Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра Информатики

Дисциплина: Программирование (КПиЯП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

Учёт рабочего времени программиста

БГУИР КП 1-53 01 07 10724 ПЗ

Студент: гр. 052003

Козлов А.В.

Руководитель: ассистент кафедры информатики Шиманский В.В.

Минск 2011

Содержание

1. Аннотация 03
2. Введение 03
3. Теоретические основы 04
4. Инкапсуляция 04
5. Абстракция данных 04
6. Наследование 05
7. Полиморфизм 05
8. Руководство для программиста 06
9. Руководство для пользователя 08
10. Заключение 15
11. Список используемой литературы 16
12. Листинг программы 17

Аннотация

Данный курсовой проект может быть использован в сфере менеджмента программных продуктов и предназначен для контроля рабочего времени программиста. Используя данный программный продукт можно легко подсчитать КПД конкретного программиста, проанализировать, в какое время суток его кпд наивысший, а также можно найти применение для подсчете заработной платы (при почасовой оплате труда).

Проект разрабатывалсяы на языке программирования С++. В данной работе использовались такие технологии, как инкапсуляция, абстракция данных, наследование, полиморфизм.

Введение

В силу необходимости учета рабочего времени прораммиста возникла идея по разработке программного обеспечения для автоматизации контроля. По приходу на работу пользователь включает компьютер и запускает программу «KeeperTime». После авторизации (путем ввода индивидуального логина/пароля) появляется окно с меню. В данном списке можно просмотреть текущие задачи, которые необходимо реализовать. Пользователь находит задачу, у которой наивысший приоритет и начинает её выполнять. После выполнения, необходимо написать отчет, в котором изложить все подробности реализации, как запустить задачу и т. п. При сохранении отчета, программа показывает «незанятое» время, часть (или собственно все) нужно установить в качестве времени, затраченном на выполнение данной задачи. «Незанятое» время — это количество времени, прошедшее с момента прихода на работу минус время, потраченное на выполнение других задач (т. е. другими словами говоря несписанное время). В идеале, конечно же, по окончанию рабочего дня «незанятого» времени не должно быть.

Над каждым программистом закреплен менеджер. В контексте данной программы, менеджер — это человек, руководящий работой небольшой командой программистов; которой по приходу на работу, заходит в свой аккаунт в программе, после чего на дисплее появляется список реализуемых его командой проектов. Выбрав проект, менеджер разбивает его на более мелкие проекты - «задачи», которые «вешает» конкретному программисту из своей команды.

Теоретические основы

**Инкапсуляция**

**Инкапсуля́ция** — свойство языка программирования, позволяющее пользователю не задумываться о сложности реализации используемого программного компонента (то, что у него внутри), а взаимодействовать с ним посредством предоставляемого интерфейса (публичных методов и членов), а также объединить и защитить жизненно важные для компонента данные. При этом пользователю предоставляется только спецификация ([интерфейс](http://ru.wikipedia.org/wiki/Интерфейс_(объектно-ориентированное_программирование))) объекта.

* Пользователь может взаимодействовать с объектом только через этот интерфейс. Реализуется с помощью ключевого слова: public.
* Пользователь не может использовать закрытые данные и методы. Реализуется с помощью ключевых слов: private, protected, internal.

Инкапсуляция — один из четырёх важнейших механизмов [объектно](http://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование)-ориентированного программирования (наряду с [абстракцией](http://ru.wikipedia.org/wiki/Абстракция_данных), [полиморфизмом](http://ru.wikipedia.org/wiki/Полиморфизм_в_языках_программирования) и [наследованием](http://ru.wikipedia.org/wiki/Наследование_(программирование))).

Сокрытие реализации целесообразно применять в следующих случаях:

* предельная локализация изменений при необходимости таких изменений,
* прогнозируемость изменений (какие изменения в коде надо сделать для заданного изменения функциональности) и прогнозируемость последствий изменений.

**Абстракция данных**

**Абстра́кция — в** [**объектно-ориентированном программировании**](http://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование) **это придание** [**объекту**](http://ru.wikipedia.org/wiki/Объект) **характеристик, которые четко определяют его** [**концептуальные**](http://ru.wikipedia.org/wiki/Концепт) **границы, отличая от всех других объектов. Основная идея состоит в том, чтобы отделить способ использования составных объектов данных от деталей их реализации в виде более простых объектов, подобно тому, как** [**функциональная**](http://ru.wikipedia.org/wiki/Функциональное_программирование) **абстракция разделяет способ использования функции и деталей её реализации в терминах более примитивных функций, таким образом, данные обрабатываются функцией высокого уровня с помощью вызова функций низкого уровня.**

Такой подход является основой [объектно-ориентированного программирования](http://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование). Это позволяет работать с [объектами](http://ru.wikipedia.org/wiki/Объект_(программирование)), не вдаваясь в особенности их реализации.

Абстракция данных — популярная и в общем неверно определяемая техника программирования. Фундаментальная идея состоит в разделении несущественных деталей реализации подпрограммы и характеристик существенных для корректного ее использования. Такое разделение может быть выражено через специальный «[интерфейс](http://ru.wikipedia.org/wiki/Интерфейс_(объектно-ориентированное_программирование))», сосредотачивающий описание всех возможных применений программы[[1]](#cite_note-0).

С точки зрения [теории множеств](http://ru.wikipedia.org/wiki/Теория_множеств), процесс представляет собой организацию для группы [подмножеств](http://ru.wikipedia.org/wiki/Подмножество) своего [множества](http://ru.wikipedia.org/wiki/Множество). См. также [Закон обратного отношения между объемом и содержанием понятия](http://ru.wikipedia.org/wiki/Закон_обратного_отношения_между_содержанием_и_объемом_понятия) .

**Наследование**

**Объе́ктно-ориенти́рованное**, или **объектное**, программи́рование (в дальнейшем ООП) — [парадигма программирования](http://ru.wikipedia.org/wiki/Парадигма_программирования), в которой основными [концепциями](http://ru.wikipedia.org/wiki/Концепция) являются понятия [объектов](http://ru.wikipedia.org/wiki/Объект_(программирование)) и [классов](http://ru.wikipedia.org/wiki/Класс_(программирование)). В случае [языков](http://ru.wikipedia.org/wiki/Язык_программирования) с [прототипированием](http://ru.wikipedia.org/wiki/Прототипное_программирование) вместо классов используются объекты-прототипы.

**Полиморфизм**

**Полиморфи́зм** (в [языках программирования](http://ru.wikipedia.org/wiki/Язык_программирования)) — возможность объектов с одинаковой спецификацией иметь различную реализацию.

Язык программирования поддерживает полиморфизм, если [классы](http://ru.wikipedia.org/wiki/Класс_(программирование)) с одинаковой спецификацией могут иметь различную реализацию — например, реализация класса может быть изменена в процессе [наследования](http://ru.wikipedia.org/wiki/Наследование_(программирование))[[1]](#cite_note-0).

Кратко смысл полиморфизма можно выразить фразой: «Один [интерфейс](http://ru.wikipedia.org/wiki/Интерфейс_программирования_приложений), множество реализаций».

Полиморфизм — один из четырёх важнейших механизмов [объектно-ориентированного программирования](http://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование) (наряду с [абстракцией](http://ru.wikipedia.org/wiki/Абстракция_данных), [инкапсуляцией](http://ru.wikipedia.org/wiki/Инкапсуляция_(программирование)) и [наследованием](http://ru.wikipedia.org/wiki/Наследование_(программирование))).

Полиморфизм позволяет писать более абстрактные программы и повысить коэффициент [повторного использования кода](http://ru.wikipedia.org/wiki/Повторное_использование_кода). Общие свойства [объектов](http://ru.wikipedia.org/wiki/Объект_(программирование)) объединяются в систему, которую могут называть по-разному — [интерфейс](http://ru.wikipedia.org/wiki/Интерфейс_(объектно-ориентированное_программирование)), [класс](http://ru.wikipedia.org/wiki/Класс_(программирование)). Общность имеет внешнее и внутреннее выражение:

* внешняя общность проявляется как одинаковый набор [методов](http://ru.wikipedia.org/wiki/Метод_(программирование)) с одинаковыми именами и [сигнатурами](http://ru.wikipedia.org/wiki/Сигнатура_функции) (именем методов и типами [аргументов](http://ru.wikipedia.org/wiki/Аргумент_(программирование)) и их количеством);
* внутренняя общность — одинаковая функциональность методов. Её можно описать интуитивно или выразить в виде строгих законов, правил, которым должны подчиняться методы. Возможность приписывать разную функциональность одному методу (функции, операции) называется [*перегрузкой*](http://ru.wikipedia.org/wiki/Перегрузка_процедур_и_функций) *метода* (*перегрузкой функций*, [*перегрузкой операций*](http://ru.wikipedia.org/wiki/Перегрузка_операторов)).

**Руководство для программиста**

*Описание разработанных классов*

|  |  |
| --- | --- |
| **Название** | **Назначение** |
| BaseRecord | Реализует базовое поведение записи БД |
| DateTime | Реализует представление даты/времени в удобном виде для просмотра, сохранения в базу, а также для операций преобразования даты/времени |
| Exception | Реализует базовое поведение обработчика ошибок |
| MySQLRecordNotFound | Exception, возникающий когда выполняется запрос выборки из базы, но результат не был выбран |
| MySQLNoUpdateRecord | Exception, возникающий в результате безуспешной попытки обновления записи в базе данных |
| MySQLNoInsertRecord | Exception, возникающий в результате безуспешной попытки вставки данных в бд |
| MySQLNoDeleteRecord | Exception, возникающий в результате безуспешной попытки удаления записи из базы |
| MySQLQueryException | Exception, возникающий в результате выполнения с ошибкой запроса к бд |
| MySQLRecordNotUnique | Exception, возникающий при попытке записать несколько строк данных с одинаковыми первичными ключами (или другими уникальными полями) которые являются уникальными для конкретной таблицы в бд |
| MySQLAuthorizationException | Exception, возникающий в результате неудачной попытки авторизации пользователя |
| Group | Класс, относящий пользователя к определенной группе (пользователи, менеджеры либо аодминистраторы) |
| Menu | Реализация конкретных видов меню для каждого пользователя в зависимости от группы |
| NumberToStringConverter | Generic-класс для конвертации различных числовых данных в строку (в качестве строк используется тип std::string) |
| Project | Класс, описывающий проект |
| Report | Работа с отчетами |
| SqlConnector | Класс, служащий для работы с БД (MySQL). Реализует интерфейс поключения, частичной обработки ошибок. Предоставляет более высокоуровневый интерфейс для отправки запросов к БД |
| Status | Служит для реализации 4х статусов проекта или задачи: Сделать, Выполнятеся, Проверить, Закрыт |
| Task | Служит для представления конкретной задачи |
| Team | Позволяет менеджеру создать свою небольшую команду программистов, после чего у него появляется право на добавление пользователей, установки им задач и т.п. |
| User | Представляет конкретного пользоватя (программиста, менеджера, админа) |
| Wages | Класс для подсчета заработной платы пользователя с учетом часовой тарифной ставки, премии и т.п. |
| WorkTime | Класс, регистрирующий рабочее время программиста |

Наиболее значимые в данном проекте классы — это SqlConnector, реализующий «промежуточный легковесный интерфейс» для работы с базой; а таже класс BaseRecord.

Класс BaseRecord — это самый главный класс в данном проекте, от которого наследуются почти все остальные классы. Данный класс представляет собой базовое отображение записи базы данных, он содежит такие поля, как id, table; а также методы Save(), Delete(), Retrieve() и т.п, таким образом реализуя базовое представление и поведение записи. За счёт использования полиморфизма, более конкретные классы (такие как например, Group, Team, User и т.д), уже реализуют данные методы без необходимости конкретизации в них самих: метод Retrieve() позволяет получить запись из базы по её id (опять же, определяемом в BaseRecord); метод Delete() - позволяет удалить запись из базы, а метод Save() - позволяет создать новую запись или же обновить уже существующую (принцип простой, если поле id заполнено значением — значит такая запись уже есть в базе данных, и её нужно обновить; если же поле id принимает дефолтное значение 0 — то создается новая запись в базе). Принцип действия очень похож на ORM-технологию. Здесь также как и в ORM можно создать, допустим нового польователя, таким образом:

User\* Alex = new User();

Alex->Name(«Alexander»);

Alex->Surname(«Kozlov»);

Alex->Login(«alexssource»);

…

Alex->Save();

В данном случае будет создан новый пользователь в базе с логином alexssource. Так как данный класс наследуется от BaseRecord, у которого есть поле id (и свойство соотвественно GetId()) можно легко узнать id нового пользователя написав:

int alexId = Alex->GetId(); // *допустим 5*

Так же просто можно и подредактировать пользователя:

User\* Alex = new User(5); // *в качестве параметра констуктора указываем id редактируемого пользователя*

Alex->Name(«Misha»);

…

Alex->Save();

Этим действием мы переименовали пользователя.

**Руководство пользователя**

При запуске программы появляется окно, изображенное на Рис 1.

Рис.1. Окно авторизации пользователя

Введем логин и пароль от аккаунта менеджера. Появилось меню менеджера, представленное на Рис.2:

Рис.2. Меню менеджера

Выберем 1й пункт и просмотрим проекты команды менеджера (Рис.3):

Рис.3. Проекты команды менеджера

Теперь выберем 2й пункт и просмотрим пользователей, входящих в команду менеджера (Рис. 4):

Рис.4. Пользователи в команде менеджера

Для просмотра активных задач, отсортированных по добавлению, начиная с последнего, и по приоритету, выберем 3й пункт меню (Рис.5):

Рис.5. Список активных задач для команды менеджера

Добавим новый проект, для этого в меню выберем 4й пункт. Появилось окно, представленное на Рис.6:

Рис.6. Создание нового проекта

Добавим новую задачу, для этого выберем пункт меню 5. Появился список проектов, в которые можно добавить новую задачу. Выберем проект с id=2, выберем пользователя, который будет выполнять задачу, заполним оставшиеся поля и создадим новую задачу (Рис. 7):

Рис. 7. Создание новой задачи

Зарегистрируем нового программиста в команду менеджера. Для этого выберем пункт меню 6 (см. Рис. 8):

Рис.8. Регистрация нового пользователя в команду менеджера

Просмотрим отчеты разработчиков по проекту 5. Для этого выберем пункт меню 7 (Рис. 9):

Рис. 9. Просмотр отчета по проекту

Для выхода выберем пункт меню 0.

