Chapter 6

**SQL Aggregate Operators**

**Review: SQL Environment**

- **Data Definition Language (DDL)**
  - Commands that define a database, including creating, altering, and dropping tables and establishing constraints
  - CREATE / DROP / ALTER, …

- **Data Manipulation Language (DML)**
  - Commands that maintain and query a database
  - INSERT, UPDATE, DELETE, SELECT, …

- **Data Control Language (DCL)**
  - Commands that control a database, including administering privileges and committing data
  - GRANT, ADD, REVOKE
Review: SQL Queries

**CREATE TABLE** table_name (field type constraints, field2 type2, CONSTRAINT name ...,);

**INSERT INTO** table (fields) **VALUES** (values);

**DELETE FROM** table **WHERE** conditions;

**UPDATE** table **SET** field = value **WHERE** conditions;

**SELECT** [DISTINCT] attribute-list **FROM** table-list **WHERE** conditions;

**SQL Aggregate Operators**
SQL Aggregate Operator: COUNT

- **How many products were on order number 1004?**

  ```sql
  SELECT COUNT(*)
  FROM OrderLine_T
  WHERE OrderID = 1004;
  ```

  *Result:* COUNT(*) : 2

- **COUNT(*)** applied on the set of rows selected by the WHERE clause.

SQL Aggregate Operators

- **How many products were on order number 1004, and what is their ID?**

  ```sql
  SELECT ProductID, COUNT(*)
  FROM OrderLine_T
  WHERE OrderID = 1004;
  ```

  *Result:* ERROR
Why ERROR?

- The problem is that ProductID has 2 values for the two rows selected, and COUNT returns one aggregate value.
- Need to ensure that ProductID returns a single value
- Solution: use the **GROUP BY** clause (discussed shortly)

### SQL Aggregate Operators

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNT(*)</td>
<td>Counts all values</td>
</tr>
<tr>
<td>COUNT(DISTINCT A)</td>
<td>Counts distinct values</td>
</tr>
<tr>
<td>SUM (DISTINCT A)</td>
<td>Sum of distinct values</td>
</tr>
<tr>
<td>AVG (DISTINCT A)</td>
<td>Average of distinct values</td>
</tr>
<tr>
<td>MAX (A)</td>
<td>Maximum value</td>
</tr>
<tr>
<td>MIN (A)</td>
<td>Minimum value</td>
</tr>
</tbody>
</table>

**Purchase_T**

<table>
<thead>
<tr>
<th>Product</th>
<th>Date</th>
<th>Price</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>2013-08-21</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>2013-10-09</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Bagel</td>
<td>2013-10-05</td>
<td>1.50</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>2013-10-04</td>
<td>0.5</td>
<td>10</td>
</tr>
</tbody>
</table>

**Examples**

- SELECT SUM(Price*Quantity) FROM Purchase_T
  - 65 (=20+5+10+30)

- SELECT SUM(Price*Quantity) FROM Purchase_T WHERE Product = ‘Bagel’
  - 50 (=20+30)
Practice: Exercise #6

Write SQL queries to answer the following question:

3. What is the smallest section number used in the first semester of 2008?

SELECT Example–Boolean Operators

AND, OR, and NOT Operators for customizing conditions in WHERE clause:

```
SELECT ProductDescription, ProductFinish, ProductStandardPrice
FROM Product_T
WHERE ProductDescription LIKE '%Desk'
   OR ProductDescription LIKE '%Table'
   AND ProductStandardPrice > 300;
```

Note: the LIKE operator allows you to compare strings using wildcards. For example, the % wildcard in ‘%Desk’ indicates that all strings that have any number of characters preceding the word “Desk” will be allowed.
SELECT Example—Boolean Operators

With parentheses…these override the normal precedence of Boolean operators

```
SELECT ProductDescription, ProductFinish, ProductStandardPrice
FROM Product_T;
WHERE (ProductDescription LIKE '%Desk'
     OR ProductDescription LIKE '%Table')
     AND ProductStandardPrice > 300;
```

By default, the AND operator takes precedence over the OR operator. With parentheses, you can make the OR take place before the AND.
Practice: Exercise #7

Write SQL queries to answer the following questions:

1. How many students are enrolled in Section 2714 in the first semester of 2008?

2. Which faculty members have qualified to teach a course since 1993? List the faculty ID, course ID, and date of qualification.

SELECT – Sorting Results with the ORDER BY Clause

- List all Customers -- Name, City and State -- ordered first by State, and within a state by Name.

```sql
SELECT CustomerName, CustomerCity, CustomerState
FROM Customer_T
WHERE CustomerState IN ('FL', 'TX', 'CA', 'HI')
ORDER BY CustomerState, CustomerName;
```

IN allows you to include only those rows whose CustomerState value is either FL, TX, CA, or HI.
Practice: Exercise #9

Write SQL queries to answer the following questions:

1. What are the courses included in the Section table? List each course only once.

2. List all students in alphabetical order by StudentName.

3. List the students who are enrolled in each course in Semester I, 2008. Order the students by the sections in which they are enrolled.

