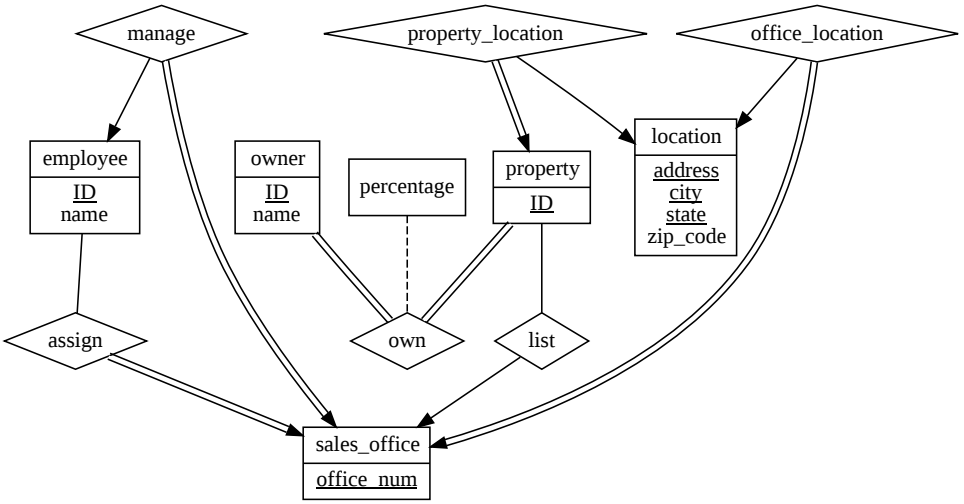
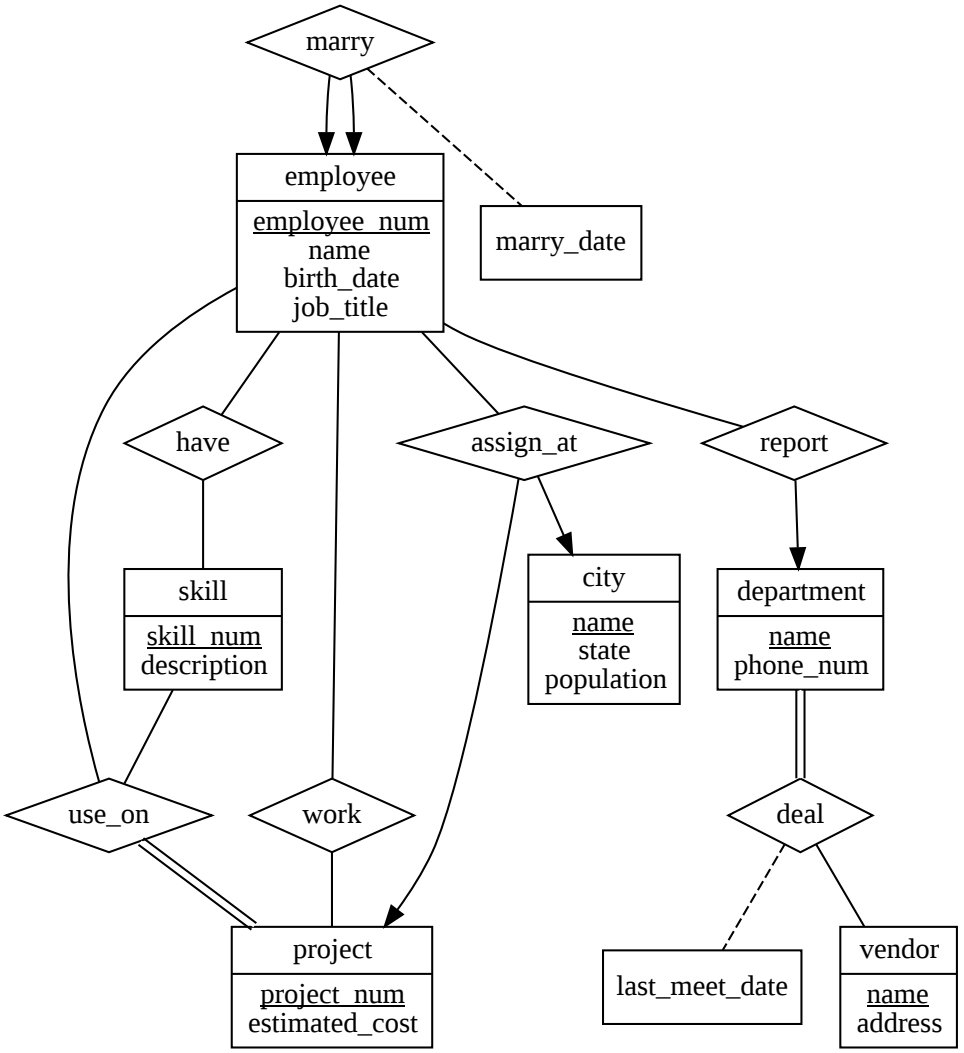
_ Name: Wei Zhong, UDID: 702044972, E-mail: zhongwei@udel.edu

