# 课程设计文档

By 2014034243005 梁期智

## 数据库的分析与设计

数据库是整个系统的各个部分能否紧密地结合在一起以及如何结合的关键所在，因此数据库是系统的核心和基础。销售系统以数据库为核心，在数据库管理系统的支持下，进行信息的收集、整理、存储、检索、更新、加工、统计和传播等操作。数据库设计的好坏直接影响到整个系统的质量和效率。

数据库的设计一般经过需求分析、概念设计、逻辑设计、物理设计四个步骤。

### 数据库的需求分析

根据功能需求分析，要有一个用户表，用户具有id、姓名、密码、身份标识四个属性，用于登陆系统；

而整个销售管理系统的核心是商品，那么就产生的商品表，商品表拥有id、姓名、价格、产地、库存、所属仓库、所属分类、删除标识八个属性；

而商品与仓库、分类均为多对一的逻辑关系，那么就产生了仓库表和分类表，仓库表具有id、名称、删除标识、排序规则四个属性，分类表具有id、名称、删除标识三个属性；

在销售管理过程中，势必会生成订单，根据功能需求分析，首先要将商品入库，这样就生成了入库单，入库单具有id、订单号、经手人id、商品所属仓库id、商品所属分类id、入库数量、商品id、删除标识八个属性；

而在销售中，每有一笔生意，就会生成销售单，需要建立销售表，销售单具有id、订单号、经手人id、商品所属仓库id、商品所属分类id、销售数量、商品id、删除标识八个属性；

在商品销售以及库存调动中又会涉及到出库，这样就生成了出库单，出库单具有id、订单号、经手人id、商品所属仓库id、商品所属分类id、出库数量、商品id、删除标识八个属性，为了数据交互性能的提升，将入库单与出库单合并产生出入库表，他们各自多了一条属性，即出入库标识；

这便是整套系统的数据库需求。

### 数据库的逻辑结构设计

⑴用户表逻辑结构

表2 用户表逻辑结构表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性 | 数据类型 | 域范围 | 允许为空 | 主外键说明 | 默认值 |
| id | 字符型 | 最大32个字符 | 否 | 主键 | 无 |
| 用户名 | 字符型 | 最大20个字符 | 否 | 无 | 无 |
| 密码 | 字符型 | 最大20个字符 | 否 | 无 | 无 |
| 身份标识 | 整型 | 最大1个字符 | 是 | 无 | 0 |

⑵分类表逻辑结构

表3 分类表逻辑结构表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性 | 数据类型 | 域范围 | 允许为空 | 主外键说明 | 默认 |
| id | 字符型 | 最大32个字符 | 否 | 主键 | 无 |
| 分类名 | 字符型 | 最大20个字符 | 否 | 无 | 无 |
| 删除标识 | 整型 | 最大1个字符 | 是 | 无 | 0 |

⑶仓库表逻辑结构

表4 仓库表逻辑结构表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性 | 数据类型 | 域范围 | 允许为空 | 主外键说明 | 默认 |
| id | 字符型 | 最大32个字符 | 否 | 主键 | 无 |
| 仓库名 | 字符型 | 最大20个字符 | 否 | 无 | 无 |
| 删除标识 | 整型 | 最大1个字符 | 是 | 无 | 0 |
| 排序规则 | 整型 | 最大10个字符 | 是 | 无 | 0 |

⑷商品表逻辑结构

表5 商品表逻辑结构表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性 | 数据类型 | 域范围 | 允许为空 | 主外键说明 | 默认 |
| id | 字符型 | 最大32个字符 | 否 | 主键 | 无 |
| 商品名 | 字符型 | 最大20个字符 | 否 | 无 | 无 |
| 价格 | 精确值类型 | 最大10个字符，两位小数 | 否 | 无 | 无 |
| 产地 | 字符型 | 最大20个字符 | 否 | 无 | 无 |
| 库存 | 实型 | 无 | 否 | 无 | 无 |
| 所属仓库id | 字符型 | 最大32个字符 | 否 | 外键 | 无 |
| 所属分类id | 字符型 | 最大32个字符 | 否 | 外键 | 无 |
| 删除标识 | 整型 | 最大1个字符 | 是 | 无 | 0 |

