Sub-queries with IN and EXISTS

Query results

- When a query is run, it returns a result set.
- Usually, it is a 2-D matrix; rows and columns.

```
Select * from stock;
```

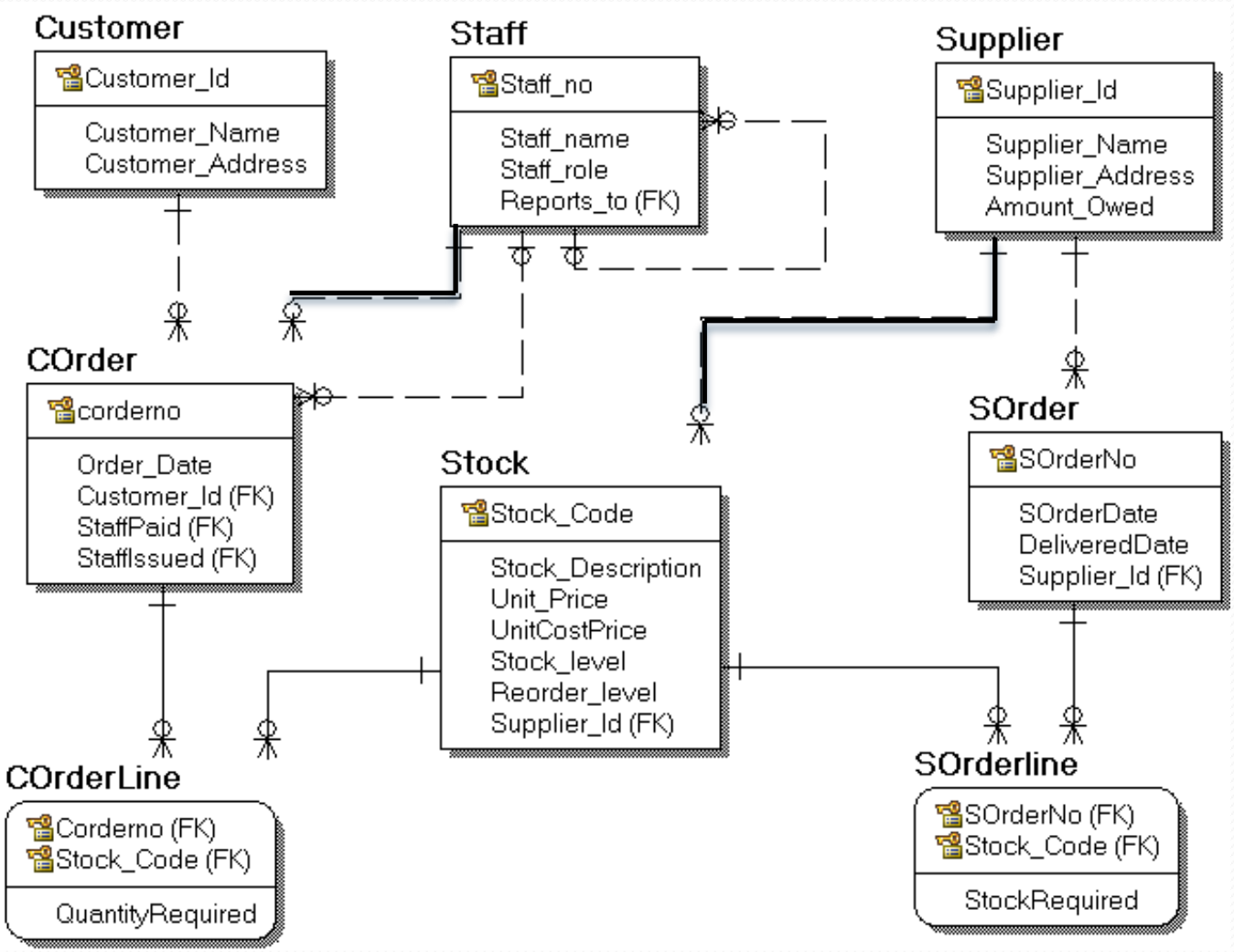
- If it is an aggregate query, with no sub-division, it will be a vector – i.e. one row, with multiple columns.

```
Select max(unit_price), count(*),
min(stock_level) from stock;
```

Query results

- It can be a vertical vector – i.e. a single column from each row.
  - **`select supplier_Id from stock;`**
- If it has only one column and one row, it is a scalar.
  - **`Select max(unit_price) from stock;`**

Selects using Builder schema



Sub-selects

- A select statement within a search_condition is called a **subselect** statement.
- The whole structure is often referred to as a **nested** select statement.
- SQL supports several built-in **predicates** which can be used to test the sub-selects for certain conditions.
- We shall consider the following predicates:
  - **in**,
  - **not in**,
  - **exists**,
  - **not exists**.

In and Not in

- The general forms of **in** and **not in** predicates:
  1. expr in (sub-select)
  2. expr in (value {, value})
  3. expr not in (sub-select)
  4. expr not in (value {, value})
- In will check the sub-query on the right for the presence of the expression on its left.

