In This Lecture

- Yet more SQL
- ORDER BY
- Aggregate functions
- GROUP BY and HAVING
- UNION etc.
- For more information
  - Connoly and Begg Chapter 5
  - Ullman and Widom Chapter 6.4

ORDER BY

- The ORDER BY clause sorts the results of a query
  - You can sort in ascending (default) or descending order
  - Multiple columns can be given
  - You cannot order by a column which isn’t in the result

```sql
SELECT <columns>
FROM <tables>
WHERE <condition>
ORDER BY <cols>
[ASCENDING | DESCENDING]|
ASC | DESC ]
```

ORDER BY Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>DBS</td>
<td>56</td>
</tr>
<tr>
<td>John</td>
<td>A1</td>
<td>72</td>
</tr>
<tr>
<td>Mary</td>
<td>DBS</td>
<td>60</td>
</tr>
<tr>
<td>Mark</td>
<td>PR1</td>
<td>43</td>
</tr>
<tr>
<td>Mark</td>
<td>PR2</td>
<td>35</td>
</tr>
<tr>
<td>Jane</td>
<td>AI</td>
<td>54</td>
</tr>
</tbody>
</table>

SELECT * FROM Grades
ORDER BY Mark

ORDER BY Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>DBS</td>
<td>56</td>
</tr>
<tr>
<td>John</td>
<td>A1</td>
<td>72</td>
</tr>
<tr>
<td>Mary</td>
<td>DBS</td>
<td>60</td>
</tr>
<tr>
<td>Mark</td>
<td>PR1</td>
<td>43</td>
</tr>
<tr>
<td>Mark</td>
<td>PR2</td>
<td>35</td>
</tr>
<tr>
<td>Jane</td>
<td>AI</td>
<td>54</td>
</tr>
</tbody>
</table>

SELECT * FROM Grades
ORDER BY Code ASC, Mark DESC

ORDER BY Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>DBS</td>
<td>56</td>
</tr>
<tr>
<td>John</td>
<td>A1</td>
<td>72</td>
</tr>
<tr>
<td>Mary</td>
<td>DBS</td>
<td>60</td>
</tr>
<tr>
<td>Mark</td>
<td>PR1</td>
<td>43</td>
</tr>
<tr>
<td>Mark</td>
<td>PR2</td>
<td>35</td>
</tr>
<tr>
<td>Jane</td>
<td>AI</td>
<td>54</td>
</tr>
</tbody>
</table>

SELECT * FROM Grades
ORDER BY Code ASC, Mark DESC

ORDER BY Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>DBS</td>
<td>56</td>
</tr>
<tr>
<td>John</td>
<td>A1</td>
<td>72</td>
</tr>
<tr>
<td>Mary</td>
<td>DBS</td>
<td>60</td>
</tr>
<tr>
<td>Mark</td>
<td>PR1</td>
<td>43</td>
</tr>
<tr>
<td>Mark</td>
<td>PR2</td>
<td>35</td>
</tr>
<tr>
<td>Jane</td>
<td>AI</td>
<td>54</td>
</tr>
</tbody>
</table>

SELECT Mark/100
FROM Grades

ORDER BY Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>DBS</td>
<td>56</td>
</tr>
<tr>
<td>John</td>
<td>A1</td>
<td>72</td>
</tr>
<tr>
<td>Mary</td>
<td>DBS</td>
<td>60</td>
</tr>
<tr>
<td>Mark</td>
<td>PR1</td>
<td>43</td>
</tr>
<tr>
<td>Mark</td>
<td>PR2</td>
<td>35</td>
</tr>
<tr>
<td>Jane</td>
<td>AI</td>
<td>54</td>
</tr>
</tbody>
</table>

SELECT Salary + Bonus
FROM Employees

ORDER BY Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>DBS</td>
<td>56</td>
</tr>
<tr>
<td>John</td>
<td>A1</td>
<td>72</td>
</tr>
<tr>
<td>Mary</td>
<td>DBS</td>
<td>60</td>
</tr>
<tr>
<td>Mark</td>
<td>PR1</td>
<td>43</td>
</tr>
<tr>
<td>Mark</td>
<td>PR2</td>
<td>35</td>
</tr>
<tr>
<td>Jane</td>
<td>AI</td>
<td>54</td>
</tr>
</tbody>
</table>

SELECT 1.175*Price
FROM Products
Aggregate Functions

- Aggregate functions compute summaries of data in a table
- Most aggregate functions (all except COUNT) work on a single column of numeric data
- Use an alias to name the result

- Aggregate functions
  - COUNT: The number of rows
  - SUM: The sum of the entries in a column
  - AVG: The average entry in a column
  - MIN, MAX: The minimum and maximum entries in a column

