Structured Query Language (SQL) - 3

Databases and Interfaces

Matthew Pike

SELECT (Advanced)

Example Tables

Student

ID	First	Last
S103	John	Smith
S104	Mary	Jones
S105	Jane	Brown
S106	Mark	Jones
S107	John	Brown

Course

Code	Title
DBS	Database Systems
PR1	Programming 1
PR2	Programming 2
IAI	Introduction to AI

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

SQL SELECT Overview

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

([] optional, | or) 4

DISTINCT and ALL

- Sometimes you end up with duplicate entries
- Using **`DISTINCT`** removes duplicates
- Using **`ALL`** retains duplicates
- **`ALL`** is used as a default if neither is supplied
- These will work over multiple columns

```
SELECT ALL Last
  FROM Student;
```

Last
Smith
Jones
Brown
Jones
Brown

```
SELECT DISTINCT Last
  FROM Student;
```

Last
Smith
Jones
Brown

WHERE Clauses

- In most cases returning all the rows is not necessary
  - A **`WHERE`** clause restricts rows that are returned
  - It takes the form of a condition
    - only rows that satisfy the condition are returned

- Example conditions:
  - `Mark < 40`
  - `First = 'John'`
  - `First <> 'John'`
  - `First = Last`
  - `(First = 'John') AND (Last = 'Smith')`
  - `(Mark < 40) OR (Mark > 70)`

WHERE Examples

```
SELECT * FROM Grade
 WHERE Mark >= 60;
```

ID	Code	Mark
S103	DBS	72
S104	PR1	68
S104	IAI	65
S107	PR1	76
S107	PR2	60

```
SELECT DISTINCT ID
    FROM Grade
    WHERE Mark >= 60;
```

ID
S103
S104
S107

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 ID numbers and Marks for students who have passed (scored 50% or more) in IAI

ID	Mark
S103	58
S104	65

Solution

```
SELECT ID, Mark FROM Grade
 WHERE (Code = 'IAI') AND
       (Mark >= 50);
```

Alias

Aliases

- Aliases rename columns or tables
  - Can make names more meaningful
  - Can shorten names, making them easier to use
  - Can resolve ambiguous names

- Two forms:
  - Column alias

```
SELECT column [AS] new-col-name
```

  - Table alias

```
SELECT * FROM table [AS] new-table-name
```

([] optional)

Alias Example

Employee

ID	Name
123	John
124	Mary

WorksIn

ID	Department
123	Marketing
124	Sales
124	Marketing

```
SELECT
   E.ID AS empID,
   E.Name, W.Department
FROM
   Employee E,
   WorksIn W
WHERE
   E.ID = W.ID;
```

Note: You cannot create a column alias in a WHERE clause

Alias Example

empID	Name	Department
123	John	Marketing
124	Mary	Sales
124	Mary	Marketing

```
SELECT
   E.ID AS empID,
   E.Name, W.Department
FROM
   Employee E,
   WorksIn W
WHERE
   E.ID = W.ID;
```

Subqueries

Subqueries

- A SELECT statement can be nested inside another query to form a subquery
- The results of the subquery are passed back to the containing query

empID	Name	Department
123	John	Marketing
124	Mary	Sales
124	Mary	Marketing

- For example, retrieve a list of names of people who are in John's department:

```
SELECT Name
    FROM Employee
WHERE Dept =
    (SELECT Deptartment
    FROM Employee
    WHERE Name = 'John')
```

Subqueries

```
SELECT Name
    FROM Employee
    WHERE Dept =
        (SELECT Dept
        FROM Employee
        WHERE
            Name = 'John')
```

- First the subquery is evaluated, returning 'Marketing'
- This value is passed to the main query

```
SELECT Name
    FROM Employee
WHERE Dept =
'Marketing';
```

Subqueries

- Often a subquery will return a set of values rather than a single value
- We cannot directly compare a single value to a set. Doing so will result in an error
- Options for handling sets
  - IN
    - checks to see if a value is in a set
  - EXISTS
    - checks to see if a set is empty
  - NOT
    - can be used with any of the above

IN

- Using IN we can see if a given value is in a set of values
- NOT IN checks to see if a given value is not in the set
- The set can be given explicitly or can be produced in a subquery

```
SELECT columns
   FROM tables
   WHERE value
      IN set;
```

```
SELECT columns
   FROM tables
   WHERE value
      NOT IN set;
```

IN

Employee

Name	Department	Manager
John	Marketing	Chris
Mary	Marketing	Chris
Chris	Marketing	Jane
Peter	Sales	Jane
Jane	Management	

```
SELECT *
   FROM Employee
   WHERE Department IN
      ('Marketing',
      'Sales');
```

Employee

Name	Department	Manager
John	Marketing	Chris
Mary	Marketing	Chris
Chris	Marketing	Jane
Peter	Sales	Jane

NOT IN

Employee

Name	Department	Manager
John	Marketing	Chris
Mary	Marketing	Chris
Chris	Marketing	Jane
Peter	Sales	Jane
Jane	Management	

```
SELECT *
    FROM Employee
    WHERE Name
    NOT IN
    ('Chris', 'Jane');
```

