

07-数据设计

数据持久化

为什么持久化

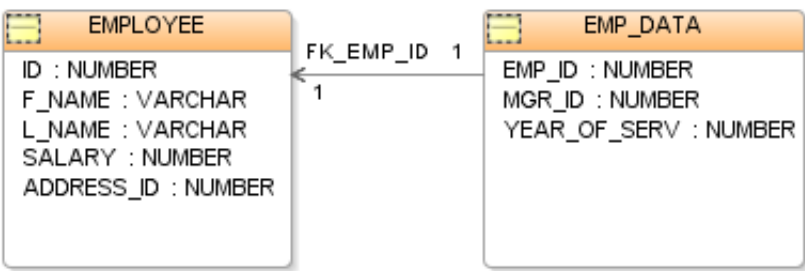
知识：数据被记录（record）、存储（storage）、回忆（recall）。

持久化历史

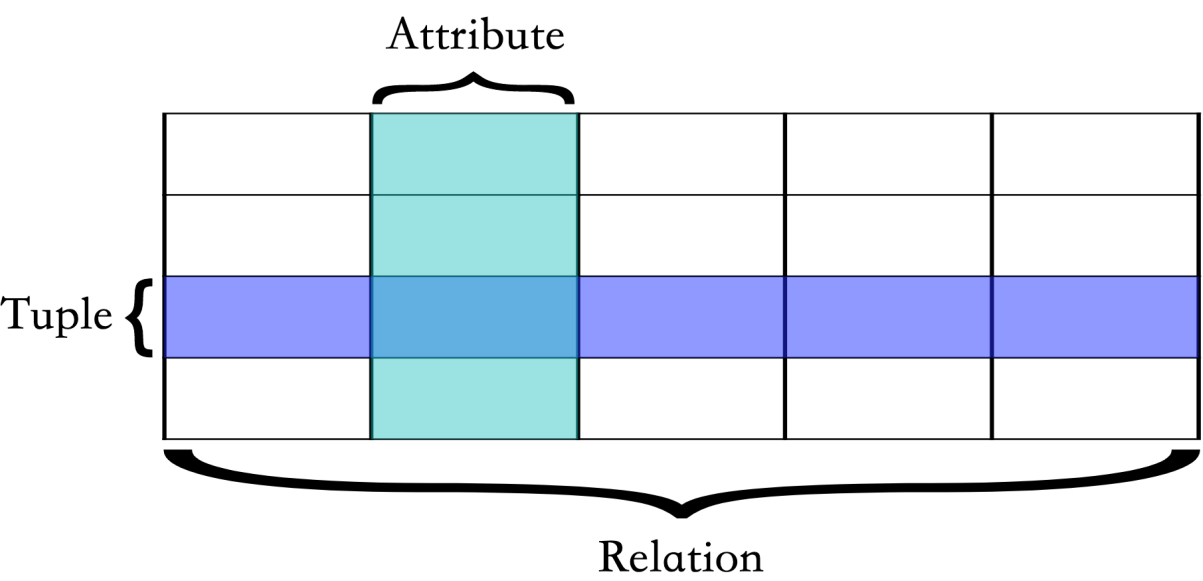
- 实物记录（绳结、泥板、竹片、绢布、纸张）
- 电子文件
- 数据库系统

关系型数据库

Relation in Concept Model

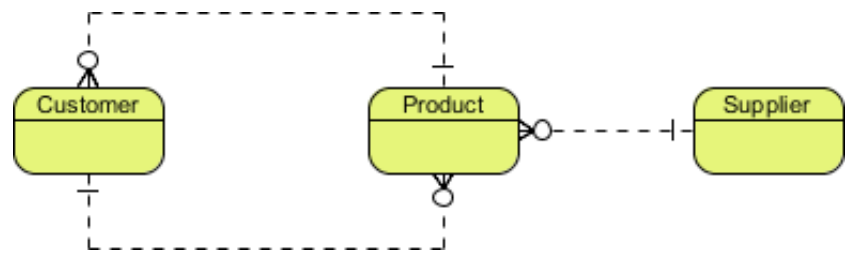


Relation in database

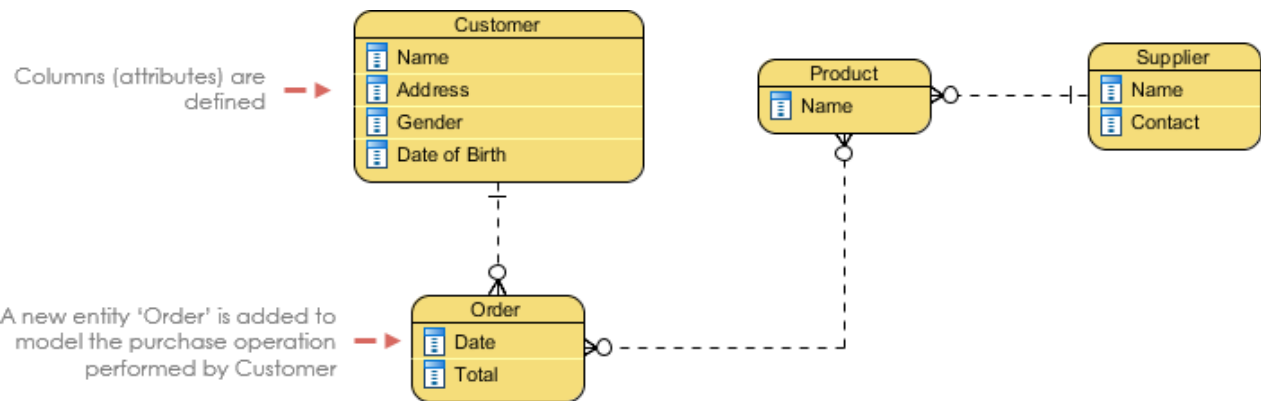


Concept model->Logic model ->Physical model

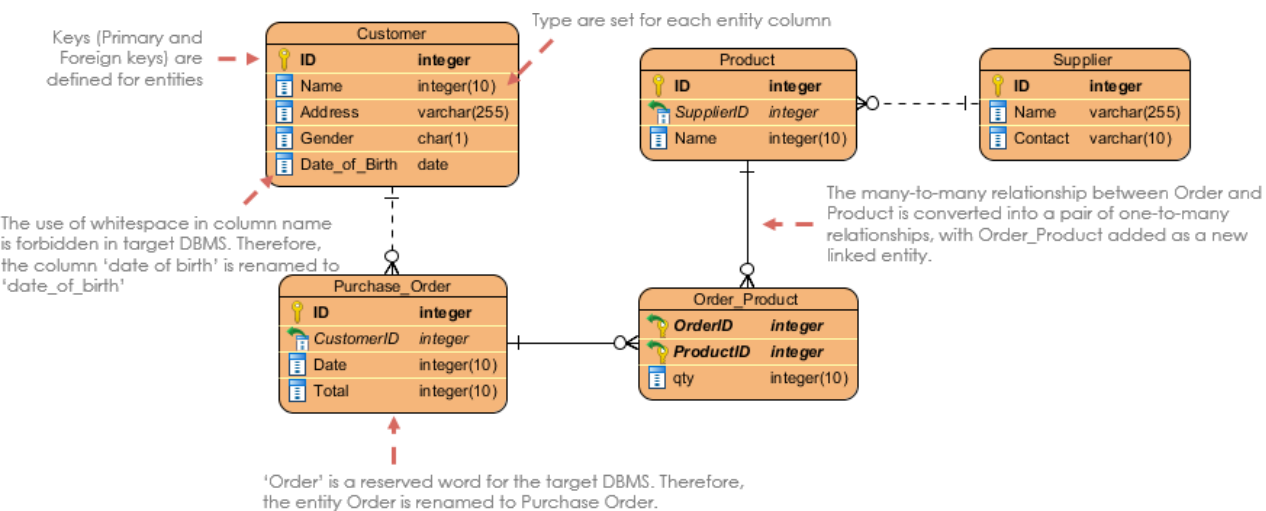
Concept model



Logic model



Physical model



