If you recall from a previous lecture, we stated that the Relational Data Model is concerned with:

- Data Structure (flat, 2D tables)
- Data Manipulation (relational algebraic operations)
- Data Integrity (ensuring that the DB accurately reflects the real world)

We have explored Relational Data Structure, and have learned about Relational Data Manipulation via our study of Relational Algebra. We will see Data Integrity concepts later in the semester.

Actual DBMS software does not use the notation of the mathematical model (Relational Algebra) to perform operations on a Database, but uses a 4GL programming language. The most popular of these languages is SQL.

**SQL (Chapters 6 & 7)**

The standard language for Relational Database products

Most relational DBMS packages (Oracle, Nomad, Ingres, DB2, Access) support some dialect of SQL (ex. Oracle SQL has certain extensions that make it slightly different from the pure standard...)
SQL consists of several parts:

- **Data Definition Language (DDL)**
  defining and modifying relational schemas, deleting relations, creating indices, etc.

- **Data Manipulation Language (DML)**
  storage and retrieval of data from already-established schemas
  inserting, updating, deleting tuples; querying the database

- **Embedded DML**
  for use in C#, Java, C++, etc. programs

- **View definition**
  create and maintain virtual relations

- **Authorization**
  define and enforce user access rights to database elements

- **Integrity**
  allows application-specific rules regarding data to be specified for the database. These rules are always checked when changes are made to the DB.

- **Transaction Control**
  support for multi-user database concurrency control

We will focus on SQL commands that provide DDL and DML operations in this chapter, and we will explore many of the other parts of SQL later in the course.
The DML features of SQL may be divided into two areas:

- Database Maintenance commands: concerned with storage of data
- Database Querying commands: concerned with retrieval of data

We will first explore the Database Querying features of SQL, assuming that relation schemas already have been defined for the DB, and that data have already been stored in the relations.

---

**SELECT statement**

The most frequently used query command in SQL

**general form:**

```sql
SELECT [DISTINCT] attribute-list
    FROM  table-list
    [ WHERE  condition ]
    [ GROUP BY  grouping-attributes ]
    [ HAVING  grouping-condition ]
    [ ORDER BY  attribute/{ASC | DESC}-pairs ]
```

Note that the meaning of the SELECT statement in SQL differs from the meaning of the selection operation (σ) in relational algebra. This is an unfortunate situation, but with care, we can avoid confusion.
Relational Algebraic operators in SQL

With the SELECT statement, most of the operators from the relational algebra that provide retrieval of data from relations may be expressed:

Restriction: $\sigma_P (R)$ - here, P is some logical condition (or predicate)

```
SELECT *
FROM R
WHERE P
```
(here, wildcard character * matches all attributes)

Projection: $\pi_{A, B, ..., N} (R)$

```
SELECT DISTINCT A, B, ..., N
FROM R
```
(Without the keyword DISTINCT, SQL allows duplicate tuples in the result. This is a violation of relational model rules, but is OK in practice.)

Cartesian Product: $R \times S$

```
SELECT R.*, S.*
FROM R, S
```

Join: $R \bowtie S$ (assume that attrs. A and B are shared by both R and S)

```
SELECT R.*, S.*
FROM R, S
WHERE R.A = S.A AND R.B = S.B
```
(Note that this isn't technically the same as the join, because the result of this SELECT would have two copies of all duplicated attributes in R and S. A more detailed attribute list would be required to exactly duplicate the join operation.)
Projection without duplicate elimination:

\[
\text{SELECT A, B, ..., N} \\
\text{FROM R}
\]

SELECT does not eliminate duplicate tuples by default

Why not?

