Xi'an Jiaotong-Liverpool University
西交利物浦大學

SQL Select II

Jianjun Chen (Jianjun.Chen@xjtlu.edu.cn)

Contents

- Joins
  - Cross, Inner, Natural, Outer
- ORDER BY to produce ordered output
- Aggregate functions
  - MIN, MAX, SUM, AVG, COUNT
- GROUP BY and HAVING

Joins

Cross, Inner, Natural, Outer

Joins

- Joins can be used to combine tables in a `SELECT` query
- **`CROSS JOIN`**: same as Cartesian product (taught in the previous lecture).
- **`INNER JOIN`**: returns pairs of rows satisfying a condition.
- **`NATURAL JOIN`**: returns pairs of rows with common values in identically named columns.
- **`OUTER JOIN`**: returns pairs of rows satisfying a condition (as `INNER JOIN`), BUT ALSO handles `NULL`s.

CROSS JOIN

- Syntax:

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

- Same as:

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

- Usually needs **WHERE** to filter out unrelated tuples

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: Example

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

INNER JOIN: Practice

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 INNER JOIN Enrolment
   USING (ID)
```

?

INNER JOIN: Practice

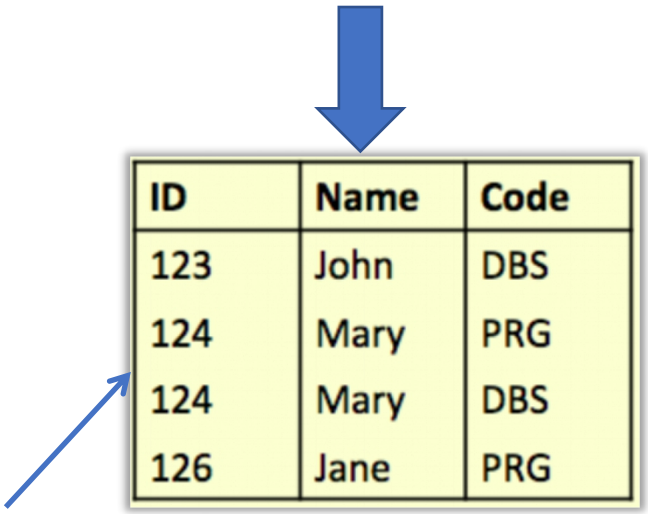
Student

ID	Name
123	John
124	Mary
125	Mark
126	Jane

Enrolment

ID	Code
123	DBS
124	PRG
124	DBS
126	PRG

```
SELECT * FROM
   Student INNER JOIN Enrolment
   USING (ID)
```



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

A single ID row will be outputted representing the equal values from both Student.ID and Enrolment.ID

NATURAL JOIN

- Syntax:

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

- A **NATURAL JOIN** is effectively a special case of an **INNER JOIN** where the **USING** clause has specified all identically named columns.

- Same as the A ⋈B in relational algebra.

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

JOINs vs WHERE Clauses

- Inner/Natural JOINs are not absolutely necessary.
- You can obtain the same results by selecting from multiple tables and using appropriate WHERE clauses.
- Should you use them?
  - They often lead to concise and elegant queries.
  - NATURAL JOINs are extremely common.
  - Support for JOINs can vary between DBMSs.

OUTER JOIN

- Syntax

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

- Where type is one of **LEFT**, **RIGHT** or **FULL**.

Example: Left Outer Join

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

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 ...

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

Example: Full 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 FULL 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
NULL	NULL	128	DBS	80

Full Outer Join in MySQL

Only Left and Right outer joins are supported in MySQL. If you really want a FULL outer join:

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

Can be achieved using:

```
(SELECT * FROM Student LEFT OUTER JOIN
      Enrolment ON Student.ID = Enrolment.ID)
UNION
(SELECT * FROM Student RIGHT OUTER JOIN
      Enrolment ON Student.ID = Enrolment.ID);
```

Why Using Outer Joins?

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

- The next few slides will consider this problem:

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

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

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

Problems with 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

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

Phone

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

For students graduating in absentia, find a list of all student IDs, names, addresses, phone numbers and their final degree classifications. What is your solution?

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

```
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

Next table to combine 

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

```
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

Next table to combine 

Degree

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

Solution Using OURTER JOIN

```
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';
```

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

Degree

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

Solution Using OURTER JOIN

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

Address

```
___ LEFT OUTER JOIN ___
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

Degree

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

```
___ INNER JOIN ___
    ON Student.ID = Degree.ID
    WHERE Grad = 'A';
```

Final Result Using OUTER JOIN

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

Order By

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

([] optional, | or)

ORDER BY

- Syntax:

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

- 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 should not choose to order by a column that is not in the result.
  - It is possible though:

```
SELECT y / 100 AS y2 FROM a ORDER BY y DESC;
```

ORDER BY: Example 1

Grades

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

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

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

ORDER BY: Example 2

Grades

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

```
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

Aggregate Functions

sum, avg, count, min, max

Aggregate Functions

- We learned that it is possible to put arithmetic expressions in SELECT.

- You can also use aggregate functions to compute summaries of data in a table.
  - **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/maximum entries in a column

- Most aggregate functions (except COUNT (*)) work on a single column of numerical data.

COUNT

Grades

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

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

?

