment, "** +  
 **":finish, :amount, :description);"**;  
 Query query = session.createSQLQuery(sqlQuery)  
 .addEntity(AssignedTask.**class**);  
 query.setParameter(**"person\_id"**, assignedTask.getPerson().getIdentifier());  
 query.setParameter(**"task\_id"**, assignedTask.getTask().getIdentifier());  
 query.setParameter(**"date\_of\_assignment"**, assignedTask.getAssignmentDate());  
 query.setParameter(**"finish"**, assignedTask.getFinished());  
 query.setParameter(**"amount"**, assignedTask.getAmount());  
 query.setParameter(**"description"**, assignedTask.getDescription());  
 query.executeUpdate();  
 transaction.commit();  
 session.close();  
 }  
 **public** List<AssignedTask> findAssignedTasks(Integer taskIdentifier) {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 String sqlQuery = **"select \* from assigned\_tasks where "** +  
 **"(date\_finish IS NULL OR finish = true) AND "** +  
 **"task\_id = "** + taskIdentifier + **";"**;  
 Query query = session.createSQLQuery(sqlQuery)  
 .addEntity(AssignedTask.**class**);  
 List<AssignedTask> list = query.list();  
 transaction.commit();  
 session.close();  
 **if** (list == **null**) {  
 **throw new** NullPointerException(**"DB returns null list"**);  
 }  
 **return** list;  
 }  
 **public void** updateAssignedTask(AssignedTask assignedTask) {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 session.update(assignedTask);  
 transaction.commit();  
 session.close();  
 }  
}

**package** DAO;  
**import** model.Person;  
**import** org.hibernate.Query;  
**import** org.hibernate.Session;  
**import** org.hibernate.Transaction;  
**import** util.HibernateUtil;  
**import** java.util.List;  
**public class** PersonDAO {  
 **public** Person findPersonByLogin(String login) {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 Query query = session.createSQLQuery(**"select \* from persons where login = :login"**)  
 .addEntity(Person.**class**);  
 query.setParameter(**"login"**, login);  
 List queryList = query.list();  
 transaction.commit();  
 session.close();  
 **if** (queryList == **null**) {  
 **throw new** NullPointerException(**"DB returns null list."**);  
 }  
 **if** (!queryList.isEmpty()) {  
 **return** (Person) queryList.get(0);  
 } **else** {  
 **return null**;  
 }  
 }  
 **public** List<Person> findAvailableWorkers() {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 String sqlQuery = **"select \* from current\_workers;"**;  
 Query query = session.createSQLQuery(sqlQuery)  
 .addEntity(Person.**class**);  
 List<Person> persons = query.list();  
 transaction.commit();  
 session.close();  
 **if** (persons == **null**) {  
 **throw new** NullPointerException(**"DB returns null list"**);  
 }  
 **return** persons;  
 }  
 **public void** updatePersonPassword(Person person) {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 String sqlQuery = **"alter user "** + person.getLogin().toLowerCase()  
 + **" with password '"** + person.getPassword() + **"';"** ;  
 Query query = session.createSQLQuery(sqlQuery);  
 query.executeUpdate();  
 transaction.commit();  
 session.close();  
 }  
}

**package** DAO;  
**import** model.Plane;  
**import** org.hibernate.Query;  
**import** org.hibernate.Session;  
**import** org.hibernate.Transaction;  
**import** util.HibernateUtil;  
**import** java.util.List;  
**public class** PlaneDAO {  
 **public** List<Plane> findAll() {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 String sqlQuery = **"select \* from planes;"**;  
 Query query = session.createSQLQuery(sqlQuery).addEntity(Plane.**class**);  
 List<Plane> list = query.list();  
 transaction.commit();  
 session.close();  
 **if** (list == **null**) {  
 **throw new** NullPointerException(**"DB returns null list."**);  
 }  
 **return** list;  
 }  
 **public void** update(Plane plane) {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 session.update(plane);  
 transaction.commit();  
 session.close();  
 }  
  