Expr in (sub-select)

Return the names of customers who have not collected their orders.

```
Select customer_name
From customer
Where Customer_id in
(select customer_id
from corder
where staffissued is null);
```

Running the inner SELECT

```
select customer_id
from corder
where staffissued
  is null;
```

```
CUSTOMER_ID
----------
5
5
5
5
5
5
5
5
5
5

10 rows selected
```

And the outer SELECT….

```
Select customer_name
  from customer
Where Customer_id in(
5
5
…
5
5
5
5)
```

```
CUSTOMER_NAME
--------------

Mary Martin


1 rows selected
```

Expr not in (sub-select)

- Return the supplier name of all suppliers that do not supply any stock:

```
SELECT supplier_name
FROM supplier
WHERE supplier_id NOT IN
(SELECT supplier_id from STOCK);
```

Check out two selects:

```
SELECT supplier_id from
  STOCK;
   SUPPLIER_ID
   -----------
   501
   501
   501
   504
   508
   510
   510
   511
   506
   501
   501
   510
   508
   505
   511
   511
```

```
Select supplier_name,
Supplier_id from Supplier;
```

SUPPLIER_NAME	SUPPLIER_ID
Buckleys	501
Brendan Moore	502
James McGovern	503
Liam Keenan	504
Mary O'Brien	505
Oliver Moore	506
Robert O'Mahony	507
Patricia O'Brien	508
June Browne	509
Paul Sloan	510
Kevin Kelly	511

And the final…

```
SELECT supplier_name
FROM supplier
WHERE supplier_id NOT IN
(SELECT supplier_id from STOCK);
```

Yields

```
SUPPLIER_NAME
-------------------------
Brendan Moore
James McGovern
Robert O'Mahony
June Browne

4 rows selected
```

BEWARE “NOT IN”

- When the list / sub-query on the right of “NOT IN” has NULL values NOT IN will not match any rows.
- In this case you can use:
  - LEFT JOIN …. WHERE … IS NULL or
  - NOT EXISTS (read on…)

…question…

EXISTS and correlated sub-queries

Correlated sub-queries

- This is where the inner select statement refers to data defined in the outer select statement.

EXISTS condition

- **This returns a Boolean value of true or false.**
- The result is true if the sub-query returns a **non-empty** set of values.
- The sub-query is generally correlated to the outer query.
  - Otherwise, the mere existence of rows in the sub-query would not normally be relevant.

SQL: EXISTS Condition

- The EXISTS condition is considered "to be met" if the sub-query returns at least one row.
- The syntax for the EXISTS condition is:

```
SELECT columns
  FROM tables
  WHERE EXISTS ( subquery );
```

- The EXISTS condition can be used in any valid SQL statement - select, insert, update, or delete.

Not correlated sub-query

- E.g.

```
select * from stock where exists
(select sysdate from dual);
```

- This is the same as

```
Select * from stock;
```

because the sub-query always returns a value, so the ‘exists’ clause is true.

Example #1

- Let's take a look at a simple example. The following is an SQL statement that uses the EXISTS condition:

```
SELECT *
  FROM supplier
  WHERE EXISTS
     (select *
        from sorder
        where supplier.supplier_id =
  sorder.supplier_id);
```

Example #1

- This select statement will return all records from the suppliers table where there is at least one record in the sorder table with the same supplier_id.
- It is a *correlated sub-query*

Example #2 - NOT EXISTS

- The EXISTS condition can also be combined with the NOT operator.
- For example,

```
SELECT *
FROM supplier
WHERE NOT EXISTS
    (SELECT * FROM sorder WHERE
supplier.supplier_id =
sorder.supplier_id
    );
```

Example #2 – NOT EXISTS

- This will return all records from the suppliers table where there are **no** records in the *orders* table for the given supplier_id.

Example #3 - DELETE Statement

- The following is an example of a delete statement that utilizes the EXISTS condition:

```
DELETE FROM supplier
  WHERE not EXISTS
     (select *
       from sorder
       where supplier.supplier_id =
  sorder.supplier_id);
```

…question…

Building queries

Sub-query returning a scalar

```
SELECT stock_code, stock_description,
  unit_price
FROM stock
WHERE unit_price > (SELECT
  AVG(unit_price) FROM stock);
```

This query returns the stock code, description and price of all stock items that cost more than the average unit price of any stock item.

Scalar SELECT with an alias

- This is where a SELECT statement that returns a SCALAR (single row and column) is used as an attribute:

```
SELECT staff_name,
   staff_role,
   (SELECT staff_name FROM staff boss WHERE
  boss.staff_no = emp.reports_to
   ) AS answers_to
FROM staff emp;
```

Correlated sub-query

```
SELECT stock_code, stock_description,
  unit_price, supplier_id
FROM stock os
WHERE unit_price > (
 SELECT AVG(unit_price) FROM stock WHERE
  supplier_id = os.supplier_id);
```

- This returns the stock code, description, price and supplier id of all stock items that cost more than the average stock item supplied by that supplier.
- Question: Why would you do that?

Demo

```
Select supplier_id, avg(unit_price)
  from stock group by supplier_id;
```

```
SUPPLIER_ID       AVG(UNIT_PRICE)
--------------------- -
504                    9.5
510                    8.46666666666667
505                    8
508                   25
506                  200
511                   98.33
501                  324.5
7 rows selected
```

Full SELECT statement

CODE	STOCK_DESC	UNIT_PRICE	SUPPLIER_ID
B111	Window Frames 2'x4'	45	508
C121	6" Nails(100)	9.95	510
D101	Workbench	250	511
E101	Cavity blocks(500)	1000	501
E141	Cavity blocks(200)	400	501
A642	4"x4" treated timber	9.5	510

6 rows selected