Structured Query Language (SQL) - 4

Databases and Interfaces

Matthew Pike & Linlin Shen

GROUP BY

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;
```

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

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

- 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 corresponding data types

UNION

Grades

Name	Code	Mark
Jane	IAI	54
John	DBS	56
John	IAI	72
James	PR1	43
James	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;
```

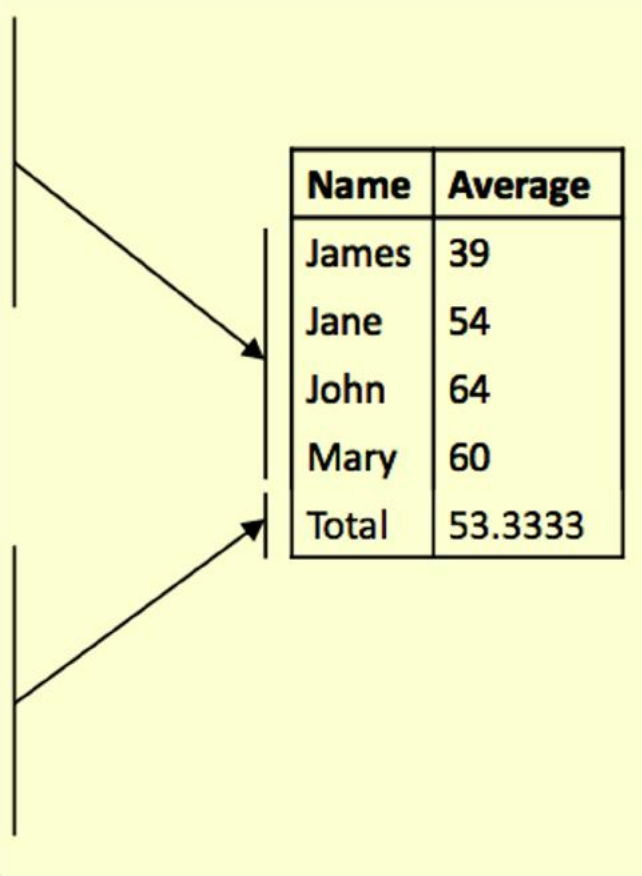
- The average overall

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

- Note - this has the same columns as 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
James	39
Jane	54
John	64
Mary	60
Total	53.3333

JOINS

Joins

- JOINs can be used to combine tables in a SELECT query
  - There are numerous types of JOIN
    - CROSS JOIN
    - INNER JOIN
    - NATURAL JOIN
    - OUTER JOIN

- A CROSS JOIN B
  - Returns all pairs of rows from A and B, the same as Cartesian Product
- A INNER JOIN B
  - Returns pairs of rows satisfying a condition
- A NATURAL JOIN B
  - Returns pairs of rows with common values in identically named columns
- A OUTER JOIN B
  - Returns pairs of rows satisfying a condition (as INNER JOIN), BUT ALSO handles NULLS

CROSS JOIN

```
SELECT * FROM
    A CROSS JOIN B
```

- Is the same as

```
SELECT * FROM A, B
```

- Usually best to use a **WHERE** clause to avoid huge result sets
  - Without a **WHERE** clause, the number of rows produced will be equal to the number of rows in A multiplied by the number of rows in B.

CROSS JOIN

Student

ID	Name
123	John
124	Mary
125	Mark
126	Jane

Enrolment

ID	Code
123	DBS
124	PRG
124	DBS
126	PRG

```
SELECT * FROM
   Student CROSS
   JOIN Enrolment
```

ID	Name	ID	Code
123	John	123	DBS
124	Mary	123	DBS
125	Mark	123	DBS
126	Jane	123	DBS
123	John	124	PRG
124	Mary	124	PRG
125	Mark	124	PRG
126	Jane	124	PRG
123	John	124	DBS
124	Mary	124	DBS

Alternative to CROSS JOIN: SELECT from Multiple Tables

- Often you need to combine information from two or more tables
- You can produce the effect of a Cartesian product using:

```
SELECT * FROM
Table1, Table2
```