⑸销售订单表逻辑结构

表6 销售订单表逻辑结构表

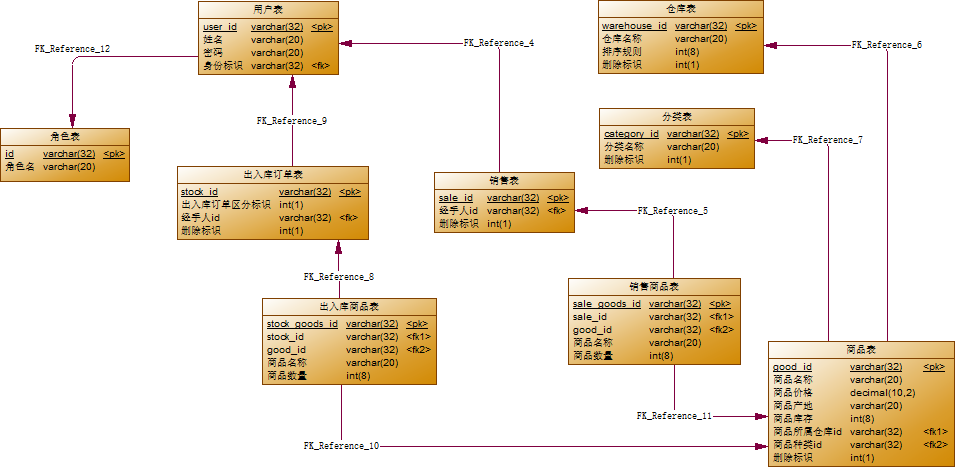
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性 | 数据类型 | 域范围 | 允许为空 | 主外键说明 | 默认 |
| id | 字符型 | 最大32个字符 | 否 | 主键 | 无 |
| 订单号 | 字符型 | 最大20个字符 | 否 | 无 | 无 |
| 经手人id | 实型 | 最大32个字符 | 否 | 外键 | 无 |
| 所属仓库id | 字符型 | 最大32个字符 | 否 | 外键 | 无 |
| 所属分类id | 字符型 | 最大32个字符 | 否 | 外键 | 无 |
| 销售数量 | 实型 | 无 | 否 | 无 | 无 |
| 商品id | 字符型 | 最大32个字符 | 否 | 外键 | 无 |
| 删除标识 | 整型 | 最大1个字符 | 是 | 无 | 0 |

⑹出入库订单表逻辑结构

表7 出入库订单表逻辑结构表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性 | 数据类型 | 域范围 | 允许为空 | 主外键说明 | 默认 |
| id | 字符型 | 最大32个字符 | 否 | 主键 | 无 |
| 订单号 | 字符型 | 最大20个字符 | 否 | 无 | 无 |
| 经手人id | 实型 | 最大32个字符 | 否 | 外键 | 无 |
| 所属仓库id | 字符型 | 最大32个字符 | 否 | 外键 | 无 |
| 所属分类id | 字符型 | 最大32个字符 | 否 | 外键 | 无 |
| 销售数量 | 实型 | 无 | 否 | 无 | 无 |
| 商品id | 字符型 | 最大32个字符 | 否 | 外键 | 无 |
| 出入库标识 | 整形 | 最大1个字符 | 否 | 无 | 无 |
| 删除标识 | 整型 | 最大1个字符 | 是 | 无 | 0 |

