X THE PLACE OF USEFUL LEARNING

University of Strathclyde Glasgow

Database Fundamentals – CS990

Database and Web Systems Development - CS952

Image source: https://www.strath.ac.uk/branding/

CS990/CS952
Database Fundamentals

Some more queries (SELECT)

Course Content

1. Introduction to Relational Databases *(Introduction + Relational Model)*
2. Data Modelling - *(Entity Relationship Modelling + The Enhanced Entity Relationship Model)*
3. Database Design and SQL - *(Logical modelling + Introduction to SQL)*
4. Further SQL - *(Advanced SQL queries + Creating tables with SQL)*
5. Normalisation - *(Normalisation to second normal form + Third normal form)*

Today

- Example
- Functions
- Subqueries
- Aliases

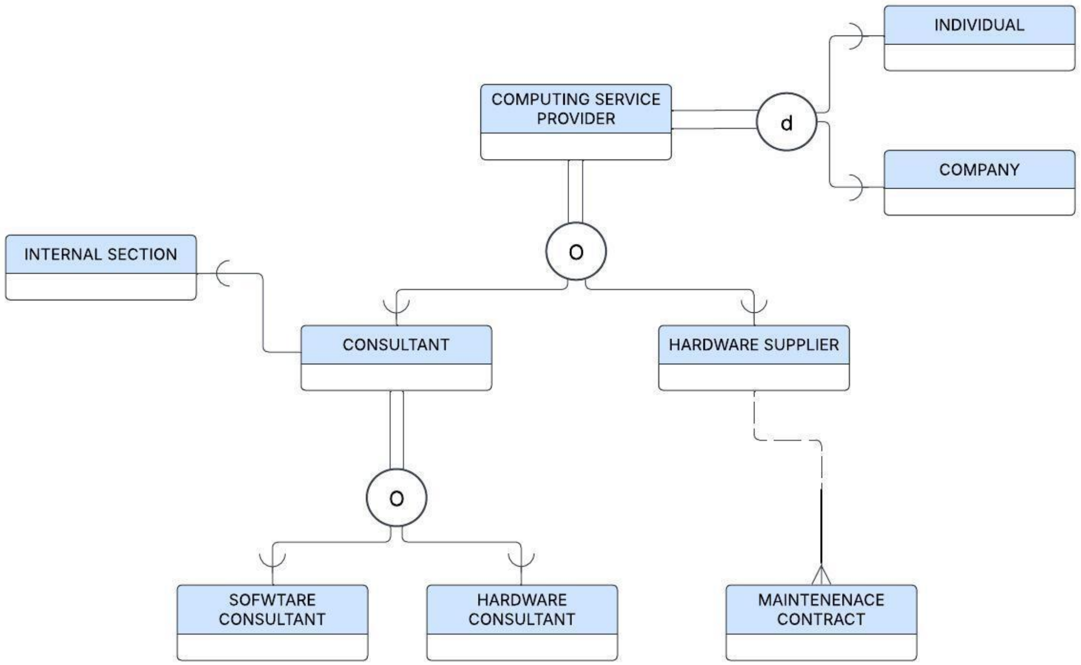
Example

Entities:

- Computing service provider
  - Individual
  - Company

  - Hardware Supplier
    - Maintenance Contract
  - Consultant
    - Software Consultant
    - Hardware Consultant

    - Internal Section



Relationships

Name	Entities		Degree	Optionality
Holds	Maintenance Contract	Hardware Supplier	N:1	**Optional** on hardware Supplier **Obligatory** on Maintenance Contract

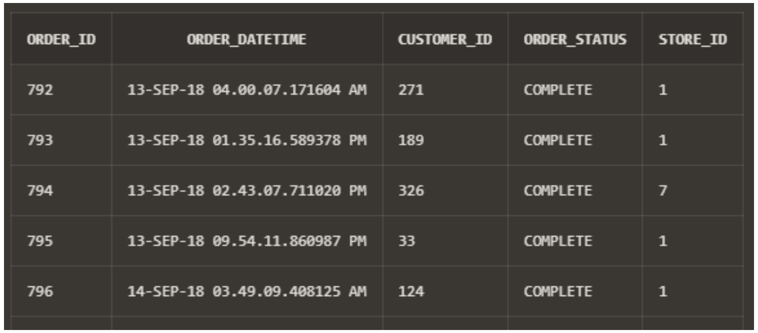
Assumption: an HW Supplier **can hold** many Maintenance Contracts

ANSI and NON-ANSI SQL

American National Standards Institute (ANSI)

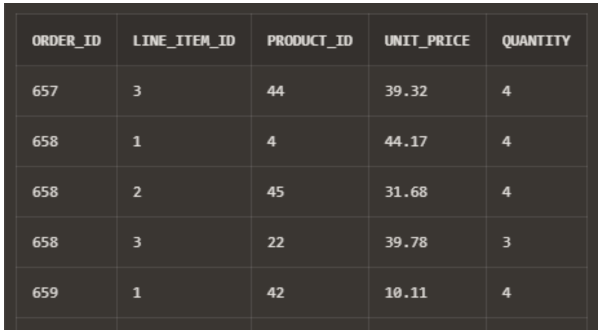
- ANSI SQL
  - Uses explicit (JOIN) syntax
  - More readable, structured, and portable across different databases
  - Queries written in ANSI SQL are more likely to work across different databases (e.g., MySQL, PostgreSQL, Oracle, SQL Server).
  - Introduced in SQL-92 standard
- NON-ANSI SQL
  - Uses explicit (JOIN) syntax
  - Complex to use
  - Database specific

ORDERS



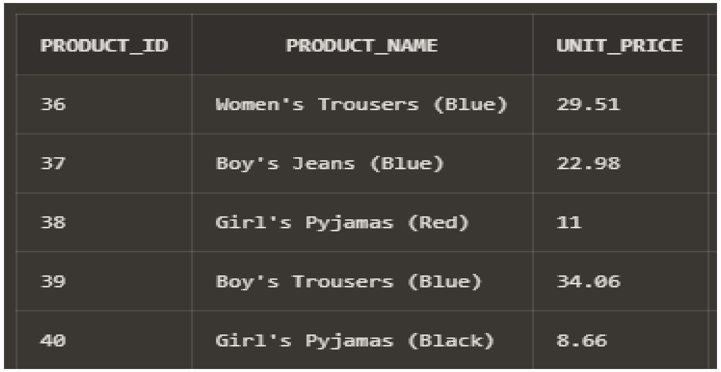
ORDER_ID	ORDER_DATETIME	CUSTOMER_ID	ORDER_STATUS	STORE_ID
792	13-SEP-18 04.00.07.171604 AM	271	COMPLETE	1
793	13-SEP-18 01.35.16.589378 PM	189	COMPLETE	1
794	13-SEP-18 02.43.07.711020 PM	326	COMPLETE	7
795	13-SEP-18 09.54.11.860987 PM	33	COMPLETE	1
796	14-SEP-18 03.49.09.408125 AM	124	COMPLETE	1

