SQL

- SQL = Structured Query Language has become something of a standard in relational systems.
- Virtually all relational DBMS's support SQL.
- They usually provide "a superset of a subset," but the differences for large systems are typically in details and newer ideas not yet included in the standard.
- In the course, we will try to stick to common features which most vendors are likely to support.
- SQL provides several types of commands:
    - Data definition
        - Define and alter the relational schema
        - Define views of the schema
    - Data control
        - Control of access
    - Data manipulation
        - Query
        - Update

In this presentation, emphasis will be placed upon data definition and queries.

Basic Syntax for queries in SQL:

```
Select  <attribute-list>
From    <relation-list>
[Where  <condition>]
```

The examples presented here will use the schema of the text (Figure 5.7; Figure 8.1, fifth and fourth editions); (Figure 7.7; Figure 8.1(a), third edition). First, the basic SPJ operations of the relational algebra are covered.

Example (Selection): List the tuples of the Employee relation which identify females.

```
Select   *
From     Employee
Where    Sex = ‘F’
```

Example (Projection): List the first, last, and middle names of all Employees.

```
Select    FName, MInit, LName
From      Employee
```

Note that the Where clause is optional, and not needed here.

Example (Combine selection and projection): List the first, last, and middle names of all female Employees.

```
Select   FName, MInit, LName
From     Employee
Where    Sex = ‘F’
```

Example (Join): Join the Employee and Department tables on the DNumber and DNo keys.

```
Select   *
From     Employee, Department
Where    DNo = DNumber
```

Example (Natural join): Suppose that the attribute DNo in the Employee relation is changed to be DNumber, exactly the same as in Department. Then it is necessary to qualify the attributes of the join. Here is the solution to the same query as above.

```
Select   *
From     Employee, Department
Where    Employee.DNumber =
         Department.DNumber
```

Example (SPJ combination): List the names of all female Employees, together with the name of the department in which they work.

```
Select   LName, FName, MInit, DName
From     Employee, Department
Where    (DNo = DNumber) and (Sex = ‘F’)
```

Example (Logical connectives): List the names of all Employees who either are female or else work in the research department.

```
Select   LName, FName, MInit
From     Employee, Department
Where    (DNo = DNumber) and
         ((Sex = ‘F’) or (DName = ‘Research’))
```

Example (Union): An alternative solution to the previous query.

```
Select   LName, FName, MInit
From     Employee
Where    Sex = ‘F’
Union
Select   LName, FName, MInit
From     Employee, Department
Where    (DNo = DNumber) and
         (DName = ‘Research’)
```

Example (Logical connectives): List the names of all Employees who are both female and work in the research department.

```
Select    LName, FName, MInit
From      Employee, Department
Where     (DNo = DNumber) and
          (Sex = ‘F’)
               and (DName = ‘Research’)
```

Example (Intersection): An alternative solution to the previous query.

```
Select    LName, FName, MInit
From      Employee
Where     Sex = ‘F’
Intersect
Select    LName, FName, MInit
From      Employee, Department
Where     (DNo = DNumber) and
               (DName = ‘Research’)
```

Example (Logical connectives): List the names of all Employees who are female but do not work in the research department.

```
Select   LName, FName, MInit
From     Employee, Department
Where    (DNo = DNumber) and
         ((Sex = ‘F’) and
           (NOT (DName = ‘Research’)))
```

Example (Set difference): An alternative solution to the previous query.

```
Select   LName, FName, MInit
From     Employee
Where    Sex = ‘F’
Except
Select   LName, FName, MInit
From     Employee, Department
Where    (DNo = DNumber) and
         (DName = ‘Research’)
```

Example (Removing duplicate elements): By default, SQL will not remove duplicate elements in a query. For example, suppose we want to obtain a list of all project locations. The following query will do the trick, but will duplicate the locations which are the homes of more than one project.

```
Select    Plocation
From      Project
```

To obtain a list with duplicates removed, the following query may be used.

```
Select    Distinct Plocation
From      Project
```

Example (Order operations): The usual order operations are available in SQL. For example, here a query which finds the names of all Employees who have a salary greater than 40000.

```
Select    LName, FName, MInit
From      Employee
Where     Salary > 40000
```

Example (Embedded queries): The “Where” part of a query may itself be a query. For example, here is another version of the query which finds the names of all Employees who work in the department which houses the “Computerization” project.

```
Select   LName, FName, MInit
From     Employee
Where    DNo In
           (Select  DNum
            From   Project
            Where
              PName=’Computerization’)
```

Example (Aliases): Sometimes, it is useful to introduce an alias name for a relation. Here is another solution to the query which lists the names of all Employees who either are female or else work in the research department.

```
Select   LName, FName, MInit
From     Employee E, Department D
Where    (E.DNo = D.DNumber) and
         ((E.Sex = ‘F’)
               or (D.DName = ‘Research’))
```

Here E is an alias for Employee, and D for Department.

Example (Aliases): Sometimes, it is essential to use aliases. This example retrieves the Last Name of each Employee, together with the Last Name of the supervisor of that Employee. Employees with no supervisor are not listed.

```
Select    E.LName, S.LName
From      Employee E, Employee S
Where     E.SuperSSN = S.SSN
```

Example (Quotient): Unfortunately, SQL does not have the quotient operation defined directly. However, it is easy to realize it. In this example, the names of those Employees who work on every project are found.

```
Select   LName, FName, MInit
From     Employee
Where    Not Exists
           (Select PNumber
            From Project
             Except
               (Select   PNo
                From     Works_On
                Where    SSN=ESSN
               )
            )
```

While Microsoft Access does not support the Except and Intersect operations, PostgreSQL does.

!!!☺!!!

