sql-intro-3

February 27, 2018

First, we create the tables for this section.
Note: Python notebooks are independent from each other

```
In [1]: %load_ext sql
        # Connect to an empty SQLite database
        %sql sqlite://
```

```
Out[1]: 'Connected: None@None'
```

```
In [2]: %%sql  -- Find all tables in the database
        SELECT name FROM sqlite_master WHERE type='table';
```

```
Done.
```

```
Out[2]: []
```

```
In [3]: %%sql

        -- Create tables
        DROP TABLE IF EXISTS Company;
        CREATE TABLE Company (
            CName varchar(255) NOT NULL PRIMARY KEY,
            StockPrice FLOAT,
            Country varchar(255)
        );

        DROP TABLE IF EXISTS Product;
        CREATE TABLE Product (
            PName VARCHAR(255) NOT NULL PRIMARY KEY,
            Price FLOAT,
            Category VARCHAR(255),
            Manufacturer VARCHAR(255)
        );

        DROP TABLE IF EXISTS Purchase;
        CREATE TABLE Purchase(
            id varchar(255) PRIMARY KEY,
```

```
        product varchar(255) NOT NULL,
        buyer varchar(255) NOT NULL,
        FOREIGN KEY (product) REFERENCES Product(PName)
    );

    -- Insert tuples
    INSERT INTO Company VALUES ('GWorks', 25, 'USA');
    INSERT INTO Company VALUES ('Canon', 65, 'Japan');
    INSERT INTO Company VALUES ('Hitachi', 15, 'Japan');
    INSERT INTO Company VALUES ('IBM', 140, 'USA');

    INSERT INTO Product VALUES ('Gizmo', 19.99, 'Gadgets', 'GWorks');
    INSERT INTO Product VALUES ('Powergizmo', 29.99, 'Gadgets', 'GWorks');
    INSERT INTO Product VALUES ('SingleTouch', 149.99, 'Photography', 'Canon');
    INSERT INTO Product VALUES ('MultiTouch', 203.99, 'Household', 'Hitachi');

    INSERT INTO Purchase VALUES (1, 'Gizmo', 'Joe Blow');
    INSERT INTO Purchase VALUES (2, 'Gizmo', 'Joe Blow');
    INSERT INTO Purchase VALUES (3, 'SingleTouch', 'Mr Smith');
    INSERT INTO Purchase VALUES (4, 'MultiTouch', 'Mr Smith');
    INSERT INTO Purchase VALUES (5, 'Gizmo', 'Mr Smith');
```

```
Done.
Done.
Done.
Done.
Done.
Done.
Done.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
```

Out[3]: []

In [4]:
```
%%sql  -- A useful way to retrieve information regarding a table
pragma table_info(Company);
```

```
Done.
```

```
Out[4]: [(0, 'CName', 'varchar(255)', 1, None, 1),
         (1, 'StockPrice', 'FLOAT', 0, None, 0),
         (2, 'Country', 'varchar(255)', 0, None, 0)]

In [5]: %%sql  -- A useful way to retrieve information regarding a table
        pragma table_info(Product);

Done.


Out[5]: [(0, 'PName', 'VARCHAR(255)', 1, None, 1),
         (1, 'Price', 'FLOAT', 0, None, 0),
         (2, 'Category', 'VARCHAR(255)', 0, None, 0),
         (3, 'Manufacturer', 'VARCHAR(255)', 0, None, 0)]

In [6]: %%sql  -- A useful way to retrieve information regarding a table
        pragma table_info(Purchase);

Done.


Out[6]: [(0, 'id', 'varchar(255)', 0, None, 1),
         (1, 'product', 'varchar(255)', 1, None, 0),
         (2, 'buyer', 'varchar(255)', 1, None, 0)]

In [7]: %%sql -- Another set of tables

        -- Create tables
        DROP TABLE IF EXISTS P;
        CREATE TABLE P (
            A INT
        );

        DROP TABLE IF EXISTS Q;
        CREATE TABLE Q (
            B INT,
            C INT
        );

        -- Insert tuples
        INSERT INTO P VALUES (1);
        INSERT INTO P VALUES (3);

        INSERT INTO Q VALUES (2,3);
        INSERT INTO Q VALUES (3,4);
        INSERT INTO Q VALUES (3,5);

Done.
Done.
```

```
Done.
Done.
Done.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
```

```
Out[7]: []
```

```
In [8]: %sql SELECT name FROM sqlite_master WHERE type='table';
```

```
Done.
```

```
Out[8]: [('Company',), ('Product',), ('Purchase',), ('P',), ('Q',)]
```

