SQL : SELECT Queries

Martin Read

UNIVERSITY OF PLYMOUTH

Aim: Introduction to SELECT Queries

- To reinforce the use of SQL statements to extract data from a database.

UNIVERSITY OF PLYMOUTH

Example SELECT Query

SELECT Artifact.[Site Code], Artifact.[Artefact Number], Artifact.[Description], [Simple Name].[Simple Name], [Full name].[Full name], Material.[Material], Artifact.[Part Number], [Number of items], Context, Artifact.[Location Type], [Location Reference], Artifact.Length, Artifact.Width, Artifact.Diameter, Artifact.Height, Artifact.Weight, Artifact.[Recovery Reference], [Object Class?] AS Expr1, [Object Type?]

FROM Material INNER JOIN ([Simple Name] INNER JOIN ([Full Name] INNER JOIN Artifact ON [Full Name].[Full name ID] = Artifact.[Object Class]) ON [Simple Name].[Simple Name ID] = Artifact.[Object Type]) ON Material.[Material ID] = Artifact.Material

WHERE Artifact.[Object Class] IN (select [Full name ID] FROM [Full Name] WHERE [Full Name] LIKE [Object Class?])

AND Artifact.[Object Type] IN (select [Simple name ID] FROM [Simple Name] WHERE [Simple Name] LIKE [Object Type?])

ORDER BY Artifact.[Artefact Number];

Basic Retrieval Command Structure

SQL structures data retrieval into 3 distinct clauses known collectively as a SELECT statement

SELECT : Lists the columns to be projected into the answer table

FROM : Identifies the tables from which answer columns will be projected and any tables used in joins to form the answer

WHERE : Includes conditions for selection and join conditions

SELECT Basic Syntax

SELECT [DISTINCT | ALL] {* | [columns/expression [AS new_name]] [,…]}

FROM <table-name/s>

[WHERE <condition>]

[ORDER BY <column-list>]

Student

student_id	firstname	lastname
S103	John	Smith
S104	Mary	Jones
S105	Jane	Brown
S106	Mark	Jones
S107	John	Brown

Module

Code	title
DBS	Database Systems
PR1	Programming 1
PR2	Programming 2
IAI	Intro to AI

Grade

student_id	module_code	mark
S103	DBS	72
S103	IAI	58
S104	PR1	
S104	IAI	65
S106	PR2	43
S107	PR1	76
S107	PR2	60
S107	IAI	35

```
SELECT student_id, firstname + ” “ + lastname AS “Student Name”
    FROM student;
```

DISTINCT and ALL

- Sometimes you end up with duplicate entries
- Using **DISTINCT** removes duplicates
- Using **ALL** retains them – this is the default

SELECT DISTINCT lastname
FROM student;

WHERE Clauses

- Usually, we do not want to display all the records:
  - **A WHERE clause restricts the records that are returned**
  - The WHERE clause takes the form of a condition - only those records that satisfy the condition are returned

- Example **WHERE** :

mark < 40

firstname = “John”

firstname <> “John”

first name = lastname

(firstname = “John”) AND (lastname = “Smith”)

(mark < 40) OR (mark >70)

UNIVERSITY OF PLYMOUTH

5 types of condition (1)

Comparison

- compare the value of a field (or expression) against the value of an expression

arithmetic operators

=	<>
<	<=
>	>=

Boolean (logical) operators

NOT, AND, OR

5 types of condition (2)

Range

- test whether the value of a field (or expression) falls within specified boundaries

```
WHERE column_expression BETWEEN lower_bound AND upper_bound


WHERE mark BETWEEN 40 AND 70;
```

5 types of condition (3)

Set membership

- test whether the value of a field (or expression) equals one in a set of values (from a subquery or list of values)

```
WHERE column_expression IN (set_value [,…] )

WHERE lastname IN (“Smith”, “Brown”)
```

Subquery

```
WHERE student_id IN (SELECT student_id FROM Grade
    WHERE module_code = “IAI”);
```

UNIVERSITY OF PLYMOUTH

5 types of condition (4)

Pattern match

- test whether the value of a field (or expression) matches a pattern (string)