ORDER_ITEMS



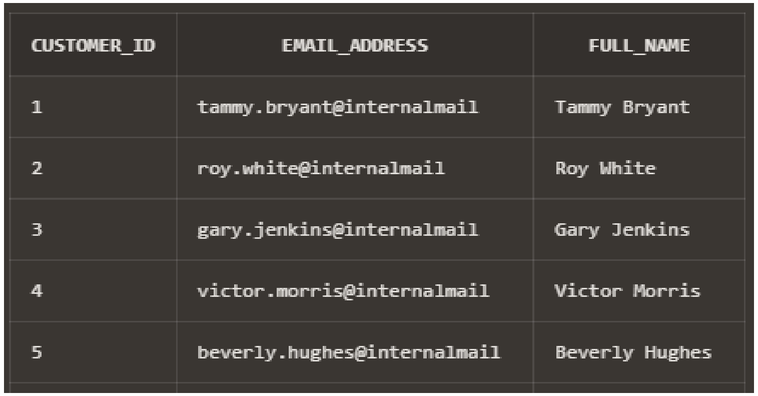
ORDER_ID	LINE_ITEM_ID	PRODUCT_ID	UNIT_PRICE	QUANTITY
657	3	44	39.32	4
658	1	4	44.17	4
658	2	45	31.68	4
658	3	22	39.78	3
659	1	42	10.11	4

PRODUCTS



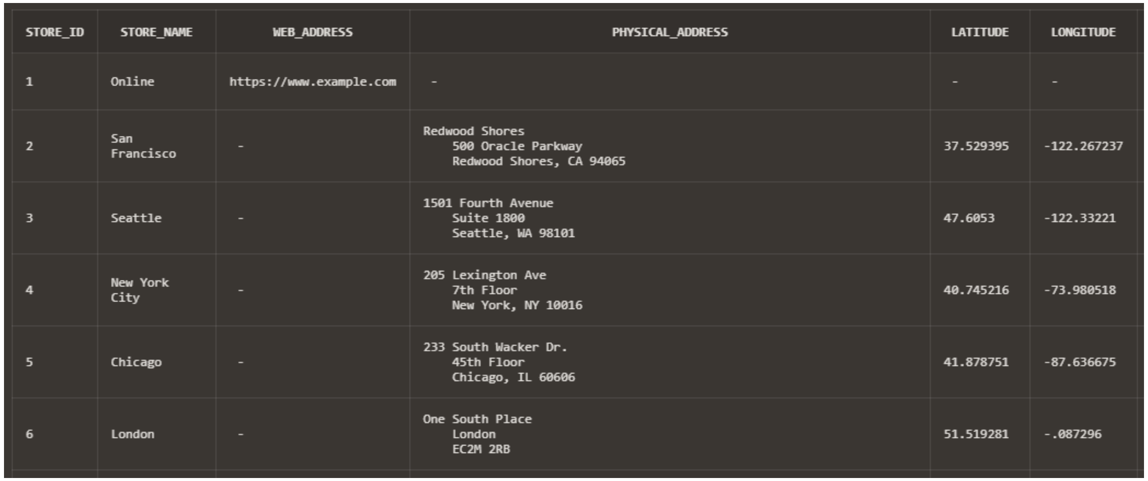
PRODUCT_ID	PRODUCT_NAME	UNIT_PRICE
36	Women's Trousers (Blue)	29.51
37	Boy's Jeans (Blue)	22.98
38	Girl's Pyjamas (Red)	11
39	Boy's Trousers (Blue)	34.06
40	Girl's Pyjamas (Black)	8.66

CUSTOMERS



CUSTOMER_ID	EMAIL_ADDRESS	FULL_NAME
1	tammy.bryant@internalmail	Tammy Bryant
2	roy.white@internalmail	Roy White
3	gary.jenkins@internalmail	Gary Jenkins
4	victor.morris@internalmail	Victor Morris
5	beverly.hughes@internalmail	Beverly Hughes

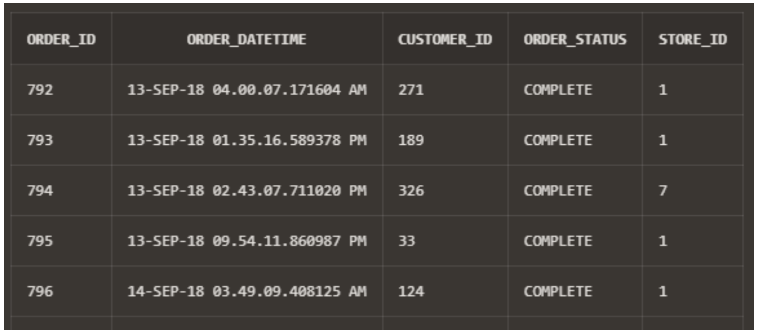
STORES



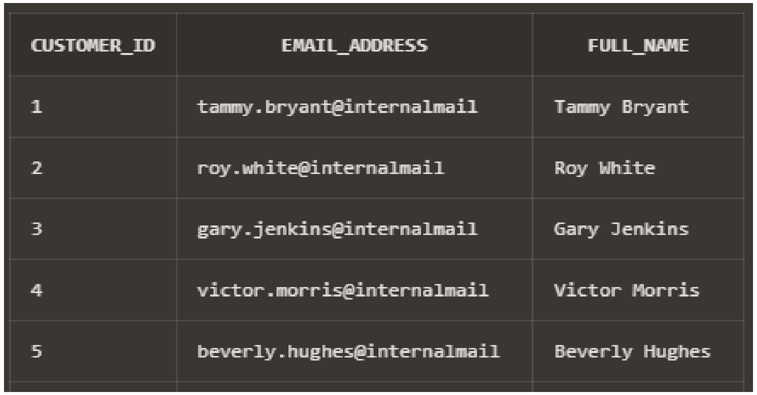
STORE_ID	STORE_NAME	WEB_ADDRESS	PHYSICAL_ADDRESS	LATITUDE	LONGITUDE
1	Online	https://www.example.com	-	-	-
2	San Francisco	-	Redwood Shores 500 Oracle Parkway Redwood Shores, CA 94065	37.529395	-122.267237
3	Seattle	-	1501 Fourth Avenue Suite 1800 Seattle, WA 98101	47.6053	-122.33221
4	New York City	-	205 Lexington Ave 7th Floor New York, NY 10016	40.745216	-73.980518
5	Chicago	-	233 South Wacker Dr. 45th Floor Chicago, IL 60606	41.878751	-87.636675
6	London	-	One South Place London EC2M 2RB	51.519281	-.087296

Non-ANSI JOIN

RELATING TWO TABLES BY A JOIN (NON-ANSI SYNTAX)



ORDER_ID	ORDER_DATETIME	CUSTOMER_ID	ORDER_STATUS	STORE_ID
792	13-SEP-18 04.00.07.171604 AM	271	COMPLETE	1
793	13-SEP-18 01.35.16.589378 PM	189	COMPLETE	1
794	13-SEP-18 02.43.07.711020 PM	326	COMPLETE	7
795	13-SEP-18 09.54.11.860987 PM	33	COMPLETE	1
796	14-SEP-18 03.49.09.408125 AM	124	COMPLETE	1



CUSTOMER_ID	EMAIL_ADDRESS	FULL_NAME
1	tammy.bryant@internalmail	Tammy Bryant
2	roy.white@internalmail	Roy White
3	gary.jenkins@internalmail	Gary Jenkins
4	victor.morris@internalmail	Victor Morris
5	beverly.hughes@internalmail	Beverly Hughes

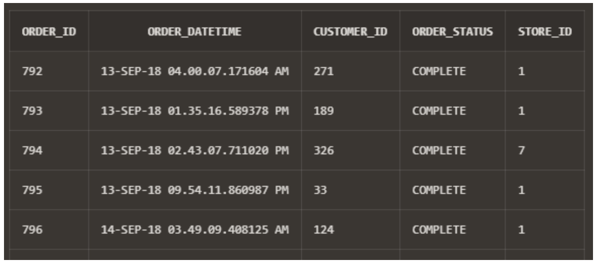
List the names of customers their order numbers and the dates on which they placed orders.

```
SELECT
      C."FULL_NAME",
      O."ORDER_ID",
      O."ORDER_DATETIME“
FROM  CO."CUSTOMERS" C,CO."ORDERS" O
WHERE C."CUSTOMER_ID" = O."CUSTOMER_ID";
```

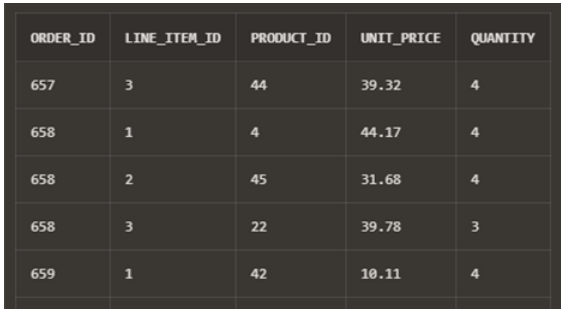
- The tables are joined on attributes mentioned in the **WHERE** clause
- Only rows that have a matching value in **CUSTOMER** and **ORDER** are returned.