- Elimination of duplicates is computationally expensive (requires sorting)
- User may wish to include duplication in result
- Duplication is required for most aggregate functions to correctly compute answers

Default state is to allow duplicates

Duplicate tuple elimination requires use of DISTINCT keyword
Set Operations

**general form:**  
\[ \text{table1} \ \text{UNION} \ \text{table2} \]
\[ \text{table1} \ \text{INTERSECT} \ \text{table2} \]
\[ \text{table1} \ \text{MINUS} \ \text{table2} \]

**rules:**  
\text{table1} \ \text{and} \ \text{table2} \ \text{must be type-compatible} \\
\text{attributes in each table must physically be in the same order}

**example:**
\[
( \ \text{SELECT} \ \text{EName} \\
 \ \text{FROM} \ \text{Employees} \\
 \ \text{WHERE} \ \text{Dept} = "toy" )
\]
\text{MINUS}
\[
( \ \text{SELECT} \ \text{EName} \\
 \ \text{FROM} \ \text{Employees} \\
 \ \text{WHERE} \ \text{Dept} = "candy" )
\]

Not all versions of SQL support MINUS and INTERSECT, as these can be expressed using different forms of SELECT. (How?) All versions of SQL do support UNION.
Renaming Attributes

allows a column name to be given to computed results

provides a new name for shared fields, so that two columns don't have the same name in some report

gives flexibility in naming attributes

The AS clause in the *attribute-list* performs this task

example:

```
SELECT  StName, QualityPts / Credits AS GPA
FROM    Student
```

Renaming Relations

Provides a shorter name for a relation for use in a SELECT statement

Helps when fully-qualified attribute names are needed (e.g., in a join operation)

The AS clause in the *table-list* performs this task

example:

```
SELECT  S.StName, C.Description
FROM    Student AS S, Courses AS C, Regis AS R
WHERE   S.StNum = R.StNum
        AND  C.CourseNum = R.CourseNum
        AND  C.Section = R.Section
```
Sorting

Not a concern in the mathematical model (no ordering of tuples is defined in relational theory)

This is a concern in the real world: ordering of information in a report is oftentimes crucial to understanding the report.

general form:

```
SELECT ....
   FROM ....
   WHERE ....
   ORDER BY order-list
```

*order-list* is a comma-separated list of *attribute order* pairs
(space separates *attribute* and *order* in the pair)

order designators: ASC or DESC (ASC is the default)

every example:

```
SELECT   *
   FROM   Student
   WHERE   GPA >= 3.5
   ORDER BY   LName DESC, FName, Middle ASC
```
Consider the parts, suppliers and shipments database:

PARTS table:

<table>
<thead>
<tr>
<th>P#</th>
<th>PName</th>
<th>Color</th>
<th>Weight</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Nut</td>
<td>Red</td>
<td>15</td>
<td>Paris</td>
</tr>
<tr>
<td>P2</td>
<td>Bolt</td>
<td>Green</td>
<td>12</td>
<td>Athens</td>
</tr>
<tr>
<td>P3</td>
<td>Cam</td>
<td>Blue</td>
<td>20</td>
<td>London</td>
</tr>
<tr>
<td>P4</td>
<td>Cog</td>
<td>Puce</td>
<td>17</td>
<td>London</td>
</tr>
</tbody>
</table>

SUPPLIERS table:

<table>
<thead>
<tr>
<th>S#</th>
<th>SName</th>
<th>City</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Smith</td>
<td>London</td>
<td>20</td>
</tr>
<tr>
<td>S2</td>
<td>Jones</td>
<td>Paris</td>
<td>10</td>
</tr>
<tr>
<td>S3</td>
<td>Blake</td>
<td>Athens</td>
<td>30</td>
</tr>
</tbody>
</table>

SHIPMENTS table:

<table>
<thead>
<tr>
<th>*</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>P#</td>
<td>S#</td>
</tr>
<tr>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td>P1</td>
<td>S1</td>
</tr>
<tr>
<td>P2</td>
<td>S1</td>
</tr>
<tr>
<td>P1</td>
<td>S2</td>
</tr>
<tr>
<td>P3</td>
<td>S2</td>
</tr>
<tr>
<td>P2</td>
<td>S3</td>
</tr>
<tr>
<td>P3</td>
<td>S3</td>
</tr>
</tbody>
</table>