- If the tables have columns with the same name, ambiguity will result
- This can be resolved by referencing columns with the table name:

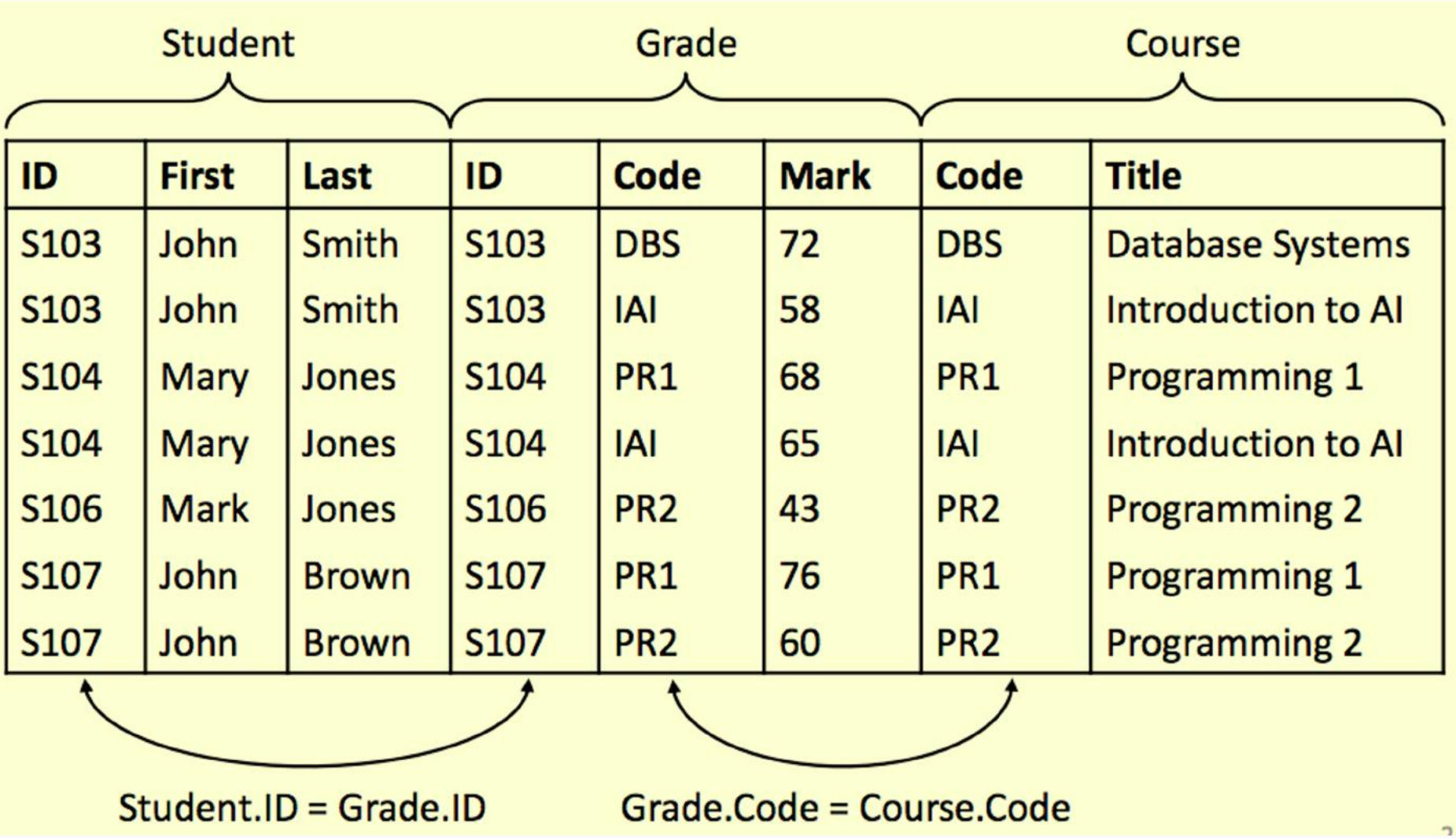
```
TableName.ColumnName
```

SELECT from Multiple Tables

- When selecting from multiple tables, it is almost always best to use a **WHERE** clause to find common values

```
SELECT *
FROM
   Student, Grade, Course
WHERE
   Student.ID = Grade.ID
AND
   Course.Code = Grade.Code
```