  WHERE column_expression LIKE “pattern_expression”

- The following “wildcard” characters can be used

% (percent) any sequence of zero or more characters

_ (underscore) any single character

[] range of wildcards matches any single character in range or set

[^] not in range matches any single character NOT in range/set

```
SELECT student_id, firstname + " " + lastname AS "Student Name"
    FROM student
    WHERE lastname LIKE "J%";
```

UNIVERSITY OF PLYMOUTH

5 types of condition (5)

Null

- test whether the value of a field (or expression) is null (=empty)

```
WHERE column_expression IS [NOT] NULL

WHERE mark IS NOT NULL;
```

Example SQL Queries

```
SELECT * FROM grade;

SELECT COUNT(*) AS “Total firsts” FROM Grade
        WHERE mark >= 70;

SELECT DISTINCT student_id
        FROM Grade
        WHERE mark >= 60;

SELECT lastname FROM Student WHERE student_id IN
(SELECT student_id FROM Grade WHERE mark IS NOT NULL
AND module_code LIKE “P%”);
```

UNIVERSITY OF PLYMOUTH

Exercise

- Write an SQL query to find a list of student_id & marks in module IAI of students who have passed (scored 40 or higher).
- Now try to find the average mark for module IAI

UNIVERSITY OF PLYMOUTH

Solution

```
SELECT student_id, mark
        FROM Grade
        WHERE module_code = “IAI”AND mark >= 40;
```

Aggregate Functions

Functions include:

- COUNT() Calculates number of records
- SUM() Sum of column values returned
- MAX(), MIN() Min & Max values in a column
- AVG() Average
- StDev() Standard deviation
- VAR Variance - square of st. deviation

- Used to produce aggregated numeric data
- COUNT, MIN & MAX can also be used on non-numeric fields

UNIVERSITY OF PLYMOUTH

```
SELECT SUM(mark)/COUNT(mark) AS “Average”
        FROM Grade
        WHERE module_code = “IAI”:

SELECT AVG(mark)
        FROM Grade
        WHERE module_code = “IAI”:
```

Precedence

- Multiplication & Division take priority
- Same priority operators evaluated L to R
- Use parenthesis to force prioritisation and clarify statements

UNIVERSITY OF PLYMOUTH

Grouping data

```
SELECT [DISTINCT | ALL] {* | [columns/expression [AS
new_name]] [,…]}
        FROM table_name/s
        [GROUP BY column_list]  [HAVING condition]
        [ORDER BY column_list];
```

- GROUP BY - groups the rows by the same column value
- HAVING - filters the groups according to the supplied condition

Group/Order Operators

- Used for output in a specific sequence
- One or more fields in each record must contain the same value for every record in the group
- GROUP BY used with aggregate functions

```
SELECT  student_id, AVG(mark)
FROM  Grade
GROUP BY student_id
ORDER BY student_id DESC;
```

- Lists average mark per student, in reverse order
- All entries in the SELECT clause (except the aggregate function) MUST be in the GROUP BY clause but not necessarily vice versa

Having Operator

- Used as a final filter
- Filters the results after the GROUP BY function

```
SELECT  student_num, sname, SUM(amount_paid)
        FROM  finance
        WHERE payment_date > #01/01/08#
        GROUP BY student_num, sname
        HAVING SUM(amount_paid)  > 1000
        ORDER BY student_num DESC;
```

UNIVERSITY OF PLYMOUTH

Command processed in following sequence

FROM	allocates the table(s) to be used
WHERE	if present
GROUP BY	if present
HAVING	if present
SELECT	specifies columns in the new (resulting) relation
ORDER BY	if present, specifies sequence of columns in the new relation

Joining tables

- In a relational DBMS, often a need to extract data from more than one table at a time

- Tables can be joined where there are common column(s)

  usually a primary key - foreign key relationship

SELECT from Multiple Tables

SELECT * FROM Table1,Table2...