0.1 SQL Aliases

Used to give a table, or a column in a table, a temporary name. * Make column names more readable. * Only exist for the duration of the query.

Alias Column Syntax

```
SELECT column_name [AS] alias_name
FROM table_name;
```

Alias Table Syntax

```
SELECT column_name(s)
FROM table_name [AS] alias_name;
```

---

1 Multi-Table Queries

In this section 1. Join: basics 2. Joins: SQL semantics 3. Set operators 4. Nested queries 5. Aggregation and GROUP BY

1.1 Joins

A **join** between tables returns all unique combinations of their tuples **which meet some specified join condition**.

```
Product(PName, Price, Category, Manufacturer)
Company(CName, StockPrice, Country)
```

Note: Omitted attribute types in schema for brevity.

Example: Find all products under $200 manufactured in Japan; return their names and prices.

```
SELECT PName, Price
FROM   Product, Company
WHERE  Manufacturer = CName
       AND Country = 'Japan'
       AND Price <= 200
```

Example: Find all products under $200 manufactured in Japan; return their names and prices.


```
SELECT PName, Price
FROM   Product, Company
WHERE  Manufacturer = CName
       AND Country = 'Japan'
       AND Price <= 200
```


Multiple equivalent ways to write a basic join in SQL. Example:

```
SELECT PName, Price,
FROM   Product
JOIN   Company ON Manufacturer = CName
                  AND Country = 'Japan'
WHERE Price <= 200
```

Example: Find all products under $200 manufactured in Japan; return their names and prices.

```
In [9]: %%sql -- Product Table
        SELECT *
        FROM   Product;

Done.


Out[9]: [('Gizmo', 19.99, 'Gadgets', 'GWorks'),
         ('Powergizmo', 29.99, 'Gadgets', 'GWorks'),
         ('SingleTouch', 149.99, 'Photography', 'Canon'),
         ('MultiTouch', 203.99, 'Household', 'Hitachi')]

In [10]: %%sql -- Company Table
         SELECT *
         FROM   Company;

Done.


Out[10]: [('GWorks', 25.0, 'USA'),
          ('Canon', 65.0, 'Japan'),
          ('Hitachi', 15.0, 'Japan'),
          ('IBM', 140.0, 'USA')]
```

Example: Find all products under $200 manufactured in Japan; return their names and prices. |

```
In [11]: %%sql
         SELECT PName, Price
         FROM   Product, Company
         WHERE  Manufacturer = CName
                AND Country = 'Japan'
                AND Price <= 200;

Done.


Out[11]: [('SingleTouch', 149.99)]
```

1.1.1 Tuple Variable Ambiguity in Multi-Table

Consider the following tables with the same attributes:

```
Person(name, address, worksfor)
Company(name, address)
```

In the query:

```
SELECT DISTINCT name, address
FROM            Person, Company
WHERE           worksfor = name
```

Which name/address does this refer to?

```
Person(name, address, worksfor)
Company(name, address)
```

Equivalent ways to resolve *variable ambiguity* :

```
SELECT DISTINCT name, address
FROM            Person, Company
WHERE           Person.worksfor = Company.name
```

```
SELECT DISTINCT name, address
FROM            Person p, Company c
WHERE           p.worksfor = c.name
```

1.2 Meaning (semantics) of SQL Queries

```
SELECT P.A
FROM   P, Q
WHERE  P.A = Q.B
```

Note the **semantics** of a join

Table P

A
1
3

Table Q

B	C
2	3
3	4
3	5

example

```
SELECT P.A FROM P, Q WHERE P.A = Q.B
```

Note: Single due to space limitations. *Best practice* is to use **multiple lines**.