Non-ANSI JOIN

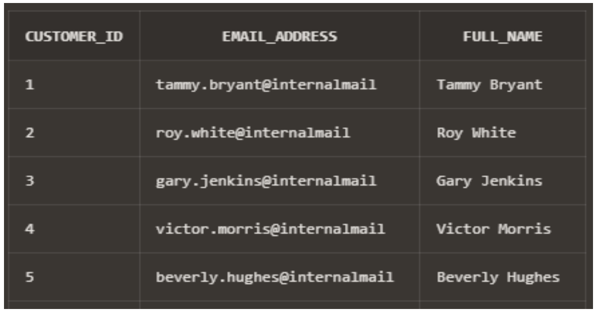
RELATING more than TWO TABLES BY A JOIN (NON-ANSI SYNTAX)



ORDER_ID	ORDER_DATETIME	CUSTOMER_ID	ORDER_STATUS	STORE_ID
792	13-SEP-18 04.00.07.171604 AM	271	COMPLETE	1
793	13-SEP-18 01.35.16.589378 PM	189	COMPLETE	1
794	13-SEP-18 02.43.07.711020 PM	326	COMPLETE	7
795	13-SEP-18 09.54.11.860987 PM	33	COMPLETE	1
796	14-SEP-18 03.49.09.408125 AM	124	COMPLETE	1



ORDER_ID	LINE_ITEM_ID	PRODUCT_ID	UNIT_PRICE	QUANTITY
657	3	44	39.32	4
658	1	4	44.17	4
658	2	45	31.68	4
658	3	22	39.78	3
659	1	42	10.11	4



CUSTOMER_ID	EMAIL_ADDRESS	FULL_NAME
1	tammy.bryant@internalmail	Tammy Bryant
2	roy.white@internalmail	Roy White
3	gary.jenkins@internalmail	Gary Jenkins
4	victor.morris@internalmail	Victor Morris
5	beverly.hughes@internalmail	Beverly Hughes

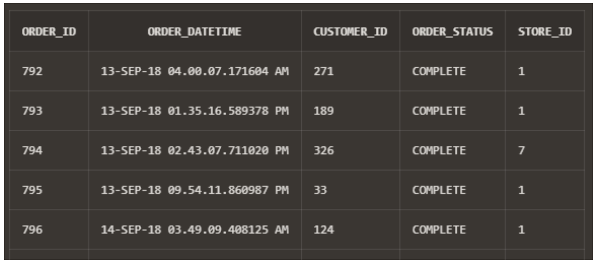
Additional tables can be added by further **WHERE** clauses.

List the customer names and the order numbers, order dates, store ID, the product IDs, and quantities.

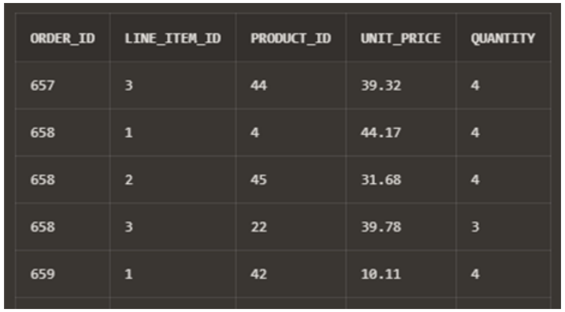
```
SELECT
    C."FULL_NAME",
    O."ORDER_ID",
    O."ORDER_DATETIME",
    O."STORE_ID",
    OI."PRODUCT_ID",
    OI."QUANTITY"
FROM CO."CUSTOMERS" C, CO."ORDERS" O, CO."ORDER_ITEMS" OI
WHERE C."CUSTOMER_ID" = O."CUSTOMER_ID"
AND O."ORDER_ID" = OI."ORDER_ID";
```

Non-ANSI JOIN

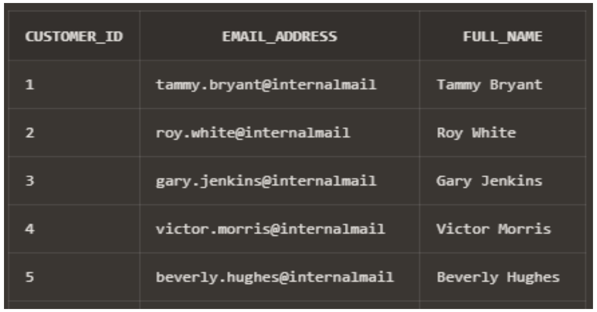
RELATING more than TWO TABLES BY A JOIN (NON-ANSI SYNTAX)



ORDER_ID	ORDER_DATETIME	CUSTOMER_ID	ORDER_STATUS	STORE_ID
792	13-SEP-18 04.00.07.171604 AM	271	COMPLETE	1
793	13-SEP-18 01.35.16.589378 PM	189	COMPLETE	1
794	13-SEP-18 02.43.07.711020 PM	326	COMPLETE	7
795	13-SEP-18 09.54.11.860987 PM	33	COMPLETE	1
796	14-SEP-18 03.49.09.408125 AM	124	COMPLETE	1



ORDER_ID	LINE_ITEM_ID	PRODUCT_ID	UNIT_PRICE	QUANTITY
657	3	44	39.32	4
658	1	4	44.17	4
658	2	45	31.68	4
658	3	22	39.78	3
659	1	42	10.11	4



CUSTOMER_ID	EMAIL_ADDRESS	FULL_NAME
1	tammy.bryant@internalmail	Tammy Bryant
2	roy.white@internalmail	Roy White
3	gary.jenkins@internalmail	Gary Jenkins
4	victor.morris@internalmail	Victor Morris
5	beverly.hughes@internalmail	Beverly Hughes

Additional conditions can also be placed in the ‘WHERE’ clause to further restrict the result. The sequence of WHERE clauses doesn’t affect the way that the query is processed.

The query lists customer names, order numbers, and order dates for orders with numbers less than 2200, using the CUSTOMERS, ORDERS, and ORDER_ITEMS tables.

```
SELECT
       C."FULL_NAME",
       O."ORDER_ID",
       O."ORDER_DATETIME",
       O."STORE_ID“
FROM CO."CUSTOMERS" C, CO."ORDERS" O, CO."ORDER_ITEMS" OI
WHERE C."CUSTOMER_ID" = O."CUSTOMER_ID“
AND O."ORDER_ID" = OI."ORDER_ID“
AND O."ORDER_ID" < 2200;
```

Comparison Examples

```
SELECT * FROM table WHERE x BETWEEN 1 AND 3;
SELECT * FROM table WHERE x IN (‘A’,’B’,’C’);
SELECT * FROM table WHERE x NOT IN (‘A’,’B’,’C’);
```

- Use brackets to enforce operator order:

```
WHERE (a=1) AND (b=2 OR b=3)
!=
WHERE (a=1 AND b=2) OR (b=3)
```

Conditions & Operators IN SQL

ANY compares a value against a set of values returned

ANY

```
SELECT DISTINCT
          Cust_name
FROM customer, order
WHERE customer.cust_num = order.cust_num
AND Order_date = ANY ( '13-SEP-10', '15-AUG-10’);
```

IN checks whether a value matches any value

IN

```
SELECT DISTINCT
   c.Cust_Name
FROM customer c
JOIN orders o ON c.cust_num = o.cust_num
WHERE o.Order_Date IN (TO_DATE('13-SEP-10', 'DD-MON-YY'),
TO_DATE('15-AUG-10', 'DD-MON-YY'));
```

Continue…

This tests whether a value lies within a specified range.