- If the tables have columns with the same name, then: ambiguity
- You resolve this by referencing columns with the table name

  SELECT Table1.column, Table2.column

UNIVERSITY OF PLYMOUTH

SELECT from Multiple Tables

*SELECT * FROM student, grade;*

- BUT records for the first entry from the Student table... Are matched with ALL of the entries from the Grade table

student_id	firstname	lastname	student_id	module_code	mark
S103	John	Smith	S103	DBS	72
S103	John	Smith	S103	IAI	58
S103	John	Smith	S104	PR1	
S103	John	Smith	S104	IAI	65
S103	John	Smith	S106	PR2	43
S104	Mary	Jones	S103	DBS	72

- And then with the second…and so on

UNIVERSITY OF PLYMOUTH

Cartesian Product
Unconditional join

SELECT from Multiple Tables

*SELECT * FROM student, grade*

WHERE (student.student_id = grade.student_id);

- **INNER JOIN** or Equi-Join - specify a condition which a pair of rows satisfy
- 2 or more tables linked via common field/s
  - Most common form of joining used in relational data manipulation
  - Frequently involves primary & foreign keys

UNIVERSITY OF PLYMOUTH

SELECT from Multiple Tables - JOINs

- JOINs can be used to combine tables
  - There are many types of JOIN
  - **CROSS JOIN**
  - **INNER JOIN**
  - **(NATURAL JOIN)**
  - **OUTER JOIN**
- **OUTER JOIN**s are linked with **NULL**s

UNIVERSITY OF PLYMOUTH

https://docs.microsoft.com/en-us/sql/relational-databases/performance/joins?view=sql-server-ver15

CROSS JOIN

- **A CROSS JOIN B**
- returns all pairs of rows from A and B
- Same as SELECT * FROM A, B
  - i.e. Cartesian join

SELECT *

FROM student CROSS JOIN grade

(NATURAL JOIN – NOT available in SQL Server)

A NATURAL JOIN B

- creates an implicit join clause based on common columns in the two tables being joined. Common columns are columns having the same name in both tables
- Can be an INNER join, LEFT OUTER join, or RIGHT OUTER join.
- Default is INNER join

    *SELECT **

    FROM student NATURAL JOIN grade;

