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 работы 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
3. Руководство для программиста 06
4. Руководство для пользователя 08
5. Заключение 15
6. Список используемой литературы 16
7. 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 разрабатывался 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.

**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;

}