15.1

select department.dep\_id, dep\_name, count(\*) as num\_employees

from department left join employee

on department.dep\_id = employee.dep\_id

group by department.dep\_id;

15.2

JOIN is used to combine two tables. To perform a join, each table must have at least one field which is used to find matching records in the other table. The join type decides which record will be included in the result set.

1. INNER JOIN. INNER JOIN will only select the records which have matching values in all joined tables. The records with missing values in one or multiple tables will not go into the result set.

2. OUTER JOIN. OUTER JOIN will not only contain the records in INNER JOIN, but also contain records which have no matching values in other tables. OUTER JOINs are divided into following subtypes:

2.1 LEFT OUTER JOIN. The result set contains every record in the left table of the LEFT OUTER JOIN statement no matter whether there is a matching record in the right table. If there is no matching record in the right table, the corresponding value in the result set will be NULL.

2.2 RIGHT OUTER JOIN. This type of JOIN is the opposite of LEFT OUTER JOIN. The result set contains every record in the RIGHT table of the RIGHT OUTER JOIN statement no matter whether there is a matching record in the left table. If there is no matching record in the left table, the corresponding value in the result set will be NULL.

2.3 FULL OUTER JOIN. The result set is the union of the LEFT OUTER JOIN and RIGHT OUTER JOIN. All records will be part of the result set. If there is no matching record in the other table, the corresponding field value will be NULL.

3. CROSS JOIN. The result set is the cartesian product of all records in the two tables. The CROSS JOIN statement have no WHERE clause, as each record in the left table will combine with every record in the right table. Thus if left table has m records and right table has n records, the result set will have m\*n records.

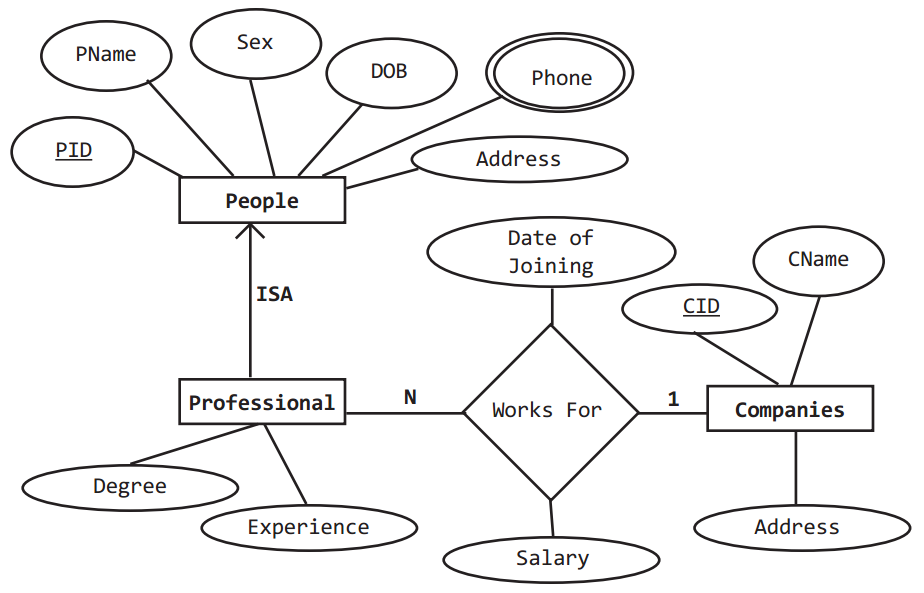
15.3

Denormalization is used to improve database performance by adding redundant data to the relations. A relational normalized database sometimes suffers heavy access load as we have to join many tables and access memory many times to get the data we want. Denormalization reduces the times of joining tables and accessing memory by adding redundant data in the relations.

A normalized design stores different but related pieces of information in separate logical tables. If these tables are stored physically as separate disk files, completing a query that extracts data from several tables will be slow. One way to deal with this is to denormalize the logical data design. The database designers modify the data design by storing additional redundant data on disk to optimize query response. But this will add designers responsibility to ensure the denormalized database consistence. This is done by creating rules in the database called constraints, that specify how the redundant copies of information must be kept synchronized. It is the increase in logical complexity of the database design and the added complexity of the additional constraints that make this approach hazardous. Moreover, constraints introduce a trade-off, speeding up reads (SELECT in SQL) while slowing down writes (INSERT, UPDATE, and DELETE). This means a denormalized database under heavy write load may actually offer worse performance than its functionally equivalent normalized counterpart.

A denormalized data model is not the same as a data model that has not been normalized, and denormalization should only take place after a satisfactory level of normalization has taken place and that any required constraints and/or rules have been created to deal with the inherent anomalies in the design.

15.4



15.5

Assume that we have 3 tables: Student, Course, CourseEnrollment. The Student table has at least two fields: Student\_ID, Student\_name, and probably has other personal information. The Course table has at least two fields: Course\_ID, Course\_name, and will likely contain the course description, the professor, etc. CourseEnrollment will pair Student and Course, and will also contain a field Course\_grade.

The SQL query to get the list of the honor roll students can be:

select Student\_name, GPA

from

(

select Student\_ID, top 10 percent avg(Course\_grade) as GPA

from CourseEnrollment

group by Student\_ID

order by avg(Course\_grade) ) Honnors

inner join Student where Student.Student\_ID = Honnors.Student\_ID;