}

**package** DAO;  
**import** model.PrimaryTask;  
**import** org.hibernate.Query;  
**import** org.hibernate.Session;  
**import** org.hibernate.Transaction;  
**import** org.hibernate.exception.ConstraintViolationException;  
**import** util.HibernateUtil  
**import** java.util.List;  
**public class** PrimaryTaskDAO {  
 **public** List<PrimaryTask> findAll() {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 Query query = session.createSQLQuery(**"select \* from primary\_tasks;"**)  
 .addEntity(PrimaryTask.**class**);  
 List list = query.list();  
 transaction.commit();  
 session.close();  
 **if** (list == **null**) {  
 **throw new** NullPointerException(**"DB returns null list."**);  
 }  
 **return** list;  
 }  
 **public** PrimaryTask findPrimaryTaskByName(String primaryTask) {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 String sqlQuery = **"select \* from primary\_tasks where short\_name = '"** + primaryTask + **"';"**;  
 Query query = session.createSQLQuery(sqlQuery)  
 .addEntity(PrimaryTask.**class**);  
 List<PrimaryTask> list = query.list();  
 transaction.commit();  
 session.close();  
 **if** (list == **null**) {  
 **throw new** NullPointerException(**"DB return null list"** );  
 }  
 **if** (list.size() > 1) {  
 System.***err***.println(**"DB return list with size "** + list.size() + **"!!!"**);  
 }  
 **return** list.get(0);  
 }  
 **public boolean** add(PrimaryTask primaryTask)  
 **throws** ConstraintViolationException {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 String sqlQuery = **"insert into primary\_tasks("** + **"cost, description, short\_name, time\_to\_complete)"** + **" values(:cost,:description,:name,:time);"**;  
 Query query = session.createSQLQuery(sqlQuery);  
 query.setParameter(**"cost"**, primaryTask.getCost());  
 query.setParameter(**"description"**, primaryTask.getDescription());  
 query.setParameter(**"name"**, primaryTask.getName());  
 query.setParameter(**"time"**,primaryTask.getTimeToComplete());  
 **int** result = 0;  
 **try** {  
 result = query.executeUpdate();  
 } **catch** (Exception e) {  
 **throw** e;  
 } **finally** {  
 transaction.commit();  
 session.close();  
 }  
 **if** (result > 0) {  
 **return true**;  
 }  
 **return false**;  
 }  
}

**package** DAO;  
**import** model.Problem;  
**import** org.hibernate.Query;  
**import** org.hibernate.Session;  
**import** org.hibernate.Transaction;  
**import** util.HibernateUtil;  
**import** java.sql.\*;  
**import** java.util.List;  
**public class** ProblemDAO {  
 **public** List<Problem> findProblemsByAssignedPerson(Integer personIdentifier) {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 String sqlQuery = **"select \* from problems where assign\_to\_person\_id = "** + personIdentifier + **";"**;  
 Query query = session.createSQLQuery(sqlQuery).addEntity(Problem.**class**);  
 List<Problem> result = query.list();  
 transaction.commit();  
 session.close();  
 **if** (result == **null**) {  
 **throw new** NullPointerException(**"DB returns null"**);  
 }  
 **return** result;  
 }  
 **public** Double getProgress(Integer problem) {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 String sqlQuery = **"select get\_project\_progress(?);"**;  
 Connection connection = HibernateUtil.*getConnection*(session);  
 PreparedStatement statement;  
 Double result = **null**;  
 ResultSet resultSet;  
 **try** {  
 statement = connection.prepareStatement(sqlQuery);  
 statement.setInt(1, problem);  
 resultSet = statement.executeQuery();  
 resultSet.next();  
 result = resultSet.getDouble(1);  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 } **finally** {  
 transaction.commit();  
 session.close();  
 }  
 **return** result;  
 }  
 **public** Long getTime(Integer problem) {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 String sqlQuery = **"select get\_time\_to\_finish\_project(?);"**;  
 Connection connection = HibernateUtil.*getConnection*(session);  
 PreparedStatement statement;  
 Long result = **null**;  
 ResultSet resultSet;  
 **try** {  
 statement = connection.prepareStatement(sqlQuery);  
 statement.setInt(1, problem);  
 resultSet = statement.executeQuery();  
 resultSet.next();  
 result = resultSet.getLong(1);  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 } **finally** {  
 transaction.commit();  
 session.close();  
 }  
 **return** result;  
 }  
}

**package** DAO;  
**import** model.Request;  
**import** org.hibernate.Query;  
**import** org.hibernate.Session;  
**import** org.hibernate.Transaction;  
**import** util.HibernateUtil;  
**import** java.util.List;  
**public class** RequestDAO {  
 **public** List<Request> findRequestsByProblem(Integer problemIdentifier) {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 String sqlQuery = **"select \* from request\_person where task\_id IN ("** +  
 **"select task\_id from tasks where problem\_id = "** + problemIdentifier + **" AND canceled = false);"**;  
 Query query = session.createSQLQuery(sqlQuery).addEntity(Request.**class**);  
 List<Request> requests = query.list();  
 transaction.commit();  
 session.close();  
 **if** (requests == **null**) {  
 **throw new** NullPointerException(**"DB returns null list"**);  
 }  
 **return** requests;  
 }  
 **public** Request findCurrentRequestByTaskID(Integer taskID) {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 String sqlQuery = **"select \* from request\_person where canceled = false "** +  
 **"AND task\_id = "** + taskID + **";"**;  
 Query query = session.createSQLQuery(sqlQuery)  
 .addEntity(Request.**class**);  
 List<Request> list = query.list();  
 transaction.commit();  
 session.close();  
 **if** (list == **null**) {  
 **throw new** NullPointerException(**"DB return null list"**);  
 }  
 **if**(list.size() > 1) {  
 System.***err***.println(**"DB RETURN + "** + list.size() + **" REQUESTS!!!"**);  
 }  
 **if** (list.isEmpty()) {  
 **return null**;  
 }  
 **return** list.get(0);  
 }  
 **public void** addRequest(Request request) {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 String sqlQuery = **"insert into request\_person(task\_id, date\_of\_request, "** +  
 **"description, canceled) values(:task, :date, :description, false);"**;  
 Query query = session.createSQLQuery(sqlQuery)  
 .addEntity(Request.**class**);  
 query.setParameter(**"task"**, request.getTask().getIdentifier());  
 query.setParameter(**"date"**, request.getDateOfRequest());  
 query.setParameter(**"description"**, request.getDescription());  
 query.executeUpdate();  
 transaction.commit();;  
 session.close();  
 }  
  
