#### **SQL SELECT III**

Database Systems Michael Pound

#### **Last Lecture**

 Find a list of names of any students who are enrolled on at least one module alongside 'Evans'

| Enrolment |        |  |
|-----------|--------|--|
| sID       | mCode  |  |
| 1         | G52ADS |  |
| 2         | G52ADS |  |
| 5         | G51DBS |  |
| 5         | G51PRG |  |
| 5         | G51IAI |  |
| 4         | G52ADS |  |
| 6         | G51PRG |  |
| 6         | G51IAI |  |

#### **Last Lecture**

SELECT \* FROM Enrolment E1, Enrolment E2
WHERE E1.mCode = E2.mCode;

| sID | mCode  | sID | mCode  |
|-----|--------|-----|--------|
| 1   | G52ADS | 1   | G52ADS |
| 2   | G52ADS | 1   | G52ADS |
| 5   | G51DBS | 1   | G52ADS |
| 5   | G51PRG | 1   | G52ADS |
| 5   | G51IAI | 1   | G52ADS |
| 4   | G52ADS | 1   | G52ADS |
| 6   | G51PRG | 1   | G52ADS |
| 6   | G51IAI | 1   | G52ADS |
| 1   | G52ADS | 2   | G52ADS |
| 2   | G52ADS | 2   | G52ADS |

#### **Last Lecture**

SELECT \* FROM

| sID | mCode  | sID |
|-----|--------|-----|
| 1   | G52ADS | 4   |
| 2   | G52ADS | 4   |
| 4   | G52ADS | 4   |

#### **Last Lecture**

SELECT SID, sName FROM Student
WHERE SID IN
(SELECT DISTINCT E1.SID
FROM Enrolment E1 INNER JOIN Enrolment E2
USING (mCode)
WHERE E2.SID =
(SELECT SID FROM Student
WHERE SName = 'Evans'))
AND SID <> (SELECT SID FROM Student
WHERE SName = 'Evans');

### This Lecture

- More SQL SELECT
  - ORDER BY
  - Aggregate functions
  - GROUP BY and HAVING
  - UNION
- Further reading
  - The Manga Guide to Databases, Chapter 4
  - Database Systems, Chapter 6

#### **SQL SELECT Overview**

#### SELECT

[DISTINCT | ALL] <column-list>
FROM <table-names>
[WHERE <condition>]
[GROUP BY <column-list>]
[HAVING <condition>]
[ORDER BY <column-list>]

([] optional, | or)

#### **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

Grades

John DBS 56 John IAI 72 Mary DBS 60

 Mary
 DBS
 60

 James
 PR1
 43

 James
 PR2
 35

 Jane
 IAI
 54

Name | Code

Mark

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

#### **ORDER BY**

SELECT \* FROM Grades ORDER BY Mark

#### Grades

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

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

#### **ORDER BY**

SELECT \* FROM Grades ORDER BY Code ASC,

Mark DESC

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

# **Constants and Arithmetic**

- As well as columns, a SELECT statement can also be used to
  - · Select constants
  - Compute arithmetic expressions
  - · Evaluate functions
- Often helpful to use an alias when dealing with expressions or functions

SELECT Mark / 100 FROM Grades

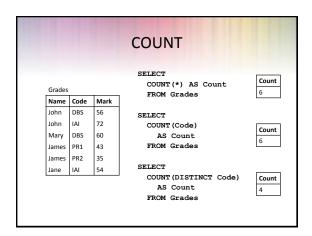
SELECT Salary + Bonus FROM Employee

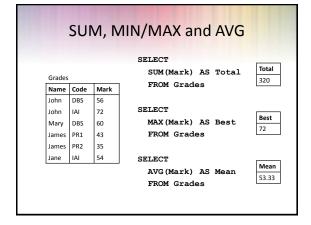
SELECT 1.175 \* Price AS 'Price inc. VAT' FROM Products

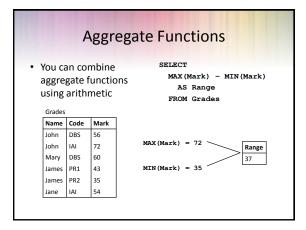
SELECT 'Constant' AS Text
FROM

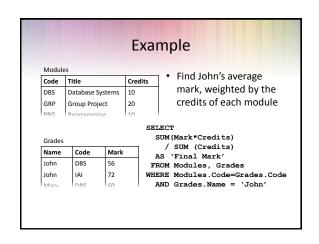
### **Aggregate Functions**

- Aggregate functions compute summaries of data in a table
  - Most aggregate functions (except COUNT (\*)) work on a single column of numerical data
- Again, it's best to use an alias to name the result
- Aggregate functions
  - COUNT: The number of rows
  - **SUM**: The sum of the entries in the column
  - AVG: The average entry in a column
  - MIN, MAX: The minimum and maximum entries in a column









# GROUP BY

- Sometimes we want to apply aggregate functions to groups of rows
- Example, find the average mark of each student individually
- The GROUP BY clause achieves this

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

#### **GROUP BY**

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

- Every entry in <cols1> should be in <cols2>, be a constant, or be an aggregate function
- You can have WHERE and 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   |
| James | PR1  | 43   |
| James | PR2  | 35   |
| Jane  | IAI  | 54   |