SELECT from Multiple Tables



INNER JOIN

- **`INNER JOIN`** specifies a condition that pairs of rows must satisfy

```
SELECT *
   FROM A INNER JOIN B
   ON condition
```

- Can also use a **`USING`** clause that will output rows with equal values in the specified columns
- **`SELECT * FROM A INNER JOIN B USING (col1, col2)`**
- **`col1`** and **`col2`** must appear in both A and B

INNER JOIN

Buyer

Name	Budget
Smith	100,000
Jones	150,000
Green	80,000

Property

Address	Price
15 High Street	85,000
12 Queen Street	125,000
87 Oak Lane	175,000

```
SELECT * FROM
    Buyer INNER JOIN
    Property ON
    Price <= Budget
```

Name	Budget	Address	Price
Smith	100,000	15 High Street	85,000
Jones	150,000	15 High Street	85,000
Jones	150,000	12 Queen Street	125,000

NATURAL JOIN

```
SELECT * FROM A NATURAL
JOIN B
```

- Is the same as

```
SELECT A.Col1, A.Col2,
    ... , A.Coln, [and all
    other columns except
    for B.Col1,...,B.Coln]
FROM A, B
    WHERE A.Col1 = B.Col1
    AND ...
    AND A.Coln = B.Coln
```

- A **`NATURAL JOIN`** is effectively a special case of an **`INNER JOIN`** where the **`USING`** clause has specified all identically named columns
- It can be written as
  - A ⋈ B
- and used in relational algebra expressions

NATURAL JOIN

Student (S)

ID	Name
123	John
124	Mary
125	Mark
126	Jane

Enrolment (E)

ID	Code
123	DBS
124	PRG
124	DBS
126	PRG

```
SELECT * FROM
    Student NATURAL JOIN
Enrolment;
```

ID	Name	Code
123	John	DBS
124	Mary	PRG
124	Mary	DBS
126	Jane	PRG

Outer Joins

- When we take the join of two relations we match up tuples which share values
  - Some tuples have no match, and are 'lost'
  - These are called 'dangles'
- Outer joins include dangles in the result and use NULLs to fill in the blanks
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN

Example: Inner Join

Student

ID	Name
123	John
124	Mary
125	Mark
126	Jane

Enrolment

ID	Code	Mark
123	DBS	60
124	PRG	70
125	DBS	50
128	DBS	80

← Dangles

Student INNER JOIN Enrolment ON Student.ID = Enrolment.ID

Example: Inner Join

Student

ID	Name
123	John
124	Mary
125	Mark
126	Jane

Enrolment

ID	Code	Mark
123	DBS	60
124	PRG	70
125	DBS	50
128	DBS	80

← Dangles

Student INNER JOIN Enrolment ON Student.ID = Enrolment.ID

ID	Name	ID	Code	Mark
123	John	123	DBS	60
124	Mary	124	PRG	70
125	Mark	125	DBS	50

Outer Join Syntax

```
SELECT cols
   FROM table1 type-OUTER-JOIN table2
   ON condition
```

Where type is one of LEFT, RIGHT or FULL

Example:

```
SELECT * FROM
   Student LEFT OUTER JOIN Enrolment
   ON Student.ID = Enrolment.ID;
```

Example: Left Outer Join

Student

ID	Name
123	John
124	Mary
125	Mark
126	Jane

Enrolment

ID	Code	Mark
123	DBS	60
124	PRG	70
125	DBS	50
128	DBS	80

← Dangles

Student LEFT OUTER JOIN Enrolment ON ...