### 数据库的物理结构设计



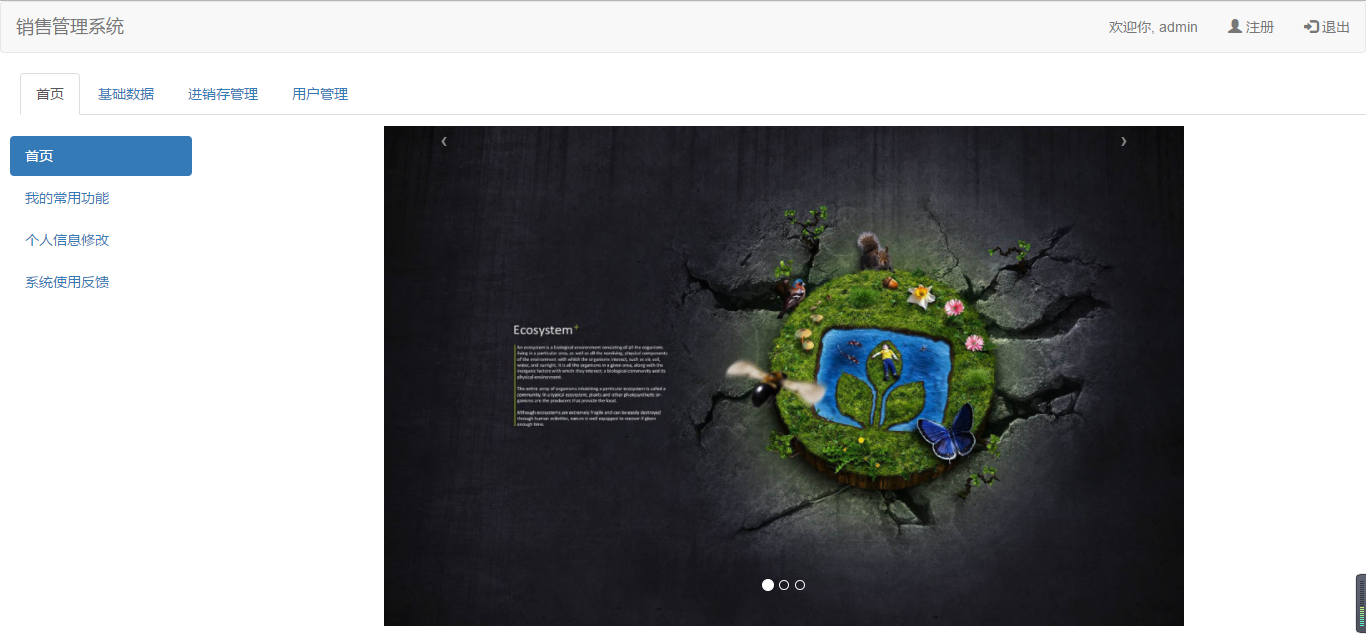
## 系统实现

### 系统界面布局设计

登陆界面：



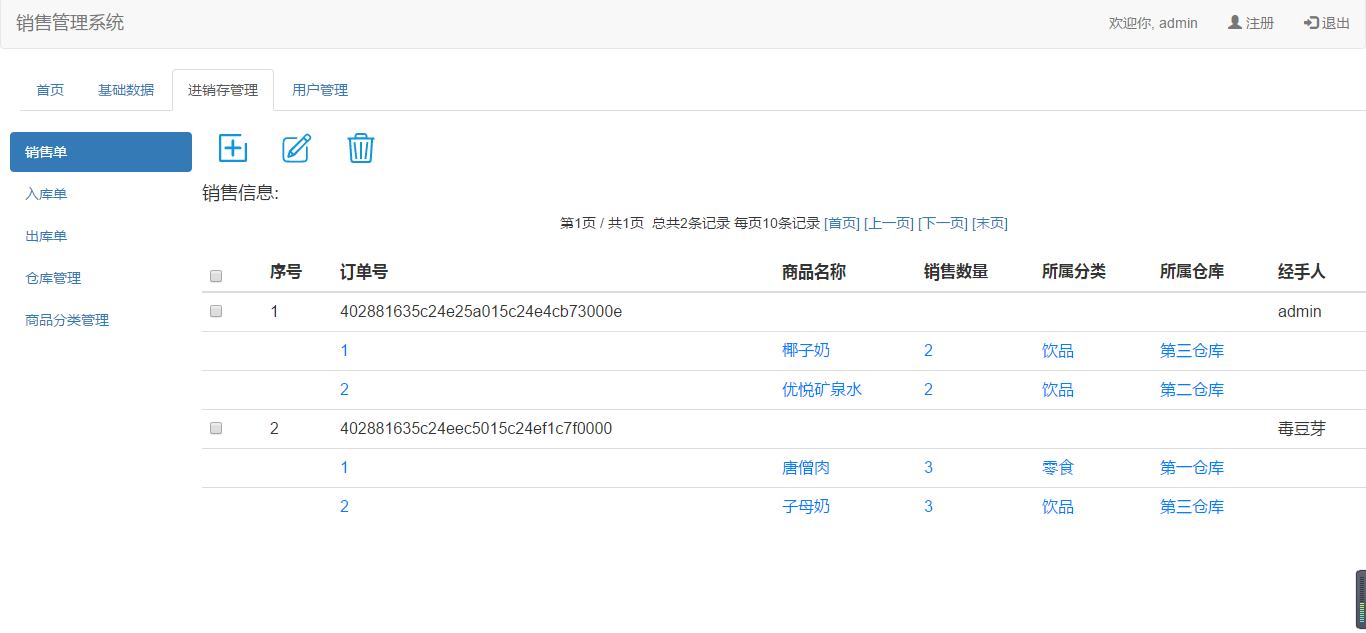
首页：



商品页面：



销售、库存、仓库、分类管理页面：

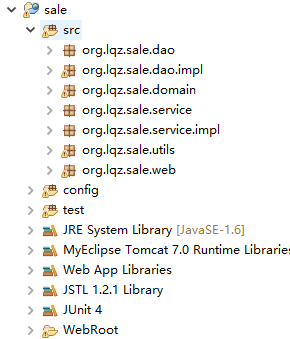


用户、权限管理页面：



### 代码实现：

**如下为系统工程结构图：**



**代码分析：**

**BaseDao**

|  |
| --- |
| package org.lqz.sale.dao;  import java.io.Serializable;  import java.util.Collection;  import java.util.List;  import org.lqz.sale.utils.Page;  public interface BaseDao {  // 查询所有，带条件查询  public <T> List<T> find(String hql, Class<T> entityClass, Object[] params);  // 获取一条记录  public <T> T get(Class<T> entityClass, Serializable id);  // 分页查询，将数据封装到一个page分页工具类对象  public <T> Page<T> findPage(String hql, Page<T> page, Class<T> entityClass, Object[] params);  // 新增和修改保存  public <T> void saveOrUpdate(T entity);  // 批量新增和修改保存  public <T> void saveOrUpdateAll(Collection<T> entitys);  // 单条删除，按id  public <T> void deleteById(Class<T> entityClass, Serializable id);  // 批量删除  public <T> void delete(Class<T> entityClass, Serializable[] ids);  } |

**BaseDaoImpl**

|  |
| --- |
| **package org.lqz.sale.dao.impl;**  **import java.io.Serializable;**  **import java.util.Collection;**  **import java.util.List;**  **import org.hibernate.Query;**  **import org.hibernate.Session;**  **import org.hibernate.SessionFactory;**  **import org.lqz.sale.dao.BaseDao;**  **import org.lqz.sale.utils.Page;**  **public class BaseDaoImpl implements BaseDao {**    **private SessionFactory sessionFactory;**  **public void setSessionFactory(SessionFactory sessionFactory) {**  **this.sessionFactory = sessionFactory;**  **}**  **public Session getSession() {**  **return sessionFactory.getCurrentSession();**  **}**  **// 带条件查询**  **public <T> List<T> find(String