1) Get part names for all parts

2) Get all part names without duplicates

3) For all parts, display the part number and weight in grams (giving the computed column the name GRAMS)
Consider the parts, suppliers and shipments database:

**PARTS table:**

<table>
<thead>
<tr>
<th>P#</th>
<th>PName</th>
<th>Color</th>
<th>Weight</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Nut</td>
<td>Red</td>
<td>15</td>
<td>Paris</td>
</tr>
<tr>
<td>P2</td>
<td>Bolt</td>
<td>Green</td>
<td>12</td>
<td>Athens</td>
</tr>
<tr>
<td>P3</td>
<td>Cam</td>
<td>Blue</td>
<td>20</td>
<td>London</td>
</tr>
<tr>
<td>P4</td>
<td>Cog</td>
<td>Puce</td>
<td>17</td>
<td>London</td>
</tr>
</tbody>
</table>

**SUPPLIERS table:**

<table>
<thead>
<tr>
<th>S#</th>
<th>SName</th>
<th>City</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Smith</td>
<td>London</td>
<td>20</td>
</tr>
<tr>
<td>S2</td>
<td>Jones</td>
<td>Paris</td>
<td>10</td>
</tr>
<tr>
<td>S3</td>
<td>Blake</td>
<td>Athens</td>
<td>30</td>
</tr>
</tbody>
</table>

**SHIPMENTS table:**

<table>
<thead>
<tr>
<th>P#</th>
<th>S#</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>S1</td>
<td>200</td>
</tr>
<tr>
<td>P2</td>
<td>S1</td>
<td>700</td>
</tr>
<tr>
<td>P1</td>
<td>S2</td>
<td>300</td>
</tr>
<tr>
<td>P3</td>
<td>S2</td>
<td>500</td>
</tr>
<tr>
<td>P2</td>
<td>S3</td>
<td>450</td>
</tr>
<tr>
<td>P3</td>
<td>S3</td>
<td>50</td>
</tr>
</tbody>
</table>

1) Get part names for all parts

   SELECT PNAME
   FROM PARTS

2) Get all part names without duplicates

   SELECT DISTINCT PNAME
   FROM PARTS

3) For all parts, display the part number and weight in grams (giving the computed column the name GRAMS)

   SELECT P#,
   WEIGHT*454 AS GRAMS
   FROM PARTS
4) Display all attributes of all suppliers

5) Show the Supplier Name for all suppliers in Paris with status more than 20

6) Print the Supplier number and status rating for all suppliers in Paris, in descending order of status

7) Print the Supplier Name for all suppliers from whom we have an order for part P3

8) What are the colors of all parts currently on order; each color should appear only once.

9) Print Part Name, Supplier Name and Quantity for each shipment
4) Display all attributes of all suppliers
   SELECT *
   FROM SUPPLIERS

5) Show the Supplier Name for all suppliers in Paris with status more than 20
   SELECT DISTINCT SNAME
   FROM SUPPLIERS
   WHERE CITY = "Paris"
   AND STATUS > 20

6) Print the Supplier number and status rating for all suppliers in Paris, in descending order of status
   SELECT S#, STATUS
   FROM SUPPLIERS
   WHERE CITY = "Paris"
   ORDER BY STATUS DESC

7) Print the Supplier Name for all suppliers from whom we have an order for part P3
   SELECT S.SNAME
   FROM SUPPLIERS AS S,
   SHIPMENTS AS SP
   WHERE S.S# = SP.S#
   AND SP.P# = "P3"

8) What are the colors of all parts currently on order; each color should appear only once.
   SELECT DISTINCT P.COLOR
   FROM SHIPMENTS AS SP,
   PARTS AS P
   WHERE SP.P# = P.P#