Теперь войдем в программу от лица обычного программиста. Для этого в окне авторизации просто вводим его данные (логин и пароль). Как видим из Рис.10, система автоматически определила нас в группу «Пользователь», и появилось соответствующее меню пользователя:

Рис. 10. Меню пользователя

Просмотрим проекты, на разработку которых определил нас менеджер. Для этого выберем 1й пункт меню (Рис. 11):

Рис. 11. Проекты, назначенные разработчику “alexssource”

Теперь просмотрим все активные задачи, которые назначил нам менеджер. Для этого из окна меню выберем 2й пункт (Рис. 12):

Рис. 12. Активные задачи для пользователя

Просмотрим, сколько времени мы уже залогинены (т.е. находимся на рабочем месте) – данное время будет «списываться» в отчет. Для этого выберем 3й пункт меню (Рис. 13):

Рис. 13. Отработанное на рабочем месте время

Отлично, теперь изменим статус задачи на «Проверить», «спишем» время, потраченное на разработку задачи, и напишем отчет о проделанной работе. Для этого выберем 4 пункт меню, и заполним все предложенные поля (Рис. 14):

Рис. 14. Перевод состояния задачи из «Сделать» в «Проверить» и написание отчета о проделанной работе.

Задача сохранена. Теперь менеджер сможет просмотреть наш отчет и начислить нам з/п!

Просмотрим что у нас «лежит» в базе данных через утилиту MySQL Query Browser (Рис. 15):

Рис. 15. Просмотр базы данных через утилиту MySQL Query Browser

Как видим, пользователь, созданный нами на Рис. 8, присутствует в базе (5 – Misha Collins), а это значит что всё действительно работает!

**Заключение**

В данной курсовой работе была разработана программа **KeeperTime**, позволяющая автоматизировать процесс контроля за работой программиста. Программа включает в себя методы, обеспечивающие выполнение всех поставленных задач для работы с базой данных. Главное меню программы позволяет обеспечить доступ к функциям программы и к сведениям, хранящимся в базе данных «KeeperTime».

Используя СУРБД MySQL 5.x мы сократили объем кода, увеличили функциональность, быстродействие и безопасность программы. Проведенное тестирование показало работоспособность программы и соответствие её требованиям задания на курсовой проект.

**Программа тестировалась на компьютерах с системными характеристиками:**

* 1. AMD Turion X2 – 2.40 GHz (64bit Processor), 2Гб ОЗУ
  2. AMD Athlon XP 2500+ (32bit Processor), 512Мб ОЗУ

2. Операционная система Ubuntu Linux 11.10

**Необходимые системные требования:**

1. Компьютер на базе процессора AMD или Intel с частотой не ниже 800 MHz и ОЗУ не менее 64Мб
2. Операционная система Linux
3. Установленные клиент и сервер СУРБД MySQL (и желательно какая-либо из утилит для администрирования БД).

**Список используемой литературы**

1. Шилдт, Герберт. С++ Базовый курс, 3-е издание: пер. с англ. – Алексей Жданов: СПб.: БХВ-Петербург, 2002. – 683 с., ил.
2. Б. Страуструп. Язык программирования С++, 3-е издание:пер. с англ. – М.: "Бином", СПб.: "Невский диалект", 1999. - 991 с*.*, ил.
3. Б. Оверленд. С++ без страха: пер. с англ. – Триумф, 2005. – 431с., ил.
4. Мэтью, Стоунс. Основы программирования в Linux. 4-е издание. – «БХВ Петербург», Санкт-Петербург, 2009.

**Листинг программы**

**BaseRecord.h**

/\*

\* File: BaseRecord.h

\* Author: Alex Kozlov <alexssource@gmail.com>

\*

\* Created on November 30, 2011, 11:37 PM

\*/

#ifndef BASERECORD\_H

#define BASERECORD\_H

#include <stdlib.h>

#include <string>

#include <vector>

#include "ERROR\_CODES.h"

#include "Exception.h"

#include "SqlConnector.h"

#include "NumberToStringConverter.h"

using namespace std;

class BaseRecord

{

protected:

/\*\*

\* The identifier of record

\* This field is readonly

\*/

int id;

/\*\*

\* Name of table for current record

\*/

string table;

/\*\*

\* The pointer to sql-connector for calling to database

\*/

SqlConnector\* const connector; // константный указатель на не константные данные

/\*\*

\* Initialize the begins parameters

\* @param table - the name of table

\* @param id - record id

\*/

void Initialize(string table, int id = 0)

{

this->table = table;

this->id = id;

}

/\*\*

\* Created the new record and assign them the unique identifier

\* @return bool result

\*/

virtual bool Create()

{

if(this->IsSet()) {

throw new MySQLRecordNotUnique();

return false;

}

if(this->CreateRecord()) {

this->id = this->connector->GetLastInsertedId();

return true;

}

else return false;

}

/\*\*

\* Updates the current record by its identifier

\* @param id - the identifier of record

\* @return result - the result of update operation

\*/

virtual bool UpdateRecord() const;

/\*\*

\* Creates a new record

\* @return <bool> res - the result of insert record process

\*/

virtual bool CreateRecord() const;

/\*\*

\* Delete the current record from database

\* @return <bool> res

\*/

virtual bool DeleteRecord() const;

/\*\*

\* Retrieve the record from database by it's id

\* All fields, which will be find in database must contain current object

\*/

virtual bool Retrieve() { };

/\*\*

\* Obtaining records from an arbitrary identifier

\* @param id - the identifier of record

\* @return BaseRecord\* - pointer to result object

\*/

virtual BaseRecord\* RetrieveRecord(int id) { };

/\*\*

\* Reset all field to their begin status

\*/

virtual void Reset() {

this->table = "";

this->id = 0;

}

/\*\*

\* Used in the derived class for in begin initialization

\* (called in current's class constructor)

\*/

virtual void InitializeRecord() { };

/\*\*

\* Check if set current record

\* @return true if record is set

\*/

virtual bool IsSet() const { return false; }

virtual string GetSqlCreate() const { };

virtual string GetSqlUpdate() const { };

virtual string GetSqlDelete() const;

public:

BaseRecord(SqlConnector\* const conn, string table): connector(conn) {

this->id = 0;

this->Initialize(table);

this->InitializeRecord();

}

BaseRecord(SqlConnector\* const conn, string table, int id): connector(conn) {

this->id = 0;

Initialize(table, id);

InitializeRecord();

Retrieve();

}

virtual ~BaseRecord() { }

/\*\*

\* Returns the identifier of current record (0 if isn't set)

\* @return id - identifier of record

\*/

int GetId() const { return this->id; }

/\*\*

\* Save or update the current record to database

\* @param id - the identifier of record (default is 0 for create a new record)

\* @return result - true if result is success

\*/

virtual bool Save()

{

return (this->id == 0) ? this->Create() : this->UpdateRecord();

}

/\*\*

\* Delete the current record

\* Need to override the method DeleteRecord in the derived type

\* @return <bool> result

\*/

virtual bool Delete()

{

bool result;

result = this->DeleteRecord();

this->Reset();

return result;

}

/\*\*

\* Retrieves the rows from current table with limit

\* @param where - the string of condition

\* @param limit - the string of limit

\* @return vector<MYSQL\_ROW> - the vector of table rows

\*/

virtual vector<string \*> RetrieveTableRows(string where = "", string limit = "") const ;

/\*\*

\* Retrieves the records from current table with various condition and limit

\* @param where - the string of condition

\* @param limit - the string of limit

\* @return vector&lt;BaseRecord \*&rt; - the vector of table rows

\*/

virtual vector<BaseRecord \*> RetrieveTableRecords(string where = "", string limit = "") const { };

};

#endif /\* BASERECORD\_H \*/

**BaseRecord.cpp**

#include "../header/BaseRecord.h"

vector<string \*> BaseRecord::RetrieveTableRows(string where, string limit) const

{

MYSQL\_RES\* res;

MYSQL\_ROW sqlrow;

string sql = "SELECT \* FROM `" + this->table + "` " + where + " " + limit;

vector<string \*> records;

try {

res = this->connector->Query(sql, SELECT);

}

catch(Exception e) {

std::cout << "Exception: " << e.what() << endl;

}

while(sqlrow = mysql\_fetch\_row(res)) {

string\* srow = new string[sizeof(sqlrow)];

for(int i = 0; i < sizeof(sqlrow); i++) {

srow[i] = (string)sqlrow[i];

}

records.push\_back(srow);

}

return records;

}

bool BaseRecord::CreateRecord() const

{

string sql = this->GetSqlCreate();

try {

this->connector->Query(sql, INSERT);

}

catch(MySQLNoInsertRecord\* nir) {

std::cout << "Error when trying to update record:" << std::endl;

std::cout << "\t" << nir->what() << std::endl;

return false;

}

return true;

}

bool BaseRecord::UpdateRecord() const

{

string sql = this->GetSqlUpdate();

try {

this->connector->Query(sql, UPDATE);

}

catch(MySQLNoUpdateRecord\* nur) {

std::cout << "Error when trying to update record:" << std::endl;

std::cout << "\t" << nur->what() << std::endl;

return false;

}

return true;

}

bool BaseRecord::DeleteRecord() const

{

string sql = this->GetSqlDelete();

try {

this->connector->Query(sql, DELETE);

}

catch(MySQLNoDeleteRecord\* ndr) {

std::cout << ndr->Message() << std::endl;

return false;

}

return true;

}

string BaseRecord::GetSqlDelete() const

{

string sql;

sql = "DELETE FROM `" + this->table + "` WHERE `id` = "

+ NumberToStringConverter<int>::Convert(this->id);

return sql;

}

**DateTime.h**

/\*

\* File: DateTime.h

\* Author: alex

\*

\* Created on December 19, 2011, 1:12 AM

\*/

#ifndef DATETIME\_H

#define DATETIME\_H

#include <time.h>

#include <string>

#include <iostream>

#include <sstream>

using namespace std;

class DateTime

{

private:

int day;

int month;

int year;

int hour;

int minute;

int seconds;

int timestamp;

void Initialize(int d, int m, int y, int h, int mins, int s);

public:

DateTime();

DateTime(int timestamp);

DateTime(int year, int month, int day = 1, int hour = 0, int minute = 0, int seconds = 0);

DateTime(const DateTime& orig) { };

virtual ~DateTime() { };

string GetDMY() const;

string ToString() const;

void AddSomeDays(int days);

void TimestampToDate();

void DateToTimestamp();

// properties

public:

int Day() const { return this->day; }

void Day(int d) { this->day = d; }

int Month() const { return this->month; }

void Month(int t) { this->month = t; }

int Year() const { return this->year; }

void Year(int t) { this->year = t; }

int Hour() const { return this->hour; }

void Hour(int t) { this->hour = t; }

int Minute() const { return this->minute; }

void Minute(int t) { this->minute = t; }

int Seconds() const { return this->seconds; }

void Seconds(int t) { this->seconds = t; }

int Timestamp() const { return this->timestamp; }

void Timestamp(int t) { this->timestamp = t; }

};

#endif /\* DATETIME\_H \*/

**DateTime.cpp**

/\*

\* File: DateTime.cpp

\* Author: alex

\*

\* Created on December 19, 2011, 1:12 AM

\*/

#include "../header/DateTime.h"

void DateTime::Initialize(int d, int m, int y, int h, int mins, int s)

{

this->day = d;

this->hour = h;

this->minute = mins;

this->month = m;

this->seconds = s;

this->year = y;

}

DateTime::DateTime()

{

this->Timestamp(time(NULL));

this->TimestampToDate();

}

DateTime::DateTime(int timestamp)

{

this->Timestamp(timestamp);

this->TimestampToDate();

}

DateTime::DateTime(int year, int month, int day, int hour, int minute, int seconds)

{

this->Initialize(day, month, year, hour, minute, seconds);

this->DateToTimestamp();

}

string DateTime::GetDMY() const

{

string time;

stringstream ss;

ss << Day() << "." << Month() << "." << Year();

return ss.str();

}

string DateTime::ToString() const

{

string time;

stringstream ss;

ss << GetDMY() << " " << Hour() << ":" << Minute() << ":" << Seconds();

return ss.str();

}

void DateTime::TimestampToDate()

{

tm\* stime;

time\_t tmsp = this->Timestamp();

stime = localtime(&tmsp);

this->Initialize(stime->tm\_mday, stime->tm\_mon + 1, stime->tm\_year + 1900,

stime->tm\_hour, stime->tm\_min, stime->tm\_sec);

}

void DateTime::DateToTimestamp()

{

tm\* stime = new tm();

time\_t timestamp;

stime->tm\_year = this->year - 1900;

stime->tm\_mon = this->month - 1;

stime->tm\_mday = this->day;

stime->tm\_hour = this->hour;

stime->tm\_min = this->minute;

stime->tm\_sec = this->seconds;

timestamp = mktime(stime);

this->Timestamp(timestamp);

}

void DateTime::AddSomeDays(int days)

{

this->Timestamp(this->Timestamp() + days\*24\*60\*60);

this->TimestampToDate();

}

**ERROR\_CODES.h**

/\*

\* File: ERROR\_CODES.h

\* Author: Alex Kozlov <alexssource@gmail.com>

\*

\* Created on December 10, 2011, 6:13 PM

\*/

#ifndef ERROR\_CODES\_H

#define ERROR\_CODES\_H

#define ERROR\_TYPE\_MYSQL "MYSQL\_ERROR"