```
SELECT
   COUNT(Code) AS Count
   FROM Grades;
```

?

```
SELECT
   COUNT(DISTINCT Code)
      AS Count
   FROM Grades;
```

?

COUNT

Grades

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

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

Count
6

```
SELECT
   COUNT(Code) AS Count
   FROM Grades;
```

Count
6

```
SELECT
   COUNT(DISTINCT Code)
      AS Count
   FROM Grades;
```

Count
4

SUM, MIN/MAX and AVG

Grades

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

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

?

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

?

```
SELECT
   AVG(Mark) AS Mean
   FROM Grades;
```

?

SUM, MIN/MAX and AVG

Grades

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

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

Total
320

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

Best
72

```
SELECT
   AVG(Mark) AS Mean
   FROM Grades;
```

Mean
53.33

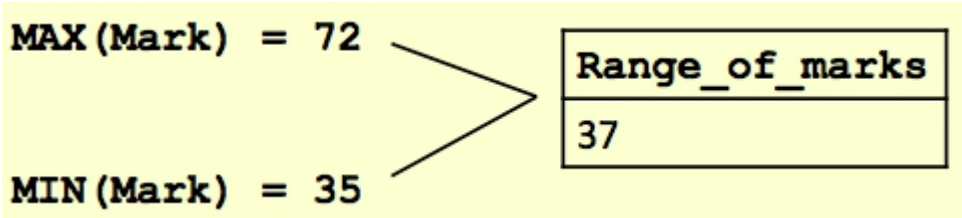
Combining Aggregate Functions

- You can combine aggregate functions using arithmetic

Grades

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

```
SELECT
  MAX(Mark) - MIN(Mark)
     AS Range_of_marks
  FROM Grades;
```



Combining AF: Example

Modules

Code	Title	Credits
DBS	Database Systems	10
IAI	Introduction to AI	20
PRG	Programming	10

Grades

Name	Code	Mark
John	DBS	56
John	IAI	72
Mary	DBS	60

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

?

Combining AF: Example

Modules

Code	Title	Credits
DBS	Database Systems	10
IAI	Introduction to AI	20
PRG	Programming	10

Grades

Name	Code	Mark
John	DBS	56
John	IAI	72
Mary	DBS	60

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

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

Aggregate Functions

- You cannot use aggregate functions in the `WHERE` clause.

```
SELECT * FROM staff
      WHERE age > AVG(age);
```

Common mistake

- But you can use them in the subqueries in the `WHERE` clause.

```
SELECT * FROM staff WHERE
age > (SELECT AVG(age) FROM staff);
```

Aggregate Functions

- The use of aggregate functions leads to all rows after the first row being truncated.

id	name	age
1	Annie	28
2	Brown	31
3	George	26
4	Kathy	21
5	Annie	21

```
SELECT *, AVG(age)
FROM staff;
```

id	name	age	AVG(age)
1	Annie	28	25.4000

GROUP BY and HAVING

Used with aggregate functions

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 column_set1 FROM tables
    WHERE predicate
    GROUP BY column_set2;
```

- Every entry in ‘column_set2’ should be in ‘column_set1’, be a constant, or be an aggregate function

GROUP BY: Example

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: Example

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	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

GROUP BY: Example

```
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

Same results, but produced in a different order

Group By

- Like aggregate functions, `GROUP BY` also truncates all rows after the first row for each sub-group.

student_id	module	marks
1	CPT103	78
1	CPT111	81
2	CPT103	66
2	CPT111	63
2	CPT105	71

```
SELECT * FROM grades
GROUP BY student_id;
```

student_id	module	marks
1	CPT103	78
2	CPT103	66

- Group by -> Aggregate functions -> truncate rows.
- The code above is just showing how `GROUP BY` works and is not really useful in real life.

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

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
    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 or aggregate functions that does not exist after the step of column selection (see below).

- Think of a query being processed as follows:
    1. Tables are joined
    2. WHERE clauses
    3. GROUP BY clauses and aggregates
    4. Column selection
    5. HAVING clauses
    6. ORDER BY