- Example
  - Find John’s average mark, weighted by the credits of each module

GROUP BY

- Sometimes we want to apply aggregate functions to groups of rows
- Example, find the average mark of each student
  
  ```sql
  SELECT COUNT(*) AS Count
  FROM Grades
  ```
  
  ```sql
  SELECT SUM(Mark) AS Total
  FROM Grades
  ```
  
  ```sql
  SELECT MAX(Mark) AS Best
  FROM Grades
  ```

- Every entry in `<cols2>` must be in `<cols1>`, be a constant, or be an aggregate function
- You can have WHERE or ORDER BY clauses as well as a GROUP BY clause

Example

```sql
SELECT COUNT(*) AS Count
FROM Grades
```

```
SELECT SUM(Mark) AS Total
FROM Grades
```

```
SELECT MAX(Mark) AS Best
FROM Grades
```

```sql
SELECT MAX(Mark)-MIN(Mark) AS Range
FROM Grades
```

```
SELECT <cols1>
FROM <tables>
GROUP BY <cols2>
```
GROUP BY

SELECT Name,
      AVG(Mark) AS Average
FROM Grades
GROUP BY Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Code</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>DBS</td>
<td>56</td>
</tr>
<tr>
<td>John</td>
<td>R1</td>
<td>72</td>
</tr>
<tr>
<td>Mary</td>
<td>DBS</td>
<td>60</td>
</tr>
<tr>
<td>Mark</td>
<td>PR1</td>
<td>43</td>
</tr>
<tr>
<td>Mark</td>
<td>PR2</td>
<td>35</td>
</tr>
<tr>
<td>Jane</td>
<td>R1</td>
<td>34</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>64</td>
</tr>
<tr>
<td>Mary</td>
<td>60</td>
</tr>
<tr>
<td>Mark</td>
<td>39</td>
</tr>
<tr>
<td>Jane</td>
<td>54</td>
</tr>
</tbody>
</table>

• Find the total value of the sales for each department in each month
• Can group by Month then Department or Department then Month
• Same results, but in a different order

GROUP BY

SELECT Month, Department,
      SUM(Value) AS Total
FROM Sales
GROUP BY Month, Department

<table>
<thead>
<tr>
<th>Month</th>
<th>Department</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>March</td>
<td>Fiction</td>
<td>20</td>
</tr>
<tr>
<td>March</td>
<td>Travel</td>
<td>30</td>
</tr>
<tr>
<td>March</td>
<td>Technical</td>
<td>40</td>
</tr>
<tr>
<td>April</td>
<td>Fiction</td>
<td>10</td>
</tr>
<tr>
<td>April</td>
<td>Travel</td>
<td>25</td>
</tr>
<tr>
<td>April</td>
<td>Fiction</td>
<td>20</td>
</tr>
<tr>
<td>May</td>
<td>Fiction</td>
<td>20</td>
</tr>
<tr>
<td>May</td>
<td>Technical</td>
<td>50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Month</th>
<th>Department</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>April</td>
<td>Fiction</td>
<td>60</td>
</tr>
<tr>
<td>March</td>
<td>Fiction</td>
<td>20</td>
</tr>
<tr>
<td>May</td>
<td>Fiction</td>
<td>20</td>
</tr>
<tr>
<td>May</td>
<td>Technical</td>
<td>50</td>
</tr>
<tr>
<td>April</td>
<td>Travel</td>
<td>25</td>
</tr>
<tr>
<td>March</td>
<td>Travel</td>
<td>30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Month</th>
<th>Department</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>April</td>
<td>Fiction</td>
<td>60</td>
</tr>
<tr>
<td>March</td>
<td>Fiction</td>
<td>20</td>
</tr>
<tr>
<td>May</td>
<td>Fiction</td>
<td>20</td>
</tr>
<tr>
<td>May</td>
<td>Technical</td>
<td>50</td>
</tr>
<tr>
<td>April</td>
<td>Travel</td>
<td>25</td>
</tr>
<tr>
<td>March</td>
<td>Travel</td>
<td>30</td>
</tr>
</tbody>
</table>

HAVING

• HAVING is like a WHERE clause, except that it applies to the results of a GROUP BY query
• It can be used to select groups which satisfy a given condition

GROUP BY

SELECT Name,
      AVG(Mark) AS Average
FROM Grades
HAVING AVG(Mark) >= 40

<table>
<thead>
<tr>
<th>Name</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>64</td>
</tr>
<tr>
<td>Mary</td>
<td>60</td>
</tr>
<tr>
<td>Jane</td>
<td>54</td>
</tr>
</tbody>
</table>

WHERE and HAVING

• WHERE refers to the rows of tables, and so cannot use aggregate functions
• HAVING refers to the groups of rows, and so cannot use columns which are not in the GROUP BY

Think of a query being processed as follows:
• Tables are combined
• WHERE clauses
• GROUP BY and Aggregates
• Column selection
• HAVING clauses
• ORDER BY

UNION, etc.