#define ERROR\_MYSQL\_CONNECT 1

#define ERROR\_MYSQL\_CHANGE\_DB 2

#define ERROR\_MYSQL\_QUERY 3

enum MYSQL\_QUERY\_TYPE { SELECT, INSERT, UPDATE, DELETE, OTHER };

#endif /\* ERROR\_CODES\_H \*/

**Exception.h**

/\*

\* File: Exception.h

\* Author: Alex Kozlov <alexssource@gmail.com>

\*

\* Created on December 10, 2011, 6:15 PM

\*/

#ifndef EXCEPTION\_H

#define EXCEPTION\_H

#include <string>

#include <exception>

#include "ERROR\_CODES.h"

using namespace std;

class Exception: public exception

{

private:

int code;

string type;

string message;

void Initialize(int c, string msg, string t = "")

{

this->code = c;

this->message = msg;

this->type = t;

}

public:

/\*\*

\* This object contained the information about your exception,

\* such as exception's code, type, message

\* @param <int> c - the code of exception (need get from ERROR\_CODES.h)

\* @param <string> msg - the message of exception

\*/

Exception(int c, string msg);

/\*\*

\* This object contained the information about your exception,

\* such as exception's code, type, message

\* @param <int> c - the code of exception (need get from ERROR\_CODES.h)

\* @param <string> t - the type of exception (example, "MYSQL\_ERROR")

\* @param <string> msg - the message of exception

\*/

Exception(int c, string t, string msg);

/\*\*

\* Remove the instance of current exception

\*/

virtual ~Exception() throw() { };

/\*\*

\* Gets the full information about your exception (example, "\_21. MYSQL\_ERROR: Unable to connect")

\* @return char\* information

\*/

virtual const char\* what() const throw();

// Properties (readonly)

/\*\*

\* Gets the exception code

\* @return int code

\*/

int Code() const { return this->code; }

/\*\*

\* Gets the exception type

\* @return string type

\*/

string Type() const { return this->type; }

/\*\*

\* Gets the exception message

\* @return string message

\*/

string Message() const { return this->message; }

};

class MySQLRecordNotFound: public Exception

{

public:

MySQLRecordNotFound(): Exception(ERROR\_MYSQL\_QUERY, ERROR\_TYPE\_MYSQL, "No Records Found") { }

virtual ~MySQLRecordNotFound() throw() { };

};

class MySQLNoUpdateRecord: public Exception

{

public:

MySQLNoUpdateRecord(string mysql\_error = ""): Exception(ERROR\_MYSQL\_QUERY, ERROR\_TYPE\_MYSQL, "No Updated Records Found: " + mysql\_error) { }

virtual ~MySQLNoUpdateRecord() throw() { };

};

class MySQLNoInsertRecord: public Exception

{

public:

MySQLNoInsertRecord(string mysql\_error = ""): Exception(ERROR\_MYSQL\_QUERY, ERROR\_TYPE\_MYSQL, "Can\'t insert the records: " + mysql\_error) { }

virtual ~MySQLNoInsertRecord() throw() { }

};

class MySQLNoDeleteRecord: public Exception

{

public:

MySQLNoDeleteRecord(string mysql\_error = ""): Exception(ERROR\_MYSQL\_QUERY, ERROR\_TYPE\_MYSQL, "Can\'t delete the record: " + mysql\_error) { }

virtual ~MySQLNoDeleteRecord() throw() { }

};

class MySQLQueryException: public Exception

{

public:

MySQLQueryException(string sql, string mysql\_say = ""): Exception(ERROR\_MYSQL\_QUERY, ERROR\_TYPE\_MYSQL,

"Error Send Query: " + sql + "\nMysql Say: " + mysql\_say) { };

virtual ~MySQLQueryException() throw() { }

};

class MySQLRecordNotUnique: public Exception

{

public:

MySQLRecordNotUnique(): Exception(ERROR\_MYSQL\_QUERY, ERROR\_TYPE\_MYSQL,

"Can't Create a record: MySQL Record Is Set (Not Unique)") { };

virtual ~MySQLRecordNotUnique() throw() { }

};

class MySQLAuthorizationException: public Exception

{

public:

MySQLAuthorizationException(string login, string pass): Exception(ERROR\_MYSQL\_QUERY, ERROR\_TYPE\_MYSQL,

"Пользователя с логином `"+login+"` и паролем `"+pass+"` не существует") { };

virtual ~MySQLAuthorizationException() throw() { }

};

#endif /\* EXCEPTION\_H \*/

**Exception.cpp**

#include "../header/Exception.h"

Exception::Exception(int c, string msg) { this->Initialize(c, msg); }

Exception::Exception(int c, string t, string msg) { this->Initialize(c, msg, t); }

const char\* Exception::what() const throw()

{

string result = "\_";

result += this->Code() + ". " + this->Type() + ": " + this->Message();

return result.c\_str();

}

**Group.h**

/\*

\* File: Group.h

\* Author: alex

\*

\* Created on December 17, 2011, 3:34 PM

\*/

#ifndef GROUP\_H

#define GROUP\_H

#include "BaseRecord.h"

using namespace std;

class Group: public BaseRecord

{

private:

// Fields and initialize methods

string name;

bool is\_manager;

bool is\_admin;

public:

// Constructors and Destructors

Group(SqlConnector\* const conn);

Group(SqlConnector\* const conn, int id);

Group(const Group& orig);

virtual ~Group();

// Properties

string Name() const { return this->name; }

void Name(string name) { this->name = name; }

bool IsManager() const { return this->is\_manager; }

void IsManager(bool is\_manager) { this->is\_manager = is\_manager; }

bool IsAdmin() const { return this->is\_admin; }

void IsAdmin(bool is\_admin) { this->is\_admin = is\_admin; }

// Section for overrides methods

protected:

virtual string GetSqlCreate() const;

virtual string GetSqlUpdate() const;

virtual string GetSqlDelete() const;

virtual void InitializeRecord();

public:

virtual bool Retrieve();

virtual BaseRecord\* RetrieveRecord(int id);

/\*\*

\* Retrieves the records from current table with various condition and limit

\* @param where - the string of condition

\* @param limit - the string of limit

\* @return vector&lt;BaseRecord \*&rt; - the vector of table rows

\*/

virtual vector<BaseRecord \*> RetrieveTableRecords(string where = "", string limit = "") const;

};

#endif /\* GROUP\_H \*/

**Group.cpp**

/\*

\* File: Group.cpp

\* Author: alex

\*

\* Created on December 17, 2011, 3:34 PM

\*/

#include "../header/Group.h"

Group::Group(SqlConnector\* const conn): BaseRecord(conn, "Group") { }

Group::Group(SqlConnector\* conn, int id): BaseRecord(conn, "Group", id) { this->Retrieve(); }

Group::Group(const Group& orig): BaseRecord(orig.connector, orig.table)

{

this->id = orig.id;

this->is\_admin = orig.is\_admin;

this->is\_manager = orig.is\_manager;

this->name = orig.name;

}

Group::~Group() { }

string Group::GetSqlCreate() const

{

string sql;

sql = "INSERT INTO `" + this->table + "` (`name`, `is\_admin`, `is\_manager`) "

+ "VALUE ('" + this->name + "', " + ((this->is\_admin) ? "1" : "0") + ", "

+ ((this->is\_manager) ? "1" : "0") + ")";

return sql;

}

string Group::GetSqlUpdate() const

{

string sql;

sql = "UPDATE `" + this->table + "` SET `name`='" + this->name + "', "

+ "`is\_admin` = " + ((this->is\_admin) ? "1" : "0") + ", "

+ "`is\_manager` = " + ((this->is\_manager) ? "1" : "0") + " "

+ "WHERE `id` = " + NumberToStringConverter<int>::Convert(this->id);

return sql;

}

string Group::GetSqlDelete() const

{

string sql;

sql = "DELETE FROM `" + this->table + "` WHERE `id` = "

+ NumberToStringConverter<int>::Convert(this->id);

return sql;

}

void Group::InitializeRecord()

{

this->is\_admin = false;

this->is\_manager= false;

}

bool Group::Retrieve()

{

MYSQL\_RES\* res;

MYSQL\_ROW row;

string sql;

sql = "SELECT `id`, `name`, `is\_admin`, `is\_manager` FROM `" + this->table

+ "` WHERE `id` = " + NumberToStringConverter<int>::Convert(this->id)

+ " LIMIT 0, 1";

try {

res = this->connector->Query(sql, SELECT);

}

catch(MySQLRecordNotFound\* rnf) {

std::cout << rnf->what() << std::endl;

return false;

}

row = mysql\_fetch\_row(res);

this->Name(row[1]);

this->IsAdmin(row[2]);

this->IsManager(row[3]);

return true;

}

BaseRecord\* Group::RetrieveRecord(int pid)

{

Group\* gr = new Group(this->connector, pid);

gr->Retrieve();

return gr;

}

vector<BaseRecord \*> Group::RetrieveTableRecords(string where, string limit) const

{

vector<BaseRecord \*> groups;

BaseRecord\* baseRecord;

MYSQL\_RES\* res;

MYSQL\_ROW row;

Group\* group;

string sql;

sql = "SELECT `id`, `name`, `is\_admin`, `is\_manager` FROM `" + this->table + "` "

+ where + " " + limit;

try {

res = this->connector->Query(sql, SELECT);

}

catch(MySQLRecordNotFound\* rnf) {

std::cout << rnf->what() << std::endl;

exit(ERROR\_MYSQL\_QUERY);

}

while(row = mysql\_fetch\_row(res)) {

group = new Group(this->connector);

group->id = atoi(row[0]);

group->Name(row[1]);

group->IsAdmin(atoi(row[2]));

group->IsManager(atoi(row[3]));

baseRecord = group;

groups.push\_back(baseRecord);

}

return groups;

}

**Menu.h**

/\*

\* File: Menu.h

\* Author: alex

\*

\* Created on December 26, 2011, 3:06 PM

\*/

#ifndef MENU\_H

#define MENU\_H

#include <iostream>

#include <string>

#include <vector>

#include "SqlConnector.h"

#include "User.h"

#include "Project.h"

#include "Status.h"

#include "NumberToStringConverter.h"

#include "Report.h"

#include <time.h>

using namespace std;

class Menu

{

protected:

SqlConnector\* connector;

User\* user;

DateTime\* timeStart;

int workedtime; // затр. время в секундах - испся как промежуточный результат

void ShowMenu();

void EnterLogin();

void ShowStatuses();

bool MenuManager();

void ManagerShowProjects();

void ManagerShowUsers();

void ManagerShowActiveTasks();

void ManagerAddProject();

void ManagerAddTask();

void ManagerAddUser();

void ManagerShowReports();

bool MenuUser();

void UserShowProjects();

void UserShowActiveTasks();

void UserShowWorkedTime();

void UserChangeStatusTask();

public:

Menu(SqlConnector\* const conn);

~Menu();

};

#endif /\* MENU\_H \*/

**Menu.cpp**

/\*

\* File: Menu.cpp

\* Author: alex

\*

\* Created on December 26, 2011, 3:06 PM

\*/

#include "../header/Menu.h"

#include "header/Task.h"

Menu::Menu(SqlConnector\* const conn): connector(conn) { this->user = NULL; this->ShowMenu(); }

Menu::~Menu() { delete this->user; }

void Menu::ShowMenu()

{

cout << "Menu:" << endl;

while(this->user == NULL) {

this->EnterLogin();

}

if(this->user->GetGroup()->Name() == "manager") {

while(this->MenuManager()) { }

}

else if(this->user->GetGroup()->Name() == "user") {

while(this->MenuUser()) { }

}

else {

cout << "Your group isn't find" << endl;

}

}

void Menu::EnterLogin()

{

string login;

string password;

User\* user;

cout << "Enter your Login: ";

cin >> login;

cout << "Enter password for '" << login << "': ";

cin >> password;

cout << "\n";

user = new User(this->connector);

user->Login(login);

user->Password(password);

try {

user->RetrieveByLoginPass();

}

catch(Exception\* e) {

cout << e->what() << endl;

return;

}

this->user = user;

this->timeStart = new DateTime();

}

void Menu::ShowStatuses()

{

Status\* status = new Status(this->connector);

vector<BaseRecord\*> statuses;

statuses = status->RetrieveTableRecords();

for(int i = 0; i < statuses.size(); i++) {

cout << "- " << statuses[i]->GetId() << ": " << ((Status\*)statuses[i])->Name() << endl;

}

delete status;

}

bool Menu::MenuManager()

{

int choice = 0;

do {

cout << "Manager:" << endl;

cout << "1. Show Projects" << endl;

cout << "2. Show Users in my team" << endl;

cout << "3. Show Active Tasks" << endl;

cout << "4. Add new project" << endl;

cout << "5. Add new task" << endl;

cout << "6. Add new user" << endl;

cout << "7. Просмотр отчетов по проектам" << endl;

cout << "0. Exit" << endl;

cout << "Выберите действие: ";

cin >> choice;

} while((choice < 1) && choice > 7);

switch(choice) {

case 1:

this->ManagerShowProjects();

return true;

break;

case 2:

this->ManagerShowUsers();

return true;

break;

case 3:

this->ManagerShowActiveTasks();

return true;

break;

case 4:

this->ManagerAddProject();

return true;

break;

case 5:

this->ManagerAddTask();

return true;

break;

case 6:

this->ManagerAddUser();

return true;

break;

case 7:

this->ManagerShowReports();

return true;

break;

case 0:

return false;

break;

}

}

void Menu::ManagerShowProjects()

{

Project\* project = new Project(this->connector);

vector<BaseRecord \*> projects;

string where = " WHERE `team` = " + NumberToStringConverter<int>::Convert(this->user->GetTeam()->GetId())

+ " ORDER BY `id` DESC, `status` ASC";

try {

projects = project->RetrieveTableRecords(where);

}

catch(Exception\* e) {

cout << e->what() << endl;