Table P

A
1
3

Table Q

B	C
2	3
3	4
3	5

1. Take the **cross product** $X = P \times Q$:

P × Q

A	B	C
1	2	3
1	3	4
1	3	5
3	2	3
3	3	4
3	3	5

- Recall: Cross product ($A \times B$) is the set of all unique tuples in A,B
- Ex: $\{a, b, c\} \times \{1, 2\} = \{(a, 1), (a, 2), (b, 1), (b, 2), (c, 1), (c, 2)\}$

```
SELECT P.A FROM P, Q WHERE P.A = Q.B
```

Table P

A
1
3

Table Q

B	C
2	3
3	4
3	5

2. Apply **selection / conditions**:

$Y = \{(p, q) \in X | p.A == q.B\}$

- Filtering!

```
SELECT P.A FROM P, Q WHERE P.A = Q.B
```

A	B	C
3	3	4
3	3	5

example

A
3
3

example

Table P

A
1
3

Table Q

B	C
2	3
3	4
3	5

3. Apply **projections** to get the final result:

$Z = \{(y.A) \text{ for } y \in Y\}$

- Return only *some* attributes

- Remembering these steps is critical for understanding the output of certain queries.
- We say *"semantics"*, not *"execution order"*
- We showed *what a join means*, **NOT** *how the DBMS executes it under the covers*

```
In [12]: %%sql
         SELECT P.A
         FROM   P, Q
         WHERE  P.A = Q.B

Done.


Out[12]: [(3,), (3,)]
```

1.2.1 A subtlety about Joins

Example: Find all countries that manufacture some product in the 'Gadgets' category.

```
Product(PName, Price, Category, Manufacturer)
Company(CName, StockPrice, Country)
```

```
In [13]: %%sql
         SELECT *
         FROM   Product;
```

```
Done.
```

```
Out[13]: [('Gizmo', 19.99, 'Gadgets', 'GWorks'),
          ('Powergizmo', 29.99, 'Gadgets', 'GWorks'),
          ('SingleTouch', 149.99, 'Photography', 'Canon'),
          ('MultiTouch', 203.99, 'Household', 'Hitachi')]
```

```
In [14]: %%sql
         SELECT *
         FROM   Company;
```

```
Done.
```

```
Out[14]: [('GWorks', 25.0, 'USA'),
          ('Canon', 65.0, 'Japan'),
          ('Hitachi', 15.0, 'Japan'),
          ('IBM', 140.0, 'USA')]
```

Example: Find all countries that manufacture some product in the 'Gadgets' category.

```
Product(PName, Price, Category, Manufacturer)
Company(CName, StockPrice, Country)

SELECT Country
FROM   Product, Company
WHERE  Manufacturer=CName AND Category='Gadgets'
```