索引

Why 索引

在数据库系统的使用过程当中，数据的查询是使用最频繁的一种数据操作。

最基本的查询算法当然是顺序查找（linear search），遍历表然后逐行匹配行值是否等于待查找的关键词，其时间复杂度为 $O(n)$ 。但时间复杂度为 $O(n)$ 的算法规模小的表，负载轻的数据库，也能有好的性能。但是数据增大的时候，时间复杂度为 $O(n)$ 的算法显然是糟糕的，性能就很快下降了。

好在计算机科学的发展提供了很多更优秀的查找算法，例如二分查找（binary search）、二叉树查找（binary tree search）等。如果稍微分析一下会发现，每种查找算法都只能应用于特定的数据结构之上，例如二分查找要求被检索数据有序，而二叉树查找只能应用于二叉查找树上，但是数据本身的结构不可能完全满足各种数据结构（例如，理论上不可能同时将两列都按顺序进行组织），所以，在数

据之外，数据库系统还维护着满足特定查找算法的数据结构，这些数据结构以某种方式引用（指向）数据，这样就可以在这些数据结构上实现高级查找算法。这种数据结构，就是索引。

索引是对数据库表中一个或多个列的值进行排序的结构。与在表中搜索所有的行相比，索引用指针指向存储在表中指定列的数据值，然后根据指定的次序排列这些指针，有助于更快地获取信息。通常情况下，只有当经常查询索引列中的数据时，才需要在表上创建索引。索引将占用磁盘空间，并且影响数据更新的速度。但是在多数情况下，索引所带来的数据检索速度优势大大超过它的不足之处。

Why B+ Tree

- 1.文件很大，不可能全部存储在内存中，故要存储到磁盘上
- 2.索引的结构组织要尽量减少查找过程中磁盘I/O的存取次数（为什么使用B-/B+Tree，还跟磁盘存取原理有关。）
- 3.局部性原理与磁盘预读，预读的长度一般为页（page）的整倍数，（在许多操作系统中，页得大小通常为4k）
- 4.数据库系统巧妙利用了磁盘预读原理，将一个节点的大小设为等于一个页，这样每个节点只需要一次I/O就可以完全载入，（由于节点中有两个数组，所以地址连续）。而红黑树这种结构，**h**明显要深的多。由于逻辑上很近的节点（父子）物理上可能很远，无法利用局部性