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

| Name  | Average |
|-------|---------|
| John  | 64      |
| Mary  | 60      |
| James | 39      |
| Jane  | 54      |

#### **GROUP BY**

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

#### **GROUP BY Rules**

- GROUP BY works slightly differently in MySQL than in other DBMSs.
- Usually, every column you name in your SELECT statement, must also appear in your GROUP BY clause. Apart from those in Aggregate functions.
- · For example:

SELECT ID, Name, AVG(Mark) FROM Students GROUP BY ID, Name

#### **GROUP BY Rules**

- In MySQL, for convenience, you are allowed to break this rule.
- You are allowed to GROUP BY a column that won't appear in the output table
- Despite this, you should follow the ISO standard where possible
  - Avoids problems if you use a different DBMS in the future
  - Can lead to peculiar output where multiple values get output as one

#### **GROUP BY Rules**

 The MySQL extension means you do not need to GROUP BY every column you're SELECTing. It also means you don't have to SELECT a column even if it's in your GROUP BY clause:

SELECT artID, artName, AVG(cdPrice) FROM Artist NATURAL JOIN CD GROUP BY artID;

#### **GROUP BY Rules**

 Be careful though, relaxed rules means you might get peculiar output if you're not careful:

SELECT cdTitle, AVG(cdPRICE)
FROM Artist NATURAL JOIN CD
GROUP BY artID;

| cdTitle                   | AVG(cdPrice) |
|---------------------------|--------------|
| For Lack of a Better Name | 11.49        |
| Version                   | 9.99         |
| The Resistance            | 10.99        |

#### **GROUP BY Rules**

What's the best way? Instead of:
 SELECT artName, AVG(cdPrice)
 FROM Artist NATURAL JOIN CD
 GROUP BY artID

Try:

SELECT artName, Average
FROM (SELECT artID, artName,
AVG(cdPrice) AS Average
FROM Artist NATURAL JOIN CD
GROUP BY artID, artName) AS SubTable;

#### **HAVING**

- HAVING is like a WHERE clause, except that it only 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, so cannot make use of aggregate functions
- HAVING refers to the groups of rows, and so cannot use columns which are not in the GROUP BY or an aggregate function
- Think of a query being processed as follows:
  - · Tables are joined
  - where clauses
  - GROUP BY clauses and aggregates
  - Column selection
  - HAVING clauses
  - · ORDER BY

#### UNION

- UNION, INTERSECT and EXCEPT
  - These treat the tables as sets and are the usual set operators of union, intersection and difference
  - We'll be concentrating on UNION
- They all combine the results from two select statements
- The results of the two selects should have the same columns and data types

#### UNION

Grades Name Code Mark Jane DBS 56 John John IAI 72 James PR1 43 James PR2 35 DBS

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

#### UNION

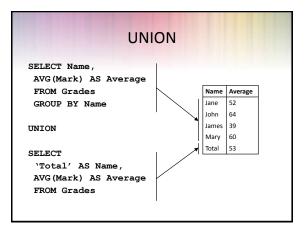
- The average for each student:
- The average overall

#### SELECT

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

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

 Note - this has the same columns as average by student



## Final SELECT 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 of the number of credits each module is worth
  - Produced by a single query

# Student | ID | First | Last | Year | | Grade | ID | Code | Mark | YearTaken | | Module | | Code | Title | Credits |

#### **Getting Started**

- Finalists should be treated differently to other years
  - Write one SELECT for the finalists
  - finalistsWrite a second SELECT for the first and second
  - Join the results using a UNION

vears

<QUERY FOR FINALISTS>

UNION

<QUERY FOR OTHERS>

#### **Table Joins**

- Both 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
  - Because we're practicing, we're going to use a standard CROSS JOIN and 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 must retrieve
  - Computed average mark, weighted 40-60 across years 2 and 3
  - First year marks must be ignored
  - The ID, Name and Year are needed as they are used for ordering
- · The average is difficult
  - We don't have any statements 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 the same 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 Others

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

#### Information for Others

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

SUM:((20\*YearTaken)/100) 'Mark\*Credits)/120 AS AverageMark
FRON 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, Last, First
UNION
SELECT Year, Student.ID, Last, First, SUM(Mark\*Credits)/120 AS AverageMark
FRON Student. Module. Grade

FROM Student, Module, Grade
WHERE Student.ID = Grade.ID AND Module.Code = Grade.Code

whose Student. ID = Grade.ID and module.Code = Grade.Co AND YearTaken = Year AND Year IN (1,2)

SELECT Year, Student.ID, Last, First,

GROUP BY Year, Student.ID, Last, First

ORDER BY Year desc, AverageMark desc, Last, First, ID

#### **Example Output** Student.ID Last AverageMark 11014456 Andrews John 81 11013891 Smith Mary 11014012 76 Jones Steven 11013204 76 Brown Amy 74 11014919 Robinson Paul Edwards Robert 73 11027871--Green 11024298 Hall David 43 11024826 40 Wood James 11027621 39 Clarke Stewart 11024978 Sarah Wilson 36 34 11026563 Matthew 11027625 Williams Paul 31

#### **Next Lecture**

- PHP
  - Variables
  - Arrays
  - IF...ELSE statements
  - Loops
  - Connecting to MySQL
- Further reading
  - W3Schools online tutorials at http://www.w3schools.com/php/