SQL 3: Alias, Subqueries, Aggregate Functions & Grouping

Databases and Interfaces

Matthew Pike & Yuan Yao

University of Nottingham Ningbo China (UNNC)

Overview

This Lecture

- In this lecture we will cover:
  - Aliases - to make queries more readable
  - Subqueries - for more complex queries
  - Aggregate functions - to summarise data
  - `GROUP BY` - grouping data and applying aggregate functions

The Database Schema for this Lecture

```
CREATE TABLE Student(
    sID INTEGER PRIMARY KEY,
    firstName VARCHAR(20) NOT NULL,
    lastName VARCHAR(20) NOT NULL
);
```

```
CREATE TABLE Module(
    mCode CHAR(8) PRIMARY KEY,
    title VARCHAR(30) NOT NULL,
    credits INTEGER NOT NULL
);
```

```
CREATE TABLE Grade(
    sID INTEGER NOT NULL,
    mCode CHAR(8) NOT NULL,
    grade INTEGER NOT NULL,
    PRIMARY KEY (sID, mCode),
    FOREIGN KEY (sID)
        REFERENCES Student(sID),
    FOREIGN KEY (mCode)
        REFERENCES Module(mCode)
);
```

The Database Content for this Lecture

sID	firstName	lastName
1	John	Smith
2	Jane	Doe
3	Mary	Jones
4	David	Smith

Table 1: `Student` Table

mCode	title	credits
COMP1036	Fundamentals	20
COMP1048	Databases	10
COMP1038	Programming	20

Table 2: `Module` Table

sID	mCode	grade
1	COMP1036	35
1	COMP1048	50
2	COMP1048	65
2	COMP1038	70
3	COMP1036	35
3	COMP1038	65

Table 3: `Grade` Table

Using Aliases to Rename Columns and Tables

Aliases

- An alias is a temporary name for a table or column. It can be used to:
  - Make queries more readable
  - Shorten column names
  - Resolve ambiguous names
- Aliases are specified using the AS keyword
  - For example:
    - `SELECT column_name AS alias_name FROM table_name;`

Using Aliases to Rename Columns

We can use aliases to rename columns in a query. This is useful for:

- Making queries more readable
- Shortening or standardising column names
- Resolving ambiguous names - for example, when joining tables (covered in a next lecture)

```
SELECT
    sID AS 'ID',
    firstName AS 'First',
    lastName AS 'Last'
FROM
    Student;
```

ID	First	Last
1	John	Smith
2	Jane	Doe
3	Mary	Jones
4	David	Smith

Table 4: Rename Columns using Aliases

Using Aliases to Rename Tables

> **!** Alias in WHERE Clause
>
> You **cannot** create a column alias in a WHERE clause, because the column value might not yet be determined when the WHERE clause is executed.

```
SELECT
    s.sID AS 'ID',
    s.lastName AS 'Last'
FROM
    Student AS s
WHERE
    s.sID < 3;
```

ID	Last
1	Smith
2	Doe

Table 5: Associating student names with grades via a table Alias.

Subqueries

Subqueries in SQL

- A `SELECT` statement can be nested inside another `SELECT` statement.
  - The inner `SELECT` statement is called a **subquery**.
  - The outer `SELECT` statement is called a **main query** or **outer query**.
- Subqueries are useful when you need to:
  - Filter a table based on the results of another query;
  - Calculate a value based on the results of another query.
- Subqueries are specified using parentheses:
  - `SELECT * FROM table WHERE column [IN|EXISTS|=] (SELECT column FROM table);`

Subqueries in SQL: Example

- The subquery will return the IDs of students who have achieved >= 70 in a module.
- The outer query will then return the names of students with those IDs.

```
SELECT
    firstName, lastName
FROM Student
WHERE sID IN (
    SELECT sID
    FROM Grade
    WHERE grade >= 70
);
```

firstName	lastName
Jane	Doe

Table 6: Names of students who have achieved >= 70 in a module.

Subqueries Processing Order

- Subqueries are processed before the outer query. Nested subqueries are processed from the innermost subquery to the outermost subquery.
- The results of the subquery are stored in a temporary table, which is then used in the outer query.

```
SELECT grade
FROM Grade
WHERE mCode = (
    SELECT mCode
    FROM Module
    WHERE title =
    'Databases'
);
```

```
SELECT grade
FROM Grade
-- The result of the subquery
-- is used in the outer query
WHERE mCode = 'COMP1048';
```

Subqueries with Sets of Values

- A subquery will often return a set of values, not just a single value.
- The subquery can be used to filter a table based on a set of values.
- When handling sets of values, the following operators are used:
  - `IN` - returns true if the value is in the set.
  - `EXISTS` - returns true if the set is not empty.
  - `NOT IN` - returns true if the value is not in the set.
  - `NOT EXISTS` - returns true if the set is empty.
- The set of values can be specified using a subquery or a list of values:
  - `SELECT * FROM table WHERE column [IN|EXISTS] (SELECT column FROM table);`
  - `SELECT * FROM table WHERE column [IN|EXISTS] (value1, value2, ...);`

Using `IN` with Subqueries

> **Note**
>
> The `IN` operator is used to check if a value is in a set of values

```
SELECT title AS 'Module Title'
FROM Module
WHERE mCode IN (
    SELECT mCode
    FROM Grade
    WHERE grade >= 70
);
```

Module Title
Programming

Table 7: The names of modules in which a student has scored >= 70.