}

if(projects[0]->GetId()) cout << "Проекты для команды " << ((Project \*)projects[0])->GetTeam()->Name() << endl;

for(int i = 0; i < projects.size(); i++) {

cout << "- " << ((Project \*)projects[i])->GetId() << ": " << ((Project \*)projects[i])->Name() << endl;

cout << ((Project \*)projects[i])->Description() << endl;

cout << "Статус: " << ((Project \*)projects[i])->GetStatus()->Name() << endl;

cout << "Приоритет: " << ((Project \*)projects[i])->Priority() << endl;

cout << "Расчётное время: " << ((Project \*)projects[i])->TimmingPlan() << " часов" << endl;

cout << "Дата созданиия: " << ((Project \*)projects[i])->CreatedAt()->ToString() << endl;

cout << "Действителен до: " << ((Project \*)projects[i])->ExpiredAt()->ToString() << endl;

cout << "Deadline: " << ((Project \*)projects[i])->Deadline()->ToString() << "\n" << endl;

}

}

void Menu::ManagerShowUsers()

{

vector<BaseRecord \*> users;

string where = " WHERE `team` = " + NumberToStringConverter<int>::Convert(this->user->GetTeam()->GetId())

+ " ORDER BY `id` ASC";

try {

users = this->user->RetrieveTableRecords(where);

}

catch(Exception\* e) {

cout << e->what() << endl;

}

if(users[0]->GetId()) cout << "Пользователи команды " << ((User \*)users[0])->GetTeam()->Name() << endl;

for(int i = 0; i < users.size(); i++) {

cout << "- " << ((User\*)users[i])->GetId() << ": " << ((User\*)users[i])->Name() << " "

<< ((User\*)users[i])->Surname() << " (" << ((User\*)users[i])->Login() << ")" << endl;

cout << "Email:\t" << ((User\*)users[i])->Email() << endl;

cout << "Группа:\t" << ((User\*)users[i])->GetGroup()->Name() << endl;

cout << "Создан:\t" << ((User\*)users[i])->CreatedAt()->ToString() << endl;

cout << "Часовая тарифная ставка:\t" << ((User\*)users[i])->Wage() << "$\\час\n" << endl;

}

}

void Menu::ManagerShowActiveTasks()

{

Task\* task = new Task(this->connector);

vector<BaseRecord \*> tasks;

string where = " INNER JOIN `Project` p ON t.project = p.id WHERE p.`team` = " + NumberToStringConverter<int>::Convert(this->user->GetTeam()->GetId())

+ " ORDER BY t.`id` DESC, t.`project` ASC, t.`status` ASC";

try {

tasks = task->RetrieveTableRecords(where);

}

catch(Exception\* e) {

cout << e->what() << endl;

}

if(tasks[0]->GetId()) cout << "Задачи команды " << ((Task \*)tasks[0])->GetProject()->GetTeam()->Name() << endl;

for(int i = 0; i < tasks.size(); i++) {

cout << "- " << ((Task \*)tasks[0])->GetId() << ": " << ((Task \*)tasks[0])->Name() << endl;

cout << ((Task \*)tasks[0])->Description() << endl;

cout << "Проект: " << ((Task \*)tasks[0])->GetProject()->Name() << endl;

cout << "Исполнитель: " << ((Task \*)tasks[0])->GetUser()->Name() << " "

<< ((Task \*)tasks[0])->GetUser()->Surname() << " ("

<< ((Task \*)tasks[0])->GetUser()->Login() << ")" << endl;

cout << "Статус: " << ((Task \*)tasks[0])->GetStatus()->Name() << endl;

cout << "Дата создания: " << ((Task \*)tasks[0])->CreatedAt()->ToString() << endl;

cout << "Приоритет: " << ((Task \*)tasks[0])->Priority() << "\n" << endl;

}

}

void Menu::ManagerShowReports()

{

Report\* report = new Report(this->connector);

vector<BaseRecord \*> reports;

string where;

int pid;

cout << "Выберите проект, по которому хотите посмотреть отчеты, и запомните его id" << endl;

this->ManagerShowProjects();

cout << "Введите id проекта: ";

cin >> pid;

where = " WHERE `project` = " + NumberToStringConverter<int>::Convert(pid);

try {

reports = report->RetrieveTableRecords(where);

}

catch(Exception\* e) {

cout << e->what() << endl;

}

if(reports[0]->GetId()) cout << "Отчеты по проекту " << ((Report\*)reports[0])->GetProject()->Name() << ":" << endl;

for(int i = 0; i < reports.size(); i++) {

cout << "- " << reports[i]->GetId() << ": From User:"

<< ((Report\*)reports[0])->GetTask()->GetUser()->Name() << " "

<< ((Report\*)reports[0])->GetTask()->GetUser()->Surname() << " ("

<< ((Report\*)reports[0])->GetTask()->GetUser()->Login() << ")" << endl;

cout << "Задача: [" << ((Report\*)reports[0])->GetTask()->GetId() << "] "

<< ((Report\*)reports[0])->GetTask()->Name() << endl;

cout << "Проект: [" << ((Report\*)reports[0])->GetProject()->GetId() << "] "

<< ((Report\*)reports[0])->GetProject()->Name() << endl;

cout << ((Report\*)reports[0])->GetReport() << endl;

cout << "Затрачено времени на выполнение проекта: " << ((Report\*)reports[0])->Hours() << " часов" << endl;

cout << "Дата создания отчета: " << ((Report\*)reports[0])->CreatedAt()->ToString() << "\n" << endl;

}

}

void Menu::ManagerAddProject()

{

Project\* project = new Project(this->connector);

string name, description;

//unsigned int team, status, created, expired, deadline

int priority, ed, em, ey, dd, dm, dy;

float timming\_plan;

cout << "Введите название проекта: ";

cin >> name;

cout << "Введите описание проекта:" << endl;

cin >> description;

cout << "Введите дату, до которой действителен проект: " << endl;

cout << "\tДень(1-31): ";

cin >> ed;

cout << "\tМесяц(1-12): ";

cin >> em;

cout << "\tГод(1900 - 2900): ";

cin >> ey;

cout << "Введите Deadline-дату: " << endl;

cout << "\tДень(1-31): ";

cin >> dd;

cout << "\tМесяц(1-12): ";

cin >> dm;

cout << "\tГод(1900 - 2900): ";

cin >> dy;

cout << "Введите приоритет (1-7): ";

cin >> priority;

cout << "Введите планируемое на разработку время: ";

cin >> timming\_plan;

project->Name(name);

project->Description(description);

project->SetTeam(this->user->GetTeam());

project->TimmingPlan(timming\_plan);

project->SetStatus(new Status(this->connector, 2));

project->CreatedAt(new DateTime());

project->ExpiredAt(new DateTime(ey, em, ed, 0, 0, 0));

project->Deadline(new DateTime(dy, dm, dd, 0, 0, 0));

project->Priority(priority);

project->Save();

if(project->GetId() != 0) {

cout << "Проект успешно сохранен!\nId проекта: " << project->GetId() << endl;

}

else {

cout << "Возникла ошибка при сохранении проекта" << endl;

}

delete project;

}

void Menu::ManagerAddTask()

{

Task\* task = new Task(this->connector);

string name, title, description;

float timming\_plan;

int project, user, status = 2, priority;

cout << "Введите название задачи: ";

cin >> name;

cout << "Заголовок задачи: ";

cin >> title;

cout << "Описание задачи: ";

cin >> description;

cout << "Найдите проект, соответствующий задаче, и запомните его id:" << endl;

this->ManagerShowProjects();

cout << "Введите id проекта: ";

cin >> project;

cout << "Найдите пользователя, которому хотите поручить данную задачу, и запомните его id:" << endl;

this->ManagerShowUsers();

cout << "Введите id пользователя: ";

cin >> user;

cout << "Введите планируемое время выполнения задачи: ";

cin >> timming\_plan;

cout << "Введите приоритет задачи: ";

cin >> priority;

task->Name(name);

task->Title(title);

task->Description(description);

task->TimmingPlan(timming\_plan);

task->SetProject(project);

task->SetUser(user);

task->SetStatus(status);

task->CreatedAt(new DateTime());

task->Priority(priority);

task->Save();

if(task->GetId() != 0) {

cout << "Задача успешно сохранена!\nId задачи: " << task->GetId() << endl;

}

else {

cout << "Возникла ошибка при сохранении задачи" << endl;

}

delete task;

}

void Menu::ManagerAddUser()

{

User\* user = new User(this->connector);

string name, surname, login, password, email;

int group = 1, wage;

cout << "Введите имя: " << endl;

cin >> name;

cout << "Фамилия: " << endl;

cin >> surname;

cout << "Логин: " << endl;

cin >> login;

cout << "Пароль: " << endl;

cin >> password;

cout << "Email: " << endl;

cin >> email;

cout << "Часовая тарифная ставка: " << endl;

cin >> wage;

user->Name(name);

user->Surname(surname);

user->Login(login);

user->Password(password);

user->Email(email);

user->SetGroup(group);

user->SetTeam(this->user->GetTeam());

user->Wage(wage);

user->CreatedAt(new DateTime());

user->Save();

if(user->GetId() != 0) {

cout << "Пользователь успешно сохранен!\nId юзера: " << user->GetId() << endl;

}

else {

cout << "Возникла ошибка при сохранении пользователя" << endl;

}

//delete user;

}

bool Menu::MenuUser()

{

int choice = 0;

do {

cout << "User:" << endl;

cout << "1. Show Projects" << endl;

cout << "2. Show Active Tasks" << endl;

cout << "3. Просмотреть время работы" << endl;

cout << "4. Изменить статус задачи" << endl;

cout << "0. Exit" << endl;

cout << "Выберите действие: ";

cin >> choice;

} while((choice < 1) && choice > 4);

switch(choice) {

case 1:

this->UserShowProjects();

return true;

break;

case 2:

this->UserShowActiveTasks();

return true;

break;

case 3:

this->UserShowWorkedTime();

return true;

break;

case 4:

this->UserChangeStatusTask();

return true;

break;

case 0:

return false;

break;

}

}

void Menu::UserShowProjects()

{

this->ManagerShowProjects();

}

void Menu::UserShowActiveTasks()

{

Task\* task = new Task(this->connector);

vector<BaseRecord \*> tasks;

string where = " INNER JOIN `Project` p ON t.project = p.id WHERE p.`team` = "

+ NumberToStringConverter<int>::Convert(this->user->GetTeam()->GetId())

+ " AND `user` = " + NumberToStringConverter<int>::Convert(this->user->GetId())

+ " ORDER BY t.`id` DESC, t.`project` ASC, t.`status` ASC";

try {

tasks = task->RetrieveTableRecords(where);

}

catch(Exception\* e) {

cout << e->what() << endl;

}

if(tasks[0]->GetId()) cout << "Задачи команды " << ((Task \*)tasks[0])->GetProject()->GetTeam()->Name() << endl;

for(int i = 0; i < tasks.size(); i++) {

cout << "- " << ((Task \*)tasks[0])->GetId() << ": " << ((Task \*)tasks[0])->Name() << endl;

cout << ((Task \*)tasks[0])->Description() << endl;

cout << "Проект: " << ((Task \*)tasks[0])->GetProject()->Name() << endl;

cout << "Исполнитель: " << ((Task \*)tasks[0])->GetUser()->Name() << " "

<< ((Task \*)tasks[0])->GetUser()->Surname() << " ("

<< ((Task \*)tasks[0])->GetUser()->Login() << ")" << endl;

cout << "Статус: " << ((Task \*)tasks[0])->GetStatus()->Name() << endl;

cout << "Дата создания: " << ((Task \*)tasks[0])->CreatedAt()->ToString() << endl;

cout << "Приоритет: " << ((Task \*)tasks[0])->Priority() << "\n" << endl;

}

}

void Menu::UserShowWorkedTime()

{

DateTime\* timeEnd = new DateTime();

int deltaSeconds = timeEnd->Timestamp() - this->timeStart->Timestamp();

float deltaHours = (float) deltaSeconds / 3600;

cout << "Вы работаете " << deltaHours << " часов" << endl;

this->workedtime = deltaSeconds;

}

void Menu::UserChangeStatusTask()

{

int tid, sid;

float time;

string rep;

Report\* report = new Report(this->connector);

Task\* task;

this->UserShowWorkedTime();

cout << "Выберите задачу: " << endl;

this->UserShowActiveTasks();

cout << "Введите id изменяемой задачи: ";

cin >> tid;

cout << "Выберите новый статус: " << endl;

this->ShowStatuses();

cout << "Введите id нового статуса: ";

cin >> sid;

do {

cout << "Введите время в часах, затраченное на выполнение задачи: " << endl;

this->UserShowWorkedTime();

cout << " -> ";

cin >> time;

} while(time\*3600 > this->workedtime);

cout << "Напишите отчет о проделанной работе:" << endl;

cin >> rep;

// не забыть сделать изменение статуса!!!

task = new Task(this->connector, tid);

task->SetStatus(sid);

task->Save();

cout << "Статус задачи изменен!" << endl;

report->SetTask(task);

report->SetProject(task->GetProject());

report->SetReport(rep);

report->Hours(time);

report->CreatedAt(new DateTime());

report->Tmpl(rep);

report->Save();

if(report->GetId() != 0) {

cout << "Отчет успешно сохранен!\nId отчета: " << report->GetId() << endl;

int oldtmstmp = this->timeStart->Timestamp();

int newtmstmp = oldtmstmp + time\*3600;

this->timeStart->Timestamp(newtmstmp);

}

else {

cout << "Возникла ошибка при сохранении отчета" << endl;

}

}

**NumberToStringConverter.h**

/\*

\* File: NumberToStringConverter.h

\* Author: alex

\*

\* Created on December 17, 2011, 6:23 PM

\*/

#ifndef NUMBERTOSTRINGCONVERTER\_H

#define NUMBERTOSTRINGCONVERTER\_H

#include <string>

#include <sstream>

using namespace std;

template <class T>

class NumberToStringConverter

{

public:

static string Convert(T base)

{

stringstream ss;

ss << base;

return ss.str();

}

};