BETWEEN

```
SELECT DISTINCT
      Cust_name
FROM customer, order
WHERE customer.cust_num = order.cust_num AND
Order_date BETWEEN `24-JUL-10` AND `30-OCT-10`;
```

This query will return all customer names whose name starts with the letter “N“.

Used for pattern matching with string values.

LIKE

```
SELECT DISTINCT
      Cust_name
FROM customer
WHERE Cust_name LIKE ‘N%’;
```

This query will return all customer names whose name starts with the letter “N“.

Example Tables

- Books

Name	***Number***
Book1	1
Book2	2
Book3	3
Book4	4
Book5	5

Borrowers

Name	***Number***	*Dept*
Anne	1	Maths
Bill	2	Maths
Claire	3	French
Duncan	4	French
Edward	5	French

- Loans

BookNumber	*PersonNumber*
1	2
3	4

SELECT and Calculations

- There are many functions that you can include in a SELECT statement, for example:
  - **`SELECT MAX(Number) FROM Borrowers;`** Shows largest borrower number (Min is also available) more to go in the next slide.
  - **`SELECT Number+1 FROM Borrowers;`** Adds 1 to each borrower number and reports it

SQL Functions

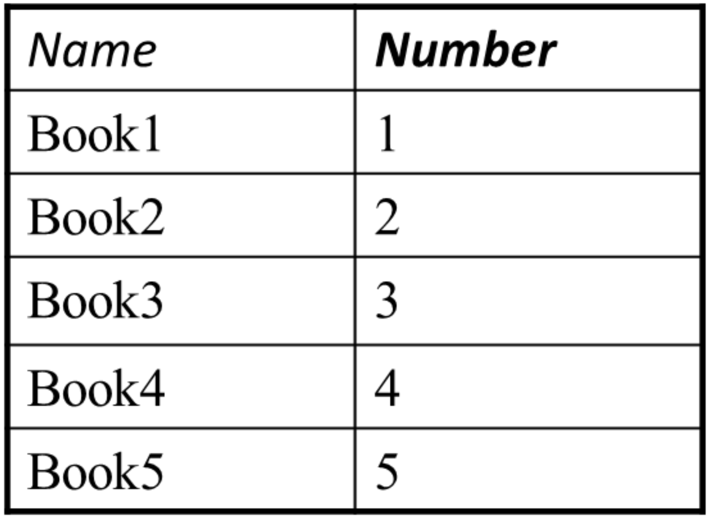
Aggregate Functions

- **AVG()** - Returns the average value
- **COUNT()** - Returns the number of rows
- **FIRST()** - Returns the first value
- **LAST()** - Returns the last value
- **MAX()** - Returns the largest value
- **MIN()** - Returns the smallest value
- **SUM()** - Returns the sum

COUNT

```
SELECT
  COUNT(Number)
from Books;
```

```
Result   = 5
```

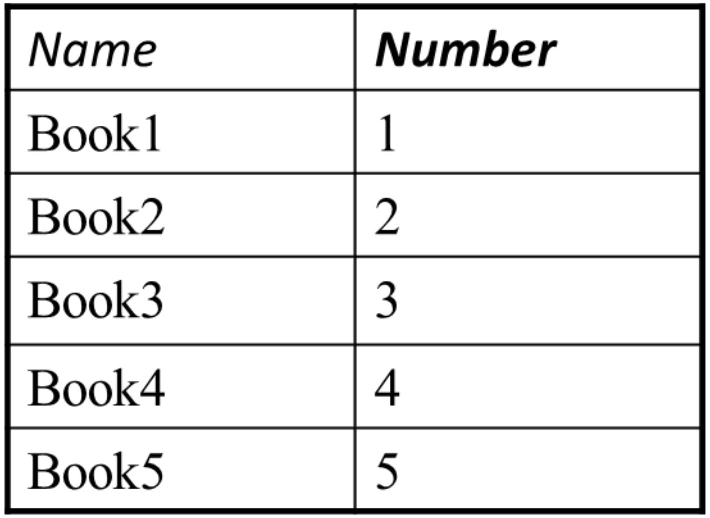


Name	***Number***
Book1	1
Book2	2
Book3	3
Book4	4
Book5	5

AVG

```
Select
  AVG(Number)
from Books;
```

Result = 3



Name	***Number***
Book1	1
Book2	2
Book3	3
Book4	4
Book5	5

Case sensitivity

Quoted strings are case sensitive

```
SELECT
        CUST_NUM,
        CUST_NAME,
        CUST_ADDRESS
FROM CUSTOMER
WHERE CUST_NUM >= 12212;
```

Is the same as:

```
select
        cust_num,
        cust_name,
        cust_address
from customer
where cust_num >= 12212;
```

But:

```
SELECT
        CUST_NUM,
        CUST_NAME,
        CUST_ADDRESS
FROM CUSTOMER
WHERE CUST_ADDRESS = ‘SPINKHILL’;
```

Is different from:

```
SELECT
CUST_NUM,
CUST_NAME,
CUST_ADDRESS
FROM CUSTOMER
WHERE CUST_ADDRESS = ‘Spinkhill’;
```

NESTED Queries

- Who has the book with the **lowest ID** number?

```
SELECT
      Borrowers.Name
FROM  Borrowers, Books, Loans
WHERE Books.Number = (SELECT MIN(Number) FROM Books)
AND Books.Number = Loans.BookNumber
AND Borrowers.Number = Loans.PersonNumber;
```

- The answer, as we would expect, is Bill.
- Note the use of brackets to enclose the sub-SELECT and use of the function **`MIN()`**

ANSI

```
SELECT
    Borrowers.Name
FROM
    Borrowers
JOIN Loans ON Borrowers.Number = Loans.PersonNumber
JOIN Books ON Books.Number = Loans.BookNumber
WHERE Books.Number = (SELECT MIN(Number) FROM Books);
```

This query retrieves the names of borrowers who have borrowed the book with the smallest Number in the Books table.