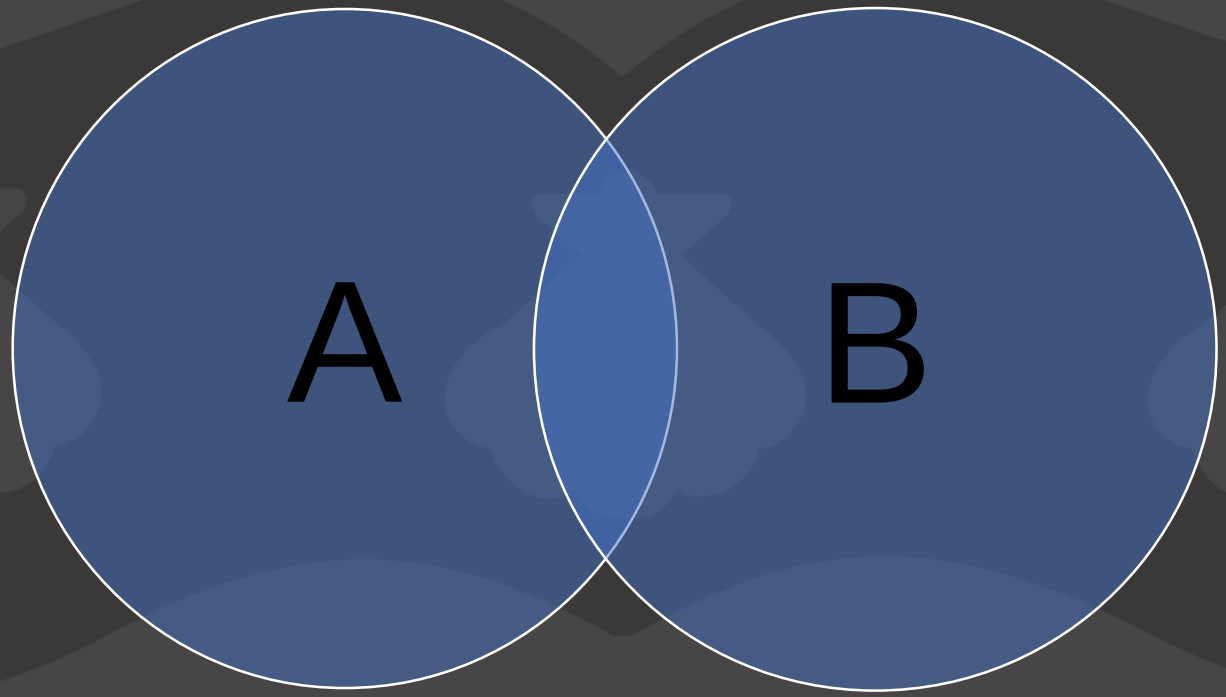
- is the same as

SELECT A.col1,… A.coln, [& all other columns apart from B.col1, …B.coln]

FROM A,B

WHERE A.col1 = B.col1 AND A.col2 =B.col2

...AND A.coln = B.col.n (this assumes that col1… coln in A & B have common names)



SELECT *
FROM A INNER JOIN B...

INNER JOIN

- specifies a condition which the pairs of rows satisfy

```
SELECT *
        FROM A INNER JOIN B
        [ ON ( join_condition ) ]
```

On other versions of SQL you can also use

```
SELECT *
        FROM A INNER JOIN B
        USING (col1,col2,…)
```

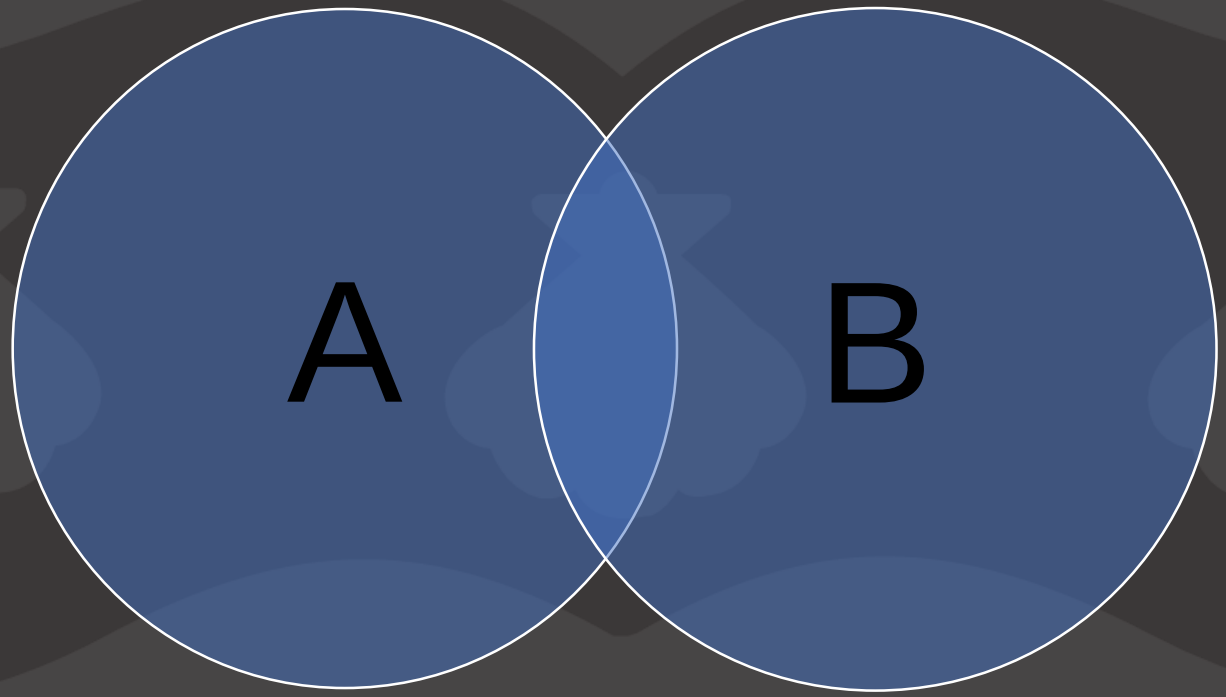
- Chooses rows where the given columns are equal