Homework 2

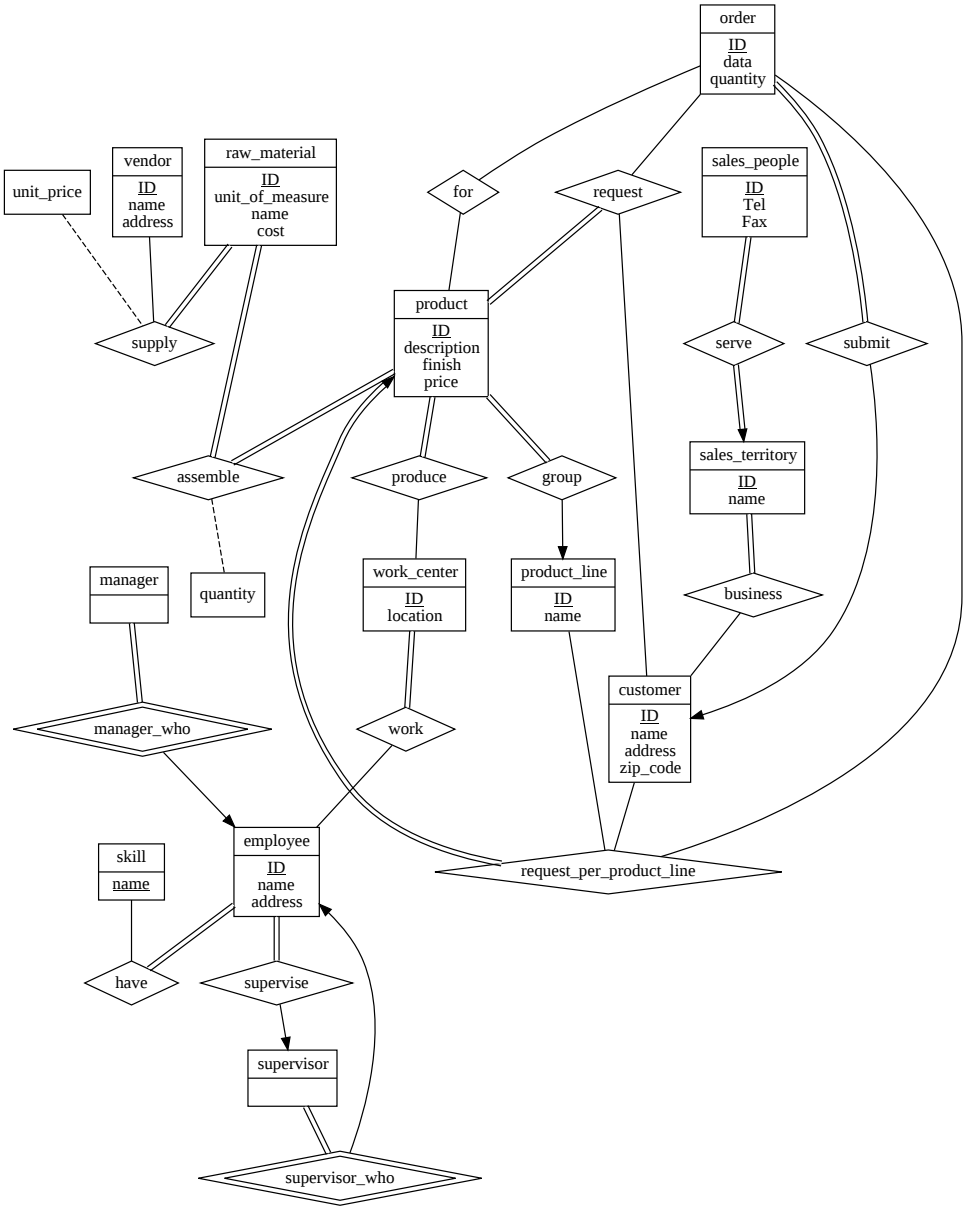
E-R diagrams (as references here)



Problem 1



Problem 2



Problem 3
(Assume "order line" is a typo, and a given customer order must request at least one product and only one product per product line)