Sub Queries

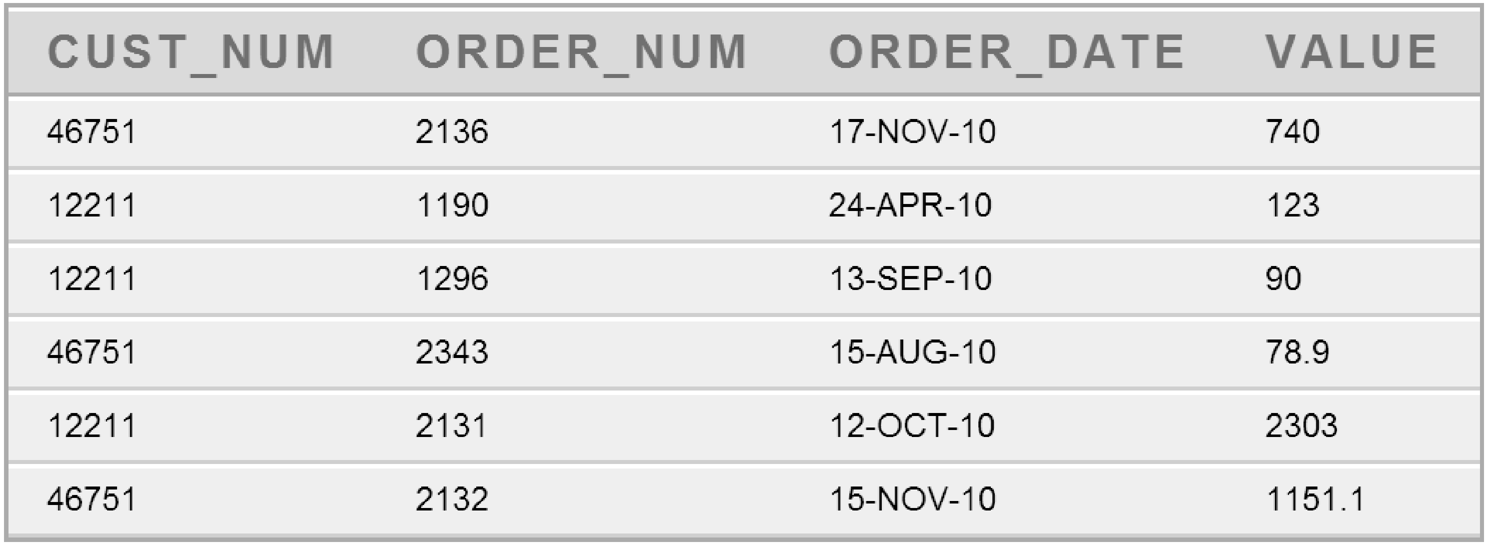
The sub query is evaluated once only:

E.g. Print the orders where the value of the order is above the average value of an order:

```
SELECT
      ORDER_NUM,
      ORDER_DATE
FROM ORDER
WHERE VALUE > ( SELECT AVG(VALUE) FROM ORDER);
```

Functions such as MAX() cannot be incorporated directly into the WHERE clause.

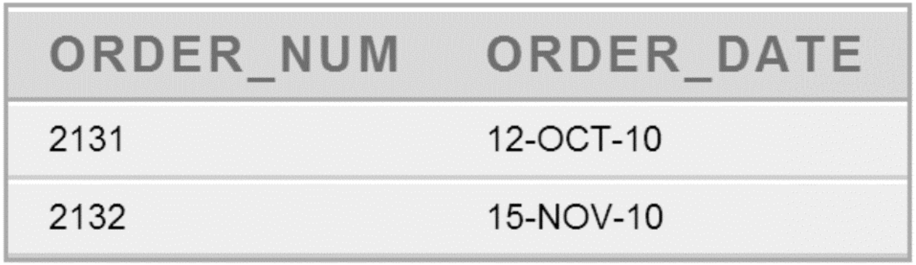
Steps



CUST_NUM	ORDER_NUM	ORDER_DATE	VALUE
46751	2136	17-NOV-10	740
12211	1190	24-APR-10	123
12211	1296	13-SEP-10	90
46751	2343	15-AUG-10	78.9
12211	2131	12-OCT-10	2303
46751	2132	15-NOV-10	1151.1

Approach to resolving a sub-query:

- Calculate the average of the value attribute (747.6)
- Retrieve each of the rows from the table where the value attribute is greater than 747.6
- Project the order_num and order_date attributes.



ORDER_NUM	ORDER_DATE
2131	12-OCT-10
2132	15-NOV-10

Example 2

Part

Part No	Description	Price
223A	*22mm clip*	*0.10*
1212	*22mm con*	*1.50*
6341SS	*3m tube*	*4.50*

List the descriptions of parts where the price is above the average price of a part.

```
SELECT
     Description
FROM part
WHERE price > ( SELECT AVG(price) FROM part);
```

More on Sub Queries

We can use a sub query, but what happens when the sub-query produces more than 1 row?

```
SELECT * FROM Staff
WHERE Name = ANY (SELECT Name FROM Borrowers);
```

Selects staff whose name appears in both Staff and Borrowers tables

```
SELECT * FROM Borrowers
WHERE Number > ALL (SELECT Number FROM Staff);
```

Selects those borrowers who have a higher number than ALL of the staff

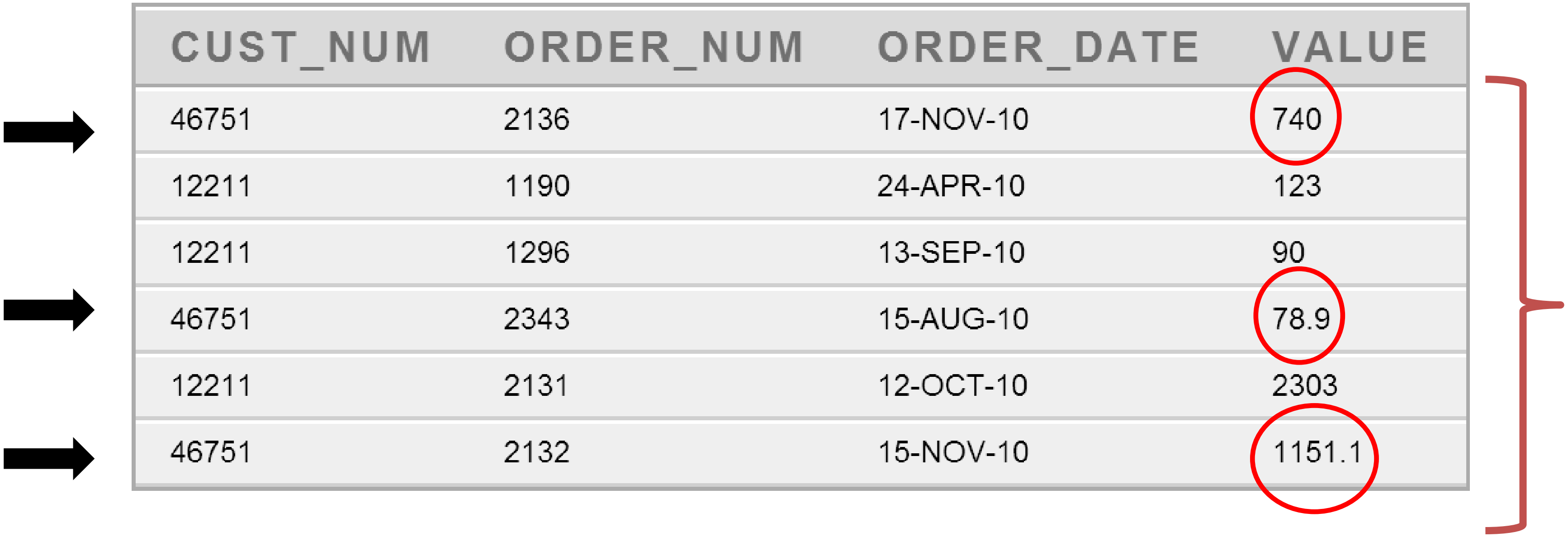
CORRELATED QUERIES

The correlated query is evaluated once for each of the tuples returned by the main query.