Example: Find all countries that manufacture some product in the 'Gadgets' category.

```
In [15]: %%sql -- Cross product
         SELECT *
         FROM   Product, Company;
```

```
Done.
```

```
Out[15]: [('Gizmo', 19.99, 'Gadgets', 'GWorks', 'GWorks', 25.0, 'USA'),
          ('Gizmo', 19.99, 'Gadgets', 'GWorks', 'Canon', 65.0, 'Japan'),
          ('Gizmo', 19.99, 'Gadgets', 'GWorks', 'Hitachi', 15.0, 'Japan'),
          ('Gizmo', 19.99, 'Gadgets', 'GWorks', 'IBM', 140.0, 'USA'),
          ('Powergizmo', 29.99, 'Gadgets', 'GWorks', 'GWorks', 25.0, 'USA'),
          ('Powergizmo', 29.99, 'Gadgets', 'GWorks', 'Canon', 65.0, 'Japan'),
          ('Powergizmo', 29.99, 'Gadgets', 'GWorks', 'Hitachi', 15.0, 'Japan'),
          ('Powergizmo', 29.99, 'Gadgets', 'GWorks', 'IBM', 140.0, 'USA'),
          ('SingleTouch', 149.99, 'Photography', 'Canon', 'GWorks', 25.0, 'USA'),
          ('SingleTouch', 149.99, 'Photography', 'Canon', 'Canon', 65.0, 'Japan'),
          ('SingleTouch', 149.99, 'Photography', 'Canon', 'Hitachi', 15.0, 'Japan'),
          ('SingleTouch', 149.99, 'Photography', 'Canon', 'IBM', 140.0, 'USA'),
```

```
 ('MultiTouch', 203.99, 'Household', 'Hitachi', 'GWorks', 25.0, 'USA'),
 ('MultiTouch', 203.99, 'Household', 'Hitachi', 'Canon', 65.0, 'Japan'),
 ('MultiTouch', 203.99, 'Household', 'Hitachi', 'Hitachi', 15.0, 'Japan'),
 ('MultiTouch', 203.99, 'Household', 'Hitachi', 'IBM', 140.0, 'USA')]
```

Example: Find all countries that manufacture some product in the 'Gadgets' category.

```
In [16]: %%sql -- Filtering
         SELECT *
         FROM   Product, Company
         WHERE  Manufacturer=CName
                AND Category='Gadgets';

Done.


Out[16]: [('Gizmo', 19.99, 'Gadgets', 'GWorks', 'GWorks', 25.0, 'USA'),
          ('Powergizmo', 29.99, 'Gadgets', 'GWorks', 'GWorks', 25.0, 'USA')]
```

Example: Find all countries that manufacture some product in the 'Gadgets' category.

```
In [17]: %%sql -- Projection
         SELECT Country
         FROM   Product, Company
         WHERE  Manufacturer=CName
                AND Category='Gadgets';

Done.


Out[17]: [('USA',), ('USA',)]
```

What is worng with the resut of the query? How do we solve this?

1.3 Explicit Set Operators

```
In [18]: # Create tables & insert some numbers
         %sql DROP TABLE IF EXISTS R; DROP TABLE IF EXISTS S; DROP TABLE IF EXISTS T;
         %sql CREATE TABLE R (A int); CREATE TABLE S (A int); CREATE TABLE T (A int);
         for i in range(1,6):
             %sql INSERT INTO R VALUES (:i)
         for i in range(1,10,2):
             %sql INSERT INTO S VALUES (:i)
         for i in range(1,11,3):
             %sql INSERT INTO T VALUES (:i)
```

```
Done.
Done.
Done.
Done.
Done.
Done.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
1 rows affected.
```

```
In [19]: %sql SELECT * FROM R;
```

```
Done.
```

```
Out[19]: [(1,), (2,), (3,), (4,), (5,)]
```

```
In [20]: %sql SELECT * FROM S;
```

```
Done.
```

```
Out[20]: [(1,), (3,), (5,), (7,), (9,)]
```

```
In [21]: %sql SELECT * FROM T;
```

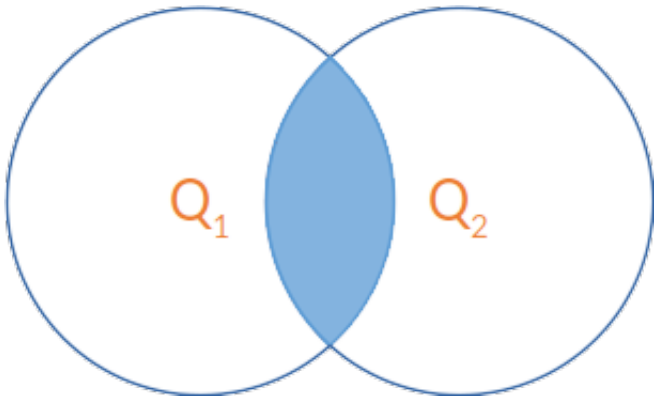
```
Done.
```

```
Out[21]: [(1,), (4,), (7,), (10,)]
```

1.4 INTERSECT

To combine two SELECT statements, returns only common rows returned by the two SELECT statements.