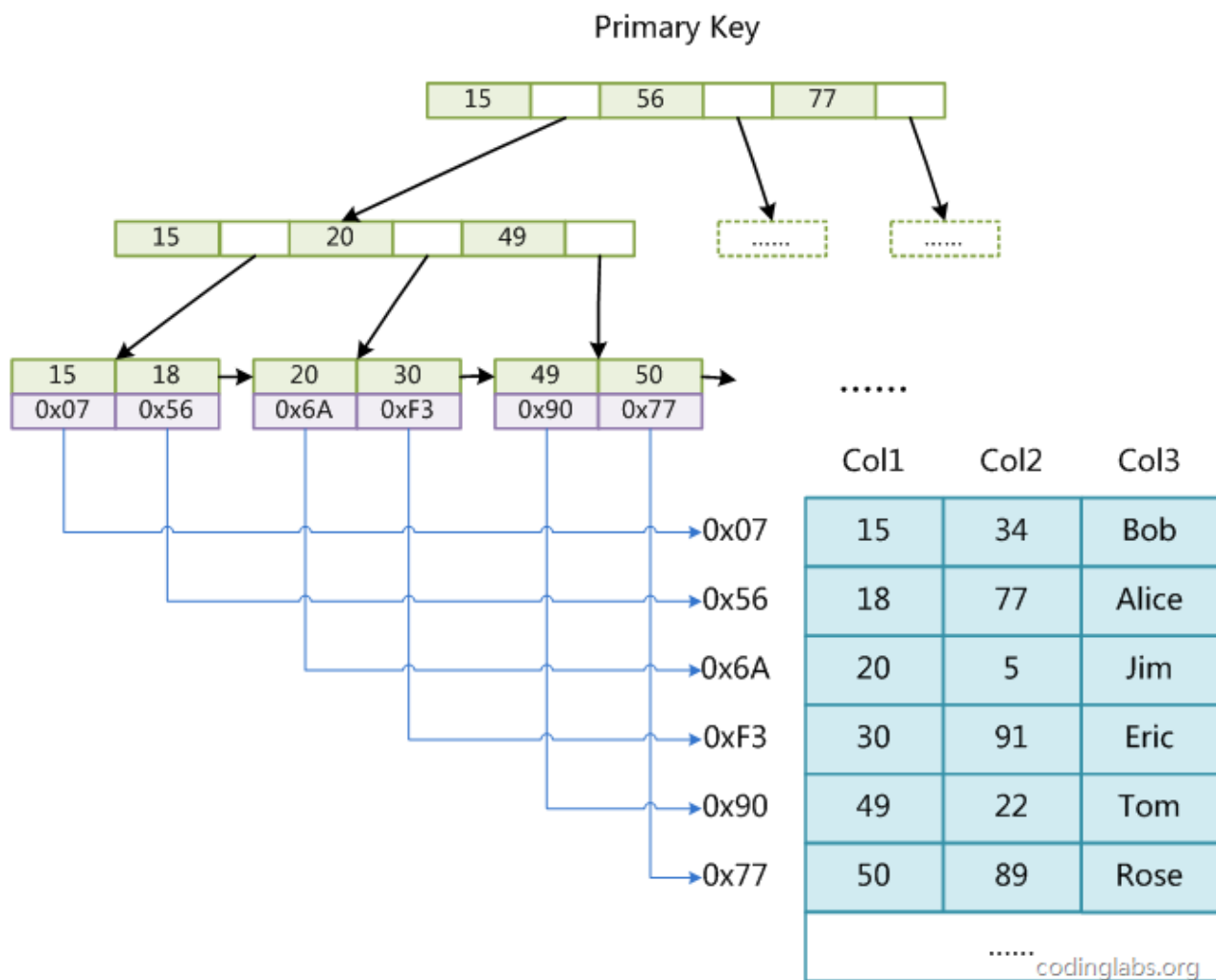
索引实现

InnoDB索引和MyISAM索引的区别：

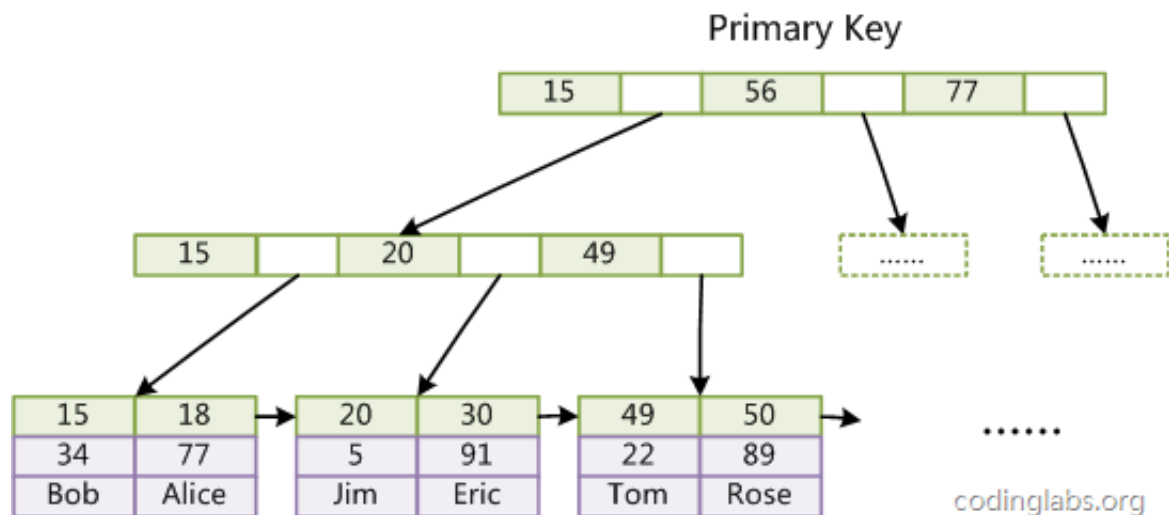
一是主索引的区别：InnoDB的数据文件本身就是索引文件。而MyISAM的索引和数据是分开的。

