

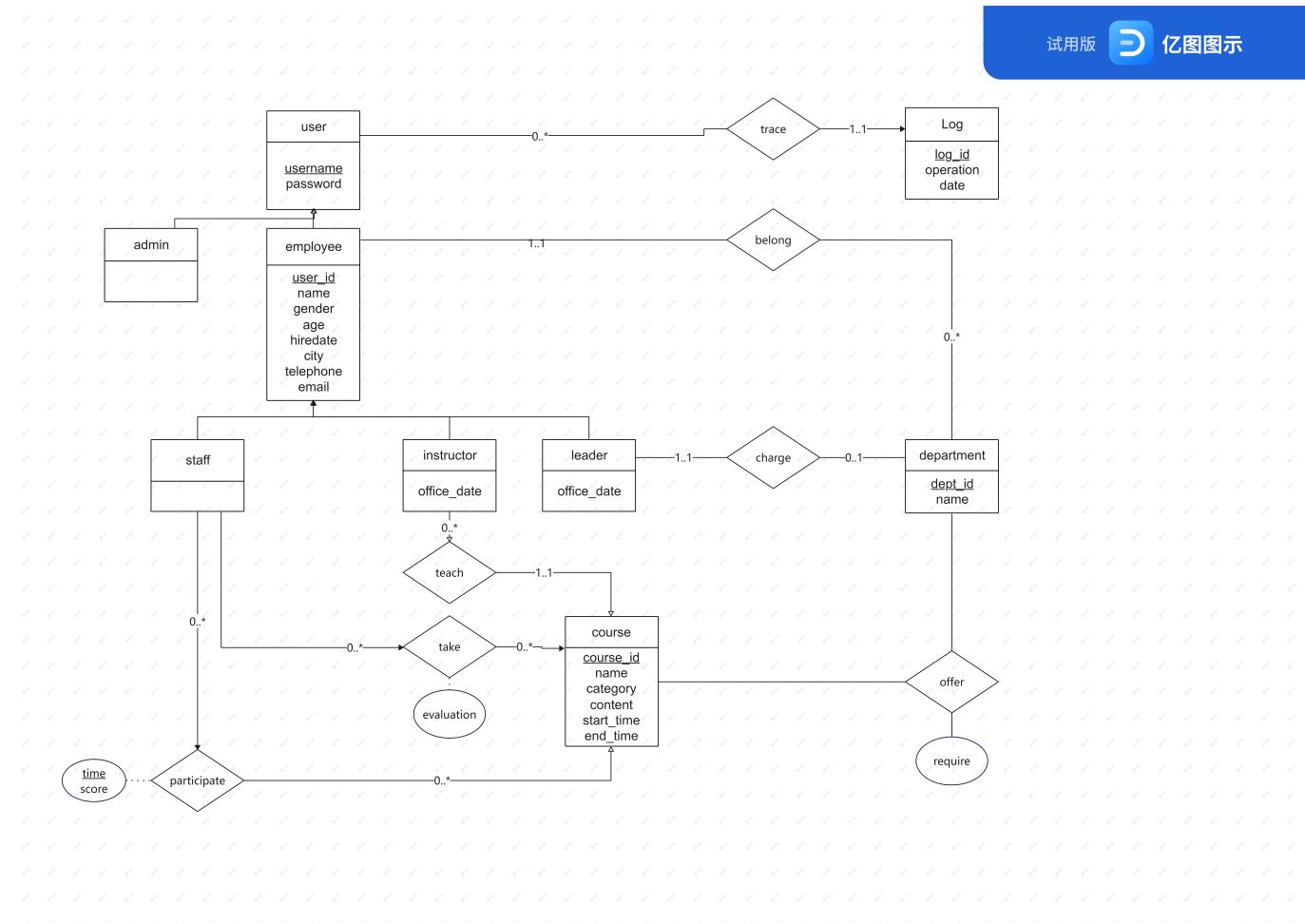
数据库设计PJ

员工培训管理系统

18307110072 赵书誉

19302010020 袁逸聪

E-R模型设计



E-R图转换为关系模式

实体集

- 部门信息 **department** (dept_id, name)
- 用户信息 **user** (username, password)
- 管理员 **admin** (username)
- 职员信息 **employee** (user_id, username, name, gender, age, hire_date, city, telephone, email, dept_name)
- 普通员工 **staff** (user_id)

- 教员信息 instructor (user_id, office_date)
- 主管信息 leader (user_id, office_date, dept_name)
- 课程信息 course (course_id, instructor_id, name, category, content, start_time, end_time)
- 日志信息 log (log_id, username, operation, date)