#endif /\* NUMBERTOSTRINGCONVERTER\_H \*/

**Project.cpp**

/\*

\* File: Preject.h

\* Author: Alex Kozlov <alexssource@gmail.com>

\*

\* Created on December 1, 2011, 2:50 PM

\*/

#ifndef PROJECT\_H

#define PROJECT\_H

#include <iostream>

#include <string>

#include "BaseRecord.h"

#include "Team.h"

#include "Status.h"

#include "DateTime.h"

using namespace std;

class Project: public BaseRecord

{

private:

string name;

string description;

float timming\_plan;

int priority;

Team\* team;

Status\* status;

DateTime\* created\_at;

DateTime\* expired\_at;

DateTime\* deadline;

public:

Project(SqlConnector\* const);

Project(SqlConnector\* const, int);

Project(const Project &);

virtual ~Project();

// Properties:

string Name() const { return this->name; }

void Name(string name) { this->name = name; }

string Description() const { return this->description; }

void Description(string desc) { this->description = desc; }

Team\* GetTeam() const { return this->team; }

void SetTeam(Team\* team) { this->team = team; }

float TimmingPlan() const { return this->timming\_plan; }

void TimmingPlan(float tp) { this->timming\_plan = tp; }

Status\* GetStatus() const { return this->status; }

void SetStatus(Status\* s) { this->status = s; }

int Priority() const { return this->priority; }

void Priority(int p) { this->priority = p; }

DateTime\* CreatedAt() const { return this->created\_at; }

void CreatedAt(DateTime\* ca) { this->created\_at = ca; }

DateTime\* ExpiredAt() const { return this->expired\_at; }

void ExpiredAt(DateTime\* ea) { this->expired\_at = ea; }

DateTime\* Deadline() const { return this->deadline; }

void Deadline(DateTime\* dl) { this->deadline = dl; }

// Oveerided methods

protected:

virtual string GetSqlCreate() const;

virtual string GetSqlUpdate() const;

virtual void InitializeRecord();

public:

virtual bool Retrieve();

virtual BaseRecord\* RetrieveRecord(int id);

/\*\*

\* Retrieves the records from current table with various condition and limit

\* @param where - the string of condition

\* @param limit - the string of limit

\* @return vector&lt;BaseRecord \*&rt; - the vector of table rows

\*/

virtual vector<BaseRecord \*> RetrieveTableRecords(string where = "", string limit = "") const;

};

#endif /\* PROJECT\_H \*/

**Project.cpp**

#include "../header/Project.h"

Project::Project(SqlConnector\* const conn): BaseRecord(conn, "Project") { }

Project::Project(SqlConnector\* const conn, int id): BaseRecord(conn, "Project", id) { this->Retrieve(); }

Project::Project(const Project& orig): BaseRecord(orig.connector, orig.table)

{

this->Name(orig.Name());

this->Description(orig.Description());

this->SetTeam(orig.GetTeam());

this->TimmingPlan(orig.TimmingPlan());

this->SetStatus(orig.GetStatus());

this->CreatedAt(orig.CreatedAt());

this->ExpiredAt(orig.ExpiredAt());

this->Deadline(orig.Deadline());

this->Priority(orig.Priority());

}

Project::~Project()

{

if(this->team != NULL) delete this->team;

if(this->status != NULL) delete this->status;

if(this->created\_at != NULL) delete this->created\_at;

if(this->expired\_at != NULL) delete this->expired\_at;

if(this->deadline != NULL) delete this->deadline;

this->InitializeRecord();

}

void Project::InitializeRecord()

{

this->team = NULL;

this->status = NULL;

this->created\_at = NULL;

this->expired\_at = NULL;

this->deadline = NULL;

}

string Project::GetSqlCreate() const

{

string sql;

sql = "INSERT INTO `" + this->table + "` (`name`, `description`, `team`, "

+ "`timming\_plan`, `status`, `created\_at`, `expired\_at`, `deadline`, `priority`) "

+ "VALUE ('" + this->Name() + "', '" + this->Description() + "', "

+ NumberToStringConverter<int>::Convert(this->GetTeam()->GetId()) + ", '"

+ NumberToStringConverter<float>::Convert(this->TimmingPlan()) + "', "

+ NumberToStringConverter<int>::Convert(this->GetStatus()->GetId()) + ", "

+ NumberToStringConverter<int>::Convert(this->CreatedAt()->Timestamp()) + ", "

+ NumberToStringConverter<int>::Convert(this->ExpiredAt()->Timestamp()) + ", "

+ NumberToStringConverter<int>::Convert(this->Deadline()->Timestamp()) + ", "

+ NumberToStringConverter<int>::Convert(this->Priority()) + ")";

return sql;

}

string Project::GetSqlUpdate() const

{

string sql;

sql = "UPDATE `" + this->table +"` SET "

+ "`name` = '" + this->Name() + "', "

+ "`description` = '" + this->Description() + "', "

+ "`team` = " + NumberToStringConverter<int>::Convert(this->GetTeam()->GetId()) + ", "

+ "`timming\_plan` = '" + NumberToStringConverter<float>::Convert(this->TimmingPlan()) + "', "

+ "`status` = " + NumberToStringConverter<int>::Convert(this->GetStatus()->GetId()) + ", "

+ "`created\_at` = " + NumberToStringConverter<int>::Convert(this->CreatedAt()->Timestamp()) + ", "

+ "`expired\_at` = " + NumberToStringConverter<int>::Convert(this->ExpiredAt()->Timestamp()) + ", "

+ "`deadline` = " + NumberToStringConverter<int>::Convert(this->Deadline()->Timestamp()) + ", "

+ "`priority` = " + NumberToStringConverter<int>::Convert(this->Priority()) + " "

+ "WHERE `id` = " + NumberToStringConverter<int>::Convert(this->GetId());

return sql;

}

bool Project::Retrieve()

{

MYSQL\_RES\* res;

MYSQL\_ROW row;

string sql;

sql = "SELECT `id`, `name`, `description`, `team`, `timming\_plan`, `status`, `created\_at`, `expired\_at`, `deadline`, `priority` "

"FROM `" + this->table + "` "

+ "WHERE `id` = " + NumberToStringConverter<int>::Convert(this->id) + " LIMIT 0, 1";

try {

res = this->connector->Query(sql, SELECT);

}

catch(MySQLRecordNotFound\* rnf) {

std::cout << rnf->what() << std::endl;

return false;

}

row = mysql\_fetch\_row(res);

this->Name(row[1]);

this->Description(row[2]);

this->SetTeam(new Team(this->connector, atoi(row[3])));

this->TimmingPlan(atof(row[4]));

this->SetStatus(new Status(this->connector, atoi(row[5])));

this->CreatedAt(new DateTime(atoi(row[6])));

this->ExpiredAt(new DateTime(atoi(row[7])));

this->Deadline(new DateTime(atoi(row[8])));

this->Priority(atoi(row[9]));

return true;

}

BaseRecord\* Project::RetrieveRecord(int id)

{

return new Project(this->connector, id);

}

vector<BaseRecord \*> Project::RetrieveTableRecords(string where, string limit) const

{

vector<BaseRecord \*> projects;

BaseRecord\* baseRecord;

MYSQL\_RES\* res;

MYSQL\_ROW row;

Project\* project;

string sql;

sql = "SELECT `id`, `name`, `description`, `team`, `timming\_plan`, `status`, `created\_at`, `expired\_at`, `deadline`, `priority` FROM `"

+ this->table + "` " + where + " " + limit;

try {

res = this->connector->Query(sql, SELECT);

}

catch(MySQLRecordNotFound\* rnf) {

std::cout << rnf->what() << std::endl;

exit(ERROR\_MYSQL\_QUERY);

}

while(row = mysql\_fetch\_row(res)) {

project = new Project(this->connector);

project->id = atoi(row[0]);

project->Name(row[1]);

project->Description(row[2]);

project->SetTeam(new Team(this->connector, atoi(row[3])));

project->TimmingPlan(atof(row[4]));

project->SetStatus(new Status(this->connector, atoi(row[5])));

project->CreatedAt(new DateTime(atoi(row[6])));

project->ExpiredAt(new DateTime(atoi(row[7])));

project->Deadline(new DateTime(atoi(row[8])));

project->Priority(atoi(row[9]));

baseRecord = project;

projects.push\_back(baseRecord);

}

return projects;

}

**Report.h**

/\*

\* File: Report.h

\* Author: Alex Kozlov <alexssource@gmail.com>

\*

\* Created on December 1, 2011, 12:48 PM

\*/

#ifndef REPORT\_H

#define REPORT\_H

#include <iostream>

#include <string>

#include "Task.h"

#include "Project.h"

#include "DateTime.h"

using namespace std;

class Report: public BaseRecord

{

private:

Task\* task;

Project\* project;

string report;

float hours;

string tmpl;

DateTime\* created\_at;

void GetTemplate();

public:

Report(SqlConnector\* const);

Report(SqlConnector\* const, int);

Report(const Report &);

virtual ~Report();

void ProcessingReport();

// Properties:

Task\* GetTask() const { return this->task; }

void SetTask(Task\* task) { this->task = task; }

Project\* GetProject() const { return this->project; }

void SetProject(Project\* project) { this->project = project; }

string GetReport() const { return this->report; }

void SetReport(string report) { this->report = report; }

float Hours() const { return this->hours; }

void Hours(float hours) { this->hours = hours; }

string Tmpl() const { return this->tmpl; }

void Tmpl(string tmpl) { this->tmpl = tmpl; }

DateTime\* CreatedAt() const { return this->created\_at; }

void CreatedAt(DateTime\* created) { this->created\_at = created; }

// Override methods

protected:

virtual string GetSqlCreate() const;

virtual string GetSqlUpdate() const;

virtual void InitializeRecord();

public:

virtual bool Retrieve();

virtual BaseRecord\* RetrieveRecord(int id);

/\*\*

\* Retrieves the records from current table with various condition and limit

\* @param where - the string of condition

\* @param limit - the string of limit

\* @return vector&lt;BaseRecord \*&rt; - the vector of table rows

\*/

virtual vector<BaseRecord \*> RetrieveTableRecords(string where = "", string limit = "") const;

virtual bool IsSet() const;

};

#endif /\* REPORT\_H \*/

**Report.cpp**

#include "../header/Report.h"

Report::Report(SqlConnector\* const conn): BaseRecord(conn, "Report") { this->GetTemplate(); }

Report::Report(SqlConnector\* const conn, int id): BaseRecord(conn, "Report", id) { this->Retrieve(); this->GetTemplate(); }

Report::Report(const Report& orig): BaseRecord(orig.connector, orig.table)

{

this->SetTask(orig.GetTask());

this->SetProject(orig.GetProject());

this->SetReport(orig.GetReport());

this->Hours(orig.Hours());

this->CreatedAt(orig.CreatedAt());

this->Tmpl(orig.Tmpl());

}

Report::~Report()

{

delete this->task;

delete this->project;

delete this->created\_at;

}

void Report::InitializeRecord()

{

this->task = NULL;

this->project = NULL;

this->created\_at = NULL;

this->hours = 0.0;

}

void Report::GetTemplate()

{

MYSQL\_RES\* res;

MYSQL\_ROW row;

res = this->connector->Query("SELECT `template` FROM `Templates` WHERE `module` = 'Report' LIMIT 0,1", SELECT);

row = mysql\_fetch\_row(res);

this->Tmpl(row[0]);

}

void Report::ProcessingReport()

{

std::cout << "Its ProcessingReport" << std::endl;

}

string Report::GetSqlCreate() const

{

string sql;

sql = "INSERT INTO `" + this->table + "` (`task`, `project`, `report`, `hours`, `created\_at`) VALUE ("

+ NumberToStringConverter<int>::Convert(this->GetTask()->GetId()) + ", "

+ NumberToStringConverter<int>::Convert(this->GetProject()->GetId()) + ", "

+ "'" + this->GetReport() + "', "

+ "'" + NumberToStringConverter<float>::Convert(this->Hours()) + "', "

+ NumberToStringConverter<int>::Convert(this->CreatedAt()->Timestamp()) + ")";

return sql;

}

string Report::GetSqlUpdate() const

{

string sql;

sql = "UPDATE `" + this->table + "` SET "

+ "`task` = " + NumberToStringConverter<int>::Convert(this->GetTask()->GetId()) + ", "

+ "`project` = " + NumberToStringConverter<int>::Convert(this->GetProject()->GetId()) + ", "

+ "`report` = '" + this->GetReport() + "', "

+ "`hours` = '" + NumberToStringConverter<float>::Convert(this->Hours()) + "', "

+ "`created\_at` = " + NumberToStringConverter<int>::Convert(this->CreatedAt()->Timestamp())

+ " WHERE `id` = " + NumberToStringConverter<int>::Convert(this->id);

return sql;

}

bool Report::Retrieve()

{

MYSQL\_RES\* res;

MYSQL\_ROW row;

string sql;

sql = "SELECT `id`, `task`, `project`, `report`, `hours`, `created\_at` FROM `"

+ this->table + "` "

+ "WHERE `id` = " + NumberToStringConverter<int>::Convert(this->id) + " LIMIT 0, 1";

try {

res = this->connector->Query(sql, SELECT);

}

catch(MySQLRecordNotFound\* rnf) {

std::cout << rnf->what() << std::endl;

return false;

}

row = mysql\_fetch\_row(res);

this->SetTask(new Task(this->connector, atoi(row[1])));

this->SetProject(new Project(this->connector, atoi(row[2])));

this->SetReport(row[3]);

this->Hours(atof(row[4]));

this->CreatedAt(new DateTime(atoi(row[5])));

return true;

}

BaseRecord\* Report::RetrieveRecord(int id)

{

return new Report(this->connector, id);

}

vector<BaseRecord \*> Report::RetrieveTableRecords(string where, string limit) const

{

vector<BaseRecord \*> reports;

BaseRecord\* baseRecord;

MYSQL\_RES\* res;

MYSQL\_ROW row;

Report\* report;

string sql;

sql = "SELECT `id`, `task`, `project`, `report`, `hours`, `created\_at` FROM `"

