Schema for examples:

```
apply(sid, cname, major, decision)
college(cname, state, enrollment)
student(sid, sname, gpa, sizehs)
```

1. **Aggregations** Use *aggregation functions* to summarize attribute values of collections of tuples. Aggregation functions include: COUNT, MIN, MAX, SUM, AVG. Find average GPA of all students.

   ```
   select avg(gpa) as averagegpa
   from student;
   ```

   Find the GPA spread (max to min) across all students from high schools with fewer than 1000 students.

   ```
   select max(gpa) - min(gpa)
   from student
   where sizehs < 1000;
   ```

2. **Groups and aggregations** Use the group by clause to group together rows with identical values in specified columns. If an attribute appears in the select clause, one the following conditions must be true: the attribute also appears in the group by clause, or the attribute value is aggregated.

   Find number of accepted applications to each college.

   ```
   select cname, count(*) as numaccepted
   from apply
   where decision = 'Y'
   group by cname;
   ```

   What if we wanted to count number of *applicants* rather than applications? Replace `count(*)` with `count (distinct sID)`.

3. **Filtering aggregated rows** Suppose we wanted only schools that admitted at least 3 applications. We *cannot* use the where clause because the where clause is evaluated *before* rows are grouped. Instead we must use the having clause.

   ```
   select cname, count(*) as numaccepted
   from apply
   where decision = 'Y'
   group by cname
   having count(*) >= 3;
   ```

4. **Joining relations** To combine multiple relations, place relation names in from clause and use where clause to filter the results. Return the locations of colleges that have admitted CS majors. Why the duplicates?

   ```
   select state
   from apply a, college c
   ```

```
where a.cname = c.cname and major = 'CS' and decision = 'Y'
```

A join can also described using the `join ... on` syntax.

```
select state
from apply a join college c on a.cname = c.cname
where major = 'CS' and decision = 'Y'
```

5. **Outer joins** In the join example above, the only tuples from apply that appear in the result are those with a match in the college relation. A left outer join will preserve all tuples in the left relation. Those without a match will be assigned `null` for columns in the right relation. (There are also right outer joins—preserve the right relation—and full outer joins—preserve both relations.)

   Student names and the majors they applied for. If student has no applications, the major field will be null.

   ```
   select distinct sname, major
   from student s left outer join apply a on s.sid = a.sid;
   ```

6. **Subqueries** The answer to every query is a relation. You can therefore compose a query on the result of a previous query.

   Find average GPA of students who applied to be CS majors.

   ```
   select avg(gpa)
   from student, (select sid from apply
                  where major = 'CS'
                  group by sid) as csmajors
   where student.sid = csmajors.sid;
   ```

7. **Subqueries using WITH clause** The subquery syntax can be hard to read. The with clause allows you to define a new temporary table upon which you can express queries. It is (a little bit?) easier to read.

   Find average GPA of students who applied to be CS majors.

   ```
   with csmajors(sid) as
        (select sid from apply
         where major = 'CS'
         group by sid)
   select avg(gpa)
   from student, csmajors
   where student.sid = csmajors.sid;
   ```

8. **Other forms of subqueries** There are other forms of subqueries, such as subqueries in the WHERE clause, subqueries in the SELECT clause. You are encouraged to see the assigned reading for more details.