关系集

- 普通员工 & 课程：培训 take (user_id, course_id, evaluation)
- 普通员工 & 课程：考试 participate (user_id, course_id, time, score)
- 部门 & 课程：要求 offer(dept_id, course_id, require)

数据库表结构说明

- department：存储和管理部门信息的数据库表，主键是部门编号 dept_id，部门名称也应该是unique 的。

```
CREATE TABLE department
(
    dept_id INT          NOT NULL AUTO_INCREMENT,
    name    VARCHAR(20) NOT NULL UNIQUE,
    PRIMARY KEY (dept_id)
);
```

- user：存储和管理用户账号的数据库表，主键是用户名 username。泛化成管理员和职员。

```
CREATE TABLE user
(
    username VARCHAR(20) NOT NULL,
    password VARCHAR(40) NOT NULL,
    PRIMARY KEY (username)
);
```

- admin：存储和管理管理员信息，主键是引用的外键——用户名 username。

```
CREATE TABLE admin
(
    username VARCHAR(20) NOT NULL,
    PRIMARY KEY (username),
    FOREIGN KEY (username) REFERENCES user (username) ON DELETE CASCADE ON UPDATE
    CASCADE
);
```

- employee：用于存储和管理职员信息的数据库表。信息包括姓名、性别、年龄、入职日期、城市、联系方式、邮箱，主键是员工号 user_id。可以泛化成普通员工、教员和主管。由于员工属于部门，和部门是

多对一的关系，因此引用了外键——部门名称 dept_name。每个员工对应一个用户账号，引用外键——用户名 username，一对—，所以是unique。

```
CREATE TABLE employee
(
    user_id CHAR(11) NOT NULL CHECK (LENGTH(user_id) = 11),
    username VARCHAR(20) NOT NULL UNIQUE,
    name VARCHAR(20) NOT NULL,
    gender VARCHAR(10) NOT NULL CHECK (gender IN ("男", "女")),
    age INTEGER NOT NULL CHECK (age >= 0),
    hire_date DATE NOT NULL,
    city VARCHAR(255) NOT NULL,
    telephone CHAR(11) NOT NULL CHECK (LENGTH(telephone) = 11),
    email VARCHAR(255) NOT NULL,
    dept_name VARCHAR(20),
    PRIMARY KEY (user_id),
    FOREIGN KEY (username) REFERENCES user (username) ON DELETE SET NULL ON UPDATE CASCADE,
    FOREIGN KEY (dept_name) REFERENCES department (name) ON DELETE SET NULL ON UPDATE CASCADE
);
```

- staff: 存储和管理普通职员信息的数据库表。主键是引用的外键——职员的员工号 user_id。

```
CREATE TABLE staff
(
    user_id CHAR(11) NOT NULL,
    FOREIGN KEY (user_id) REFERENCES employee (user_id) ON DELETE CASCADE ON UPDATE CASCADE,
    PRIMARY KEY (user_id)
);
```

- instructor: 存储和管理教员信息的数据库表。信息还包含任职日期，主键是引用的外键——职员的员工号 user_id。

```
CREATE TABLE instructor
(
    user_id CHAR(11) NOT NULL,
    office_date DATE NOT NULL,
    FOREIGN KEY (user_id) REFERENCES employee (user_id) ON DELETE CASCADE ON UPDATE CASCADE,
    PRIMARY KEY (user_id)
);
```

- leader: 存储和管理主管信息的数据库表。信息还包含任职日期与管理的部门，主键是引用的外键——职员的员工号 user_id。其次还引用了外键——部门名称 dept_name，因为是一对一的关系所以为

unique。

```
CREATE TABLE leader
(
    user_id      CHAR(11) NOT NULL,
    office_date  DATE      NOT NULL,
    dept_name    VARCHAR(20) UNIQUE,
    FOREIGN KEY (user_id) REFERENCES employee (user_id) ON DELETE CASCADE ON
UPDATE CASCADE,
    FOREIGN KEY (dept_name) REFERENCES department (name) ON DELETE CASCADE ON
UPDATE CASCADE,
    PRIMARY KEY (user_id)
);
```

- course：存储和管理课程信息的数据库表。包含了课程名称，课程内容，课程类型，开始时间，结束时间和教员员工号。主键是课程号 course_id，引用了外键——教员员工号，因为一门课程只能由一个教员教授，是多对一的关系，相关外键的删除和更新都是级联的。

```
CREATE TABLE course
(
    course_id    CHAR(5)      NOT NULL,
    name         VARCHAR(20)   NOT NULL,
    content       VARCHAR(255)  NOT NULL,
    category     VARCHAR(20)   NOT NULL,
    start_time   DATE        NOT NULL,
    end_time     DATE        NOT NULL,
    instructor_id CHAR(11)    NOT NULL,
    PRIMARY KEY (course_id),
    FOREIGN KEY (instructor_id) REFERENCES instructor (user_id) ON DELETE CASCADE
ON UPDATE CASCADE
);
```