 **public void** update(Request request) {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 session.update(request);  
 transaction.commit();  
 session.close();  
 }  
}

**package** DAO;  
**import** model.Skill;  
**import** org.hibernate.Query;  
**import** org.hibernate.Session;  
**import** org.hibernate.Transaction;  
**import** util.HibernateUtil;  
**import** java.util.List;  
**public class** SkillDAO {  
 **public** List<Skill> getSkillsByPrimaryTask(Integer primaryTaskIdentifier) {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 String sqlQuery = **"select \* from skills where skill\_id IN (select skill\_id "** +  
 **"from con\_primary\_task\_skill where primary\_task\_id = "** + primaryTaskIdentifier + **");"**;  
 Query query = session.createSQLQuery(sqlQuery).addEntity(Skill.**class**);  
 List<Skill> result = query.list();  
 transaction.commit();  
 session.close();  
 **if** (result == **null**) {  
 **throw new** NullPointerException(**"DB return null list"**);  
 }  
 **return** result;  
 }  
 **public** List<Skill> findAllSkills() {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 String sqlQuery = **"select \* from skills;"**;  
 Query query = session.createSQLQuery(sqlQuery).addEntity(Skill.**class**);  
 List<Skill> result = query.list();  
 transaction.commit();  
 session.close();  
 **if** (result == **null**) {  
 **throw new** NullPointerException(**"DB return null list"**);  
 }  
 **return** result;  
 }  
 **public void** addSkillsToPrimaryTasks(Integer skill, Integer primaryTask) {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 String sqlQuery = **"insert into con\_primary\_task\_skill(skill\_id, primary\_task\_id)"** +  
 **" values(:skill, :task);"**;  
 Query query = session.createSQLQuery(sqlQuery);  
 query.setParameter(**"skill"**, skill);  
 query.setParameter(**"task"**, primaryTask);  
 query.executeUpdate();  
 transaction.commit();  
 session.close();  
 }  
}

**package** DAO;  
**import** model.Task;  
**import** org.hibernate.Query;  
**import** org.hibernate.Session;  
**import** org.hibernate.Transaction;  
**import** util.HibernateUtil;  
**import** java.util.List;  
**public class** TaskDAO {  
 **public** List<Task> findTasksByProblem(Integer problemIdentifier) {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 String sqlQuery = **"select \* from tasks where problem\_id = "** + problemIdentifier + **";"**;  
 Query query = session.createSQLQuery(sqlQuery).addEntity(Task.**class**);  
 List<Task> result = query.list();  
 transaction.commit();  
 session.close();  
 **if** (result == **null**) {  
 **throw new** NullPointerException(**"DB returns null"**);  
 }  
 **return** result;  
 }  
 **public void** addTask(Task task) {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 String sqlQuery = **"insert into tasks(problem\_id, primary\_task\_id, "** + **"date\_create, date\_deadline, amount\_of\_primary\_task,"** + **"priority) values(:problem, :primaryTask, "** + **":creation, :deadline, :amount, :priority);"**;  
 Query query = session.createSQLQuery(sqlQuery).addEntity(Task.**class**);  
 query.setParameter(**"problem"**, task.getProblem().getIdentifier());  
 query.setParameter(**"primaryTask"**, task.getPrimaryTask().getIdentifier());  
 query.setParameter(**"creation"**, task.getCreateDate());  
 query.setParameter(**"deadline"**, task.getDeadline());  
 query.setParameter(**"amount"**, task.getAmount());  
 query.setParameter(**"priority"**, task.getPriority());  
 query.executeUpdate();  
 transaction.commit();  
 session.close();  
 }  
 **public** Task findTaskByIdentifier(Integer taskIdentifier) {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 String sqlQuery = **"select \* from tasks where task\_id = "** + taskIdentifier + **";"**;  
 Query query = session.createSQLQuery(sqlQuery)  
 .addEntity(Task.**class**);  
 List<Task> result = query.list();  
 transaction.commit();  
 session.close();  
 **if** (result == **null**) {  
 **throw new** NullPointerException(**"DB returns null list"**);  
 }  
 **if** (result.size() > 1) {  
 System.***err***.println(**"DB returns "** + result.size() + **" tasks by task\_id = "** + taskIdentifier);  
 }  
  
 **if** (result.size() == 0) {  
 **return null**;  
 }  
 **return** result.get(0);  
 }  
 **public void** updateTask(Task task) {  
 Session session = HibernateUtil.*getSession*();  
 Transaction transaction = session.beginTransaction();  
 session.update(task);  
 transaction.commit();  
 session.close();}  
}