Here is an alternate way to realize the division operator:

```
Select    LName, FName, MInit
From      Employee
Where     Not Exists
             (Select PNumber
              From Project
               Where Not Exists
                 (Select *
                 From Works_On
                  Where (PNumber=PNo)
                         and (SSN = ESSN)
                 )
              )
```

Example (Pattern matching): The Like operator effects pattern matching. Here is a query which finds all Employees whose first names begin with the letter “J”.

```
Select    LName, FName, MInit
From      Employee
Where     FName like 'J%'
```

Example (Pattern matching): Here is a similar example; this time the first name must contain exactly five characters as well.

```
Select LName, FName, MInit
From Employee
Where FName like 'J____'
```

Example (Pattern matching): To search for values in the range 40% to 49%, use the following

```
Select Percentage, Item
From Sales
Where Percentage like '4_~%'  Escape '~”
```

- Since PostgreSQL is case sensitive, it supports the “ilike” operator which provides case-insensitive maching.

- Pattern-matching features vary substantially from system to system. Check the manual for the flavor of SQL which you are using.

Example (Ordering the result): The following query orders the Employee names by salary, smallest to largest.

```
Select    LName, FName, MInit, Salary
From      Employee
Order by Salary
```

Example (Sense of ordering): Here is how to list the Employees with largest-to-smallest salary ordering.

```
Select    LName, FName, MInit, Salary
From      Employee
Order by Salary Desc
```

Example (Sort fields): It is not necessary to include the sort field in the select component.

```
Select    LName, FName, MInit
From      Employee
Order by Salary Desc
```

Example (Aggregation operations): Here is a query which provides the minimum, maximum, and average salary of the Employees.

```
Select    Min(Salary), Max(Salary),
          Avg(Salary)
From      Employee
```

Example (Naming columns): The above query will not name the columns with anything useful. Here is how to provide explicit names.

```
Select    Min(Salary) as MIN_Salary,
          Max(Salary) as MAX_Salary,
          Avg(Salary) as AVG_Salary
From      Employee
```

Example (Grouping aggregated results): This query groups minimum, maximum, and average salary by supervisor identity number.

```
Select    Min(Salary) as MIN_Salary,
          Max(Salary) as MAX_Salary,
          Avg(Salary) as AVG_Salary,
          SuperSSN
From      Employee
Group by SuperSSN
```

Example (The Having operation: Grouping aggregation with conditions): This is similar to the previous example, except that only aggregations for those supervisors who supervise an Employee who earns less than 30000 are listed.

```
Select   Min(Salary) as MIN_Salary,
         Max(Salary) as MAX_Salary,
         Avg(Salary) as AVG_Salary,
         SuperSSN
From     Employee
Group by SuperSSN
Having Min(Salary) < 30000
```

Example (Counting): The following query counts the number of Employees for each supervisor.

```
Select   SuperSSN,
         Count(SSN) as No_Supervisees
From     Employee
Group by SuperSSN
```

Example (Eliminating null indicators): The following query does the same as the above, save that it does not show the tuple for which the supervisor’s SSN is null.

```
Select    SuperSSN,
          Count(SSN) as No_Supervisees
From      Employee
Where     SuperSSN Is Not Null
Group by SuperSSN
```

Further Comments on the *Group By* and *Having* Directives

When using these directives, every field in the *Select* clause must be either an attribute which is used in the *Group By* directive, or an aggregation. Thus, the following query is not legal, even though it makes perfect sense.

```
Select      E.SSN, E.Salary
From        Employee as E
Group by    E.SSN
Having      E.Salary > 30000;
```

The following query, which has exactly the same semantics, is legal.

```
Select      E.SSN, Avg(E.Salary)
From        Employee as E
Group by    E.SSN
Having      Avg(E.Salary) > 30000;
```

Of course, the following is a simpler way to achieve the same result.

```
Select      E.SSN
From        Employee as E
Where       E.Salary > 30000;
```

If the *Having* directive is used without a *Group By* directive, the effect is to collect all tuples into a single group. Thus, the following query is illegal, and does not make sense.

```
Select      E.SSN, E.Salary
From        Employee as E
Having      E.Salary > 30000;
```

To obtain a list of all Employees with a salary greater than 30000, use the following query.

```
Select      E.SSN, E.Salary
From        Employee as E
Where       E.Salary > 30000;
```

Remarks on the *Contains* directive

- Query 3 on page 260 (page 233 in the fourth edition and page 263 in the third Edition) of the textbook illustrates the use of the *Contains* directive.

- Note that this directive is NOT part of SQL, as is noted on page 260 (233 in the fourth edition and 264 in the third edition).

- Do not use it in any submissions or on examinations answers; it will be marked as incorrect.

Defining Relations in SQL

General Format:

```
Create Table <Table Name>
    (<Attribute>        <Format>     <Condition>,
         .                  .              .
         .                  .              .
[Constraint <constraint_name>]
    Primary Key (<Attribute>+),
[Constraint <constraint_name>]
    Unique(<Attribute>),
[Constraint <constraint_name>]
    Foreign Key (<Attribute>+)
        References <TableName>(<Attribute>+),
[Constraint <constraint_name>]
      Foreign Key …
      )
```

Notes:

- […] denotes zero or one occurrences of the enclosed item.
- …+ denotes one or more occurrences of the preceding item.

• SQL syntax for actions on updates

```
   Create Table <name>

   .
.

Foreign Key <attribute> References <attribute>
   On Delete {Set Null, Set Default}
   On Update Cascade
```

Note: {…} means select one.

Update Operations in Relational Systems:

- There are three main update operations:
  - Insert
  - Delete
  - Modify

- These are tuple-at-a-time operations semantics are self-evident.

- Note that a modify operation may not be equivalent to a delete operation followed by an insert operation, because integrity constraints may be violated after the deletion but before the insertion.