Using `NOT IN` with Subqueries

> **Note**
>
> The `NOT IN` operator is used to check if a value is not in a set of values

```sql
SELECT
    firstName AS 'First',
    lastName AS 'Last'
FROM Student
WHERE sID NOT IN (1, 2);
```

First	Last
Mary	Jones
David	Smith

Table 8: The names of students who do not have the IDs 1 or 2.

Using EXISTS with Subqueries

> **Note**
>
> The EXISTS operator checks whether a subquery returns any row.

```
SELECT
    firstName AS 'First',
    lastName AS 'Last'
FROM Student s
WHERE EXISTS (
    SELECT sID
    FROM Grade
    WHERE sID = s.sID
);
```

First	Last
John	Smith
Jane	Doe
Mary	Jones

Table 9: Get the names of students with at least one grade.

Using `NOT EXISTS` with Subqueries

> **Note**
>
> The `NOT EXISTS` operator is used to check if a set of values is empty

```
SELECT
    title AS 'Module Title'
FROM Module m
WHERE NOT EXISTS (
    SELECT mCode
    FROM Grade
    WHERE grade >= 70
    AND mCode = m.mCode
);
```

Module Title
Fundamentals
Databases

Table 10: The names of modules in which a student have not scored >= 70.

`IN` vs `EXISTS`

- `IN` is used to check if a value is in a set of values
  - `IN` is suited to static or small sets of values and is more efficient than `EXISTS` for these cases
- `EXISTS` is used to check if a set of values is not empty
  - `EXISTS` is suited to dynamic or large sets of values and is more efficient than `IN` for these cases

> **DBMS Query Optimisation**
>
> The DBMS will optimise the query to use the most efficient method, meaning that the performance of the query will depend on the DBMS and the data.

Nested Subqueries

- The results of the innermost subquery are stored in a temporary table
- The results of the next subquery are stored in another temporary table, and so on

```
SELECT firstName AS 'First',
        lastName AS 'Last'
FROM Student WHERE sID IN (
    SELECT sID FROM Grade
    WHERE mCode IN (
        SELECT mCode FROM Module
        WHERE title IN
        ('Fundamentals',
        'Programming')
    ));
```

First	Last
John	Smith
Jane	Doe
Mary	Jones

Table 11: The names of students who have enrolled in the Fundamentals or Programming modules.

An Aside: Testing for `NULL` Values

Testing for NULL Values

We cannot use `=` or `!=` to test for `NULL` values. Instead, we must use the `IS NULL` or `IS NOT NULL` operators.

- `NULL` is a special value in SQL that represents an unknown value
- `NULL` is not equal to any other value, including `NULL` (!)
- Therefore, we cannot use `=` or `!=` to test for `NULL` values
- Instead, we must use the `IS NULL` or `IS NOT NULL` operators
  - `SELECT * FROM table WHERE column IS NULL;`
  - `SELECT * FROM table WHERE column IS NOT NULL;`
- The following query represents a very common mistake:
  - `SELECT * FROM table WHERE column = NULL;` will not work

Aggregate Functions

Arithmetic Operators

- Arithmetic operators are used to perform calculations on numeric values
- The following arithmetic operators are available:
  - `+` - addition
  - `-` - subtraction
  - `*` - multiplication
  - `/` - division
  - `%` - modulus (remainder of division)
  - `^` - exponentiation

Handling Spaces in Names

Column or alias names with spaces must be enclosed in single quotes, square brackets or backticks. You are suggested to use single quotes.

```
SELECT
    Grade AS 'Original',
    Grade - 5 AS 'G - 5',
    Grade + 5 AS [G + 5],
    MIN(100,Grade * 2)
        AS `x2`
FROM Grade
 -- Only show the first 5 rows
LIMIT 5;
```

Original	G - 5	G + 5	x2
35	30	40	70
50	45	55	100
65	60	70	100
70	65	75	100
35	30	40	70

Table 12: Adjusting all grades.

Aggregate Functions

- Aggregate functions are used to perform calculations on a set of values
- The following aggregate functions are available (not an exhaustive list):
  - `COUNT` - returns the number of rows
  - `SUM` - returns the sum of a column
  - `AVG` - returns the average of a column
  - `MIN` - returns the minimum value of a column
  - `MAX` - returns the maximum value of a column
- We can also control the presentation of the results using the `ROUND` function
  - `ROUND` - rounds a number to a specified number of decimal places

Aggregate Functions: Example

```
SELECT
    SUM(grade) AS 'Sum',
    AVG(grade) AS 'AVG',
    ROUND(AVG(grade),2)
         AS 'Rounded',
    MIN(grade) AS 'Low',
    MAX(grade) AS 'High'
FROM Grade;
```

Sum	AVG	Rounded	Low	High
320	53.33333	53.33	35	70

Table 13: Summative Grade Statistics

Using COUNT

- The COUNT function returns the number of rows in a table
- The COUNT function can be used with or without a column name
- When used without a column name, the function returns the number of rows in the table

```
SELECT COUNT(*)
    AS 'Number of Students'
FROM Student;
```

Number of Students
4

Table 14: How many students are in our DB?

