HW2 Wanzhu Chen 702040054

Problem 4:

Steps:

1. Define relations for each entity set

   Sales_office(**office_no**, location)
   Property(**p_id,** address, city, state zip_code)
   Owner(**o_id**, o_name)
   Employee(**e_id,** e_name)

2. Define relations for each relationship set

   Manage**(e_id** references employee(e_id),
   office_no references sales_office(office_no))
   Work(**e-id** references employee(e_id),
   office_no references sales_office(office_no))
   List (**p_id** references property(p_id),
   office_no references sales_offce(office_no))
   Own(**o_id** references owner(o_id),
   p_id references sales_office(office_no),
   perentage)

3. Modify key to capture mapping constraints

   Based on my ER diagram, I would change the relations of each relationship as follows:
   //manage is a one-to-one relationship, so I let office_no as the primary key while let e_id as unique.
   Manage**(office_no** references sales_office(office_no)
   (UNIQUE) e_id references employee(e_id))

   //work is a many to one relationship, so I let e_id from the many side as the primary key.
   Work(**e-id** PRIMARY KEY references employee(e_id),
   office_no references sales_office(office_no))

   //list is a many to one relationship, so I let p_id from the many side as the primary key.
   List (**p_id** references property(p_id),
   office_no references sales_offce(office_no))

   //own is many to many relationship, so no charge on it.
   Own(**o_id** references owner(o_id),
   p_id references sales_office(office_no),
   perentage)

4. Eliminate Redundant relations

Since there are no pairs of relations that have the same primary key and other attributes, I will not do elimination at this point.

5. Merge relations to capture participation constraints.

   Sales_office(**office_no**, location)
   Manage**((UNIQUE)** e_id references employee(e_id),
   office_no references sales_office(office_no))

   Merge Sales_office with Manage, since they have the same primary key and they have a strong relationship. Then set the foreign keys as not null. So I have:

   Sales_office(**office_no** primary key,
   e_id_**UNIQUE NOT NULL** references employee(e_id),
   location)

Result:

Thus, I have:

(1) Employee(**e_id** primary key, e_name)
(2) Property(**p_id** primary key, address, city, state zip_code)
(3) Owner(**o_id** primary key, o_name)
(4) Sales_office(**office_no** primary key ,
e_id_**UNIQUE NOT NULL** references employee(e_id),
location)
(5) Work(**employee_id** PRIMARY KEY, references employee(e_id),
office_no references sales_office(office_no))
//since table work has the same primary key e_id with table employee, I change the name from e_id to employee_id.
(6) List (**property_id** primary key, references property(p_id),
office_no references sales_offce(office_no))
//since table list has the same primary key p_id with table property, I change the name from p_id to property_id.

(7) Own(**o_id** references owner(o_id),
p_id references sales_office(office_no),
perentage,
primary key(o_id, p_id))