+ this->table + "` " + where + " " + limit;

try {

res = this->connector->Query(sql, SELECT);

}

catch(MySQLRecordNotFound\* rnf) {

std::cout << rnf->what() << std::endl;

exit(ERROR\_MYSQL\_QUERY);

}

while(row = mysql\_fetch\_row(res)) {

report = new Report(this->connector);

report->id = atoi(row[0]);

report->SetTask(new Task(this->connector, atoi(row[1])));

report->SetProject(new Project(this->connector, atoi(row[2])));

report->SetReport(row[3]);

report->Hours(atof(row[4]));

report->CreatedAt(new DateTime(atoi(row[5])));

baseRecord = report;

reports.push\_back(baseRecord);

}

return reports;

}

bool Report::IsSet() const

{

MYSQL\_RES\* res;

string sql;

sql = "SELECT `id` FROM `" + this->table + "` WHERE "

+ "`task` = " + NumberToStringConverter<int>::Convert(this->GetTask()->GetId()) + " "

+ "AND `project` = " + NumberToStringConverter<int>::Convert(this->GetProject()->GetId()) + " "

+ "LIMIT 0, 2";

res = this->connector->Query(sql, SELECT);

if(mysql\_num\_rows(res) != 0) {

return true;

}

else {

return false;

}

}

**SqlConnector.h**

/\*

\* File: SqlConnector.h

\* Author: alex

\*

\* Created on December 5, 2011, 12:15 PM

\*/

#ifndef SQLCONNECTOR\_H

#define SQLCONNECTOR\_H

#include <iostream>

#include <string>

#include <cstdlib>

#include "mysql/mysql.h"

#include "ERROR\_CODES.h"

#include "Exception.h"

using namespace std;

class SqlConnector

{

private:

MYSQL connection;

string user;

string pass;

string host;

string database;

protected:

bool Connect();

void Initialize(string host, string user, string pass, string database);

void SetCharset();

public:

SqlConnector(string host, string user, string pass, string database);

virtual ~SqlConnector();

MYSQL\_RES\* Query(string sql, MYSQL\_QUERY\_TYPE type);

MYSQL GetConnection() const { return this->connection; }

int GetLastInsertedId();

};

#endif /\* SQLCONNECTOR\_H \*/

**SqlConnector.cpp**

/\*

\* File: SqlConnector.cpp

\* Author: alex

\*

\* Created on December 5, 2011, 12:15 PM

\*/

#include "../header/SqlConnector.h"

SqlConnector::SqlConnector(string host, string user, string pass, string database)

{

this->Initialize(host, user, pass, database);

if(!this->Connect()) {

string error = "MySQL ERROR: ";

error.append(mysql\_error(&this->connection));

throw new Exception(ERROR\_MYSQL\_CONNECT, ERROR\_TYPE\_MYSQL, error);

}

this->SetCharset();

}

SqlConnector::~SqlConnector()

{

mysql\_close(&this->connection);

}

void SqlConnector::Initialize(string host, string user, string pass, string database)

{

this->host = host;

this->user = user;

this->pass = pass;

this->database = database;

}

void SqlConnector::SetCharset()

{

this->Query("SET NAMES `UTF8`", OTHER);

}

bool SqlConnector::Connect()

{

mysql\_init(&this->connection);

return mysql\_real\_connect(&this->connection, this->host.c\_str(), this->user.c\_str(),

this->pass.c\_str(), this->database.c\_str(), 0, NULL, 0);

}

MYSQL\_RES\* SqlConnector::Query(string sql, MYSQL\_QUERY\_TYPE type)

{

int result;

MYSQL\_RES\* res = NULL;

result = mysql\_query(&this->connection, sql.c\_str());

if(result != 0) throw new MySQLQueryException(sql, mysql\_error(&this->connection));

res = mysql\_store\_result(&this->connection);

// cout << mysql\_num\_rows(res) << endl;

if(!res && type == SELECT) {

throw new MySQLRecordNotFound();

return res;

}

if(!mysql\_affected\_rows(&this->connection)) {

switch(type) {

case UPDATE:

throw new MySQLNoUpdateRecord(mysql\_error(&this->connection));

break;

case INSERT:

throw new MySQLNoInsertRecord(mysql\_error(&this->connection));

break;

case DELETE:

throw new MySQLNoDeleteRecord(mysql\_error(&this->connection));

break;

}

}

return res;

}

int SqlConnector::GetLastInsertedId()

{

MYSQL\_RES\* res;

MYSQL\_ROW sqlrow;

try {

res = this->Query("SELECT LAST\_INSERT\_ID()", SELECT);

}

catch(MySQLQueryException\* qexc) {

std::cout << qexc->what() << std::endl;

}

sqlrow = mysql\_fetch\_row(res);

return atoi(sqlrow[0]);

}

**Status.h**

/\*

\* File: Status.h

\* Author: Alex Kozlov <alexssource@gmail.com>

\*

\* Created on December 1, 2011, 1:59 PM

\*/

#ifndef STATUS\_H

#define STATUS\_H

#include <iostream>

#include <string>

#include "BaseRecord.h"

#include "SqlConnector.h"

using namespace std;

class Status: public BaseRecord

{

private:

string name;

public:

Status(SqlConnector\* const);

Status(SqlConnector\* const, int);

Status(const Status&);

virtual ~Status();

// Properties:

string Name() const { return this->name; }

void Name(string name) { this->name = name; }

// Section for overrides methods

// Oveerided methods

protected:

virtual string GetSqlCreate() const;

virtual string GetSqlUpdate() const;

virtual bool IsSet() const;

public:

virtual bool Retrieve();

virtual BaseRecord\* RetrieveRecord(int id);

/\*\*

\* Retrieves the records from current table with various condition and limit

\* @param where - the string of condition

\* @param limit - the string of limit

\* @return vector&lt;BaseRecord \*&rt; - the vector of table rows

\*/

virtual vector<BaseRecord \*> RetrieveTableRecords(string where = "", string limit = "") const;

};

#endif /\* STATUS\_H \*/

**Status.cpp**

#include "../header/Status.h"

Status::Status(SqlConnector\* const conn): BaseRecord(conn, "Status") { }

Status::Status(SqlConnector\* const conn, int id): BaseRecord(conn, "Status", id) { this->Retrieve(); }

Status::~Status() { }

Status::Status(const Status& orig): BaseRecord(orig.connector, orig.table) {

if(orig.GetId()) {

this->id = orig.GetId();

}

this->Name(orig.Name());

}

string Status::GetSqlCreate() const

{

string sql;

sql = "INSERT INTO `" + this->table + "` (`name`) VALUE ('" + this->Name() + "')";

return sql;

}

string Status::GetSqlUpdate() const

{

string sql;

sql = "UPDATE `" + this->table +"` SET `name` = '" + this->Name() + "' WHERE `id` = "

+ NumberToStringConverter<int>::Convert(this->GetId());

return sql;

}

bool Status::Retrieve()

{

MYSQL\_RES\* res;

MYSQL\_ROW row;

string sql;

sql = "SELECT `id`, `name` FROM `" + this->table + "` "

+ "WHERE `id` = " + NumberToStringConverter<int>::Convert(this->id)

+ " LIMIT 0, 1";

try {

res = this->connector->Query(sql, SELECT);

}

catch(MySQLRecordNotFound\* rnf) {

std::cout << rnf->what() << std::endl;

return false;

}

row = mysql\_fetch\_row(res);

this->Name(row[1]);

return true;

}

BaseRecord\* Status::RetrieveRecord(int id)

{

Status\* status = new Status(this->connector, id);

return status;

}

vector<BaseRecord \*> Status::RetrieveTableRecords(string where, string limit) const

{

vector<BaseRecord \*> statuses;

BaseRecord\* baseRecord;

MYSQL\_RES\* res;

MYSQL\_ROW row;

Status\* status;

string sql;

sql = "SELECT `id`, `name` FROM `" + this->table + "` " + where + " " + limit;

try {

res = this->connector->Query(sql, SELECT);

}

catch(MySQLRecordNotFound\* rnf) {

std::cout << rnf->what() << std::endl;

exit(ERROR\_MYSQL\_QUERY);

}

while(row = mysql\_fetch\_row(res)) {

status = new Status(this->connector);

status->id = atoi(row[0]);

status->Name(row[1]);

baseRecord = status;

statuses.push\_back(baseRecord);

}

return statuses;

}

bool Status::IsSet() const

{

MYSQL\_RES\* res;

string sql;

sql = "SELECT `id` FROM `" + this->table + "` WHERE `name` = '" + this->Name() + "' LIMIT 0, 2";

res = this->connector->Query(sql, SELECT);

if(mysql\_num\_rows(res) != 0) {

return true;

}

else {

return false;

}

}

**Task.h**

/\*

\* File: Task.h

\* Author: Alex Kozlov <alexssource@gmail.com>

\*

\* Created on December 1, 2011, 2:31 PM

\*/

#ifndef TASK\_H

#define TASK\_H

#include <iostream>

#include <string>

#include "BaseRecord.h"

#include "SqlConnector.h"

#include "Project.h"

#include "User.h"

#include "Status.h"

#include "DateTime.h"

using namespace std;

class Task: public BaseRecord

{

private:

string name;

string title;

string description;

int priority;

float timming\_plan;

Project\* project;

User\* user;

Status\* status;

DateTime\* created\_at;

public:

Task(SqlConnector\* const);

Task(SqlConnector\* const, int);

Task(const Task &);

virtual ~Task();

// Properties:

string Name() const { return this->name; }

void Name(string name) { this->name = name; }

string Title() const { return this->title; }

void Title(string title) { this->title = title; }

string Description() const { return this->description; }

void Description(string description) { this->description = description; }

float TimmingPlan() const { return this->timming\_plan; }

void TimmingPlan(float tp) { this->timming\_plan = tp; }

Project\* GetProject() const { return this->project; }

void SetProject(Project\* project) { this->project = project; }

void SetProject(int pid) { this->project = new Project(this->connector, pid); }

User\* GetUser() const { return this->user; }

void SetUser(User\* user) { this->user = user; }

void SetUser(int uid) { this->user = new User(this->connector, uid); }

Status\* GetStatus() const { return this->status; }

void SetStatus(Status\* status) { this->status = status; }

void SetStatus(int sid) { this->status = new Status(this->connector, sid); }

int Priority() const { return this->priority; }

void Priority(int p) { this->priority = p; }

DateTime\* CreatedAt() const { return this->created\_at; }

void CreatedAt(DateTime\* ca) { this->created\_at = ca; }

// Oveerided methods

protected:

virtual string GetSqlCreate() const;

virtual string GetSqlUpdate() const;

virtual void InitializeRecord();

public:

virtual bool Retrieve();

virtual BaseRecord\* RetrieveRecord(int id);

/\*\*

\* Retrieves the records from current table with various condition and limit

\* @param where - the string of condition

\* @param limit - the string of limit

\* @return vector&lt;BaseRecord \*&rt; - the vector of table rows

\*/

virtual vector<BaseRecord \*> RetrieveTableRecords(string where = "", string limit = "") const;

};

#endif /\* TASK\_H \*/

**Task.cpp**

#include "../header/Task.h"

Task::Task(SqlConnector\* const conn): BaseRecord(conn, "Task") { }

Task::Task(SqlConnector\* const conn, int id): BaseRecord(conn, "Task", id) { this->Retrieve(); }

Task::Task(const Task& orig): BaseRecord(orig.connector, orig.table)

{

this->Name(orig.Name());

this->Title(orig.Title());

this->Description(orig.Description());

this->TimmingPlan(orig.TimmingPlan());

this->SetProject(orig.GetProject());

this->SetUser(orig.GetUser());

this->SetStatus(orig.GetStatus());

this->CreatedAt(orig.CreatedAt());

this->Priority(orig.Priority());

}

Task::~Task()

{

delete this->project;

delete this->user;

delete this->status;

delete this->created\_at;

}

void Task::InitializeRecord()

{

this->project = NULL;

this->user = NULL;

this->status = NULL;

this->created\_at = NULL;

this->InitializeRecord();

}

string Task::GetSqlCreate() const

{

string sql;

sql = "INSERT INTO `" + this->table + "` (`name`, `title`, `description`, `timming\_plan`, "

+ "`project`, `user`, `status`, `created\_at`, `priority`) VALUE ("

+ "'" + this->Name() + "', "

+ "'" + this->Title() + "', "

+ "'" + this->Description() + "', "

+ "'" + NumberToStringConverter<float>::Convert(this->TimmingPlan()) + "', "

+ NumberToStringConverter<int>::Convert(this->GetProject()->GetId()) + ", "

+ NumberToStringConverter<int>::Convert(this->GetUser()->GetId()) + ", "

+ NumberToStringConverter<int>::Convert(this->GetStatus()->GetId()) + ", "

+ NumberToStringConverter<int>::Convert(this->CreatedAt()->Timestamp()) + ", "

+ NumberToStringConverter<int>::Convert(this->Priority()) + ")";

return sql;

}

string Task::GetSqlUpdate() const

{

string sql;

sql = "UPDATE `" + this->table +"` SET "

+ "`name` = '" + this->Name() + "', "

+ "`title` = '" + this->Title() + "', "

+ "`description` = '" + this->Description() + "', "

+ "`timming\_plan` = '" + NumberToStringConverter<float>::Convert(this->TimmingPlan()) + "', "

+ "`project` = " + NumberToStringConverter<int>::Convert(this->GetProject()->GetId()) + ", "

+ "`user` = " + NumberToStringConverter<int>::Convert(this->GetUser()->GetId()) + ", "

+ "`status` = " + NumberToStringConverter<int>::Convert(this->GetStatus()->GetId()) + ", "

+ "`created\_at` = " + NumberToStringConverter<int>::Convert(this->CreatedAt()->Timestamp()) + ", "

+ "`priority` = " + NumberToStringConverter<int>::Convert(this->Priority()) + " "

+ "WHERE `id` = " + NumberToStringConverter<int>::Convert(this->GetId());

return sql;