二是辅助索引的区别：InnoDB的辅助索引data域存储相应记录主键的值而不是地址。而MyISAM的辅助索引和主索引没有多大区别。

MyISAM



innoDB

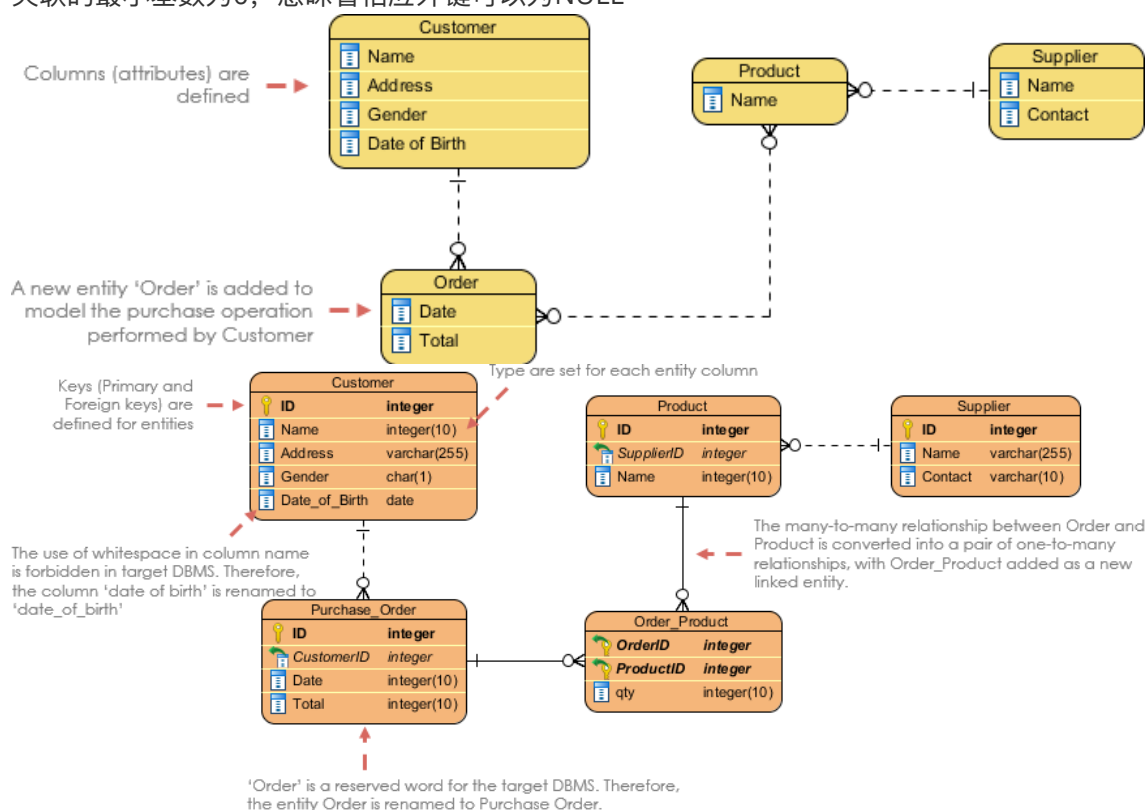


将类图映射为关系表

ORM(Object-Relation Mapping)