SQL in details:

```
CREATE TABLE `employee` (
        `e_id` varchar(8) NOT NULL,
        `e_name` varchar(20),
        PRIMARY KEY (`e_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `sales_office`(
        `office_no` numeric(6,0) NOT NULL,
        `e_id` varchar(8) NOT NULL,
        `location` varchar(16),
        PRIMARY KEY (`office_no`),
```

```
        UNIQUE KEY `e_id` (`e_id`),
       CONSTRAINT `sales_office_ibfk_1` FOREIGN KEY (`e_id`) REFERENCES
`employee` (`e_id`)
) ENGINE=InnoDB;

CREATE TABLE `property`(
        `p_id` varchar(8) NOT NULL,
        `address` varchar(50),
        `city` varchar(30),
        `state` varchar(30),
        `zip_code` int(11),
        PRIMARY KEY (`p_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;


CREATE TABLE `owner` (
        `o_id` varchar(8) NOT NULL,
        `o_name` varchar(20),
        PRIMARY KEY (`o_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `work` (
        `employee_id` varchar(8) NOT NULL,
        `office_no` numeric(6,0) DEFAULT NULL,
        PRIMARY KEY (`employee_id`),
       CONSTRAINT `work_ibfk_1` FOREIGN KEY (`employee_id`) REFERENCES
`employee` (`e_id`),
        CONSTRAINT `work_ibfk_2` FOREIGN KEY (`office_no`) REFERENCES
`sales_office` (`office_no`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;



CREATE TABLE `list` (
        `property_id` varchar(8) NOT NULL,
        `office_no` numeric(6,0) DEFAULT NULL,
        PRIMARY KEY (`property_id`),
        KEY `office_no` (`office_no`),
        CONSTRAINT `list_ibfk_1` FOREIGN KEY (`property_id`) REFERENCES
`property` (`p_id`),
        CONSTRAINT `list_ibfk_2` FOREIGN KEY (`office_no`) REFERENCES
`sales_office` (`office_no`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `own` (
        `percentage` decimal(4,2),
```

```
	`o_id` varchar(8) NOT NULL,
	`p_id` varchar(8) NOT NULL,
	PRIMARY KEY (`p_id`,`o_id`),
	CONSTRAINT `own_ibfk_1` FOREIGN KEY (`o_id`) REFERENCES `owner` (`o_id`),
	CONSTRAINT `own_ibfk_2` FOREIGN KEY (`p_id`) REFERENCES `property` (`p_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Problem5:

(a)

1. sales_office: office_no→location
2. employee: e_id→e_name
3. property: p_id→ address, city, state, zip_code
4. property: p_id, address→address, city, state, zip_code
5. owner: o_id→o_name
6. work relationship: e_id→office_no
7. list relationship: p_id→office_no
8. manage relationship: office_no→e_id

(b)

1. employee: e_no, e_name→e_no, e_name, date_of_birth
2. employee: e_no→e_name, data_of_birth
3. skill: s_no→ description
4. project: p_no→est_cost
5. city: c_name→state, population
6. department: d_name→phone
7. vendors: v_name→address
8. give relationship: e_no→title_name

(c)

1. submit relationship: o_id→c_id
2. serve relationship: e_id→ s_id
3. group relationship: p_id→pl_id
4. supply relationship: r_id, v_id→ unity_price
5. assemble relatioship: p_id, r_id→ a_quantity
6. supervise relationship: e_id → e_id
7. request relationship: o_id, p_id→pl_id
8. request relationship: p_id, pl_id, o_id→r_quantity

Problem6:

Part1: From relational algebrea expression to sql queries:

(1) SELECT title FROM course WHERE credits>=3 AND dept_name=”MATH”;
What are the titles of courses that are offered by Math department and have greater than or equal to 3 credits?

(2) SELECT id, name FROM takes NATURAL JOIN student WHERE course_id=”CISC437”;
What are the ids and names of students who have taken CISC437?

(3) SELECT b.id, b.name FROM (SELECT * FROM takes WHERE course_id=”CISC437”) a NATURAL JOIN (SELECT student.id, student.name FROM student) b;
What are the ids and names of all students who have taken CISC437?

(4) SELECT AVG(capacity) FROM (SELECT dept_name, capacity FROM classroom NATURAL JOIN section NATURAL JOIN course) a GROUP BY dept_name;
For each department, find the average capacity of all courses in all sections offered by that department.

Part2: Write relational algebrea expression and its sql queries:

(5)select title from course, teaches, instructor
where instructor.name="WU"
and course.course_id=teaches.course_id
and teaches.id=instructor.id;

$$\pi_{title}(\sigma_{name="WU"}(course \bowtie_{course.course\_id=teaches.course\_id} teaches \bowtie_{teaches.id=instructor.id} instructor))$$

(6)select distinct name from student, takes, course
where dept_name="Math"
and student.id=takes.id
and takes.course_id=course.course_id;
(Suppose that Math course means the course is from Math department.)

$$\pi_{name}\sigma_{dept\_name="Math"}(student \bowtie_{student.id==takes.id} takes \bowtie_{takes.course\_id==course.course\_id} course)$$

(7) select id, name from instructor left outer join
(select distinct id from teaches join instructor
where teaches.id=instructor.id
and ((year<2009) or (year=2009 and (semester=”summer” or semester=”spring”))) x
where x.id is not null;

$$\pi_{id,name}\left(instructor \bowtie_{id} \left(\sigma_{(year<2009)\ or\ (year=2009\ and\ (semester="summer"\ or\ semester="Spring"))}(teaches \bowtie_{teaches.id=instructor.id} instructor)\right)\right)$$

(8) select dept_name,avg(salary) from instructor group by dept_name;

$$dept\_name\ \mathcal{G}_{average(salary)}(instructor)$$

(9) select (capacity-count(takes.id)) from takes natural join section natural join classroom
where course_id="CISC637"
and semester="Spring"
and year=2015
group by course_id, semester, year;

$$(course\_id,semester,year)\ \mathcal{G}_{(capacity-count(ID))}\left(\sigma_{course\_id="CISC637"\ and\ semester="Spring"\ and\ year=2015}(takes \bowtie section \bowtie classroom)\right)$$