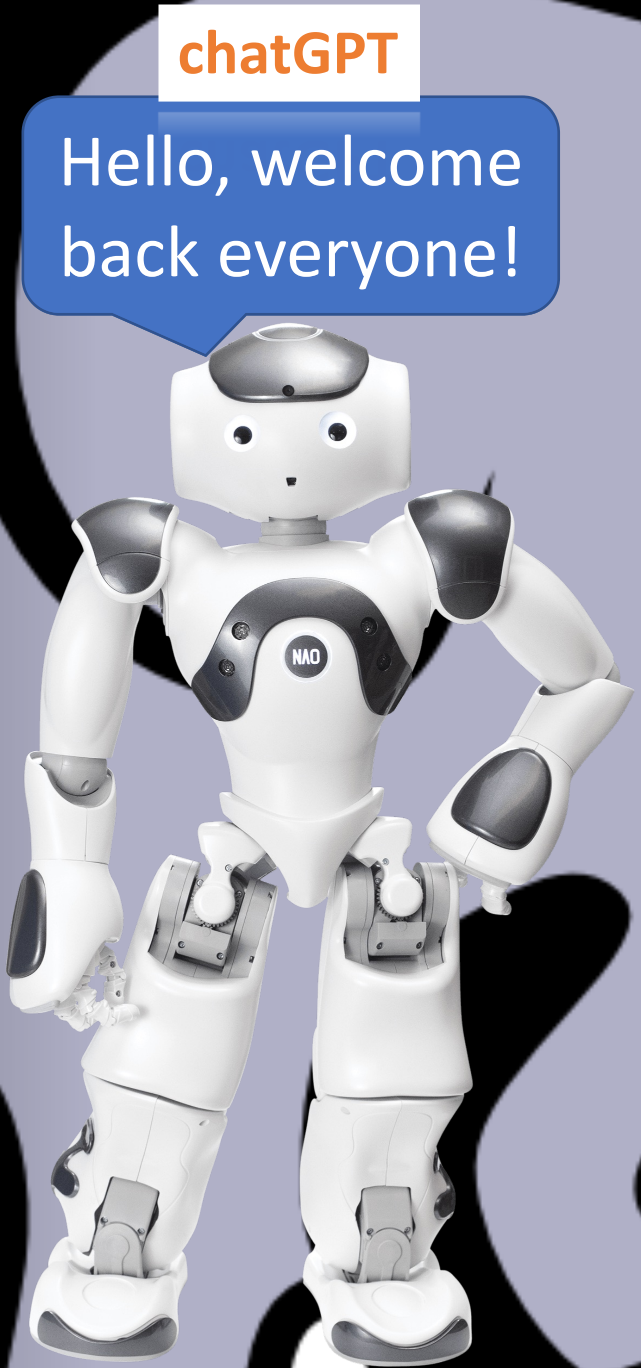
Introduction to Database Systems
I2DBS – Spring 2023

- Week 3:
- Division
- More about Joins & Nulls
- Set Operation
- Subqueries
- Views

Jorge-Arnulfo Quiané-Ruiz



Readings:

