表结构说明

SQL框架

核心功能的SQL语句

Admin系统管理员

Instructor教员

Employee员工

Manager管理员

表结构说明

数据库内共分为9个表，如下所示：

1. department

```
department(
    dept_id   varchar(8)  not null,
    dept_name varchar(20) not null, -- 部门编号和名称不能为空
    primary key (dept_id)
)
```

   - ***department*** 简单地使用 `dept_id` 作为主键，用于标识一个部门

2. employee

```
employee(
    id              varchar(11)     not null,
    name            varchar(10)     not null, -- 员工的id和姓
名不能为空
    gender          varchar(1)      check ( gender in
(null, '男', '女') ),-- 性别可以不填，或者填入'男'、'女'
    age             int,
    date_employed   TIMESTAMP       default
current_timestamp, -- 默认认为该员工是今天才录取的
    city            varchar(4),
    phone_number    varchar(11),
```

```
    email           varchar(50),
    dept_id         varchar(8)      not null,
    primary key (id),
    index (name),
    index (dept_id),-- 为name和department_id建立索引
    foreign key (dept_id) references department (dept_id)
        on delete cascade
        on update cascade
)
```

- ***employee*** 使用了员工号即 `id` 唯一标识一个员工。dept_id用作外键，引用了department表的中的dept_id，保证员工属于一个存在的部门。该外键使用了**级联删除与更新**，即在一个员工未被移出其部门之前，部门的编号改变或部门被删除都将导致员工信息从*employee* 表中删除。（设想公司的大规模裁员中一个整个部门都被裁掉，这种情况是可能的）

  注意其中对 `name` 的非空约束限制了一个员工必须具有姓名，最长10位的限制符合对人名的直观认知。

  对于枚举属性 `gender` ，通过一个check子句将值限制在"男"、"女"或null中（这允许一个员工不在个人信息透露性别，隐私保护get√-）

3. manager

```
manager(
    employee_id     varchar(11)     not null,
    name            varchar(10)     not null, -- name其实是一个冗余属性，但可以便于获取一个部门主管的名字
    dept_id         varchar(8)      not null, -- 部门主管对应的部门id不能为空
    primary key (employee_id),
    unique (dept_id), -- 一个部门只能有一个部门主管，所以department_id需要唯一性约束
    foreign key (employee_id) references employee (id)
        on delete cascade -- 当employee信息变动，manager对应的信息也级联改动
        on update cascade,
    foreign key (dept_id) references department (dept_id)
      on delete cascade
      on update cascade
)
```

- ***manager*** 使用 `employee_id` 作为主键，并引用*employee* 表中的id，表示manager也是一个现存的员工。同*employee* 中的 `dept_id` 一样，*manager* 中也维护着其对应的 `dept_id` 。这两个外键均设置级联更新和删除。正如在*employee* 中提到的一样，可能发生manager的个人信息改变或部门信息的改变使得manager需要做出相应改变。

  manager 表中的 `dept_id` 添加了 `唯一性约束` ，这代表了一个部门只能有一个部门主管。在mysql中，添加唯一性约束的键，会自动设置一个索引，而不需再手动设置索引。可以预见到，这个索引将在manager通过 `dept_id` 找到本部门的所有员工或指定员工时，起到重要作用。并且，一个部门主管所在 `部门的编号很少变动` ，在dept_id上形成索引 `不会经受频繁的索引结构重构` 。

  注意在*manager* 中的 `name` 和 `department` 属性：这其实是一个违背设计范式的冗余，因为它们可以通过查询*employee* 表获得（使用employee_id的自然连接）。但是在此处，考虑到系统的可扩展性：一个系统用户（比如总裁）在将来可能需要获知所有部门主管的id和名字，在*manager* 表中维护一个 `name` 与 `department` 属性（与employee表中一致）显然更加直观。这将为将来的系统维护者带来更大的好处。

4. instructor

```
instructor(
    employee_id     varchar(11)     not null,
    name            varchar(10)     not null,
    date_registered TIMESTAMP   default current_timestamp,
    primary key (employee_id),
    foreign key (employee_id) references employee (id)
        on delete cascade
        on update cascade
)
```