Example: Print the orders where the value of the order is above the average value of the orders for that Customer.

```
SELECT *
FROM ORDERV2
WHERE VALUE >  ( SELECT AVG(VALUE)
                 FROM ORDERV2 X
                 WHERE ORDERV2.CUST_NUM = X.CUST_NUM);
```

X IS AN ALIAS FOR **ORDER**

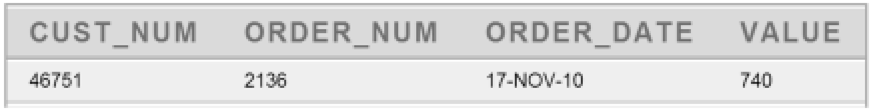


CUST_NUM	ORDER_NUM	ORDER_DATE	VALUE
46751	2136	17-NOV-10	740
12211	1190	24-APR-10	123
12211	1296	13-SEP-10	90
46751	2343	15-AUG-10	78.9
12211	2131	12-OCT-10	2303
46751	2132	15-NOV-10	1151.1

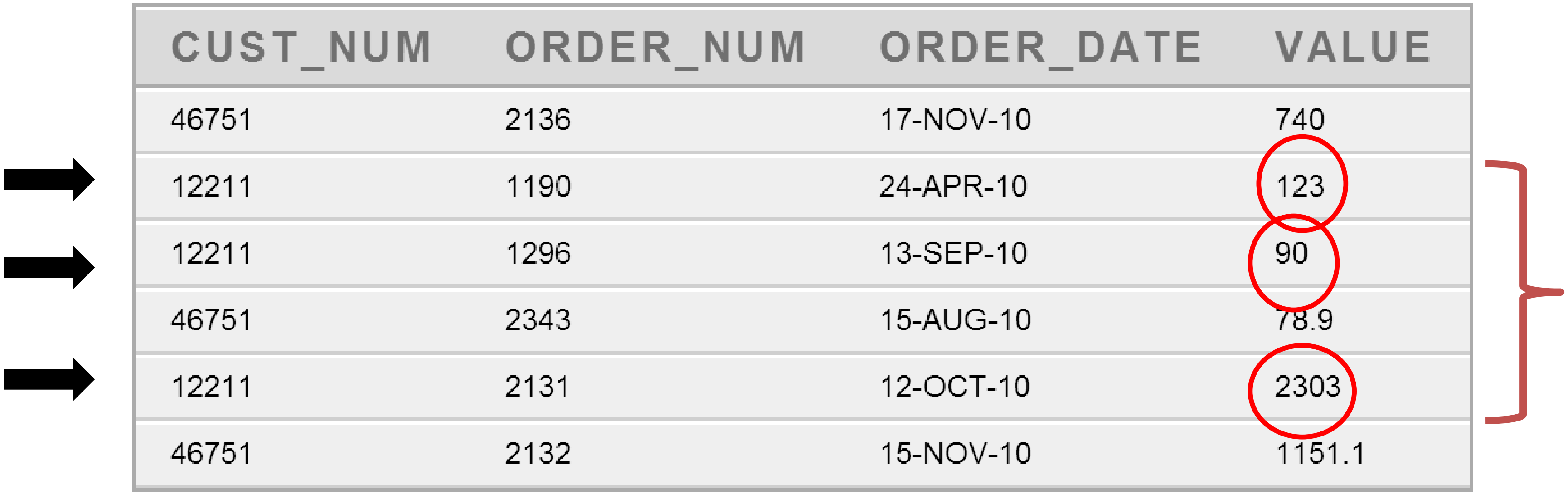
Average = 656.7

Approach to resolving a correlated-query:

1. Retrieve the **first** row
2. Get the **cust_num**
3. Calculate the **average** of the value attribute for all the rows with this cust_num
4. If the value attribute of the current row is greater than the average value calculated, **return the row**
5. **Repeat** for all rows in the table



CUST_NUM	ORDER_NUM	ORDER_DATE	VALUE
46751	2136	17-NOV-10	740

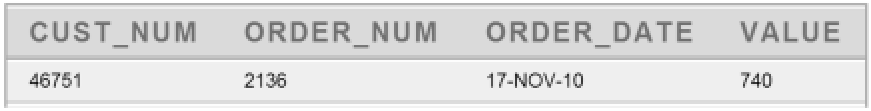


CUST_NUM	ORDER_NUM	ORDER_DATE	VALUE
46751	2136	17-NOV-10	740
12211	1190	24-APR-10	123
12211	1296	13-SEP-10	90
46751	2343	15-AUG-10	78.9
12211	2131	12-OCT-10	2303
46751	2132	15-NOV-10	1151.1

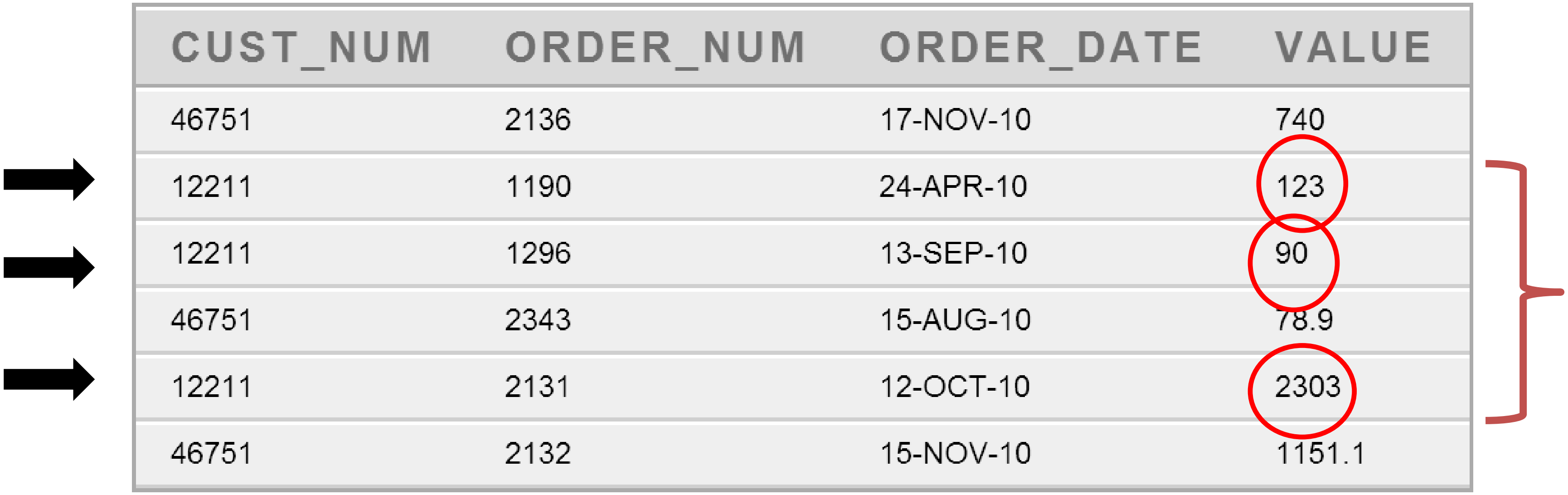
Average = 838.7

Approach to resolving a correlated-query:

1. Retrieve the **first** row
2. Get the **cust_num**
3. Calculate the **average** of the value attribute for all the rows with this cust_num
4. If the value attribute of the current row is greater than the average value calculated, **return the row**
5. **Repeat** for all rows in the table



CUST_NUM	ORDER_NUM	ORDER_DATE	VALUE
46751	2136	17-NOV-10	740

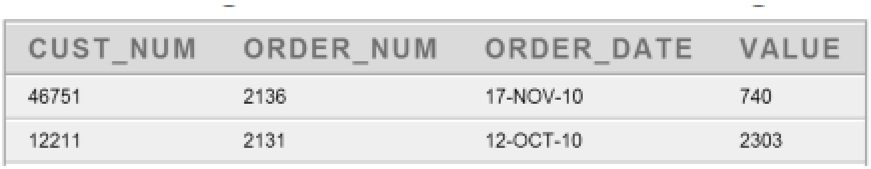


CUST_NUM	ORDER_NUM	ORDER_DATE	VALUE
46751	2136	17-NOV-10	740
12211	1190	24-APR-10	123
12211	1296	13-SEP-10	90
46751	2343	15-AUG-10	78.9
12211	2131	12-OCT-10	2303
46751	2132	15-NOV-10	1151.1