Syntax



intersect

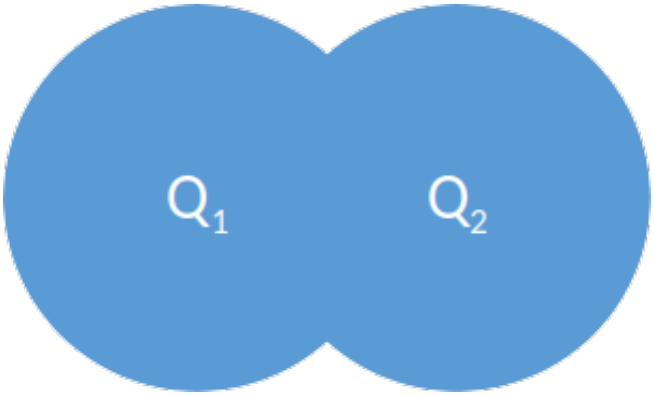
R.A	S.A	T.A
1	1	1
2	3	4
3	5	7
4	7	10
5	9	

example

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]

INTERSECT

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

INTERSECT
$\{r.A | r.A = s.A\} \cap \{r.A | r.A = t.A\}$

```
In [22]: %%sql
         SELECT R.A
         FROM   R, S
         WHERE  R.A=S.A
         INTERSECT
         SELECT R.A
         FROM   R, T
         WHERE  R.A=T.A;

Done.
```



union

R.A
1
2
3
4
5

S.A
1
3
5
7
9

T.A
1
4
7
10

example

```
Out[22]: [(1,)]
```

1.5 UNION

To combine the results of two or more SELECT statements without returning any duplicate rows.

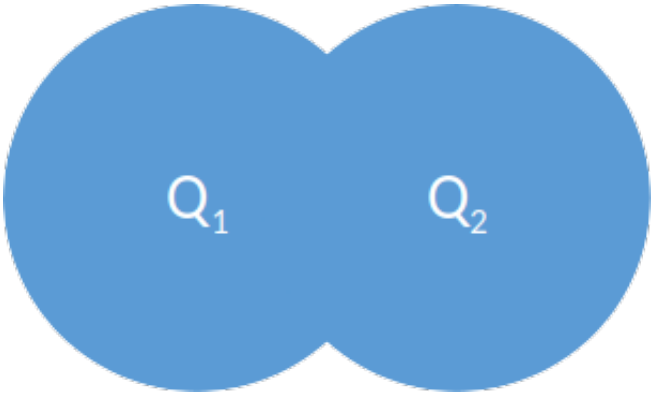
Syntax

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

UNION
$\{r.A|r.A = s.A\} \cup \{r.A|r.A = t.A\}$
To use this UNION clause, each SELECT statement must have

- The same number of columns projected
- The same data type and
- Have them in the same order

```
In [23]: %%sql
         SELECT R.A
         FROM   R, S
         WHERE  R.A=S.A
```



union_all

```
          UNION
          SELECT R.A
          FROM   R, T
          WHERE  R.A=T.A;
```

```
Done.
```

```
Out[23]: [(1,), (3,), (4,), (5,)]
```

What if we want duplicates?

1.6 UNION ALL

The UNION ALL operator is used to combine the results of two SELECT statements including duplicate rows.

The same rules that apply to the UNION clause will apply to the UNION ALL operator.

Syntax

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

UNION ALL

$\{r.A | r.A = s.A\} \cup \{r.A | r.A = t.A\}$

```
In [24]: %%sql
          SELECT R.A
          FROM   R, S
          WHERE  R.A=S.A
          UNION ALL
          SELECT R.A
          FROM   R, T
          WHERE  R.A=T.A;
```

R.A
1
2
3
4
5

S.A
1
3
5
7
9

T.A
1
4
7
10

example

```
Done.
```

```
Out[24]: [(1,), (3,), (5,), (1,), (4,)]
```

1.7 EXCEPT

To combine two SELECT statements, returns only the rows which are not available in the second SELECT statement.