ID	Name	ID	Code	Mark
123	John	123	DBS	60
124	Mary	124	PRG	70
125	Mark	125	DBS	50
126	Jane	NULL	NULL	NULL

Example: Right Outer Join

Student

ID	Name
123	John
124	Mary
125	Mark
126	Jane

Enrolment

ID	Code	Mark
123	DBS	60
124	PRG	70
125	DBS	50
128	DBS	80

Dangles

Student RIGHT OUTER JOIN Enrolment ON ...

Example: Right Outer Join

Student

ID	Name
123	John
124	Mary
125	Mark
126	Jane

Enrolment

ID	Code	Mark
123	DBS	60
124	PRG	70
125	DBS	50
128	DBS	80

← Dangles

Student RIGHT OUTER JOIN Enrolment ON ...

ID	Name	ID	Code	Mark
123	John	123	DBS	60
124	Mary	124	PRG	70
125	Mark	125	DBS	50
NULL	NULL	128	DBS	80

SQLite Support

- SQLite does not support `RIGHT OUTER JOIN`
  - https://www.sqlite.org/omitted.html
- We can simulate a `RIGHT OUTER JOIN` by simply switching the order of the two tables that are used in a `LEFT OUTER JOIN`
  - `SELECT * FROM A RIGHT OUTER JOIN B`
  - Becomes:
  - `SELECT * FROM B LEFT OUTER JOIN A`

Examples (For Reference)

WHERE

WHERE Examples

Given the table:

Grade

ID	Code	Mark
S103	DBS	72
S103	IAI	58
S104	PR1	68
S104	IAI	65
S106	PR2	43
S107	PR1	76
S107	PR2	60
S107	IAI	35

- Write an SQL query to find a list of the students IDs for both the IAI and PR2 module

ID
S103
S104
S106
S107
S107

Solution

```
SELECT ID FROM Grade WHERE
      (Code = 'IAI'
    OR Code = 'PR2');
```

Example: SELECT from Multiple Tables

Examples

Student

sID	sName	sAddress	sYear
1	Smith	5 Arnold Close	2
2	Brooks	7 Holly Avenue	2
3	Anderson	15 Main Street	3
4	Evans	Flat 1a, High Street	2
5	Harrison	Newark Hall	1
6	Jones	Southwell Hall	1

Module

mCode	mCredits	mTitle
G51DBS	10	Database Systems
G51PRG	20	Programming
G51IAI	10	Artificial Intelligence
G52ADS	10	Algorithms

Enrolment

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

Examples

Write SQL statements to do the following:

1. Produce a list of all student names and all their enrolments (module codes)
2. Find a list of module titles being taken by the student named “Harrison”
3. Find a list of module codes and titles for all modules currently being taken by first year students

Solutions

1. `SELECT sName, mCode FROM Student, Enrolment WHERE Student.sID = Enrolment.sID;`

1. `SELECT mTitle FROM Module, Student, Enrolment WHERE (Module.mCode = Enrolment.mCode) AND (Student.sID = Enrolment.sID) AND Student.sName = "Harrison";`

1. `SELECT Module.mCode, mTitle FROM Enrolment, Module, Student WHERE (Module.mCode = Enrolment.mCode) AND (Student.sID = Enrolment.sID) AND sYear = 1;`

OUTER JOIN Examples

Example

- Sometimes an outer join is the most practical approach. We may encounter NULL values, but may still wish to see the existing information

- For students graduating in absentia, find a list of all student IDs, names, addresses, phone numbers and their final degree classifications.