Problem 4

1. Define relations for each entity set:

employee(ID, name)
owner(ID, name)
sales_office(office_num)
property(ID)
location(address, city, state, zip_code)

2. Define relations for each relationship set (all pointers are defined as foreign key):

manage(e_ID, office_num)
assign(e_ID, office_num)
own(o_ID, p_ID, percentage)
list(p_ID, office_num)
property_location(p_ID, address, city, state)
office_location(office_num, address, city, state)

3. Modify keys to capture mapping constraints:

modified: manage(e_ID UNIQUE, office_num)
modified: assign(e_ID, office_num)
no change: own(o_ID, p_ID, percentage)
modified: list(p_ID, office_num)
modified: property_location(p_ID, (address, city, state) UNIQUE)
modified: office_location(office_num, (address, city, state) UNIQUE)

4. Eliminate Redundancy:
(Nothing can be done in this step)

5. Merge relations to capture participation constrains:
(1). Relations with same primary key:
employee & assign
sales_office & manage
property & list
sales_office & office_location
property & property_location

(2). Merge relations with total participation and define foreign key not NULL:
merge: sales_office & manage & office_location
=> sales_office(office_num, foreign key manager_ID references to employee(ID) UNIQUE NOT NULL, foreign key (address, city, state) references to location(address, city, state) UNIQUE NOT NULL)

merge: property & property_location
=> property(ID, foreign key (address, city, state) references to location(address, city, state) UNIQUE NOT NULL)

final results:
employee(ID, name)
owner(ID, name)
sales_office(office_num, foreign key manager_ID references to employee(ID) UNIQUE NOT NULL, foreign key (address, city, state) references to location(address, city, state) UNIQUE NOT NULL)
property(ID, foreign key (address, city, state) references to location(address, city, state) UNIQUE NOT NULL)
location(address, city, state, zip_code)
assign(e_ID references to employee(ID), foreign key (office_num) references to sales_office(office_num))
own(o_ID references to owner(ID), p_ID, percentage)
list(p_ID references to property(ID), foreign key (office_num) references to sales_office(office_num))

Cardinality and Participation Constraints
For the tree constraints below:

1. Each sale office must be assign greater than 1 employee(s).
2. Each owner has at least one property.
3. Each property has at least one owner.

We cannot ensure their total participations because the only way to ensure this (at least as much as we have learnt in SQL) is to define a NOT NULL foreign key with references to another. To do this, they should eventually be merged together, however, because they either have a many-to-many or one-to-many relationship, none of them can be merged.

As for the constraint that the co-owners has their percentages sum to be 100, we cannot ensure this cardinality constraints either. Because the percentage of ownership unit is associated with particular "own" relationship, and there may be many instances of "own" table with different percentage values, so there is no simple way to enforce their percentages sum to be 100.

For the following tree constraints:

1. Given one sales_office, there is exact one employee being manager.
2. An employee can only be assign to one sales_office.
3. Each property be listed with only one sales_office.

We can enforce these constrains by defining manager_ID as primary key in "manage", employee ID as primary

key in "assign" and property ID as primary in "list" relations. So that primary key constrain will ensure there is no two lines with duplicated primary key values for each relation.

SQL statements to implement the relational schema:

```
CREATE TABLE `employee` (
  `ID` int(11) NOT NULL,
  `name` varchar(30),
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB;

CREATE TABLE `owner` (
  `ID` int(11) NOT NULL,
  `name` varchar(30),
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB;

CREATE TABLE `location` (
  `address` varchar(120) NOT NULL,
  `city` varchar(30) NOT NULL,
  `state` varchar(30) NOT NULL,
  `zip_code` int(11),
  PRIMARY KEY (`address`,`city`,`state`)
) ENGINE=InnoDB;

CREATE TABLE `sales_office` (
  `office_num` int(11) NOT NULL,
  `manager_ID` int(11) NOT NULL,
  `address` varchar(120) NOT NULL,
  `city` varchar(30) NOT NULL,
  `state` varchar(30) NOT NULL,
  PRIMARY KEY (`office_num`),
  UNIQUE (manager_ID),
  UNIQUE (address, city, state),
  FOREIGN KEY (`manager_ID`) REFERENCES `employee` (`ID`),
  FOREIGN KEY (`address`, `city`, `state`)
  REFERENCES `location` (`address`, `city`, `state`)
) ENGINE=InnoDB;

CREATE TABLE `property` (
  `ID` int(11) NOT NULL,
  `address` varchar(120) NOT NULL,
  `city` varchar(30) NOT NULL,
  `state` varchar(30) NOT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE (address, city, state),
  FOREIGN KEY (`address`, `city`, `state`)
  REFERENCES `location` (`address`, `city`, `state`)
) ENGINE=InnoDB;

CREATE TABLE `own` (
  `o_ID` int(11) NOT NULL,
  `p_ID` int(11) NOT NULL,
  `percentage` int(11),
  PRIMARY KEY (`o_ID`,`p_ID`),
  FOREIGN KEY (`o_ID`) REFERENCES `owner` (`ID`),
  FOREIGN KEY (`p_ID`) REFERENCES `property` (`ID`)
) ENGINE=InnoDB;

CREATE TABLE `assign` (
  `e_ID` int(11) NOT NULL,
  `office_num` int(11) NOT NULL,
  PRIMARY KEY (`e_ID`),
  FOREIGN KEY (`e_ID`) REFERENCES `employee` (`ID`),
  FOREIGN KEY (`office_num`) REFERENCES `sales_office` (`office_num`)
) ENGINE=InnoDB;

CREATE TABLE `list` (
  `p_ID` int(11) NOT NULL,
  `office_num` int(11) NOT NULL,
  PRIMARY KEY (`p_ID`),
  FOREIGN KEY (`p_ID`) REFERENCES `property` (`ID`),
  FOREIGN KEY (`office_num`) REFERENCES `sales_office` (`office_num`)
) ENGINE=InnoDB;
```

