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

SQL Select III

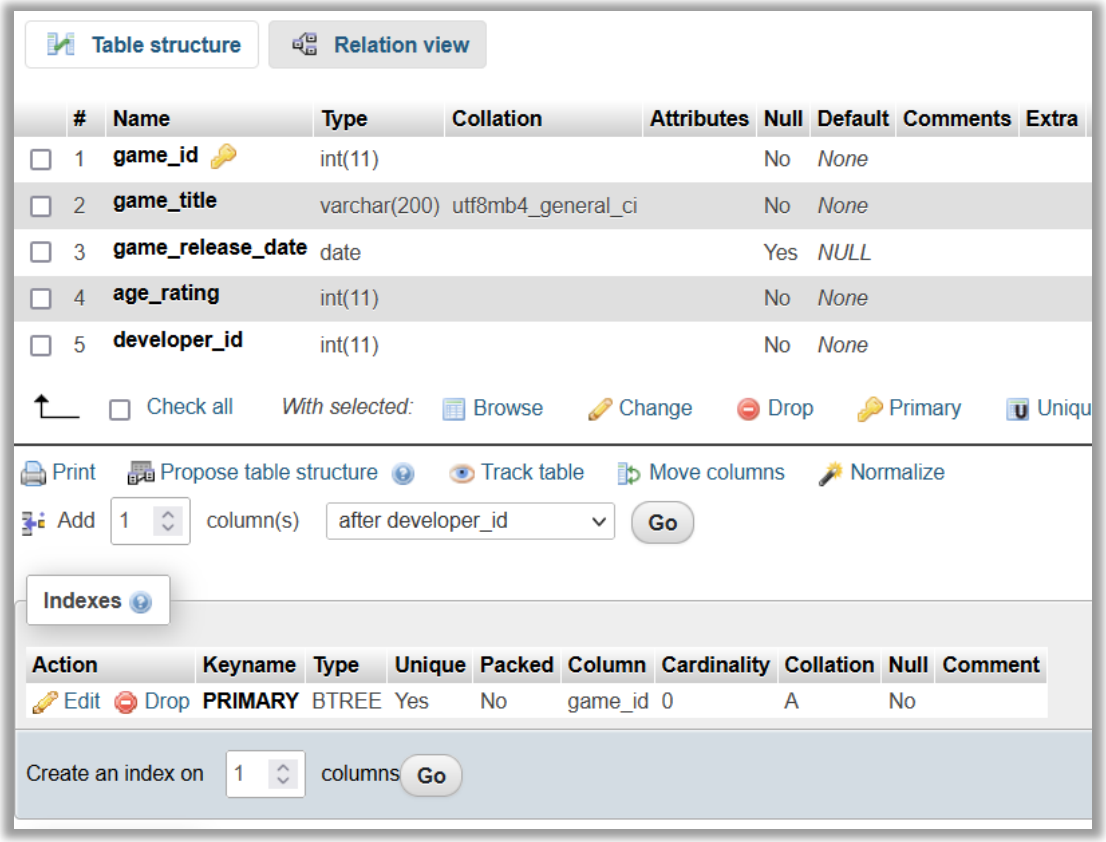
Jianjun Chen (Jianjun.Chen@xjtlu.edu.cn)

Content

- Index
- Set Operations
- Missing information:
  - Dealing with nulls.
  - Making use of default values.

Index

- What are indices and why are they needed:
- https://www.youtube.com/watch?v=fsG1XaZEa78



Index

- An index helps to speed up select queries and where clauses.
- but it slows down data input, with the update and the insert statements.

- Primary key and unique key affects the availability of certain functions of SQL.
- For example:
  - _rowid is a special column that is only accessible when you have created a primary key or a unique key.

Set Operations

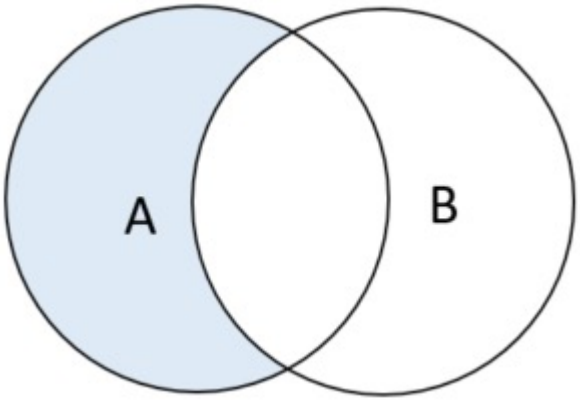
UNION, INTERSECT and EXCEPT

SET operations

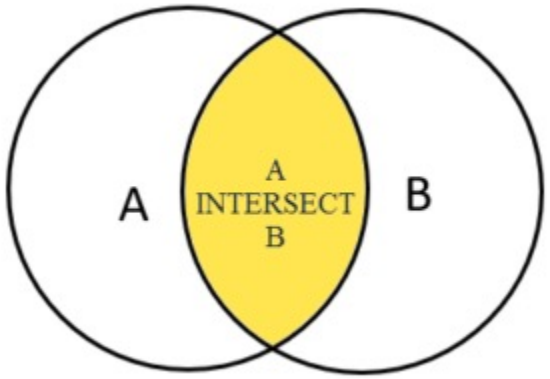
- `UNION`, `INTERSECT` and `EXCEPT`
  - Treat the tables as sets and are the usual set operators of union, intersection and difference
  - Only `UNION` is supported in `MySQL`. The other two can be simulated with subqueries.

