* SQL is used for managing and ~~querying~~ querying relational database.

(*) Data Manipulation Language (DML): Allowing querying, inserting, deleting and modifying tuples in the database.

(*) Data Definition Language (DDL): Used to define schemas, integrity constraints, views & control transactions.

(*) Integrity constraints: Ensures data accuracy (primary key, foreign key, not null constraints)

(*) DDL - Data Defination Language

- Used for specifying relations (tables), including attributes & types & integrity constraints.

Example:

```
Create table instructor(
        ID char(5),
        name varchar(20),
        dept_name varchar(20),
        salary numeric(8,2)
        );
```

Domain type

- char(n) : Fixed length character string
- varchar(n) : variable length character string.
- int, smallint : Integer values.
- numeric(p,d) : Fixed point numbers.
- real, double precision : Floating point numbers.

Integrity constraints

- Primary key — Uniquely identifies rows.
- Foreign key — References another table to maintain relationships.
- Not null : Ensures a column cannot have null values.

SQL Queries :

1. Select clause : Used to specify the columns to retrieve

```
select name from instructor;
```

- use distinct to remove dublicates and * to select all column.

2. Where clause: Applies conditions to filter results.

Example:

```
select name from instructor where
                    dept_name = 'comp. sci.';
```

3. From clause: Specifies the tables involved in the query.

Example: select * from instructor

4. Join operators: Combining rows from two or more tables based on a related column.

5. Rename operation: SQL allows renaming relation and attribute using the as clause.

example

```
select distinct table name
from instructor as instructor table,
instructor as salary
where table salary). s.salary and s.dept_na
                    = 'comp. sci';
```

④ String operation:

- The like operator is used for pattern matching
  - '%' : matches any substring
  - '_' : a single character.

example:

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

Set operation

① Union: Combine results from two queries.

Example:

```
(select course_id from section where
    sem = 'Fall' and year = 2017)
union
(select course_id from section where
    sem = 'Spring' & year = 2018);
```

② Intersects: Returns common results between two queries.

Example:

```
(select course_id from section where sem='Fall'
    and year = 2017)
intersects
(select course_id from section where sem = 'Spring' &
    year = 2018);
```

• Except : Return the difference between two queries.

Exm:

```
(select course_id from section where sem = 'Fall' and
    year = 2017)
except
(select course_id from section where sem = 'Spring' and
    year = 2018);
```

Null values :

- SQL allows null values, representing unknown or missing data.
- Use null or is not null to check for nulls.

Exm: select name from instructor where salary is null.

Aggregate function

- Operate on sets of values to return a single value.
  - avg() → calculates average.
  - min() → finds the minimum
  - max() → " " maximum
  - sum() → sum the values.
  - count() → counts the number of values.

example :

```
select avg(salary) from instructor where dept_name =
                                    'comp. sci.';
```

Group by & Having

- Group by : Used to group rows sharing a value.
- Having clause : Applies conditions to groups after aggregation.

Example:

```
select dept_name, avg(salary) as avg_salary
    from instructor grouped by dept_name
    having avg(salary) > 42000;
```

Nested subqueries:

- SQL allows queries with in queries(subqueries)

Exm:

```
select name from instructor where salary >
    (select avg(salary) from instructor);
```

Group by : Group rows sharing a value

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

Having : Filter groups after aggregation:

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

Subqueries in the from clause:

- SQL allows the use of subquery expressions in the from clause, treating the results as a temporary relation.

Example:

```
select dept_name, avg_salary
from (select dept_name, avg(salary) as avg_salary
      from instructor
      group by dept_name)
where avg_salary > 42000;
```

With clause

- with clause in SQL allows the creation of temporary relations that can be reused within a query.

exm:

```
with max_budget(value) as (
        select max(budget)
        from department
    )
select department.name
from department, max_budget
where department.budget = max_budget.value;
```

Complex queries using with clause

- With clause can be used to break complex queries into smaller, more manageable parts.

example:

```
with dept_total(dept_name, value) as (
    select dept_name, sum(salary)
    from instructor
    group by dept_name
    );
dept_total_avg(value) as (
    select avg(value)
    from dept_total
)
select dept_name
from dept_total, dept_total_avg
where dept_total.value > dept_total_avg.
                         value;
```

Scalar subquery

A scalar subquery is a subquery that returns a single value, typically used where a single value is expected like in the select, where or from clause.

example:

```
select dept_name,
    (select count(*)
    from instructor
    where department.dept_name = instructor.dept_name)
    as num_instructors
    from department;
```

Modification of the database:

- Insert: Add data to a table

example: insert into instructor values ('10211', 'Smith', 'Biology', 66000);

- Delete: Delete data from a table

example: delete from instructor where dept_name = 'Finance';

- Update: Modify existing data.

example: update instructor set salary = salary * 1.05 where salary < 70000;

Case statement:

- conditional logic inside SQL queries.

example:

```
Update instructor
set salary = case
    when salary <= 10000 then salary * 1.05
    else salary * 1.03
    end;
```