Problem 5

(a).

1. employee: employee.ID -> employee.name
2. owner: owner.ID ->owner.name
3. location: address, city, state -> zip_code
4. assign: employee.ID -> office_num
5. office_location: office_num -> address
6. office_location: office_num -> city
7. office_location: office_num -> state

8. office_location: office_num -> zip_code

(b).

1. employee: employee_num - > employee.name
2. employee: employee_num - > birth_date
3. employee: employee_num - > job_title
4. project: project_num -> estimated_cost
5. skill: skill_num -> description
6. department: department.name -> phone_num
7. assign_at: employee_num, city.name -> project_num
8. assign_at: employee_num, project_num -> city.name

(c).

1. submit: order.ID -> customer.ID
2. submit: order.ID -> customer.name
3. submit: order.ID -> customer.address
4. submit: order.ID -> customer.zip_code
5. group: product.ID -> product_line.ID
6. group: product.ID -> product_line.name
7. serve: sales_people.ID -> sales_territory.ID
8. serve: sales_people.ID -> sales_territory.name

Problem 6

part 1

- List all the course titles from math department with greater than or equal to 3 credits:

```
select title from course where credits >= 3 and dept_name = 'Math';
```

- List the ID and name of students who take CISC437:

```
select ID, name from student natural join takes
       where course_id = 'CISC437';
```

- List the ID and name of students who take CISC437:

```
select * from (select ID from takes where course_id = 'CISC437') X
         natural join (select ID, name from student) Y;
```

- Calculate the average classroom capacity of all course sections from each department:

```
select dept_name, avg(capacity) as average_capacity from
       (select dept_name, capacity from classroom
       natural join section natural join course) X
       group by dept_name;
```

part 2

-

```
select title from course join teaches using (course_id)
       join instructor using (ID) where name = 'Wu';
```

$$\pi_{\text{title}}\,\sigma_{\text{name=Wu}}\,(\text{course}\bowtie_{\text{course\_id}}\text{teaches}\bowtie_{\text{ID}}\text{instructor})$$

-

```
select distinct(s.name) from student s join
       takes using (ID) join course c using (course_id)
       where c.dept_name = 'Math';
```

(Assume the "Math course" means courses from Math department)

$$\pi_{\text{student.name}}\,\sigma_{\text{course.dept\_name=Math}}\,(\text{student}\bowtie_{\text{ID}}\text{takes}\bowtie_{\text{course\_id}}\text{course})$$

-

```
select ID, name from instructor i left outer join
      (select distinct(ID) from teaches join instructor
       using (ID) where year < 2009 or
        (year = 2009 and (semester='Summer' or semester='Spring'))
      ) X using (ID) where X.ID is not NULL;
```

$$\pi_{\text{ID,name}}\left(\text{instructor} \rtimes_{\text{ID}} \left(\sigma_{\text{year<2009 or (year=2009 and (semester=Summer or semester=Spring))}}(\text{teaches} \bowtie_{\text{ID}} \text{instructor})\right)\right)$$

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

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

- ```
  select capacity - count(ID) as open_seat from takes natural join
         section natural join classroom where
          course_id = 'CISC637' and semester= 'Spring' and year=2015
         group by course_id, semester, year;
  ```

  (Assume no two different sec_ids for CISC637 in Spring 2015)

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