• UNION, INTERSECT, and EXCEPT
• These treat the tables as sets and are the usual set operators of union, intersection, and difference
• We’ll concentrate on UNION
• Oracle has MINUS instead of EXCEPT

• They all combine the results from two select statements
• The results of the two selects must have the same columns and data types
UNION

• Find, in a single query, the average mark for each student, and the average mark overall

Grades
Name  Code  Mark
Jane  AI   52
John  BS   56
John  BI   72
Mark  PR1  43
Mark  PR2  55
Mary  BS   60

UNION

• The average for each student:
  SELECT Name,
      AVG(Mark) AS Average
  FROM Grades
  GROUP BY Name

• The average overall
  SELECT
      'Total' AS Name,
      AVG(Mark) AS Average
  FROM Grades

• Note - this has the same columns as the average by student

A Final Example

• Examiners’ reports
  • We want a list of students and their average mark
  • For first and second years the average is for that year
  • For finalists it is 40% of the second year plus 60% of the final year average.

• We want the results
  • Sorted by year then average mark (High to low) then last name, first name, and finally ID
  • To take into account the number of credits each module is worth
  • Produced by a single query

We’ll Need a UNION

• Finalists are treated differently
  • Write one query for the finalists
  • Write a second query for the first and second years
  • Use a UNION to join them together

<QUERY FOR FINALISTS>
UNION
<QUERY FOR OTHERS>
We’ll need to Join the Tables

- Both of the subqueries need information from all the tables
  - The student ID, name and year
  - The marks for each module and the year taken
  - The number of credits for each module

- This is a natural join operation
  - We could use a NATURAL JOIN statement, and hope that our version of SQL can do it
  - Safer to just use a WHERE clause

The Query So Far

```
SELECT <some information>
FROM Student, Module, Grade
WHERE Student.ID = Grade.ID
AND Module.Code = Grade.Code
AND <student is in third year>
UNION
SELECT <some information>
FROM Student, Module, Grade
WHERE Student.ID = Grade.ID
AND Module.Code = Grade.Code
AND <student is in first or second year>
```

Information for Finalists

- We need to retrieve
  - Compute average mark, weighted 40-60 across years 2 and 3
  - First year marks need to be ignored
  - The ID, Name, and Year are needed as they are used for ordering

- The average is hard
  - We don’t have any statement to separate years 2 and 3 easily
  - We can exploit the fact that 40 = 20*2 and 60 = 20*3, so YearTaken and the weighting have a simple relationship

```
SELECT Year, Student.ID, Last, First,
SUM((20*YearTaken/100)*Mark*Credits)/120 AS AverageMark
FROM Student, Module, Grade
WHERE Student.ID = Grade.ID
AND Module.Code = Grade.Code
AND YearTaken IN (2,3)
AND Year = 3
GROUP BY Year, Student.ID, First, Last
```

Information for Other Students

- Other students are easier than finalists
  - We just need to average their marks where YearTaken and Year are the same
  - As before we need the ID, Name, and Year for ordering

```
SELECT Year, Student.ID, Last, First,
SUM(Mark*Credits)/120 AS AverageMark
FROM Student, Module, Grade
WHERE Student.ID = Grade.ID
AND Module.Code = Grade.Code
AND YearTaken = Year
AND Year IN (1,2)
GROUP BY Year, Student.ID, First, Last
```
The Final Query

```sql
SELECT Year, Student.ID, Last, First,
       SUM((20*YearTaken/100)*Mark*Credits)/120 AS AverageMark
FROM Student, Module, Grade
WHERE Student.ID = Grade.ID AND Module.Code = Grade.Code
AND YearTaken IN (2,3) AND Year = 3
GROUP BY Year, Student.ID, First, Last
UNION
SELECT Year, Student.ID, Last, First,
       SUM(Mark*Credits)/120 AS AverageMark
FROM Student, Module, Grade
WHERE Student.ID = Grade.ID AND Module.Code = Grade.Code
AND YearTaken = Year AND Year IN (1,2)
GROUP BY Year, Student.ID, First, Last
ORDER BY Year desc, AverageMark desc, First, Last, ID
```

Next Lecture

- Missing Information
- NULLs and three-valued logic
- NULLs and the relational model
- OUTER JOINs
- Default values
- For more information
  - Ullman and Widom 6.1.5, 6.1.6, 6.3.8

Yet More SQL SELECT