- 通常一个类/对象映射为一张表
- 类的属性映射为表的列名
- 类的实例对象就是表的行

- 要为每个转换后的表建立主键
 - 类/对象通过引用来唯一标识自己
 - 关系/表通过主键类唯一标识自己
- 处理关联
 - 类/对象通过链接实现关联
 - 关系/表通过主键/外键对实现关联
 - 1:1关联：其中一个表的主键，作为另一个表的外键；两端等价
 - 1:N关联：将1端表的主键，作为N端表的外键
 - M:N关联：建立中间表，将M的主键和N的主键都作为中间表的外键
 - 在包含/聚合关联中，将整体的主键放在部分中作为外键
 - 关联的最小基数为0，意味着相应外键可以为NULL



Mysql

安装

MySQL Community Downloads

MySQL Community Server

General Availability (GA) Releases Archives ⓘ

MySQL Community Server 8.0.19

Select Operating System:
macOS

Looking for previous GA versions?

! Packages for Catalina (10.15) are compatible with Mojave (10.14)

macOS 10.15 (x86, 64-bit), DMG Archive (mysql-8.0.19-macos10.15-x86_64.dmg)	8.0.19	376.8M	Download
MD5: c2c592e87c8ac07d643e50ff9a7a1ebc Signature			

启动

MySQL 搜索

Instances Configuration

ACTIVE INSTANCE
● MySQL 8.0.13

INSTALLED INSTANCES
● MySQL 8.0.13

DATA DIRECTORIES


8.0.13
/usr/local/mysql-8.0.13-macos10.14-x86_64

Start MySQL Server

☒ Start MySQL when your computer starts up

Initialize Database

Uninstall



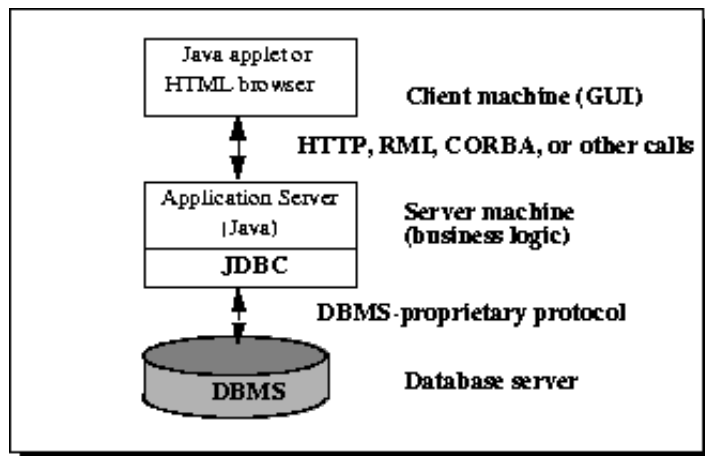
用初始密码登录

```
mysql -u root -p
```

修改初始密码

```
ALTER USER 'root'@'localhost' IDENTIFIED BY '[your new password]';
```