- log：存储与管理日志信息的数据库表，主键为log_id。包含了操作，时间，用户名，相关外键的删除和更新都是级联的。

```
CREATE TABLE log
(
    log_id      INT          NOT NULL AUTO_INCREMENT,
    username    VARCHAR(20),
    operation   VARCHAR(255) NOT NULL,
    date        TIMESTAMP    NOT NULL,
    PRIMARY KEY (log_id),
    FOREIGN KEY (username) REFERENCES user (username) ON DELETE SET NULL ON UPDATE
CASCADE
);
```

- take: 普通员工培训情况对应的数据库表。主键是user_id和course_id。相关外键的删除和更新都是级联的。特别的含有属性是 evaluation 表示该课状态 (NULL值说明未结课)。

```
CREATE TABLE take
(
    user_id      CHAR(11) NOT NULL,
    course_id    CHAR(5)  NOT NULL,
    evaluation   VARCHAR(20) CHECK (evaluation IS NULL OR evaluation IN ("通过", "未通过")),
    PRIMARY KEY (user_id, course_id),
    FOREIGN KEY (user_id) REFERENCES staff (user_id) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (course_id) REFERENCES course (course_id) ON DELETE CASCADE ON UPDATE CASCADE
);
```

- participate: 普通员工考试情况对应的数据库表。主键是user_id和course_id和录入时间 time。相关外键的删除和更新都是级联的。特别的含有属性是 score 表示考试成绩。

```
CREATE TABLE participate
(
    user_id      CHAR(11) NOT NULL,
    course_id    CHAR(5)  NOT NULL,
    time         TIMESTAMP NOT NULL,
    score        INT       NOT NULL CHECK (score >= 0 AND score <= 100),
    PRIMARY KEY (user_id, course_id, time),
    FOREIGN KEY (user_id, course_id) REFERENCES take (user_id, course_id) ON DELETE CASCADE ON UPDATE CASCADE
);
```

- offer: 部门对课程要求对应的数据库表。主键是dept_id 和 course_id。相关外键的删除和更新都是级联的。特别的含有属性是 need 表示必修还是选修。

```
CREATE TABLE offer
(
    dept_id      INT       NOT NULL,
    course_id    CHAR(5)  NOT NULL,
    need         VARCHAR(10) NOT NULL CHECK (need IN ("必修", "选修"))
),
    PRIMARY KEY (dept_id, course_id),
    FOREIGN KEY (dept_id) REFERENCES department (dept_id) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (course_id) REFERENCES course (course_id) ON DELETE CASCADE ON UPDATE CASCADE
);
```

索引定义说明

表均采用主键索引

函数依赖与范式分析

- 所有关系的属性的域都是原子的，不包含任何组合属性，所以上述关系模式都属于第一范式。
- 没有任何非主属性部分依赖于键，所以也符合第二范式。
- 没有任何非主属性传递依赖于键，所以符合第三范式。
- 没有任何属性传递依赖于键，所以符合BC范式。

使用指南

程序入口为flask/app.py,运行后在本机5000端口即可访问Web前端

Flask与工程

工程使用Python Web轻量框架Flask实现

设计上分为UI层-Service层-Dao层

1. UI层

- 由Jinja2引擎渲染html页面,能方便地处理Python所传参数
- 由于时间关系,部分功能的回显未能以Web形式做完,只能退而求其次显示Json文本结果

2. Service层

- 面向需求地定义了HTTP入口,并按照角色划分模块
- Service层的主要功能是路由+业务
- route.py承担所有的页面渲染出口

3. Dao层

- 相比于概念意义上的Dao,本项目的Dao中夹杂了部分业务成分

数据库与分层的思考

在项目实现过程中,模块划分给我们带来较大困扰,即"数据库到底只是提供数据读写的接口,还是承担部分业务逻辑"

本着学了就多用用的初心,我们能用sql语句就用sql语句,并且尝试了一些复杂的嵌套sql语句

然而,这样的开发方式一方面让人不太适应,另一方面也很难完善错误处理,因为数据库会告诉我们哪一步出错了,出错的语句大概是什么样的,我们却很难预料这些错误的具体内容并提供解释与回显

简而言之,复杂的sql虽然功能强大(相比也比拆成多句sql再用编程语言拼装要来得高效),但也让整个执行流程变得更不透明,更难以界入和界分

如果没有强有力的理由,可以想象,程序员们几乎一定会偷懒按照自己熟悉的理解方式,把数据库当成Array或Map使用

可惜在本PJ中,这样的理由不太能提现得出~