}

bool Task::Retrieve()

{

MYSQL\_RES\* res;

MYSQL\_ROW row;

string sql;

sql = "SELECT `id`, `name`, `title`, `description`, `timming\_plan`, `project`, `user`, `status`, `created\_at`, `priority` FROM `"

+ this->table + "` "

+ "WHERE `id` = " + NumberToStringConverter<int>::Convert(this->id) + " LIMIT 0, 1";

try {

res = this->connector->Query(sql, SELECT);

}

catch(MySQLRecordNotFound\* rnf) {

std::cout << rnf->what() << std::endl;

return false;

}

row = mysql\_fetch\_row(res);

this->Name(row[1]);

this->Title(row[2]);

this->Description(row[3]);

this->TimmingPlan(atof(row[4]));

this->SetProject(new Project(this->connector, atoi(row[5])));

this->SetUser(new User(this->connector, atoi(row[6])));

this->SetStatus(new Status(this->connector, atoi(row[7])));

this->CreatedAt(new DateTime(atoi(row[8])));

this->Priority(atoi(row[9]));

return true;

}

BaseRecord\* Task::RetrieveRecord(int id)

{

return new Task(this->connector, id);

}

vector<BaseRecord \*> Task::RetrieveTableRecords(string where, string limit) const

{

vector<BaseRecord \*> tasks;

BaseRecord\* baseRecord;

MYSQL\_RES\* res;

MYSQL\_ROW row;

Task\* task;

string sql;

sql = "SELECT t.`id`, t.`name`, t.`title`, t.`description`, t.`timming\_plan`, t.`project`, t.`user`, t.`status`, t.`created\_at`, t.`priority` FROM `"

+ this->table + "` t " + where + " " + limit;

try {

res = this->connector->Query(sql, SELECT);

}

catch(MySQLRecordNotFound\* rnf) {

std::cout << rnf->what() << std::endl;

exit(ERROR\_MYSQL\_QUERY);

}

while(row = mysql\_fetch\_row(res)) {

task = new Task(this->connector);

task->id = atoi(row[0]);

task->Name(row[1]);

task->Title(row[2]);

task->Description(row[3]);

task->TimmingPlan(atof(row[4]));

task->SetProject(new Project(this->connector, atoi(row[5])));

task->SetUser(new User(this->connector, atoi(row[6])));

task->SetStatus(new Status(this->connector, atoi(row[7])));

task->CreatedAt(new DateTime(atoi(row[8])));

task->Priority(atoi(row[9]));

baseRecord = task;

tasks.push\_back(baseRecord);

}

return tasks;

}

**Team.h**

/\*

\* File: Team.h

\* Author: alex

\*

\* Created on December 18, 2011, 7:32 PM

\*/

#ifndef TEAM\_H

#define TEAM\_H

#include "BaseRecord.h"

#include "DateTime.h"

class Team: public BaseRecord

{

private:

string name;

DateTime\* created\_at;

public:

Team(SqlConnector\* const conn);

Team(SqlConnector\* const conn, int id);

Team(const Team& orig);

virtual ~Team();

// Properties:

string Name() const { return this->name; }

void Name(string name) { this->name = name; }

DateTime\* CreatedAt() const { return this->created\_at; }

void CreatedAt(DateTime\* created\_at) { this->created\_at = created\_at; }

void CreatedAt(int timestamp) { this->created\_at = new DateTime(timestamp); }

void CreatedAt(int year, int month, int day = 1, int hour = 0, int minute = 0, int seconds = 0) {

this->created\_at = new DateTime(year, month, day, hour, minute, seconds);

}

// Oveerided methods

protected:

virtual string GetSqlCreate() const;

virtual string GetSqlUpdate() const;

public:

virtual bool Retrieve();

virtual BaseRecord\* RetrieveRecord(int id);

virtual vector<BaseRecord \*> RetrieveTableRecords(string where = "", string limit = "") const;

};

#endif /\* TEAM\_H \*/

**Team.cpp**

/\*

\* File: Team.cpp

\* Author: alex

\*

\* Created on December 18, 2011, 7:32 PM

\*/

#include "../header/Team.h"

Team::Team(SqlConnector\* const conn): BaseRecord(conn, "Team") { }

Team::Team(SqlConnector\* const conn, int id): BaseRecord(conn, "Team", id) { this->Retrieve(); }

Team::Team(const Team& orig): BaseRecord(orig.connector, orig.table)

{

this->Name(orig.Name());

this->CreatedAt(orig.CreatedAt());

}

Team::~Team() { }

string Team::GetSqlCreate() const

{

string sql;

sql = "INSERT INTO `" + this->table + "` (`name`, `created\_at`) "

+ "VALUE ('" + this->name + "', "

+ NumberToStringConverter<int>::Convert(this->CreatedAt()->Timestamp()) + ")";

return sql;

}

string Team::GetSqlUpdate() const

{

string sql;

sql = "UPDATE `" + this->table + "` SET `name`='" + this->name + "', "

+ "`created\_at` = " + NumberToStringConverter<int>::Convert(this->CreatedAt()->Timestamp()) + " "

+ "WHERE `id` = " + NumberToStringConverter<int>::Convert(this->id);

return sql;

}

bool Team::Retrieve()

{

MYSQL\_RES\* res;

MYSQL\_ROW row;

string sql;

sql = "SELECT `id`, `name`, `created\_at` FROM `" + this->table

+ "` WHERE `id` = " + NumberToStringConverter<int>::Convert(this->id)

+ " LIMIT 0, 1";

try {

res = this->connector->Query(sql, SELECT);

}

catch(MySQLRecordNotFound\* rnf) {

std::cout << rnf->what() << std::endl;

return false;

}

row = mysql\_fetch\_row(res);

this->Name(row[1]);

this->CreatedAt(new DateTime(atoi(row[2])));

return true;

}

BaseRecord\* Team::RetrieveRecord(int id)

{

Team\* tm = new Team(this->connector, id);

return tm;

}

vector <BaseRecord \*> Team::RetrieveTableRecords(string where, string limit) const

{

vector<BaseRecord \*> teams;

BaseRecord\* baseRecord;

MYSQL\_RES\* res;

MYSQL\_ROW row;

Team\* team;

string sql;

sql = "SELECT `id`, `name`, `created\_at` FROM `" + this->table + "` "

+ where + " " + limit;

try {

res = this->connector->Query(sql, SELECT);

}

catch(MySQLRecordNotFound\* rnf) {

std::cout << rnf->what() << std::endl;

exit(ERROR\_MYSQL\_QUERY);

}

while(row = mysql\_fetch\_row(res)) {

team = new Team(this->connector);

team->id = atoi(row[0]);

team->Name(row[1]);

team->CreatedAt(new DateTime(atoi(row[2])));

baseRecord = team;

teams.push\_back(baseRecord);

}

return teams;

}

**User.h**

/\*

\* File: User.h

\* Author: Alex Kozlov <alexssource@gmail.com>

\*

\* Created on December 1, 2011, 11:02 AM

\*/

#ifndef USER\_H

#define USER\_H

#include "BaseRecord.h"

#include "DateTime.h"

#include "Group.h"

#include "Team.h"

using namespace std;

class User: public BaseRecord

{

private:

string name;

string surname;

string login;

string password;

string email;

float wage; // часовая тарифная ставка

Group\* group;

Team\* team;

DateTime\* created\_at;

public:

User(SqlConnector\* const);

User(SqlConnector\* const, int);

User(const User&);

virtual ~User();

// Properties:

string Name() const { return this->name; }

void Name(string name) { this->name = name; }

string Surname() const { return this->surname; }

void Surname(string surname) { this->surname = surname; }

string Login() const { return this->login; }

void Login(string login) { this->login = login; }

string Password() const { return this->password; }

void Password(string pass) { this->password = pass; }

string Email() const { return this->email; }

void Email(string email) { this->email = email; }

float Wage() const { return this->wage; }

void Wage(float w) { this->wage = w; }

Group\* GetGroup() const { return this->group; }

void SetGroup(Group\* group) { this->group = group; }

void SetGroup(int group\_id) { this->group = new Group(this->connector, group\_id); }

Team\* GetTeam() const { return this->team; }

void SetTeam(Team\* team) { this->team = team; }

void SetTeam(int team\_id) { this->team = new Team(this->connector, team\_id); }

DateTime\* CreatedAt() const { return this->created\_at; }

void CreatedAt(DateTime\* created\_at) { this->created\_at = created\_at; }

void CreatedAt(int timestamp) { this->created\_at = new DateTime(timestamp); }

void CreatedAt(int year, int month, int day = 1, int hour = 0, int minute = 0, int seconds = 0) {

this->created\_at = new DateTime(year, month, day, hour, minute, seconds);

}

bool RetrieveByLoginPass();

// Section for overrides methods

// Oveerided methods

protected:

virtual string GetSqlCreate() const;

virtual string GetSqlUpdate() const;

virtual void InitializeRecord();

virtual bool IsSet() const;

public:

virtual bool Retrieve();

virtual BaseRecord\* RetrieveRecord(int id);

/\*\*

\* Retrieves the records from current table with various condition and limit

\* @param where - the string of condition

\* @param limit - the string of limit

\* @return vector&lt;BaseRecord \*&rt; - the vector of table rows

\*/

virtual vector<BaseRecord \*> RetrieveTableRecords(string where = "", string limit = "") const;

};

#endif /\* USER\_H \*/

**User.cpp**

#include "../header/User.h"

User::User(SqlConnector\* const conn): BaseRecord(conn, "User") { }

User::User(SqlConnector\* const conn, int id): BaseRecord(conn, "User", id) { this->Retrieve(); }

User::User(const User& orig): BaseRecord(orig.connector, orig.table)

{

this->Name(orig.Name());

this->Surname(orig.Surname());

this->Login(orig.Login());

this->Password(orig.Password());

this->Email(orig.Email());

this->Wage(orig.Wage());

this->SetGroup(orig.GetGroup());

this->SetTeam(orig.GetTeam());

this->CreatedAt(orig.CreatedAt());

}

User::~User()

{

if(this->team != NULL) {

delete this->team;

this->team = NULL;

}

if(this->group != NULL) {

delete this->group;

this->group = NULL;

}

}

void User::InitializeRecord()

{

this->group = NULL;

this->team = NULL;

}

string User::GetSqlCreate() const

{

string sql;

sql = "INSERT INTO `" + this->table + "` (`name`, `surname`, `login`, "

+ "`password`, `email`, `group`, `team`, `created\_at`, `wage`) "

+ "VALUE ('" + this->Name() + "', '" + this->Surname() + "', '" + this->Login() + "', "

+ "'" + this->Password() + "', '" + this->Email() + "', "

+ NumberToStringConverter<int>::Convert(this->GetGroup()->GetId()) + ", "

+ NumberToStringConverter<int>::Convert(this->GetTeam()->GetId()) + ", '"

+ NumberToStringConverter<int>::Convert(this->CreatedAt()->Timestamp()) + "', '"

+ NumberToStringConverter<float>::Convert(this->Wage()) + "')";

return sql;

}

string User::GetSqlUpdate() const

{

string sql;

sql = "UPDATE `" + this->table + "` SET `name`='" + this->Name() + "', "

+ "`surname` = '" + this->Surname() + "', "

+ "`login` = '" + this->Login() + "', "

+ "`password` = '" + this->Password() + "', "

+ "`email` = '" + this->Email() + "', "

+ "`group` = " + NumberToStringConverter<int>::Convert(this->GetGroup()->GetId()) + ", "

+ "`team` = " + NumberToStringConverter<int>::Convert(this->GetTeam()->GetId()) + ", "

+ "`created\_at` = '" + NumberToStringConverter<int>::Convert(this->CreatedAt()->Timestamp()) + "', "

+ "`wage` = '" + NumberToStringConverter<float>::Convert(this->Wage()) + "' "

+ "WHERE `id` = " + NumberToStringConverter<int>::Convert(this->id);

return sql;

}

bool User::Retrieve()

{

MYSQL\_RES\* res;

MYSQL\_ROW row;

string sql;

sql = "SELECT `id`, `name`, `surname`, `login`, `password`, `email`, `group`, `team`, `created\_at`, `wage` FROM `"

+ this->table + "` "

+ "WHERE `id` = " + NumberToStringConverter<int>::Convert(this->id)

+ " LIMIT 0, 1";

try {

res = this->connector->Query(sql, SELECT);

}

catch(MySQLRecordNotFound\* rnf) {

std::cout << rnf->what() << std::endl;

return false;

}

row = mysql\_fetch\_row(res);

this->Name(row[1]);

this->Surname(row[2]);

this->Login(row[3]);

this->Password(row[4]);

this->Email(row[5]);

this->group = new Group(this->connector, atoi(row[6]));

this->team = new Team(this->connector, atoi(row[7]));

this->created\_at = new DateTime(atoi(row[8]));

this->Wage(atof(row[9]));

return true;

}

BaseRecord\* User::RetrieveRecord(int id)

{

User\* usr = new User(this->connector, id);

return usr;

}

vector<BaseRecord \*> User::RetrieveTableRecords(string where, string limit) const

{

vector<BaseRecord \*> users;

BaseRecord\* baseRecord;

MYSQL\_RES\* res;

MYSQL\_ROW row;

User\* user;

string sql;

sql = "SELECT `id`, `name`, `surname`, `login`, `password`, `email`, `group`, `team`, `created\_at`, `wage` FROM `"

+ this->table + "` "

+ where + " " + limit;

try {

res = this->connector->Query(sql, SELECT);

}

catch(MySQLRecordNotFound\* rnf) {

std::cout << rnf->what() << std::endl;

exit(ERROR\_MYSQL\_QUERY);

}

while(row = mysql\_fetch\_row(res)) {

user = new User(this->connector);

user->id = atoi(row[0]);

user->Name(row[1]);

user->Surname(row[2]);

user->Login(row[3]);

user->Password(row[4]);

user->Email(row[5]);

user->SetGroup(atoi(row[6]));

user->SetTeam(atoi(row[7]));

user->CreatedAt(atoi(row[8]));

user->Wage(atof(row[9]));

baseRecord = user;

users.push\_back(baseRecord);