JDBC



主要步骤

- 登记并加载JDBC驱动器。
- 建立与SQL数据库的连接；
- 传送一个SQL操作；
- 获得数据结果；

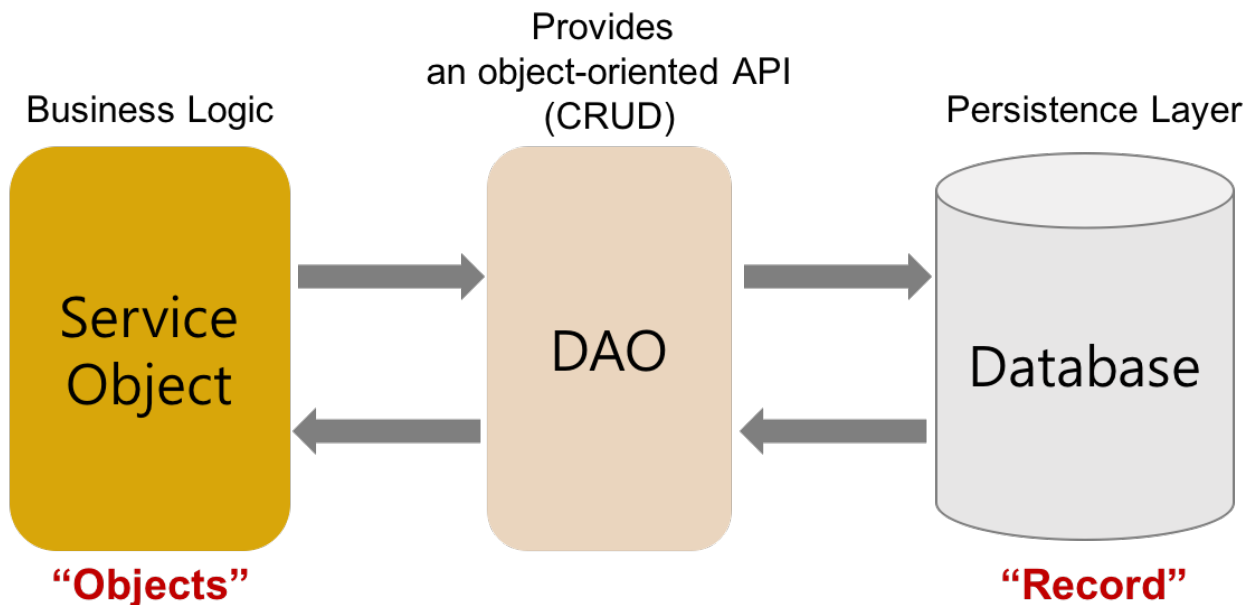
API

```
//1.加载驱动程序
Class.forName("com.mysql.jdbc.Driver");
//2. 获得数据库连接
Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
//3.操作数据库，实现增删改查
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT user_name, age FROM imooc_goddess");
//如果有数据，rs.next()返回true
while(rs.next()){
    System.out.println(rs.getString("user_name")+" 年龄: "+rs.getInt("age"));
}
```

DAO

DAO模式

DAO (Data Access objects 数据存取对象)是指位于业务逻辑和持久化数据之间实现对持久化数据的访问。通俗来讲，就是将数据库操作都封装起来。



对外提供相应的接口

在面向对象设计过程中，有一些"套路"用于解决特定问题称为模式。DAO 模式提供了访问关系型数据库系统所需操作的接口，将数据访问和业务逻辑分离对上层提供面向对象的数据访问接口。

从以上 DAO 模式使用可以看出，DAO 模式的优势就在于它实现了两次隔离。

1、隔离了数据访问代码和业务逻辑代码。业务逻辑代码直接调用DAO方法即可，完全感觉不到数据库表的存在。分工明确，数据访问层代码变化不影响业务逻辑代码,这符合单一职能原则，降低了藕合性，提高了可复用性。2、隔离了不同数据库实现。采用面向接口编程，如果底层数据库变化，如由 MySQL 变成 Oracle 只要增加 DAO 接口的新实现类即可，原有 MySQL 实现不用修改。这符合 "开-闭" 原则。该原则降低了代码的藕合性，提高了代码扩展性和系统的可移植性。

组成部分

1、DAO接口：把对数据库的所有操作定义成抽象方法，可以提供多种实现。2、DAO 实现类：针对不同数据库给出DAO接口定义方法的具体实现。3、实体类：用于存放与传输对象数据。4、数据库连接和关闭工具类：避免了数据库连接和关闭代码的重复使用，方便修改。

样例代码

DAO 接口:

```
public interface PetDao {
    /**
     * 查询所有宠物
     */
    List<Pet> findAllPets() throws Exception;
}
```

DAO 实现类:


```

public class PetDaoImpl extends BaseDao implements PetDao {
    /**
     * 查询所有宠物
     */
    public List<Pet> findAllPets() throws Exception {
        Connection conn=BaseDao.getConnection();
        String sql="select * from pet";
        PreparedStatement stmt= conn.prepareStatement(sql);
        ResultSet rs= stmt.executeQuery();
        List<Pet> petList=new ArrayList<Pet>();
        while(rs.next()) {
            Pet pet=new Pet(
                rs.getInt("id"),
                rs.getInt("owner_id"),
                rs.getInt("store_id"),
                rs.getString("name"),
                rs.getString("type_name"),
                rs.getInt("health"),
                rs.getInt("love"),
                rs.getDate("birthday")
            );
            petList.add(pet);
        }
        BaseDao.closeAll(conn, stmt, rs);
        return petList;
    }
}