- ***instructor*** 类似于manager，我们同样地增加了 `name` 这一冗余属性以增强直观性和可扩展性，以及对外键 `employee_id` 设置的级联属性。除此以外的 `data_registered` 属性代表教员的任教时间，是教员的重要特征之一。

5. course

```
course(
    course_id       varchar(7)      not null,
    title           varchar(20)     not null, -- 课程编号和名称不能为空
    type            varchar(10)     default '尚无分类', -- 课程类型
    content         varchar(50)     default '尚无描述',
    instructor_id   varchar(11)     not null, -- 一门课只有一个教员教授，以instructor_id作为course的属性即可
    primary key (course_id),
    foreign key (instructor_id) references instructor (employee_id)
        on delete cascade
        on update cascade
)
```

   - ***course*** 将课程编号作为主键，并设置了非空属性 `title` 和 `instructor_id`，分别代表课程名称和教员的员工号。

     对于course与教授该课程的instructor之间的关系，一个可能的想法是用一个关联表记录instructor与course之间的一对多关系。然而一个course只能有一个instructor的限制使得将 `instructor_id` 作为course的外键属性成为可能。这甚至可能更符合直觉：`一个instructor是一个course的重要特征之一`。

     类似上述表中所做的一样，我们也为 `instructor_id` 设置了级联删除与级联更新，这表示一个course的instructor信息不会因为instructor的员工号改变而丢失。

6. test_record

```
test_record( -- 单次测试的记录
    record_id       bigint          not null
 auto_increment, -- 测试记录可能会很多，使用bigint
    course_id       varchar(7)      not null,
    employee_id     varchar(11)     not null,
    score           int             not null    check (
score >= 0 and score <= 100 ), -- 当某次测试的score达到60需要更
新takes表中对应记录为已通过
    PRIMARY KEY (record_id),
    foreign key (course_id) references course (course_id)
        on delete cascade
        on update cascade,
    foreign key (employee_id) references employee (id)
        on delete cascade
        on update cascade
)
```

- ***test_record*** 记录了培训成绩。一个employee可能参加了多个course的多次测试，这将导致即使是对同一个employee和同一个其修读的课程，也可能不止有一个培训成绩。`(course_id, employee_id)`的二元组显然不能成为*test_record*的主键。因此我们采用了针对单次培训成绩的唯一性标识符`record_id`作为主键。course_id和employee_id作为外键，并都设置了级联属性。

  test_record可能很多

  `score`属性记录了一次培训的成绩数值，值在0~100之间。当录入一条test_record，其score不能为空。

7. log

```
log(
    log_id          bigint              auto_increment,
    employee_id     varchar(10)     default 'admin', -- 操作
者的id，若不是员工操作，默认值为admin操作
    content         varchar(100)     not null,
    time            TIMESTAMP       default
current_timestamp, -- 日志时间戳
    primary key (log_id)
)
```

- ***log*** 为日志信息表，记录了系统的操作日志。使用自增`bigint`属性`log_id`作为

主键，因为日志信息随着系统使用，数量可能会增长到非常大。

8. takes

```
takes(
    employee_id     varchar(11)     not null,
    course_id       varchar(7)      not null,
    completed       int             default 0   check (
completed in (0, 1)),-- 结课状态，默认为未结课。
    -- 结课状态针对的是(employee_id, course_id)即一个员工是否学习
完了该课程，而不是该课程是否已经结束并注销
    -- 直到管理员将此课程从course表中移除，代表该课程已注销
    is_passed       int             check ( is_passed in
 (null, 0, 1)), -- 培训的通过状态
    primary key (employee_id, course_id),
    foreign key (employee_id) references employee (id)
        on delete cascade
        on update cascade,
    foreign key (course_id) references course(course_id)
        on delete cascade
        on update cascade
)
```

   - ***takes*** 记录了选课情况。考虑到 `(employee_id, course_id)` 的二元组足以唯一标识一个选课情况，选择它作为*takes* 的主键。另外的属性包括：`completed` 表示结课状态（0为未结课，1为已结课，默认为0）；is_passed表示通过状态（0表示未通过，1表示已通过）