Example

Student

ID	Name	aID	pID	Grad
123	John	12	22	C
124	Mary	23	90	A
125	Mark	19	NULL	A
126	Jane	14	17	C
127	Sam	NULL	101	A

Phone

pID	pNumber	pMobile
17	1111111	07856232411
22	2222222	07843223421
90	3333333	07155338654
101	4444444	07213559864

Degree

ID	Classification
123	1
124	2:1
125	2:2
126	2:1
127	3

Address

aID	aStreet	aTown	aPostcode
12	5 Arnold Close	Nottingham	NG12 1DD
14	17 Derby Road	Nottingham	NG7 4FG
19	1 Main Street	Derby	DE1 5FS
23	7 Holly Avenue	Nottingham	NG6 7AR

Example: INNER JOINs

- An Inner Join with Student and Address will ignore Student 127, who doesn't have an address record
- An Inner Join with Student and Phone will ignore student 125, who doesn't have a phone record

Student

ID	Name	aID	pID	Grad
123	John	12	22	C
124	Mary	23	90	A
125	Mark	19	NULL	A
126	Jane	14	17	C
127	Sam	NULL	101	A

Example

```
SELECT ...
   FROM Student LEFT OUTER JOIN Phone
        ON Student.pID = Phone.pID
   ...
```

Student

ID	Name	aID	pID	Grad
123	John	12	22	C
124	Mary	23	90	A
125	Mark	19	NULL	A
126	Jane	14	17	C
127	Sam	NULL	101	A

Phone

pID	pNumber	pMobile
22	2222222	07843223421
90	3333333	07155338654
NULL	NULL	NULL
17	1111111	07856232411
101	4444444	07213559864

Example

```
SELECT ...
   FROM Student LEFT OUTER JOIN Phone
        ON Student.pID = Phone.pID
   LEFT OUTER JOIN Address
        ON Student.aID = Address.aID
   ...
```

Student / Phone / Address

ID	Name	aID	pID	Grad	pNumber	pMobile	aStreet	aTown	aPostcode
123	John	12	22	C	2222222	07843223421	5 Arnold...	Notts	NG12 1DD
124	Mary	23	90	A	3333333	07155338654	7 Holly...	Notts	NG6 7AR
125	Mark	19	NULL	A	NULL	NULL	1 Main...	Derby	DE1 5FS
126	Jane	14	17	C	1111111	07856232411	17 Derby...	Notts	NG7 4FG
127	Sam	NULL	101	A	4444444	07213559864	NULL	NULL	NULL

Example

```
SELECT ID, Name, aStreet, aTown, aPostcode,
    pNumber, Classification
FROM Student
    LEFT OUTER JOIN Phone
        ON Student.pID = Phone.pID
    LEFT OUTER JOIN Address
        ON Student.aID = Address.aID
INNER JOIN Degree
    ON Student.ID = Degree.ID
    WHERE Grad = 'A';
```

Example

ID	Name	aStreet	aTown	aPostcode	pNumber	Classification
124	Mary	7 Holly Avenue	Nottingham	NG6 7AR	3333333	2:1
125	Mark	1 Main Street	Derby	DE1 5FS	NULL	2:2
127	Sam	NULL	NULL	NULL	4444444	3

The records for students 125 and 127 have been preserved despite missing information

Examiners Report Example

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 averages
- We want the results:
  - Sorted by year (desc), then by average mark (high to low) then by 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

Example Output

Year	Student.ID	Last	First	AverageMark
3	11014456	Andrews	John	81
3	11013891	Smith	Mary	78
3	11014012	Brown	Amy	76
3	11013204	Jones	Steven	76
3	11014919	Robinson	Paul	74
2	[illegible]	Edwards	Robert	73
⋮	⋮	⋮	⋮	⋮
1	11027871	Green	[illegible]	[illegible]
1	11024298	Hall	David	43
1	11024826	Wood	James	40
1	11027621	Clarke	Stewart	39
1	11024978	Wilson	Sarah	36
1	11026563	Taylor	Matthew	34
1	11027625	Williams	Paul	31

Tables for the Example

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
  - Write a second SELECT for the first and second years
  - Merge the results using a UNION

```
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
  - But because we're practicing, we're going to use a standard CROSS JOIN and WHERE clause
  - Exercise:
    - repeat the query using natural join

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

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
...
```

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 I**D, Name** and **Year** for ordering

The Query so Far

```
...
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

```
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, Last, First
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, Last, First

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