4. List the courses available. Order them by course prefix. (ISM is the only prefix shown, but there are many others throughout the university.)
Motivation for Grouping

- So far, we have applied aggregate operators to all rows. What if we want to apply them to only a subset of rows?

<table>
<thead>
<tr>
<th>Product</th>
<th>Date</th>
<th>Price</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>2013-08-21</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>2013-10-09</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Bagel</td>
<td>2013-10-05</td>
<td>1.50</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>2013-10-04</td>
<td>0.5</td>
<td>10</td>
</tr>
</tbody>
</table>

- Example:  
  *Find the average price for every product purchased after 2013-10-02*
  - We may not know how many (or what) products are.
  - Group the rows by Product, then SELECT from each group!

```
SELECT AVG(Price) FROM Purchase_T
WHERE Date > '2013-10-02'
GROUP BY Product;
```

Motivation for Grouping

- *Find the average price for every product purchased after 2013-10-02*

```
SELECT AVG(Price)
FROM Purchase_T
WHERE Date > '2013-10-02'
GROUP BY Product;
```

- Evaluation order:
  - FROM: select table
  - WHERE: select rows
  - GROUP BY: group rows
  - SELECT: applied on groups
Motivation for Grouping

- **Find the average price for every product purchased after 2013-10-02**

<table>
<thead>
<tr>
<th>Product</th>
<th>Date</th>
<th>Price</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>2013-08-21</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>2013-10-09</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Bagel</td>
<td>2013-10-05</td>
<td>1.50</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>2013-10-04</td>
<td>0.5</td>
<td>10</td>
</tr>
</tbody>
</table>

```
SELECT AVG(Price) 
FROM Purchase_T 
WHERE Date > '2013-10-02' 
GROUP BY Product;
```

---

**SELECT– Categorizing Results Using GROUP BY**

For use with aggregate functions

- **Scalar aggregate**: single value returned from SQL query with aggregate function
- **Vector aggregate**: multiple values returned from SQL query with aggregate function (via GROUP BY)

```
SELECT CustomerState, COUNT (CustomerState) 
FROM Customer_T 
GROUP BY CustomerState;
```

Note: you can use single-value fields with aggregate functions if they are included in the GROUP BY clause
**SELECT— Filtering Categories Using HAVING**

For use with GROUP BY

```
SELECT CustomerState, COUNT (CustomerState)
FROM Customer_T
GROUP BY CustomerState
HAVING COUNT (CustomerState) > 1;
```

Like a WHERE clause, but it operates on groups (categories), not on individual rows. Here, only those groups with total numbers greater than 1 will be included in final result.
Practice: Exercise #8

Write SQL queries to answer the following questions:

1. Which students are enrolled in Database and Networking? 
   (Hint: Use SectionNo for each class so you can determine the answer from the Registration table by itself.)

2. Which instructors cannot teach both Syst Analysis and Syst Design?
Using and Defining Views

Views provide users controlled access to tables
Base Table – Table containing the raw data

**Dynamic View**
- A “virtual table” created dynamically upon request by a user
- No data actually stored; contents materialized when referenced
- Based on SQL SELECT statement on base tables or other views

**Materialized View**
- Copy or replication of data
- Data actually stored on the disk
- Refreshed periodically to match corresponding base tables

Defining Views

- Useful for presenting different info to different users

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Department</th>
<th>Project</th>
<th>Salary</th>
</tr>
</thead>
</table>

CREATE VIEW Developer_V AS
SELECT Name, Project
FROM Employee_T
WHERE Department = "Development"

- Payroll has access to Employee_T, others to Developer_V only
**Querying a View**

We can later use this view:

```sql
SELECT *
FROM Developer_V;
```

```sql
INSERT INTO Developer_V
VALUES('Mike', 'Gadget Design');
```

```sql
INSERT INTO Employee_T
VALUES(NULL, 'Mike', 'Development', 'Gadget Design', NULL);
```

*Most views are non-updateable.*

**Views – Another Example**

```sql
CREATE VIEW ExpensiveStuff_V
AS
SELECT ProductID, ProductDescription, ProductStandardPrice
FROM Product_T
WHERE ProductStandardPrice > 300
WITH CHECK OPTION;
```

- CHECK OPTION works only for updateable views and prevents updates that would create rows not included in the view.
Advantages of Views

- Simplify query commands
- Assist with data security (but don't rely on views for security, there are more important security measures)
- Enhance programming productivity
- Contain most current base table data
- Use little storage space
- Provide customized view for user
- Establish physical data independence

Disadvantages of Views

- Use processing time each time view is referenced
- May or may not be directly updateable
Practice: Exercise #2

Define the following view:

<table>
<thead>
<tr>
<th>StudentID</th>
<th>StudentName</th>
</tr>
</thead>
<tbody>
<tr>
<td>38214</td>
<td>Letersky</td>
</tr>
<tr>
<td>54907</td>
<td>Altvater</td>
</tr>
<tr>
<td>66324</td>
<td>Aiken</td>
</tr>
<tr>
<td>70542</td>
<td>Marra</td>
</tr>
</tbody>
</table>