9. course_to_dept

```
course_to_dept( -- 课程与部门的关联表
    course_id       varchar(7)      not null,
    dept_id         varchar(8)      not null,
    required        varchar(2)      not null    check (
required in ('必修', '选修')),
    primary key (course_id, dept_id),
    foreign key (course_id) references course (course_id)
        on delete cascade
        on update cascade,
    foreign key (dept_id) references department (dept_id)
        on delete cascade
        on update cascade
)
```

- ***course_to_dept*** 记录了一个course与一个department的关联关系，表示该课程对该部门可见。显然对于这样一个关联表，可以使用(course_id, dept_id)的二元组作为主键。对于外键引用，同样使用级联属性保证课程与部门的关联不会因课程或部门的编号变动而丢失。

  `required` 属性表示该课程对该部门是否必修。

没有使用 `trigger` 功能，因为trigger除了支持性不好之外还有一个重要的问题：难以在合作中约定。对于一个会同时影响其他表的的命令到底是应该认为trigger会同时完成其实什么都不需要额外做，还是需要后续处理，这在约定里容易造成不必要的麻烦和误解。因此我们决定不用trigger功能，而是在业务层做一些额外处理，这样处理的灵活性也比较高。

SQL框架

项目使用了MyBatis框架管理DAO层与数据库的交互，可以避免将SQL语句的编写与系统的功能逻辑串接在一起。并且通过传参形式，可以轻松优雅地实现 `动态SQL` 语句。

核心功能的SQL语句

Admin系统管理员

系统管理员的操作主要增删改查，这部分的SQL语句直接对响应表进行操作。由于我们在外键上都设置了级联属性，这些操作不会破坏引用完整性。

```
<insert id="addEmployee">
        insert into employee(id, name, dept_id) values (#{id}, #
{name}, #{did});
</insert>
...
```

需要注意的是系统管理员进行日志记录时，使用的是AdminMapper Interface的addLog方法，该方法缺省地将日志记录者设置为`'admin'`，用于标记这是一个admin日志。

Instructor教员

Instructor需要的操作主要是获取学员、关联课程与部门和录入成绩。

```
<insert id="addTestRecord">
        insert into test_record(course_id, employee_id, score)
                        values (#{cid}, #{id}, #{score});
</insert>
<insert id="associateCourse">
         insert into course_to_dept values (#{cid}, #{did}, #
{required})
</insert>
...
```

主要需要注意的是录入成绩时，先确保结课状态为已结课，然后录入test_record，最后根据成绩决定是否将通过状态更新为已通过。

这一切需要通过事务的`all-or-nothing`特性保证，否则会出现数据的不一致。因此需要用`commit()`和`rollback()`保证事务的完整提交与失败后的回滚，并在配置文件中设置事务不自动提交。

Employee员工

Employee需要的操作包括查看和更新自己的信息，查看已选课程与培训成绩记录。

```
<select id="getEmployeeById" >
    select * from employee where id = #{id};
</select>
<update id="update">
     update employee set name = #{new_name} where id = #{id};
</update>
<select id="getCourse" resultType="Course">
       select course.*
       from course, takes
       where course.course_id = takes.course_id
         and takes.employee_id = #{id};
</select>
...
```

Manager管理员

Manager的操作比较多，包括：1、获得部门中的所有员工 2、获取本部门可见的所有课程 3、分配给指定员工指定课程 4、查看指定员工的培训成绩记录 5、将指定员工转至指定部门 6、查询已通过的课程与对应员工 7、查看所有员工必修课的修读情况 8、查看未通过某门课程指定次数以上的员工 9、查看可以转部门的员工（即该员工在当前部门所选的课程全部通过） 10、查看即将转入的部门的必修课

```
<select id="getManagerByUid" resultType="Manager">
      select * from manager where employee_id = #{id}
</select>
<select id="getEmployeeByDeptId">
      select * from employee where dept_id = #{dept_id}
</select>
<select id="hsaTaken" resultType="int">
      select count(*) from takes
      where employee_id = #{id} and course_id = #{course_id}
</select>
<select id="getCoursesByDeptId">
      select course.*
      from course, course_to_dept
```

```
      where course.course_id = course_to_dept.course_id;
</select>
...
```