hql, Class<T> entityClass, Object[] params) {**  **Query query = this.getSession().createQuery(hql);**  **if (params != null) {**  **for (int i = 0; i < params.length; i++) {**  **query.setParameter(i, params[i]);**  **}**  **}**  **return (List<T>) query.list();**  **}**  **// 获取一条，根据主键id**  **public <T> T get(Class<T> entityClass, Serializable id) {**  **return (T) this.getSession().get(entityClass, id);**  **}**  **// 分页查询，查询两次，一次查询总数，一次查询分页记录**  **public <T> Page<T> findPage(String hql, Page<T> page, Class<T> entityClass, Object[] params) {**  **Query query = this.getSession().createQuery(hql);**  **if (params != null) {**  **for (int i = 0; i < params.length; i++) {**  **query.setParameter(i, params[i]);**  **}**  **}**  **// 查询一次，获取记录总数**  **int count = query.list().size(); // 可以优化？？？？ "select count(\*) " +hql**  **page.setTotalRecord(count);**  **// 设置分页**  **query.setFirstResult((page.getPageNo() - 1) \* page.getPageSize()); // 设置开始位置**  **query.setMaxResults(page.getPageSize()); // 设置获取几条**  **page.setResults((List<T>) query.list());**  **return page;**  **}**  **// 新增和修改，hibernate根据id是否为null自动判断**  **public <T> void saveOrUpdate(T entity) {**  **this.getSession().saveOrUpdate(entity);**  **}**  **// 集合保存，这时新增还是修改，就自动判断，调用时是否简洁。适合批量新增和修改时。（Mrecord控件）**  **public <T> void saveOrUpdateAll(Collection<T> entitys) {**  **for (T entity : entitys) {**  **this.saveOrUpdate(entity);// 为什么hibernate批量操作时，要用循环一条一条记录去更新？**  **}**  **}**  **// 按主键id删除**  **public <T> void deleteById(Class<T> entityClass, Serializable id) {**  **this.getSession().delete(get(entityClass, id));**  **}**  **// 批量删除**  **public <T> void delete(Class<T> entityClass, Serializable[] ids) {**  **for (Serializable s : ids) {**  **deleteById(entityClass, s);**  **}**  **}**  **}** |