- 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 compatible.



A EXCEPT B



A INTERSECT B

A UNION B

UNION: Example

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

Grades

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

UNION

1. The average for each student:

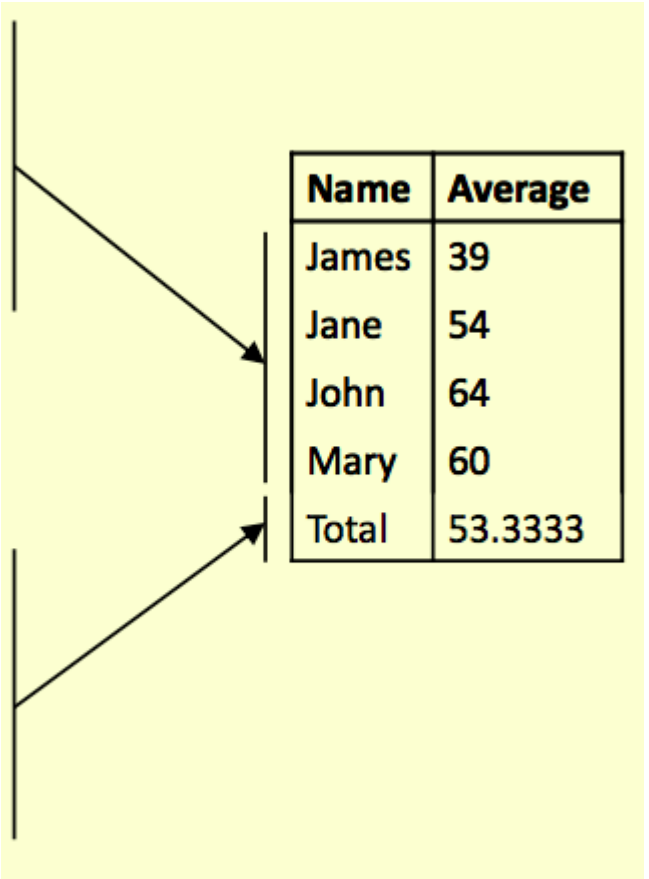
```
SELECT Name, AVG(Mark) AS Average
   FROM Grades
   GROUP BY Name;
```

2. The average overall:

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

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

Missing Information

Dealing with nulls in SQL

Making use of default values

Missing Information

- Sometimes we don’t know what value an entry in a relation should have.
  - Case 1: We know that there is a value, but don’t know what it is.
  - Case 2: There is no value at all that makes any sense.

- Two main methods have been proposed to deal with this
  - `NULLs` can be used as markers to show that information is missing .
  - A default value can be used to represent the missing value.

NULL

- NULL Represents a state for an attribute that is currently unknown or is not applicable for this tuple.
  - NULLs are a way to deal with incomplete or exceptional data.
  - NULL is a placeholder for missing or unknown value of an attribute. It is not itself a value.
  - E.g. A new staff is just added, but hasn't been decided which branch he belongs to.

- Codd proposed to distinguish two types of NULLs:
  - **A-marks**: data Applicable but not known (for example, someone's age)
  - **I-marks**: data is Inapplicable (telephone number for someone who does not have a telephone, or spouse's name for someone who is not married)

Problems with NULLs

- Problems extending relational algebra operations to NULLs:
  - Selection operation: if we check tuples for "Mark > 40" and for some tuple Mark is NULL, do we include it?
  - Comparing tuples in two relations: are two tuples (with NULLs) and the same or not?

- Additional problems for SQL:
  - NULLs treated as duplicates?
  - Inclusion of NULLs in count, sum, average? If yes, how?
  - Arithmetic operations behaviours with argument NULL?

Theoretical Solutions

- Use three-valued logic instead of classical two-valued logic to evaluate conditions:
  - When there are no `NULL`s around, conditions evaluate to true or false, but if a `NULL` is involved, a condition might evaluate to the third value ('undefined', or 'unknown')

a	b	a OR b	a AND b	a == b
True	True	True	True	True
True	False	True	False	False
True	Unknown	True	Unknown	Unknown
False	True	True	False	False
False	False	False	False	True
False	Unknown	Unknown	False	Unknown
Unknown	True	True	Unknown	Unknown
Unknown	False	Unknown	False	Unknown
Unknown	Unknown	Unknown	Unknown	Unknown

SQL NULLs in Conditions

Employee

Name	Salary
John	25,000
Mark	15,000
Anne	20,000
Chris	NULL

```
SELECT * FROM Employee
    Where Salary > 15,000;
```

Name	Salary
John	25,000
Anne	20,000

WHERE clause of SQL SELECT uses three-valued logic: only tuples where the condition evaluates to true are returned.