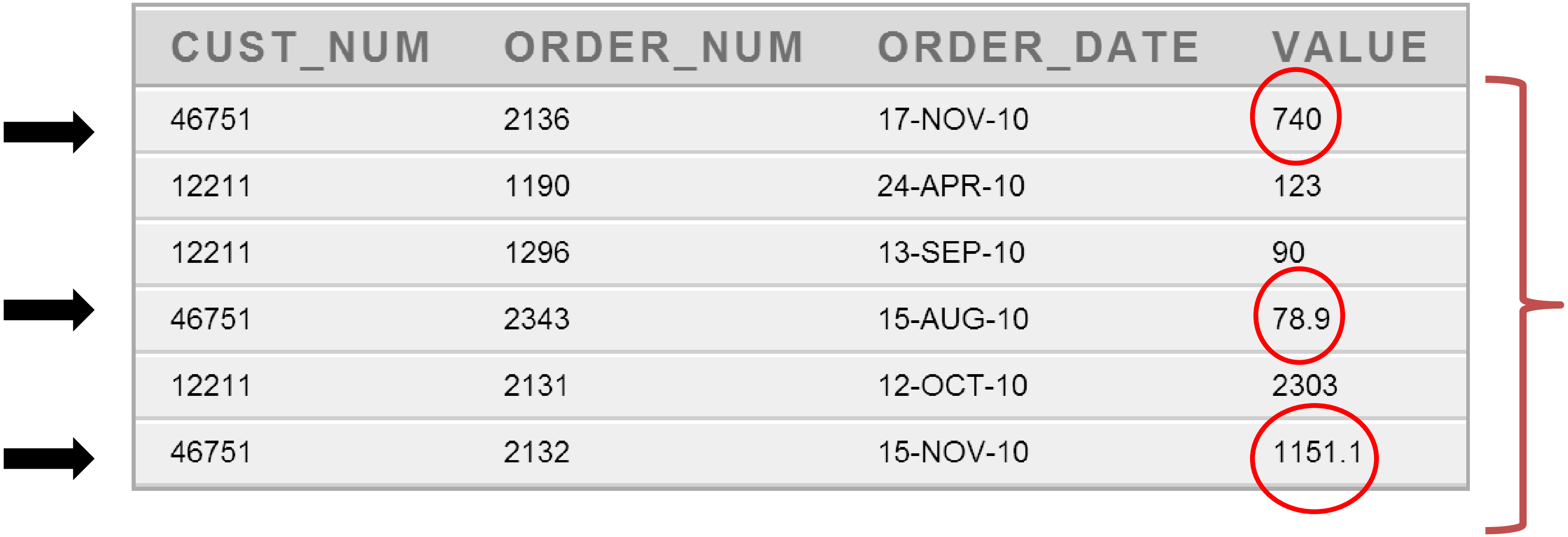
Average = 838.7

Approach to resolving a correlated-query:

1. Retrieve the **first** row
2. Get the **cust_num**
3. Calculate the **average** of the value attribute for all the rows with this cust_num
4. If the value attribute of the current row is greater than the average value calculated, **return the row**
5. **Repeat** for all rows in the table



CUST_NUM	ORDER_NUM	ORDER_DATE	VALUE
46751	2136	17-NOV-10	740
12211	2131	12-OCT-10	2303

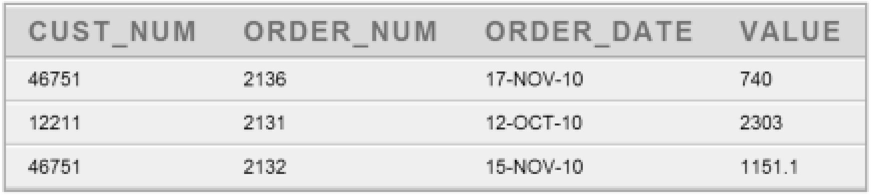


CUST_NUM	ORDER_NUM	ORDER_DATE	VALUE
46751	2136	17-NOV-10	740
12211	1190	24-APR-10	123
12211	1296	13-SEP-10	90
46751	2343	15-AUG-10	78.9
12211	2131	12-OCT-10	2303
46751	2132	15-NOV-10	1151.1

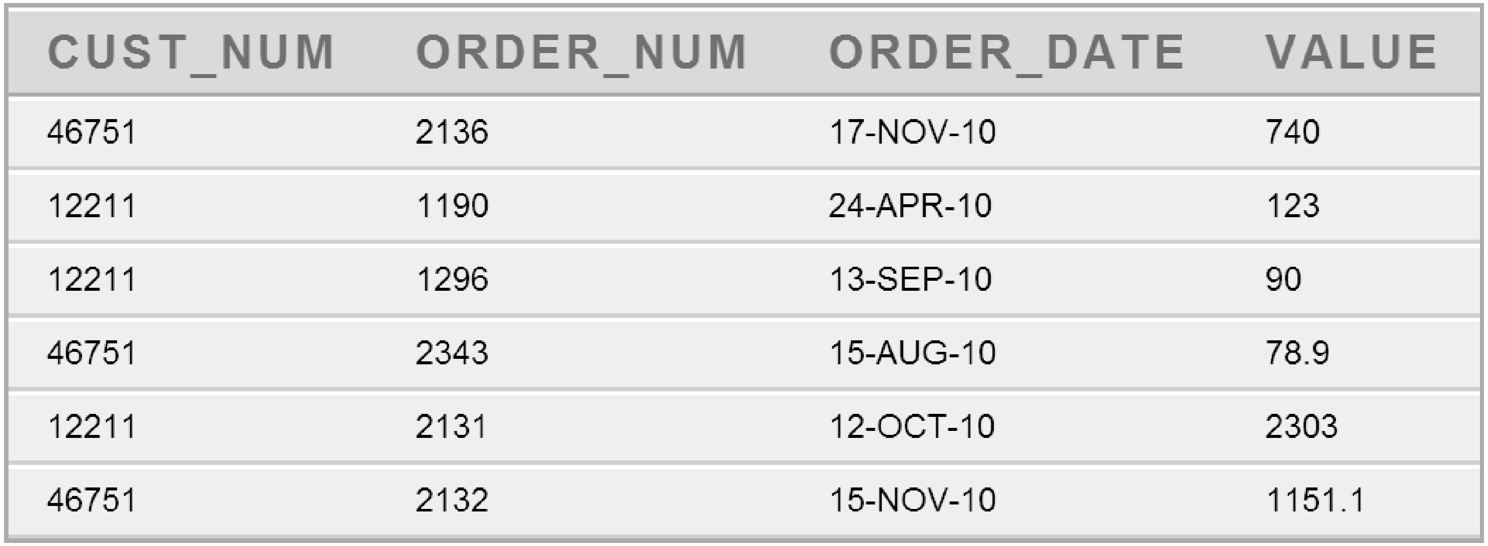
Average = 656.7

Approach to resolving a correlated-query:

1. Retrieve the **first** row
2. Get the **cust_num**
3. Calculate the **average** of the value attribute for all the rows with this cust_num
4. If the value attribute of the current row is greater than the average value calculated, **return the row**
5. **Repeat** for all rows in the table