Syntax

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]

EXCEPT

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Note: EXCEPT, not supported by MySQL

EXCEPT

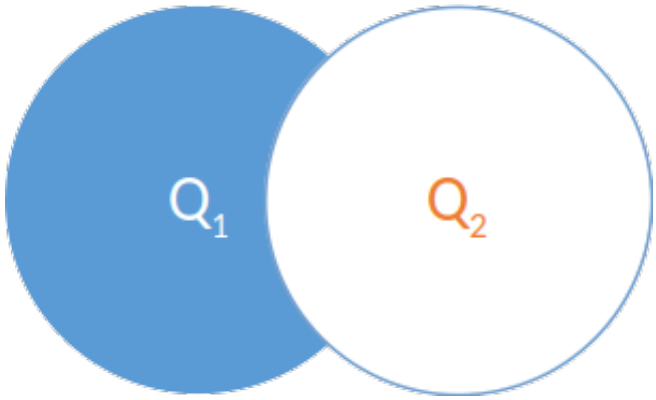
$\{r.A | r.A = s.A\} \setminus \{r.A | r.A = t.A\}$

(Set Difference)

```
In [25]: %%sql
         SELECT R.A
         FROM   R, S
         WHERE  R.A=S.A
         EXCEPT
         SELECT R.A
         FROM   R, T
         WHERE  R.A=T.A;
```

```
Done.
```

```
Out[25]: [(3,), (5,)]
```



except

R.A	S.A	T.A
1	1	1
2	3	4
3	5	7
4	7	10
5	9	

example

1.8 INTERSECT: some subtle problems

Consider the following relations

```
Company(name, hq_city) AS C
Product(pname, maker, factory_loc) AS P
```

Example: Return the headquarters of companies which make gizmos in US **AND** China

```
SELECT hq_city              -- Query 1
FROM   Company, Product
WHERE  maker = name
       AND factory_loc = 'US'
INTERSECT
SELECT hq_city              -- Query 2
FROM   Company, Product
WHERE  maker = name
       AND factory_loc = 'China'
```

Example: Return the headquarters of companies which make gizmos in US **AND** China

```
SELECT hq_city              -- Query 1
```

```
FROM   Company, Product
WHERE  maker = name
       AND factory_loc = 'US'
INTERSECT
SELECT hq_city              -- Query 2
FROM   Company, Product
WHERE  maker = name
       AND factory_loc = 'China'
```

- What if two companies have HQ in US: BUT one has factory in China (but not US) and vice versa?
- **What goes wrong?**
  `C JOIN P on maker = name`

C.name	C.hq_city	P.pname	P.maker	P.factory_loc
X Co.	Seattle	X	X Co.	U.S.
Y Inc.	Seattle	X	Y Inc.	China

Example: Return the headquarters of companies which make gizmos in US **AND** China

`C JOIN P on maker = name`

C.name	C.hq_city	P.pname	P.maker	P.factory_loc
X Co.	Seattle	X	X Co.	U.S.
Y Inc.	Seattle	X	Y Inc.	China

- X Co. has a factory in the US (but not China)
- Y Inc. has a factory in China (but not US)

But Seattle is returned by the query!
We did the INTERSECT on the wrong attributes.