}

return users;

}

bool User::IsSet() const

{

MYSQL\_RES\* res;

string sql;

sql = "SELECT `id` FROM `" + this->table + "` WHERE `login` = '" + this->Login() + "' LIMIT 0, 2";

res = this->connector->Query(sql, SELECT);

if(mysql\_num\_rows(res) != 0) {

return true;

}

else {

return false;

}

}

bool User::RetrieveByLoginPass()

{

MYSQL\_RES\* res;

MYSQL\_ROW row;

string sql;

sql = "SELECT `id`, `name`, `surname`, `login`, `password`, `email`, `group`, `team`, `created\_at`, `wage` FROM `"

+ this->table + "` "

+ "WHERE `login` = '" + this->Login() + "' "

+ "AND `password` = '" + this->Password() + "' "

+ "LIMIT 0, 1";

res = this->connector->Query(sql, SELECT);

if(mysql\_num\_rows(res) == 0) {

throw new MySQLAuthorizationException(this->Login(), this->Password());

}

row = mysql\_fetch\_row(res);

this->id = atoi(row[0]);

this->Name(row[1]);

this->Surname(row[2]);

this->Password(row[4]);

this->Email(row[5]);

this->SetGroup(atoi(row[6]));

this->SetTeam(atoi(row[7]));

this->CreatedAt(atoi(row[8]));

this->Wage(atof(row[9]));

return true;

}

**Wages.h**

/\*

\* File: Wages.h

\* Author: alex

\*

\* Created on December 26, 2011, 2:20 AM

\*/

#ifndef WAGES\_H

#define WAGES\_H

#include "SqlConnector.h"

#include "WorkTime.h"

class Wages

{

private:

float wages;

float premium;

float hours;

User\* user;

DateTime\* dateStart;

DateTime\* dateEnd;

SqlConnector\* connector;

void Initialize();

void PostConstructor();

public:

Wages(SqlConnector\* const conn, User\* const user);

Wages(SqlConnector\* const conn, User\* const user, DateTime\* const dateStart, DateTime\* const dateEnd);

Wages(const Wages& orig);

virtual ~Wages();

// Properties

float GetWages() const { return this->wages; }

void SetWages(float w) { this->wages = w; }

float GetWagesWithPremium() const { return this->GetWages() + this->Premium(); }

float Premium() const { return this->premium; }

void Premium(float p) { this->premium = p; }

User\* GetUser() const { return this->user; }

void SetUser(User\* u) { this->user = u; }

DateTime\* DateStart() const { return this->dateStart; }

void DateStart(DateTime\* ds) { this->dateStart = ds; }

DateTime\* DateEnd() const { return this->dateEnd; }

void DateEnd(DateTime\* de) { this->dateEnd = de; }

float Hours() const { return this->hours; }

void Hours(float delta) { this->hours += delta; }

};

#endif /\* WAGES\_H \*/

**Wages.cpp**

/\*

\* File: Wages.cpp

\* Author: alex

\*

\* Created on December 26, 2011, 2:20 AM

\*/

#include "../header/Wages.h"

Wages::Wages(SqlConnector\* const conn, User\* u): connector(conn)

{

DateTime\* datestart = new DateTime(); // current DateTime -

datestart->AddSomeDays(-31); // minus 31 day

DateTime\* dateend = new DateTime(); // current DateTime

this->Initialize();

this->SetUser(u);

this->DateStart(datestart);

this->DateEnd(dateend);

this->PostConstructor();

}

Wages::Wages(SqlConnector\* const conn, User\* u, DateTime\* ds, DateTime\* de): connector(conn)

{

this->Initialize();

this->SetUser(u);

this->DateStart(ds);

this->DateEnd(de);

this->PostConstructor();

}

Wages::Wages(const Wages& orig)

{

this->Initialize();

this->SetWages(orig.GetWages());

this->Premium(orig.Premium());

this->SetUser(orig.GetUser());

this->DateStart(orig.DateStart());

this->DateEnd(orig.DateEnd());

}

Wages::~Wages()

{

delete this->user;

delete this->dateStart;

delete this->dateEnd;

this->Initialize();

}

void Wages::Initialize()

{

this->wages = 0;

this->premium = 0;

this->hours = 0;

this->SetUser(NULL);

this->DateStart(NULL);

this->DateEnd(NULL);

}

void Wages::PostConstructor()

{

vector<BaseRecord \*> works;

WorkTime\* work;

this->Hours(0);

string where = "WHERE `user` = " + NumberToStringConverter<int>::Convert(this->GetUser()->GetId())

+ " AND `day` > " + NumberToStringConverter<int>::Convert(this->DateStart()->Timestamp())

+ " AND `day` < " + NumberToStringConverter<int>::Convert(this->DateEnd()->Timestamp());

WorkTime\* userWork = new WorkTime(this->connector);

works = userWork->RetrieveTableRecords(where);

for(int i = 0; i < works.size(); i++) {

work = (WorkTime \*) works[i];

this->Hours(work->Hours());

}

// Calculate the wages

this->SetWages(this->GetUser()->Wage() \* hours);

}

**WorkTime.h**

/\*

\* File: WorkTime.h

\* Author: Alex Kozlov <alexssource@gmail.com>

\*

\* Created on December 5, 2011, 12:05 PM

\*/

#ifndef WORKTIME\_H

#define WORKTIME\_H

#include <string>

#include "BaseRecord.h"

#include "DateTime.h"

#include "User.h"

using namespace std;

class WorkTime: public BaseRecord

{

private:

DateTime\* day;

float hours;

User\* user;

public:

WorkTime(SqlConnector\* const);

WorkTime(SqlConnector\* const, int);

WorkTime(const WorkTime&);

virtual ~WorkTime();

void AddTodayTime(float delta = 0.016);

// Properties:

DateTime\* Day() const { return this->day; }

void Day(DateTime\* d) { this->day = d; }

float Hours() const { return this->hours; }

void Hours(float h) { this->hours = h; }

User\* GetUser() const { return this->user; }

void SetUser(User\* u) { this->user = u; }

void SetUser(int uid) { this->user = new User(this->connector, uid); }

// Override methods

protected:

virtual string GetSqlCreate() const;

virtual string GetSqlUpdate() const;

virtual void InitializeRecord();

public:

virtual bool Retrieve();

virtual BaseRecord\* RetrieveRecord(int id);

/\*\*

\* Retrieves the records from current table with various condition and limit

\* @param where - the string of condition

\* @param limit - the string of limit

\* @return vector&lt;BaseRecord \*&rt; - the vector of table rows

\*/

virtual vector<BaseRecord \*> RetrieveTableRecords(string where = "", string limit = "") const;

virtual bool IsSet() const;

};

#endif /\* WORKTIME\_H \*/

**WorkTime.cpp**

#include "../header/WorkTime.h"

WorkTime::WorkTime(SqlConnector\* const conn): BaseRecord(conn, "WorkTime") { }

WorkTime::WorkTime(SqlConnector\* const conn, int id): BaseRecord(conn, "WorkTime", id) { this->Retrieve(); }

WorkTime::WorkTime(const WorkTime& orig): BaseRecord(orig.connector, orig.table)

{

this->Day(orig.Day());

this->Hours(orig.Hours());

this->SetUser(orig.GetUser());

}

WorkTime::~WorkTime()

{

delete this->day;

delete this->user;

this->InitializeRecord();

}

void WorkTime::InitializeRecord()

{

this->day = NULL;

this->user = NULL;

this->hours = 0.0;

}

void WorkTime::AddTodayTime(float delta)

{

this->hours += delta;

}

string WorkTime::GetSqlCreate() const

{

string sql;

sql = "INSERT INTO `" + this->table + "` (`day`, `hours`, `user`) VALUE ("

+ NumberToStringConverter<int>::Convert(this->Day()->Timestamp()) + ", "

+ "'" + NumberToStringConverter<float>::Convert(this->Hours()) + "', "

+ NumberToStringConverter<int>::Convert(this->GetUser()->GetId()) + ")";

return sql;

}

string WorkTime::GetSqlUpdate() const

{

string sql;

sql = "UPDATE `" + this->table + "` SET "

+ "`day` = " + NumberToStringConverter<int>::Convert(this->Day()->Timestamp()) + ", "

+ "`hours` = '" + NumberToStringConverter<float>::Convert(this->Hours()) + "', "

+ "`user` = " + NumberToStringConverter<int>::Convert(this->GetUser()->GetId()) + " "

+ "WHERE `id` + " + NumberToStringConverter<int>::Convert(this->id);

return sql;

}

bool WorkTime::Retrieve()

{

MYSQL\_RES\* res;

MYSQL\_ROW row;

string sql;

sql = "SELECT `id`, `day`, `hours`, `user` FROM `"

+ this->table + "` "

+ "WHERE `id` = " + NumberToStringConverter<int>::Convert(this->id) + " LIMIT 0, 1";

try {

res = this->connector->Query(sql, SELECT);

}

catch(MySQLRecordNotFound\* rnf) {

std::cout << rnf->what() << std::endl;

return false;

}

row = mysql\_fetch\_row(res);

this->Day(new DateTime(atoi(row[1])));

this->Hours(atof(row[2]));

this->SetUser(new User(this->connector, atoi(row[3])));

return true;

}

BaseRecord\* WorkTime::RetrieveRecord(int id)

{

return new WorkTime(this->connector, id);

}

vector<BaseRecord \*> WorkTime::RetrieveTableRecords(string where, string limit) const

{

vector<BaseRecord \*> works;

BaseRecord\* baseRecord;

MYSQL\_RES\* res;

MYSQL\_ROW row;

WorkTime\* work;

string sql;

sql = "SELECT `id`, `day`, `hours`, `user` FROM `"

+ this->table + "` " + where + " " + limit;

try {

res = this->connector->Query(sql, SELECT);

}

catch(MySQLRecordNotFound\* rnf) {

std::cout << rnf->what() << std::endl;

exit(ERROR\_MYSQL\_QUERY);

}

while(row = mysql\_fetch\_row(res)) {

work = new WorkTime(this->connector);

work->id = atoi(row[0]);

work->Day(new DateTime(atoi(row[1])));

work->Hours(atof(row[2]));

work->SetUser(new User(this->connector, atoi(row[3])));

baseRecord = work;

works.push\_back(baseRecord);

}

return works;

}

bool WorkTime::IsSet() const

{

MYSQL\_RES\* res;

string sql;

DateTime\* dayStart = new DateTime();

DateTime\* dayEnd = new DateTime();

dayStart->Hour(0);

dayStart->Minute(0);

dayStart->Minute(0);

dayStart->DateToTimestamp();

dayEnd->Hour(23);

dayEnd->Minute(59);

dayEnd->Seconds(59);

dayEnd->DateToTimestamp();

sql = "SELECT `id` FROM `" + this->table + "` WHERE "

+ "`user` = " + NumberToStringConverter<int>::Convert(this->GetUser()->GetId()) + " "

+ "AND `day` > " + NumberToStringConverter<int>::Convert(dayStart->Timestamp()) + " "

+ "AND `day` < " + NumberToStringConverter<int>::Convert(dayEnd->Timestamp()) + " "

+ "LIMIT 0, 2";

res = this->connector->Query(sql, SELECT);

if(mysql\_num\_rows(res) != 0) {

return true;

}

else {

return false;

}

}

**main.cpp**

/\*

\* File: main.cpp

\* Author: alex

\*

\* Created on November 17, 2011, 12:15 PM

\*/

/\*\*

\* Console Presentation

\*/

#include <iostream>

#include <string>

#include <vector>

#include "../header/ERROR\_CODES.h"

#include "../header/SqlConnector.h"

#include "../header/Group.h"

#include "../header/BaseRecord.h"

#include "../header/DateTime.h"

#include "header/User.h"

#include "header/Status.h"

#include "header/Project.h"

#include "header/Task.h"

#include "header/WorkTime.h"

#include "header/Report.h"

#include "header/Wages.h"

#include "header/Menu.h"

using namespace std;

int main()

{

vector<BaseRecord \*> groups;

SqlConnector\* SqlConn;

try {

SqlConn = new SqlConnector("localhost", "keepertime", "KeeperTime", "KeeperTime");

}

catch(Exception\* e) {

cout << e->what() << endl;

delete SqlConn;

return e->Code();

}

////// Logic For MyProgram //////

Menu\* menu = new Menu(SqlConn);

delete menu;

////// End Logic MyProgram //////

cout << "Hi, its console!" << endl;

delete SqlConn;

return 0;

}

**KeeperTime-Makefile.mk**

# Environment

MKDIR=mkdir

CP=cp

CCADMIN=CCadmin

CXXFLAGS=-I/usr/include/mysql -I/usr/local/include/mysql++

# build

build: .build-post

.build-pre:

# Add your pre 'build' code here...

.build-post: .build-impl

# Add your post 'build' code here...

# clean

clean: .clean-post

.clean-pre:

# Add your pre 'clean' code here...

.clean-post: .clean-impl

# Add your post 'clean' code here...

# clobber

clobber: .clobber-post

.clobber-pre:

# Add your pre 'clobber' code here...

.clobber-post: .clobber-impl

# Add your post 'clobber' code here...

# all

all: .all-post

.all-pre:

# Add your pre 'all' code here...

.all-post: .all-impl

# Add your post 'all' code here...

# build tests

build-tests: .build-tests-post

.build-tests-pre:

# Add your pre 'build-tests' code here...

.build-tests-post: .build-tests-impl

# Add your post 'build-tests' code here...

# run tests

test: .test-post

.test-pre:

# Add your pre 'test' code here...

.test-post: .test-impl

# Add your post 'test' code here...

# help

help: .help-post

.help-pre:

# Add your pre 'help' code here...

.help-post: .help-impl

# Add your post 'help' code here...

# include project implementation makefile

include nbproject/Makefile-impl.mk

# include project make variables

include nbproject/Makefile-variables.mk