9) Print Part Name, Supplier Name and Quantity for each shipment
   SELECT P.PNAME,
   S.SNAME, SP.QTY
   FROM PARTS AS P,
   SUPPLIERS AS S,
   SHIPMENTS AS SP
   WHERE S.S# = SP.S#
   AND P.P# = SP.P#
Legal Comparison Operators in a WHERE clause

>  <  >=  <=  =  <>

BETWEEN low AND high
IN ( list-of-values )
NOT IN ( list-of-values )

ex. SELECT Student#, Name, Major
    FROM Student
    WHERE gpa BETWEEN 2.40 AND 2.95

ex. SELECT Student#, Name, gpa
    FROM Student
    WHERE Major IN ("BI", "CH", "MA", "PY")

ex. SELECT Student#, Name, gpa
    FROM Student
    WHERE Major NOT IN ("CS", "IS", "MC")

Simple conditions can be combined using logical operators:

AND    OR    NOT

ex. SELECT Student#, Name
    FROM Student
    WHERE gpa >= 3.5
    AND Major IN ("CS", "IS", "MC")
    AND Class <> "Senior"
Advanced SQL Features

String pattern matching with wild-card characters (pp. 125 - 126):
New comparison operator for use in WHERE clauses: LIKE
ex. SELECT *
    FROM Student
    WHERE StName LIKE "Smi%h"
this query matches any tuple in Student whose name starts with
"Smi", ends with "h", and has zero or more characters (of any
value) between those strings.
So, it would match "Smith", Smich", "Smih", "Smitten by Smith", "SmiBoyAmISickOfThisExampleh", etc.
These wildcards act like Unix (or discrete math) regular expressions, not WinDOS wildcards

Wildcards in Standard SQL:
% → matches 0 or more characters (any value)
_ → matches exactly 1 character (any value)
ex. WHERE StrField LIKE "__X_%" would be true
for any tuple that has a StrField value that starts
with any two characters, has an "X" as the third
character, and is at least 4 characters long (but may
be more than 4 characters long).

Wildcards in Access SQL: (Access supports neither % nor _)
* → matches 0 or more characters (any value)
? → matches exactly 1 character (any value)
[abc] → matches exactly 1 character (must be one of the
characters between the square brackets)
Special character as the first one in the brackets: ^
meaning: match any character EXCEPT the ones in this set

ex. SELECT *
    FROM PARTS
    WHERE LOCATION LIKE "?[aeiou]*[^aeiou]"

this would match any string that has a vowel in the second position, and ends with any character other than a vowel (the ^ is not part of the set)

How to specify a pattern that actually contains a *, ?, [ , or ] character?
escape sequence: WHERE StrFld LIKE "\\*?\\[^*""
          What strings would this pattern match?

How to include the ^ character in a set?
Put it anywhere in the set except the first position:

    WHERE StrFld LIKE "??[#&^]"
          What strings would this pattern match?
          What is the length of all strings that match this pattern?
JOIN operator (in a table-list)

A Join (bowtie) operation can be performed in SQL with an appropriate WHERE clause, testing for equality between shared attributes in two tables:

```
SELECT * 
  FROM  Student, Regis 
  WHERE  Student.StNum = Regis.StNum 
      AND  GPA < 2.2;
```

The WHERE clause in this example serves two purposes: one part is doing the joining, and another is doing a tuple-test for inclusion (the sigma operation from relational algebra).

To alleviate any confusion this might cause, and for efficiency in query processing, SQL provides the JOIN operation to perform joins in a query. This is equivalent to the query above:

```
SELECT * 
  FROM   Student JOIN Regis ON  Student.StNum = Regis.StNum 
  WHERE   GPA < 2.2;
```

Some SQL versions use “INNER JOIN” instead of “JOIN” (Access is one of these)

There are other types of JOINs (Left Join, Right Join, Outer Join), but we will not look into these at this point in the semester.

What would be the result of the following?

```
SELECT DISTINCT P.COLOR 
  FROM   PARTS AS P JOIN SUPPLIERS AS SP 
     ON   P.P# = SP.P#
```