1.9 Nested Queries

- A solution for the previous example are **Nested Queries**
- A nested query is just a SELECT statement inside of another
- Nested queries are **always** enclosed in parenthesis ()

```
Company(name, hq_city) AS C
Product(pname, maker, factory_loc) AS P
```

Example: Return the headquarters of companies which make gizmos in US **AND** China

```
SELECT DISTINCT hq_city
FROM Company, Product
```

```
WHERE maker = name
      AND name IN (SELECT maker
                   FROM Product
                   WHERE factory_loc = 'US')
      AND name IN (SELECT maker
                   FROM Product
                   WHERE factory_loc = 'China')
```

1.9.1 High level note on nested Queries

- We can do nested queries because SQL is ***compositional***
- Everything (input/outputs) is represented as multisets
- The output of one query can be used as the input of another (nesting)
- This is **extremely** useful

1.10 Sub-queries Returning Relations

```
Company(CName, StockPrice, Country)
Product(PName, Price, Category, Manufacturer)
Purchase(id, product, buyer)
```

Example: Countries where one can find companies that manufacture products bought by 'Joe Blow'

```
SELECT c.country
FROM   Company c
WHERE  c.name IN (SELECT pr.maker
                  FROM Purchase p, Product pr
                  WHERE p.product = pr.name
                        AND p.buyer = 'Joe Blow')
```

```
Company(CName, StockPrice, Country)
Product(PName, Price, Category, Manufacturer)
Purchase(id, product, buyer)
```

Example: Countries where one can find companies that manufacture products bought by 'Joe Blow'

```
In [26]: %sql SELECT * FROM Company;

Done.


Out[26]: [('GWorks', 25.0, 'USA'),
          ('Canon', 65.0, 'Japan'),
          ('Hitachi', 15.0, 'Japan'),
          ('IBM', 140.0, 'USA')]
```

```
Company(CName, StockPrice, Country)
Product(PName, Price, Category, Manufacturer)
Purchase(id, product, buyer)
```

Example: Countries where one can find companies that manufacture products bought by 'Joe Blow'

```
In [27]: %%sql
         SELECT c.Country
         FROM Company c
         WHERE CName IN (SELECT pr.Manufacturer
                         FROM Purchase p, Product pr
                         WHERE p.product = pr.PName
                               AND p.buyer = 'Joe Blow');

Done.


Out[27]: [('USA',)]
```

```
Company(CName, StockPrice, Country)
Product(PName, Price, Category, Manufacturer)
Purchase(id, product, buyer)
```

Example: Countries where one can find companies that manufacture products bought by 'Joe Blow'

Is this query equivalent?

```
In [28]: %%sql
         SELECT c.Country
         FROM   Company c, Product pr, Purchase p
         WHERE  c.CName = pr.Manufacturer
                AND pr.PName = p.product
                AND p.buyer = 'Joe Blow';

Done.


Out[28]: [('USA',), ('USA',)]
```

Beware of duplicates!

1.11 Sub-queries Returning Relations

You can also use operations of the form: * s > **ALL** R * s < **ANY** R * **EXISTS** R

Note: ANY and ALL, not supported by SQLite

```
Product(PName, Price, Category, Manufacturer)
```

Example: Find products that are more expensive than all those produced by 'GWorks'

```
SELECT PName
FROM Product
WHERE price > ALL(SELECT price
                  FROM Product
                  WHERE maker = 'GWorks')
```

Example: Find products that are more expensive than all those produced by 'GWorks'

Is there a workaround for SQLite? **YES**

```
In [29]: %%sql
         SELECT *
         FROM   Product;

Done.


Out[29]: [('Gizmo', 19.99, 'Gadgets', 'GWorks'),
          ('Powergizmo', 29.99, 'Gadgets', 'GWorks'),
          ('SingleTouch', 149.99, 'Photography', 'Canon'),
          ('MultiTouch', 203.99, 'Household', 'Hitachi')]
```

```
In [30]: %%sql
         -- Workaround for SQLite
         SELECT PName
         FROM   Product
         WHERE  Price > (SELECT MAX(Price)
                         FROM   Product
                         WHERE  Manufacturer = 'GWorks');

Done.


Out[30]: [('SingleTouch',), ('MultiTouch',)]
```

- The **EXISTS** condition is used in combination with a subquery and is considered to be met, if the subquery returns **at least** one row.