**Page（工具类）**

|  |
| --- |
| **package org.lqz.sale.utils;**  **import java.util.HashMap;**  **import java.util.List;**  **import java.util.Map;**  **/\*\***  **\* 分页辅助类：对分页的基本数据进行一个简单的封装**  **\* 用来传递分页参数和查询参数params**  **\*/**  **public class Page<T> {**  **private int pageNo = 1; //页码，默认是第一页**  **private int pageSize = SysConstant.PAGE\_SIZE; //每页显示的记录数，默认是10**  **private int totalRecord; //总记录数**  **private int totalPage; //总页数**  **private List<T> results; //对应的当前页记录**  **private Map<String, Object> params = new HashMap<String, Object>(); //其他的参数我们把它分装成一个Map对象**    **public int getPageNo() {**  **return pageNo;**  **}**    **public void setPageNo(int pageNo) {**  **this.pageNo = pageNo;**  **}**    **public int getPageSize() {**  **return pageSize;**  **}**    **public void setPageSize(int pageSize) {**  **this.pageSize = pageSize;**  **}**    **public int getTotalRecord() {**  **return totalRecord;**  **}**    **public void setTotalRecord(int totalRecord) {**  **this.totalRecord = totalRecord;**  **//在设置总页数的时候计算出对应的总页数，在下面的三目运算中加法拥有更高的优先级，所以最后可以不加括号。**  **int totalPage = totalRecord%pageSize==0 ? totalRecord/pageSize : totalRecord/pageSize + 1;**  **this.setTotalPage(totalPage);**  **}**    **public int getTotalPage() {**  **return totalPage;**  **}**    **public void setTotalPage(int totalPage) {**  **this.totalPage = totalPage;**  **}**    **public List<T> getResults() {**  **return results;**  **}**    **public void setResults(List<T> results) {**  **this.results = results;**  **}**    **public Map<String, Object> getParams() {**  **return params;**  **}**    **public void setParams(Map<String, Object> params) {**  **this.params = params;**  **}**    **public String toString() {**  **StringBuilder builder = new StringBuilder();**  **builder.append("Page [pageNo=").append(pageNo).append(", pageSize=").append(pageSize).append(", results=").append(results).append(", totalPage=").append(totalPage).append(", totalRecord=").append(totalRecord).append("]");**  **return builder.toString();**  **}**    **/\* 页面链接 \*/**  **public String url; //分页按钮中的转向链接**  **public void setUrl(String url) {**  **this.url = url;**  **}**  **public String links;**  **public String getLinks() {**  **StringBuffer sBuf = new StringBuffer();**  **int curPageNo = this.pageNo; //当前页**      **sBuf.append("<span class=\"noprint\" style=\"padding:5px;\">");**    **//利用js动态设置分页页码**  **sBuf.append("<script language=\"javascript\">");**  **sBuf.append(" function setPageNo( value ){");**  **sBuf.append(" document.getElementById(\"page.pageNo\").value = value;");**  **// sBuf.append(" alert(document.getElementById(\"page.pageNo\").value);");**  **sBuf.append(" }");**  **sBuf.append("</script>");**  **sBuf.append("<input type=\"hidden\" id=\"page.pageNo\" name=\"page.pageNo\" value=\"").append(curPageNo).append("\">"); //分页参数：当前页隐藏域**    **sBuf.append("&nbsp;第").append(curPageNo).append("页 / 共").append(this.totalPage).append("页&nbsp;");**  **sBuf.append("&nbsp;总共").append(this.totalRecord).append("条记录 每页").append(this.pageSize).append("条记录&nbsp;");**    **sBuf.append("<a href=\"#").append("\" onclick=\"setPageNo(1);formSubmit('").append(url).append("','\_self')");**  **sBuf.append("\">[首页]");**  **sBuf.append("</a>&nbsp;");**    **if(this.pageNo<=1){**  **curPageNo = 1;**  **}else{**  **curPageNo = this.pageNo - 1;**  **}**  **sBuf.append("<a href=\"#").append("\" onclick=\"setPageNo(").append(curPageNo).append(");formSubmit('").append(url).append("','\_self')");**  **sBuf.append("\">[上一页]");**  **sBuf.append("</a>&nbsp;");**      **if(this.pageNo>=this.totalPage){**  **curPageNo = this.totalPage;**  **}else{**  **curPageNo = this.pageNo + 1;**  **}**  **sBuf.append("<a href=\"#").append("\" onclick=\"setPageNo(").append(curPageNo).append(");formSubmit('").append(url).append("','\_self')");**  **sBuf.append("\">[下一页]");**  **sBuf.append("</a>&nbsp;");**    **sBuf.append("<a href=\"#").append("\" onclick=\"setPageNo(").append(this.totalPage).append(");formSubmit('").append(url).append("','\_self')");**  **sBuf.append("\">[末页]");**  **sBuf.append("</a>&nbsp;");**    **sBuf.append("</span>");**    **return sBuf.toString();**  **}**    **}** |

**其他代码看源工程吧，以上只贴一些重要代码。**