Using COUNT with DISTINCT

- The COUNT function can be used with the DISTINCT keyword to count the number of unique values in a column

```
SELECT COUNT(DISTINCT mCode)
    AS 'Number of Modules'
FROM Grade
WHERE grade < 40;
```

Number of Modules
1

Table 15: How many modules have students with a grade of <40?

Combining Aggregate Functions

- Aggregate functions can be combined with other functions
- This can be useful for calculating statistics or generating reports

```
SELECT
    MAX(Grade) - MIN(Grade)
        AS 'Range of Marks',
    AVG(Grade) - MIN(Grade)
        AS 'Average - Lowest'
FROM Grade;
```

Range of Marks	Average - Lowest
35	18.33333

Table 16: Student Grade Statistics

String Functions

- String functions are used to perform calculations on string values
- The following string functions are available:
  - `||` - concatenates two or more strings
  - `LENGTH` - returns the length of a string
  - `LOWER` - converts a string to lowercase
  - `UPPER` - converts a string to uppercase

Example: Student Names as a Single Column

Concatenating Strings

We can use the `||` (concatenation) function to combine the first and last names of students into a single column. Note that other DBMSs have a `CONCAT` function for this purpose.

```
SELECT
    firstName||' '||UPPER(lastName)
        AS 'Student Name',
    LENGTH (firstName)+LENGTH(lastName)
        AS 'Length'
FROM Student;
```

Student Name	Length
John SMITH	9
Jane DOE	7
Mary JONES	9
David SMITH	10

Table 17: Combining first and last names.

Grouping Data

Grouping Data using GROUP BY

- Often we want to perform calculations on subsets of data
- The GROUP BY clause is used to group data by one or more columns
- The GROUP BY clause is used in conjunction with aggregate functions

Example: Find the Average Grade for Each Module

```
SELECT
    mCode
        AS 'Module Code',
    AVG(grade)
        AS 'Average Grade'
FROM Grade
GROUP BY mCode;
```

Module Code	Average Grade
COMP1036	35.0
COMP1038	67.5
COMP1048	57.5

Table 18: Average Grade for Each Module

HAVING Clause

When to use HAVING vs WHERE

- The `WHERE` clause is used to filter rows before grouping
- The `HAVING` clause is used to filter groups after grouping

- The `HAVING` clause is used to filter groups of data
- The `HAVING` clause is used in conjunction with aggregate functions
- The `HAVING` clause is used after the `GROUP BY` clause

Example: Average Grade for Modules Whose Average Grade >= 60

```
SELECT
    mCode
        AS 'Module Code',
    AVG(grade)
        AS 'Average Grade'
FROM Grade
GROUP BY mCode
HAVING AVG(grade) >= 60;
```

Module Code	Average Grade
COMP1038	67.5

Table 19: Average Grade for Each Module with a Grade >= 60

The `UNION` Clause

The `UNION` Clause: Adding Rows

We can think of the `UNION` clause as a way of adding rows to a result table, whereas the other clauses we have seen so far are used to filter rows or columns.

- The `UNION` clause is used to combine the results of two or more `SELECT` statements
- This is useful for combining data from different tables or results
- We can use the `UNION` clause to generate reports, for example

Example: Generate a Report of Module and Overall Average Grades

```
SELECT
    mCode AS 'Module Code',
    ROUND(AVG(grade),2)
        AS 'Average Grade'
FROM Grade
GROUP BY mCode
UNION
SELECT
    'Overall' AS 'Module Code',
    ROUND(AVG(grade),2)
        AS 'Average Grade'
FROM Grade;
```

Module Code	Average Grade
COMP1036	35.00
COMP1038	67.50
COMP1048	57.50
Overall	53.33

Table 20: Average Grade for Each Module and Overall Average Grade

References

Learning Resources

Online Tutorials

These are clickable links to the online tutorials:

- SQLite `IN`
- SQLite `EXISTS`
- SQLite Subqueries
- SQLite `Aggregate Functions`
- SQLite `UNION` Operator
- SQLite `GROUP BY`
- SQLite `HAVING`

Textbooks and Documentation

- Chapters 6 & 7 of the Database Systems textbook
- SQLite Documentation

Reference Materials

Markl, Volker, Vijayshankar Raman, David Simmen, Guy Lohman, Hamid Pirahesh, and Miso Cilimdzic. 2004. “Robust Query Processing Through Progressive Optimization.” In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, 659–70.

Zhao, Yihong, Prasad M Deshpande, and Jeffrey F Naughton. 1997. “An Array-Based Algorithm for Simultaneous Multidimensional Aggregates.” In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, 159–70.