Name	Department	Manager
John	Marketing	Chris
Mary	Marketing	Chris
Peter	Sales	Jane

EXISTS

- The EXISTS operator is a logical operator that checks whether a subquery returns any row.
- The NOT EXISTS operator returns true if the subquery returns no row.

```
SELECT columns
   FROM tables
   WHERE EXISTS set;

SELECT columns
   FROM tables
   WHERE NOT EXISTS set;
```

EXISTS and NOT EXISTS Examples

- Find information on students who **do** have any registered grades:
  - `SELECT * FROM Student a WHERE EXISTS ( SELECT 1 FROM Grade WHERE ID = a.ID);`
- Find information on students who **do not** have any registered grades:
  - `SELECT * FROM Student a WHERE NOT EXISTS ( SELECT 1 FROM Grade WHERE ID = a.ID);`

Student

ID	First	Last
S103	John	Smith
S104	Mary	Jones
S105	Jane	Brown
S106	Mark	Jones
S107	John	Brown

Course

Code	Title
DBS	Database Systems
PR1	Programming 1
PR2	Programming 2
IAI	Introduction to AI

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

EXISTS

Employee

Name	Department	Manager
John	Marketing	Chris
Mary	Marketing	Chris
Chris	Marketing	Jane
Peter	Sales	Jane
Jane	Management	

```
SELECT *
   FROM Employee AS E1
   WHERE EXISTS(
      SELECT * FROM
         Employee AS E2
         WHERE E1.Name =
            E2.Manager);
```

- Retrieve all the info for those employees who are also managers

EXISTS

```
SELECT * FROM Employee AS E1 WHERE EXISTS (SELECT * FROM
Employee AS E2 WHERE E1.Name = E2.Manager);
```

Employee E1

Name	Dept	Manager
John	Marketing	Chris
Mary	Marketing	Chris
Chris	Marketing	Jane
Peter	Sales	Jane
Jane	Management	

Employee E2

Name	Dept	Manager
John	Marketing	Chris
Mary	Marketing	Chris
Chris	Marketing	Jane
Peter	Sales	Jane
Jane	Management	

EXISTS

Name	Dept	Manager	Name	Dept	Manager
John	Marketing	Chris	John	Marketing	Chris
John	Marketing	Chris	Mary	Marketing	Chris
John	Marketing	Chris	Chris	Marketing	Jane
John	Marketing	Chris	Peter	Sales	Jane
John	Marketing	Chris	Jane	Management	
Mary	Marketing	Chris	John	Marketing	Chris
Mary	Marketing	Chris	Mary	Marketing	Chris
Mary	Marketing	Chris	Chris	Marketing	Jane
Mary	Marketing	Chris	Peter	Sales	Jane
Mary	Marketing	Chris	Jane	Management	
Chris	Marketing	Jane	John	Marketing	Chris
Chris	Marketing	Jane	Mary	Marketing	Chris
Chris	Marketing	Jane	Chris	Marketing	Jane

EXISTS

```
SELECT * FROM Employee AS E1 WHERE EXISTS (SELECT * FROM
    Employee AS E2 WHERE E1.Name = E2.Manager);
```

Name	Dept	Manager	Name	Dept	Manager
Chris	Marketing	Jane	John	Marketing	Chris
Chris	Marketing	Jane	Mary	Marketing	Chris
Jane	Management		Chris	Marketing	Jane
Jane	Management		Peter	Sales	Jane

Name	Dept	Manager
Chris	Marketing	Jane
Chris	Marketing	Jane
Jane	Management	
Jane	Management	

EXISTS

Employee

Name	Department	Manager
John	Marketing	Chris
Mary	Marketing	Chris
Chris	Marketing	Jane
Peter	Sales	Jane
Jane	Management	

```
SELECT *
    FROM Employee AS E1
    WHERE EXISTS (
        SELECT * FROM
        Employee AS E2
        WHERE E1.Name =
            E2.Manager);
```

Name	Dept	Manager
Chris	Marketing	Jane
Jane	Management	

ORDER BY

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

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

ORDER BY

Grade

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

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

ORDER BY

Grade

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

```
SELECT * FROM Grade
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

Grade

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

```
SELECT * FROM Grade
   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

Arithmetic

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

```
SELECT Mark / 100 FROM
Grade;

SELECT Salary + Bonus FROM
Employee;


SELECT 1.20 * Price
    AS 'Price inc. VAT'
    FROM Products;
```

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

COUNT

Grade

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 Grade;

SELECT
    COUNT(Code)
        AS Count
    FROM Grade;

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

COUNT

Grade

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 Grade;
```

Count
6

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

Count
6

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

Count
4

SUM, MIN/MAX and AVG

Grade

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 Grade;

SELECT
    MAX(Mark) AS Best
    FROM Grade;

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

SUM, MIN/MAX and AVG

Grade

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 Grade;
```

Total
320

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

Best
72

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

Mean
53.33

Aggregate Functions

You can combine aggregate functions using arithmetic

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

Grade

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