  Example: Find 'copycat' products, i.e. products made by competitors with the same names as products made by "GWorks"

```
Product(PName, Price, Category, Manufacturer)
```

```
SELECT p1.PName
FROM   Product p1
WHERE  p1.Manufacturer = 'GWorks'
       AND EXISTS(SELECT p2.PName
                  FROM Product p2
                  WHERE p2.Manufacturer <> 'GWorks'
                        AND p1.PName = p2.PName)
```

R.A
1
2
3
4
5

S.A
1
3
5
7
9

T.A
1
4
7
10

intersect

Note the scoping of the variables. * p1 and p2 both make reference to *Product*

Example: Find 'copycat' products, i.e. products made by competitors with the same names as products made by "GWorks"

```
Product(PName, Price, Category, Manufacturer)
```

```
In [31]: %%sql
         SELECT p1.PName
         FROM   Product p1
         WHERE  p1.Manufacturer = 'GWorks'
                AND EXISTS(SELECT p2.PName
                           FROM Product p2
                           WHERE p2.Manufacturer <> 'GWorks'
                                 AND p1.PName = p2.PName);
```

```
Done.
```

```
Out[31]: []
```

1.12 Nested Queries alternatives for INTERSECT and EXCEPT

- INTERSECT and EXCEPT are not implemented in some DBMSs

INTERSECT

```
In [32]: %%sql
         SELECT R.A
         FROM   R
         INTERSECT
         SELECT S.A
         FROM   S;
```

```
Done.
```

```
Out[32]: [(1,), (3,), (5,)]
```

R.A
1
2
3
4
5

S.A
1
3
5
7
9

T.A
1
4
7
10

intersect

R.A
1
2
3
4
5

S.A
1
3
5
7
9

T.A
1
4
7
10

intersect

INTERSECT

```
In [33]: %%sql
         SELECT R.A
         FROM   R
         WHERE  EXISTS(SELECT *
                       FROM S
                       WHERE R.A=S.A);
```

```
Done.
```

```
Out[33]: [(1,), (3,), (5,)]
```

EXCEPT

```
In [34]: %%sql
         SELECT R.A
         FROM   R
         EXCEPT
         SELECT S.A
         FROM   S;
```

```
Done.
```

```
Out[34]: [(2,), (4,)]
```

R.A
1
2
3
4
5

S.A
1
3
5
7
9

T.A
1
4
7
10

intersect

EXCEPT

```
In [35]: %%sql
         SELECT R.A
         FROM   R
         WHERE NOT EXISTS(SELECT *
                          FROM S
                          WHERE R.A=S.A);
```

```
Done.
```

```
Out[35]: [(2,), (4,)]
```

1.13 Correlated Queries

Also known as a *Synchronized Subquery* * Uses values from the outer query * Can be very powerful * Harder to optimize, the subquery may be evaluated once for each row processed by the outer query

> **Example:** Find products (and their manufacturers) that are more expensive than all products made by the same manufacturer before 1972

```
Product(name, price, category, manufacturer, year)
```

```
SELECT DISTINCT x.name, x.manufacturer
FROM   Product AS x
WHERE  x.price > ALL(SELECT y.price
                     FROM Product AS y
                     WHERE x.manufacturer = y.manufacturer
                           AND y.year < 1972)
```

1.13.1 Exercise I

An organism that sells tickets for football matches uses a database with the following relational schema:

```
Match(Match_ID, Date, Hour, Stadium_ID, Team_ID)
Team(Team_ID, Name, City)
Stadium(Stadium_ID, Name, Address, Capacity, Team_ID)
Ticket(Ticket_ID, Match_ID, Place_Number, Category, Price)
Sell(Sell_ID, Sell_Date, Ticket_ID, Payment_Method)
```

Write the following query in SQL: > What is the date for a match where 'Barcelona F.C.' plays at the 'Parc des Princes' stadium?

```
In [36]: # Modify the css style
         # from IPython.core.display import HTML
         # def css_styling():
         #     styles = open("./style/custom.css").read()
         #     return HTML(styles)
         # css_styling()
```