CUST_NUM	ORDER_NUM	ORDER_DATE	VALUE
46751	2136	17-NOV-10	740
12211	2131	12-OCT-10	2303
46751	2132	15-NOV-10	1151.1



CUST_NUM	ORDER_NUM	ORDER_DATE	VALUE
46751	2136	17-NOV-10	740
12211	1190	24-APR-10	123
12211	1296	13-SEP-10	90
46751	2343	15-AUG-10	78.9
12211	2131	12-OCT-10	2303
46751	2132	15-NOV-10	1151.1

List the details of the most recent order for each customer based on the latest ORDER_DATE

```
SELECT
      CUST_NUM,
      ORDER_NUM,
      VALUE
FROM ORDERV2
WHERE ORDER_DATE = (SELECT MAX(ORDER_DATE)
                   FROM ORDERV2 X
                   WHERE ORDER.CUST_NUM = X.CUST_NUM);
```

SUB QUERIES CAN BE USED TO CREATE JOINS.

```
SELECT DISTINCT
       CUST_NAME
FROM CUSTOMER JOIN “ORDER” ON CUSTOMER.CUST_NUM= ORDER.CUST_NUM
WHERE ORDER_DATE > '12-AUG-20’;
```

has the same effect as:

```
SELECT CUST_NAME
FROM CUSTOMER
WHERE CUST_NUM IN ( SELECT CUST_NUM
                      FROM “ORDER”
                      WHERE ORDER_DATE> '12-AUG-20’);
```

and

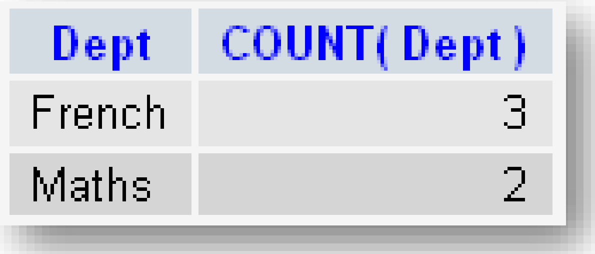
```
SELECT
       CUST_NAME
FROM CUSTOMER C
WHERE EXISTS( SELECT *
               FROM “ORDER”
               WHERE C.CUST_NUM= “ORDER”.CUST_NUM AND ORDER_DATE>
       '12-AUG-20’);
```

The exists (**Boolean operator**) condition is true if the result of evaluating the inner query is not empty.

Groups

- We may want to summarise the data in the database, and there are some statistical functions available.
- We have already seen **MAX, MIN, AVG and COUNT**
- For example, how many people are there from each department?

```
SELECT
      Dept,
      COUNT(Dept)
FROM Borrowers
GROUP BY Dept;
```



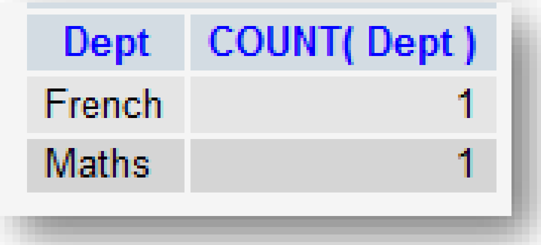
Dept	COUNT(Dept)
French	3
Maths	2

How Many Books Does Each Department Have?

- We could show the list and count them ourselves:

```
SELECT
       Borrowers.Dept,
       Books.Name
       FROM Borrowers, Books, Loans
       WHERE  Books.Number = Loans.BookNumber
       AND Borrowers.Number = Loans.PersonNumber;
```

- But SQL can do it for us:



Dept	COUNT(Dept)
French	1
Maths	1

```
SELECT
       Dept,
       COUNT(Dept)
       FROM Borrowers, Books, Loans
       WHERE  Books.Number = Loans.BookNumber
       AND Borrowers.Number = Loans.PersonNumber
       GROUP BY Dept;
```

Refining Your Selection

- You can select from the resultant table using **HAVING**:

```
SELECT
    Dept,
    COUNT(Dept)
FROM Borrowers, Books, Loans
WHERE  Books.Number=Loans.BookNumber
AND Borrowers.Number=Loans.PersonNumber
GROUP BY Dept
HAVING COUNT(Dept) > 2;
```

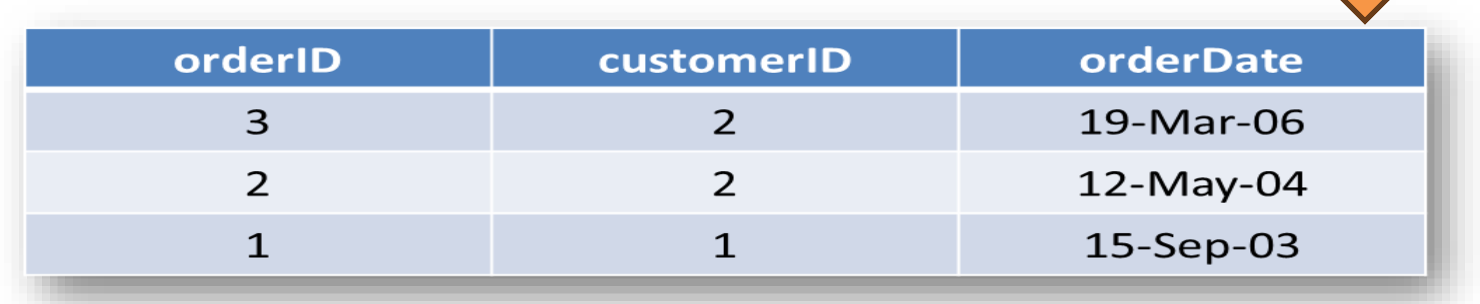
Selection Manipulation

Some things you can do to the selected list:

- Sort the results by one or more fields

```
ORDER BY field, field, …  [DESC]
```

```
SELECT * FROM CustomerOrder BY CustomerName ASC;
```



orderID	customerID	orderDate
3	2	19-Mar-06
2	2	12-May-04
1	1	15-Sep-03

- Force a query to contain unique entries only:

```
SELECT
        DISTINCT CustomerName
FROM Customer;
```

customerName
Apple
Microsoft
Google

Aliases

- You can give an alias to a table or column to make it easier to refer to later in a query or to make it easier to read in the output:

```
SELECT
    AVG(Number)
FROM books;
```

```
AVG(Number)    3
```

```
SELECT
    AVG(Number) AS Average
FROM books;
```

```
Average    3
```

```
SELECT
    AVG(‟Number”) AS "Average Value‟
FROM books;
```

Number might be a reserved keyword in Oracle, so use double quotes.

Aliases (contd)

```
SELECT Role, COUNT(Role)
     FROM staff
     GROUP BY Role
     HAVING COUNT(Role) > 1;
```

Or

AS Count

Can be re-written as:

```
SELECT Role, COUNT(Role) AS COUNT
     FROM Staff
     GROUP BY Role
     HAVING COUNT (Role)> 1;
```

- Where count is an alias for COUNT(Role)

Aliases (contd)

- These are also aliases we have been doing…

```
SELECT *
      FROM Books a, Borrowers b
      WHERE a.Number = b.Number;
```

- We have made aliases for the tables to make reference to them easier.

Course Content

1. Introduction to Relational Databases *(Introduction + Relational Model)*
2. Data Modelling - *(Entity Relationship Modelling + The Enhanced Entity Relationship Model)*
3. Database Design and SQL - *(Logical modelling + Introduction to SQL)*
4. Further SQL - *(Advanced SQL queries + Creating tables with SQL)*
5. Normalisation - *(Normalisation to second normal form + Third normal form)*

Thank you

X THE PLACE OF USEFUL LEARNING

University of Strathclyde Glasgow

Database Fundamentals – CS990

Database and Web Systems Development - CS952

Image source: https://www.strath.ac.uk/branding/