UNIVERSITY OF PLYMOUTH

https://www.sqlservertutorial.net/sql-server-basics/sql-server-inner-join/

```
SELECT *
        FROM Student INNER JOIN Grade
        ON (Student.student_id = Grade.student_no);
```

UNIVERSITY OF PLYMOUTH



OUTER JOINs

OUTER JOIN

Selects non-matching records as well

Can be used to see rows not meeting the join condition

Combines records from two tables according to a given JOIN condition

- Where matching record not present in second table
  - Results table is padded out with null values

! those joins that are not *outer* joins are *inner* joins

! LEFT, RIGHT and FULL outer joins are possible

```
SELECT s.student_id, lastname, course_code, mark
        FROM Grade AS g RIGHT OUTER JOIN Student s
        ON g.student_id = s.student_id
        ORDER BY lastname;
```

Will show names for all students

- including those without courses

LEFT OUTER JOIN

- Where matching record not present in second table
- Results table padded out with null values

FULL OUTER JOIN

Select rows from both relations even if they have no match

Does not work in ACCESS

Code	title	Average
DBS	Database Systems	72
PR1	Programming 1	76
PR2	Programming 2	52
IAI	Intro to AI	51
HCI	Human Computer Interaction	

student_id	module_code	mark
S103	DBS	72
S103	IAI	58
S104	PR1	
S104	IAI	65
S106	PR2	43
S107	PR1	76
S107	PR2	60
S107	IAI	35

UNIVERSITY OF PLYMOUTH

```
SELECT m.Code, title, AVG(mark) AS “Average”
    FROM Module m LEFT OUTER JOIN Grade g
    ON m.code = g.module_code
    GROUP BY module_code;
```

- A relational operation on a table returns a ‘results table’
- A relational operation on a results table returns another results table

- Allows nesting of operations

Joining three tables

- Join statements can be nested:

```
SELECT module_code, title, s.student_id, lastname, mark
    FROM (Student s INNER JOIN Grade g
    ON s.student_id = g.student_id)
    INNER JOIN Module m ON g.[Module_code] =
        m.[Code];
```

JOINs vs WHERE Clauses

- JOINs (so far) are not needed
- You can have the same effect by selecting from multiple tables with an appropriate WHERE clause
- So should you use JOINs?

- They often lead to concise queries
- NATURAL JOINs are very common

BUT

- Support for JOINs varies a fair bit among SQL dialects

UNIVERSITY OF PLYMOUTH

JOINs vs Subqueries

```
SELECT s.student_id, lastname
    FROM student s, grade g
    WHERE s.student_id =
  g.student_id
    AND module_code = "IAI";
```

```
SELECT student_id, lastname
    FROM student
    WHERE student_id IN
     (SELECT student_id
      FROM grade
      WHERE module_code = "IAI);
```

UNIVERSITY OF PLYMOUTH

```
SELECT Artifact.[Site Code], Artifact.[Artefact Number], Artifact.[Description], [Simple Name].[Simple Name], [Full name].[Full name], Material.[Material], Artifact.[Part Number], [Number of items], Context, Artifact.[Location Type], [Location Reference], Artifact.Length, Artifact.Width, Artifact.Diameter, Artifact.Height, Artifact.Weight, Artifact.[Recovery Reference], [Object Class?] AS Expr1, [Object Type?]

FROM Material INNER JOIN ([Simple Name] INNER JOIN ([Full Name] INNER JOIN Artifact ON [Full Name].[Full name ID] = Artifact.[Object Class]) ON [Simple Name].[Simple Name ID] = Artifact.[Object Type]) ON Material.[Material ID] = Artifact.Material

WHERE Artifact.[Object Class] IN (select [Full name ID] FROM [Full Name] WHERE [Full Name] LIKE [Object Class?])
AND Artifact.[Object Type] IN (select [Simple name ID] FROM [Simple Name] WHERE [Simple Name] LIKE [Object Type?])
ORDER BY Artifact.[Artefact Number];
```


University of Plymouth
Advancing knowledge, transforming lives
UNIVERSITY OF
PLYMOUTH