```

宠物实体类(里面get/set方法就不列出了)

```

public class Pet {
    private Integer id;
    private Integer ownerId;    //主人ID
    private Integer storeId;    //商店ID
    private String name;       //姓名
    private String typeName;    //类型
    private int health;        //健康值
    private int love;          //爱心值
    private Date birthday;     //生日
}

```

连接数据库

```

public class BaseDao {
    private static String driver="com.mysql.jdbc.Driver";
    private static String url="jdbc:mysql://127.0.0.1:3306/epet";
    private static String user="root";
    private static String password="root";
}

```

```

        static {
            try {
                Class.forName(driver);
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }
        }

        public static Connection getConnection() throws SQLException {
            return DriverManager.getConnection(url, user, password);
        }

        public static void closeAll(Connection conn, Statement stmt, ResultSet rs)
        throws SQLException {
            if(rs!=null) {
                rs.close();
            }
            if(stmt!=null) {
                stmt.close();
            }
            if(conn!=null) {
                conn.close();
            }
        }

        public int executeSQL(String preparedSql, Object[] param) throws
        ClassNotFoundException {
            Connection conn = null;
            PreparedStatement pstmt = null;
            /* 处理SQL,执行SQL */
            try {
                conn = getConnection(); // 得到数据库连接
                pstmt = conn.prepareStatement(preparedSql); // 得到
                PreparedStatement对象
                if (param != null) {
                    for (int i = 0; i < param.length; i++) {
                        pstmt.setObject(i + 1, param[i]); // 为预编译sql设置参数
                    }
                }
                ResultSet num = pstmt.executeQuery(); // 执行SQL语句
            } catch (SQLException e) {
                e.printStackTrace(); // 处理SQLException异常
            } finally {
                try {
                    BaseDao.closeAll(conn, pstmt, null);
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }

```

```

    }
    return 0;
}

}

```

Mybatis

MyBatis 是支持定制化 SQL、存储过程以及高级映射的优秀持久层框架。MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。MyBatis 可以对配置和原生 Map 使用简单的 XML 或注解，将接口和 Java 的 POJOs (Plain Old Java Objects, 普通的 Java 对象) 映射成数据库中的记录。

Mybatis 样例代码

```

@Mapper
@Repository
public interface AdminMapper {

    int addManager(User user);

    List<User> getAllManagers();
}

```

```

public class User { //省略getter、setter
    private Integer id;
    private String email;
    private String password;
    private String userName;
    private String phoneNumber;
    private double credit;
    private UserType userType;
}

```

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.hotel.data.admin.AdminMapper">
    <insert id="addManager" parameterType="com.example.hotel.po.User"
useGeneratedKeys="true" keyProperty="id">
        insert into User(email,password,usertype)
        values(#{email},#{password},#{userType})
    </insert>
    <select id="getAllManagers" resultMap="User">
        select * from User where usertype='HotelManager'
    </select>

```

```
<resultMap id="User" type="com.example.hotel.po.User">
  <id column="id" property="id"></id>
  <result column="email" property="email"></result>
  <result column="password" property="password"></result>
  <result column="username" property="userName"></result>
  <result column="phonenumber" property="phoneNumber"></result>
  <result column="credit" property="credit"></result>
  <result column="usertype" property="userType"></result>
</resultMap>
</mapper>
```

原文链接

<https://zh.wikipedia.org/wiki/%E5%85%B3%E7%B3%BB%E6%95%B0%E6%8D%AE%E5%BA%93>

<https://www.cnblogs.com/xyxxs/p/4440187.html>

<https://www.visual-paradigm.com/cn/guide/data-modeling/what-is-entity-relationship-diagram/>

<https://www.runoob.com/note/27029> <https://www.runoob.com/w3cnote/jdbc-use-guide.html>