Salary > 15,000 evaluates to 'unknown' on the last tuple – not included

SQL NULLs in Conditions

Employee

Name	Salary
John	25,000
Mark	15,000
Anne	20,000
Chris	NULL

```
SELECT * FROM Employee
    Where Salary > 15,000
       OR Name = 'Chris';
```

Name	Salary
John	25,000
Anne	20,000
Chris	NULL

`Salary > 15,000 OR Name = 'Chris'` is essentially **Unknown OR TRUE** on the last tuple:

a	b	a OR b
Unknown	True	True

SQL NULLs in Arithmetic

Employee

Name	Salary
John	25,000
Mark	15,000
Anne	20,000
Chris	NULL

```
SELECT Name,
  Salary * 0.05 AS Bonus
FROM Employee;
```

Name	Bonus
John	1,250
Mark	750
Anne	1,000
Chris	NULL

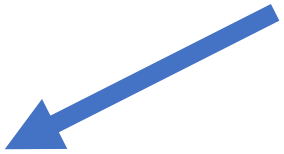
Arithmetic operations applied to NULLs result in NULLS

SQL NULLs in Aggregation

Employee

Name	Salary
John	25,000
Mark	15,000
Anne	20,000
Chris	NULL

```
SELECT
   AVG(Salary) AS Average,
   COUNT(Salary) AS Count,
   SUM(Salary) AS Sum
FROM Employee;
```



Average = 20,000
Count = 3
Sum = 60,000

Using `COUNT(*)` would give 4, even if the name of Chris is changed to `NULL`.

SQL NULLs in GROUP BY

Employee

Name	Salary
John	25,000
Mark	15,000
Anne	20,000
Jack	NULL
Sam	20,000
Chris	NULL

```
SELECT Salary,
       COUNT(Name) AS Count
       FROM Employee
       GROUP BY Salary;
```

Salary	Count
NULL	2
15,000	1
20,000	2
25,000	1

NULLs are treated as equivalents in GROUP BY clauses

SQL NULLs in ORDER BY

Employee

Name	Salary
John	25,000
Mark	15,000
Anne	20,000
Jack	NULL
Sam	20,000
Chris	NULL

```
SELECT *
   FROM Employee
   ORDER BY Salary;
```

Employee

Name	Salary
Chris	NULL
Jack	NULL
Mark	15,000
Anne	20,000
Sam	20,000
John	25,000

NULLs are considered and reported in ORDER BY clauses

Default Values

- Default values are an alternative to the use of NULLs
  - You can choose a value that makes no sense in normal circumstances.
  - These are actual values

```
age INT DEFAULT -1,
```

- Default values can have more meaning than NULLs. The followings are example default values of VARCHAR.
  - ‘none’
  - ‘unknown’
  - ‘not supplied’
  - ‘not applicable’

- Not all defaults represent missing information. It depends on the situation

Default Values: Example

- Default values are
  - "Unknown" for Name
  - -1 for Weight and Quantity
- -1 is used for Wgt and Qty as it is not sensible otherwise
- There are still problems:

```
UPDATE Parts SET
Quantity = Quantity + 5
```

Parts

ID	Name	Weight	Quantity
1	Nut	10	20
2	Bolt	15	-1
3	Nail	3	100
4	Pin	-1	30
5	Unknown	20	20
6	Screw	-1	-1
7	Brace	150	0

SQL Support

- SQL allows both NULLs and defaults:
  - A table to hold data on employees
  - All employees have a name
  - All employees have a salary (default 10000)
  - Some employees have phone numbers, if not we use NULLs

```
CREATE TABLE Employee
(
    Name VARCHAR(50) NOT NULL,
    Salary INT DEFAULT 10000 NOT NULL,
    Phone VARCHAR(15) NULL
);
```

SQL Support

- SQL allows you to insert NULLs:

```
INSERT INTO Employee VALUES
     ('John', 12000, NULL);
UPDATE Employee SET Phone = NULL
     WHERE Name = 'Mark';
```

- You can also check for NULLs:

```
SELECT Name FROM Employee
     WHERE Phone IS NULL;
SELECT Name FROM Employee
     WHERE Phone IS NOT NULL;
```

SQL: The Final Example

Increasing the difficulty

The 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 (third year) 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

Student

ID	First	Last	Year

Grade

ID	Code	Mark	YearTaken

Module

Code	Title	Credits

Example Output

Student

ID	First	Last	Year

Grade

ID	Code	Mark	YearTaken

Module

Code	Title	Credits

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
3	[illegible]	[illegible]	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

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 ID, Name and Year for ordering

```
...
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
```

```
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;
```


Questions?