Yet More SQL SELECT

Database Systems Lecture 10

Munawar, PhD

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*

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

ORDER BY Example

Grades

Name	Code	Mark
John	DBS	56
John	IAI	72
Mary	DBS	60
Mark	PR1	43
Mark	PR2	35
Jane	IAI	54

```
SELECT * FROM Grades
ORDER BY Mark
```

Name	Code	Mark
Mark	PR2	35
Mark	PR1	43
Jane	IAI	54
John	DBS	56
Mary	DBS	60
John	IAI	72

ORDER BY Example

```
SELECT * FROM Grades
  ORDER BY Code ASC,
           Mark DESC
```

Grades

Name	Code	Mark
John	DBS	56
John	IAI	72
Mary	DBS	60
Mark	PR1	43
Mark	PR2	35
Jane	IAI	54

Name	Code	Mark
Mary	DBS	60
John	DBS	56
John	IAI	72
Jane	IAI	54
Mark	PR1	43
Mark	PR2	35

Constants and Arithmetic

- As well as column names, you can select constants, compute arithmetic expressions and evaluate functions in a **SELECT** statement

```
SELECT Mark/100
  FROM Grades

SELECT
  Salary + Bonus
  FROM Employee

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

Aggregate Functions

Grades

Name	Code	Mark
John	DBS	56
John	IAI	72
Mary	DBS	60
Mark	PR1	43
Mark	PR2	35
Jane	IAI	54

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

Count
6

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

Total
320

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

Best
72

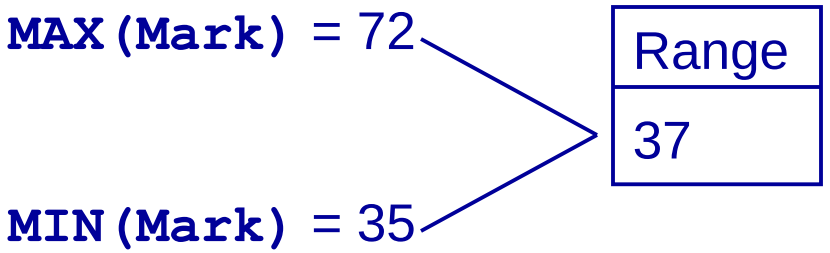
Aggregate Functions

- You can combine aggregate functions using arithmetic

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

Grades

Name	Code	Mark
John	DBS	56
John	IAI	72
Mary	DBS	60
Mark	PR1	43
Mark	PR2	35
Jane	IAI	54



Example

Modules

Code	Title	Credits
DBS	Database Sys.	10
GRP	Group Project	20
PRG	Programming	10

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

Grades

Name	Code	Mark
John	DBS	60
Mark	GRP	47
Mary	PRG	71

```
SELECT
  SUM(Mark*Credits)/SUM(Credits)
 FROM Modules, Grades
WHERE Modules.Code=Grades.Code
  AND Grades.Name = 'John'
```

GROUP BY

- Sometimes we want to apply aggregate functions to groups of rows
- Example, find the average mark of each student

- The **`GROUP BY`** clause does this

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

GROUP BY

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

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

GROUP BY

Grades

Name	Code	Mark
John	DBS	56
John	IAI	72
Mary	DBS	60
Mark	PR1	43
Mark	PR2	35
Jane	IAI	54

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

Name	Average
John	64
Mary	60
Mark	39
Jane	54

GROUP BY

Sales

Month	Department	Value
March	Fiction	20
March	Travel	30
March	Technical	40
April	Fiction	10
April	Fiction	30
April	Travel	25
April	Fiction	20
May	Fiction	20
May	Technical	50

- 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
```

Month	Department	Total
April	Fiction	60
April	Travel	25
March	Fiction	20
March	Technical	40
March	Travel	30
May	Fiction	20
May	Technical	50

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

Month	Department	Total
April	Fiction	60
March	Fiction	20
May	Fiction	20
March	Technical	40
May	Technical	50
April	Travel	25
March	Travel	30

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

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

Name	Average
John	64
Mary	60
Jane	54

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

Grades

Name	Code	Mark
Jane	IAI	52
John	DBS	56
John	IAI	72
Mark	PR1	43
Mark	PR2	35
Mary	DBS	60

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

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
```

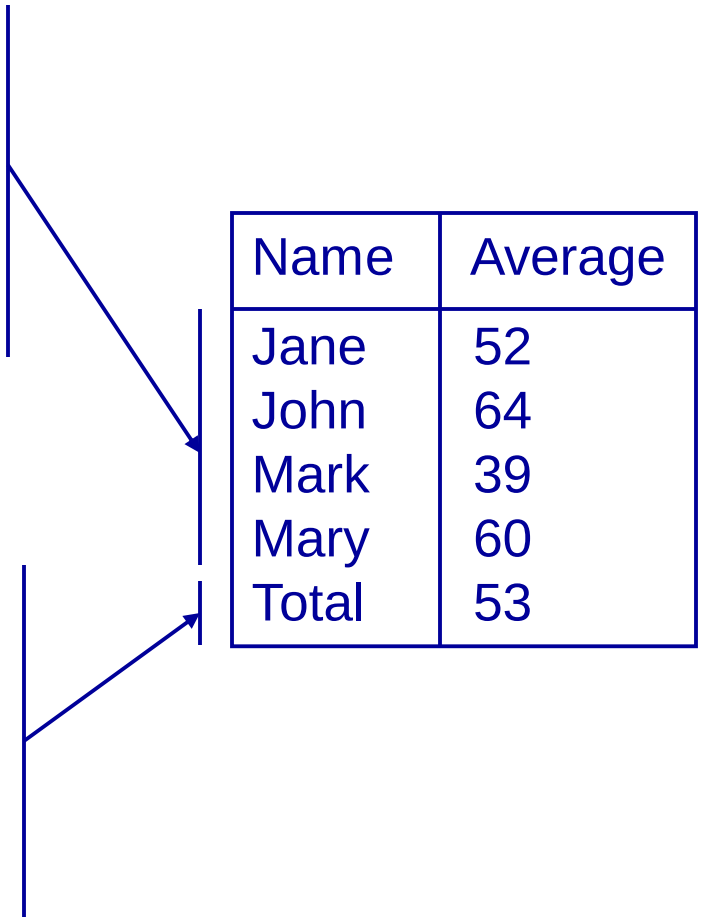
- Note - this has the same columns as the average by student

UNION

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

UNION

SELECT
'Total' as Name,
AVG(Mark) AS Average
FROM Grades
```



Name	Average
Jane	52
John	64
Mark	39
Mary	60
Total	53

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

Tables for the Example

Student

ID	First	Last	Year

Grade

ID	Code	Mark	YearTaken

Module

Code	Title	Credits

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

Information for Finalists

```
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

Information for Other Students

```
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

```
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