4
>> Types:
    char(n). Fixed length character
    varchar(n). Variable length with maximum length n.
    int. Integer
    smallint. Small integer
    numeric(p,d). Fixed point number, with user-specified precision of p digits, with d digits to the right of decimal point.
    real, double precision. Floating point and double-precision floating point numbers
    float(n). Floating point number at least n digits.

>>
```
    create table instructor (
        ID          char(5),
        name        varchar(20) not null,
        dept_name   varchar(20),
        salary      numeric(8,2),
        primary key (ID, name),
        foreign key (dept_name) references department,
        foreign key (salary) references teacher
    );
```

primary key automatically not null

>> Remove all rows from the student table
```
    delete from student
```

>> Remove table
```
    drop table r
```

>> Add attribute A with Domain D to table r
```
    alter table r add A D
```

>> Delete attribute A fro table r
```
    alter table r drop A
```

>> Remove duplicated values of A in r
```
    select distinct A
    from r
```

>> Not remove duplicated values of A in r
```
    select all A
    from r
```

>> select all attributes of r
```
    select *
    from r
```

>> Create a table with attribute A and has a row that has 437 as a value

```
    select "437" as A

>> Result is a table with one column and N rows (number of tuples in the
instructors table), each row with value “A”
    select  "A"
    from r

>> would return a relation that is the same as the instructor relation, except that
the value of the attribute salary is divided by 12
    select salary / 12
    from instructor

>> Rename A to B
    select A as B = select A B

    from instructor as T, instructor as S

# WHERE is for adding condition to the queries that can be combined using the
logical connectives and, or, not
# The from clause lists the relations involved in the query

>> Cartesian product of A and B
    from A, B

>> These queries are equal (just product the rows that has the same value of the
same keys)
    select *
    from instructor, teaches
    where instructor.ID = teaches.ID
    =
    select *
    from instructor natural join teaches

>> like uses patterns that are described using two special characters (% and _)
(Patterns are case sensitive):
    percent ( % ). matches any substring.
    underscore ( _ ). matches any character.
    note:
        Match the string “100%”: like ‘100\%'  escape  '\'

    select name
    from instructor
    where name like '%dar%'

>> Ordering the Display of Tuples (desc for descending order or asc for ascending)
    select A
    from r
    order by A desc

    Can sort on multiple attributes:
```

```
        order by A, B

>> A < x < B
    where x between A and B

>> Tuple comparison
    (A.ID, text) = (B.ID, "hi")

>> duplicates :?

>> Set Operations
    or
        (select course_id
        from section
        where sem = ‘Fall’ and year = 2009)
        union
        (select course_id
        from section
        where sem = ‘Spring’ and year = 2010)

    and
        (select course_id
        from section
        where sem = ‘Fall’ and year = 2009)
        intersect
        (select course_id
        from section
        where sem = ‘Spring’ and year = 2010)

    not
        (select course_id
        from section
        where sem = ‘Fall’ and year = 2009)
        except
        (select course_id
        from section
        where sem = ‘Spring’ and year = 2010)

    # To retain all duplicates use the corresponding multiset versions union all,
intersect all and except all.

# Any comparison with null returns unknown that it is treated as false

>> Aggregate Functions
    avg: average value
    min: minimum value
    max: maximum value
    sum: sum of values
    count: number of values
```

```
        select count (distinct ID)
        from teaches
        where semester = ’Spring’ and year = 2010;
```

>> Attributes in select clause outside of aggregate functions must appear in group by list

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

>> Having Clause: predicates in the having clause are applied after the formation of groups whereas predicates in the where clause are applied before forming groups

```
        select dept_name, avg (salary)
        from instructor
        group by dept_name
        having avg (salary) > 42000;
```

All aggregate operations except count(*) ignore tuples with null values on the aggregated attributes

>> Nested Subqueries: A subquery is a select-from-where expression that is nested within another query.

>> in the Where Clause

>> Set Membership: in

```
        select distinct course_id
        from section
        where semester = ’Fall’ and year= 2009 and
            course_id in (select course_id
                        from section
                        where semester = ’Spring’ and year= 2010);
```

>> Set Comparison: some

```
        select name
        from instructor
        where salary > some (select salary

                            from instructor

                            where dept name = ’Biology’);
```

>> Set Comparison: all

```
        select name
        from instructor
        where salary > all (select salary
                            from instructor
                            where dept name = ’Biology’);
```

>> Test for Empty Relations: exists returns true if subquery is nonempty

```
        select course_id
```

```
            from section as S
            where semester = 'Fall' and year = 2009 and
                exists (select *
                        from section as T
                        where semester = 'Spring' and year= 2010
                            and S.course_id = T.course_id);

        >> Test for Absence of Duplicate Tuples: unique returns true if subquery
contains no duplicates
            select T.course_id
            from course as T
            where unique (select R.course_id
                        from section as R
                        where T.course_id= R.course_id

                            and R.year = 2009);

    >> in the From Clause
        select dept_name, avg_salary
        from (select dept_name, avg (salary)
                from instructor
                group by dept_name) as dept_avg (dept_name, avg_salary)
        where avg_salary > 42000;

        # do not need to use the having clause

    >> in the Select Clause: Subqueries in select clause most return a scalar
        select dept_name,
            (select count(*)
            from instructor
            where department.dept_name = instructor.dept_name) as num_instructors
                from department;

>> with clause: provides a way of defining a temporary relation
    with max_budget (value) as
        (select max(budget)
        from department)
    select department.dep_name
    from department, max_budget
    where department.budget = max_budget.value;

>> Deletion
    >> delete table
        delete from r

    >>
        delete from instructor
        where dept_name= 'Finance';

>> Insertion
```

```
    >>
        insert into course
            values (’CS-437’, ’Database Systems’, ’Comp. Sci.’, 4);

    >>
        insert into course (course_id, title, dept_name, credits)
            values (’CS-437’, ’Database Systems’, ’Comp. Sci.’, 4);

    >>
        insert into student
            values (’3003’, ’Green’, ’Finance’, null);

    >>
        insert into student
            select ID, name, dept_name, 0
            from instructor

>> Updates
    >>
        update instructor
        set salary = salary * 1.03
        where salary > 100000;
```