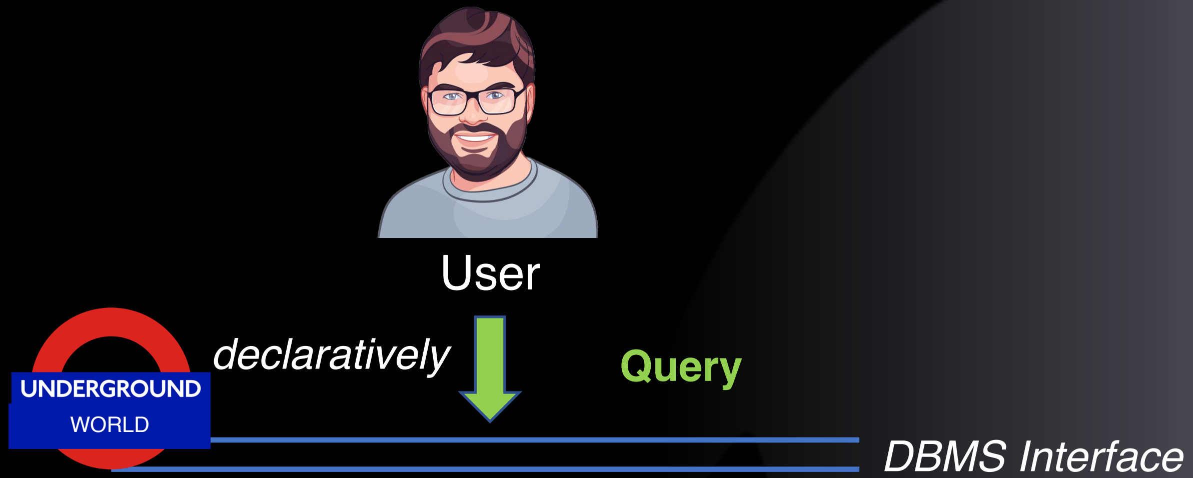
PDBM 7.3, 7.4

Profile of the Week

Raymond F. Boyce

SQL & BCNF Co-Inventor

- **1946:** Born in San Jose California
- **1972:** PhD in Computer Science from Purdue University
- **1972:** Moved to IBM
- **Mid-1970:** Co-Invented SQL
- **Mid-1974:** Co-developed the Boyce-Codd normal form.
- **1974:** R.I.P.



User

declaratively

Query

UNDERGROUND WORLD

DBMS Interface

SQL – Part 2

Readings:

PDBM 1

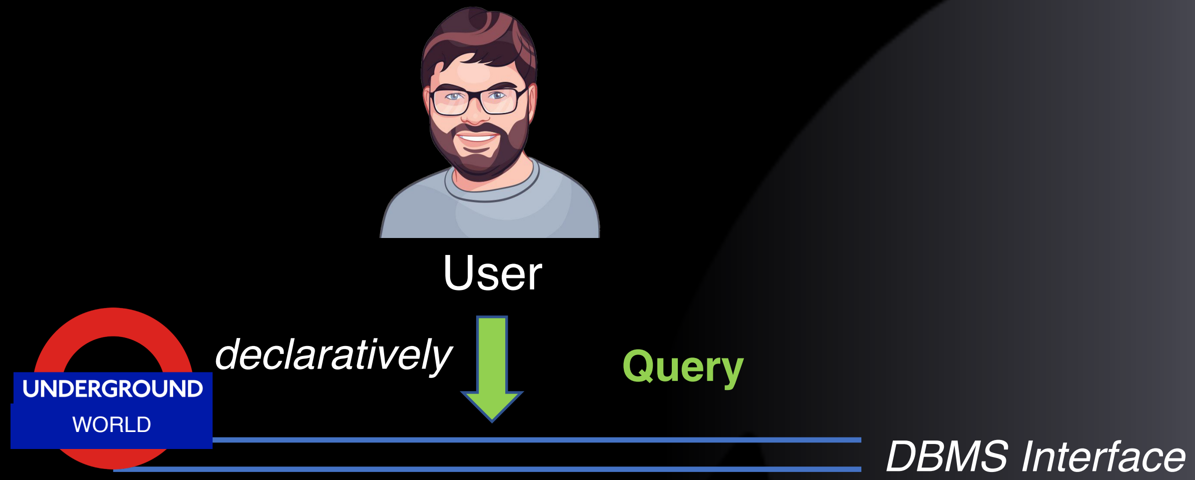
Database Schema

Running Example

- Coffees(name, manufacturer)
- Coffeehouses(name, address, license)
- Drinkers(name, address, phone)
- Likes(drinker, coffee)
- Sells(coffeehouse, coffee, price)
- Frequents(drinker, coffeehouse)

Note

Scripts on learnIT (in Week 3)



SQL DML -- Division

Readings:

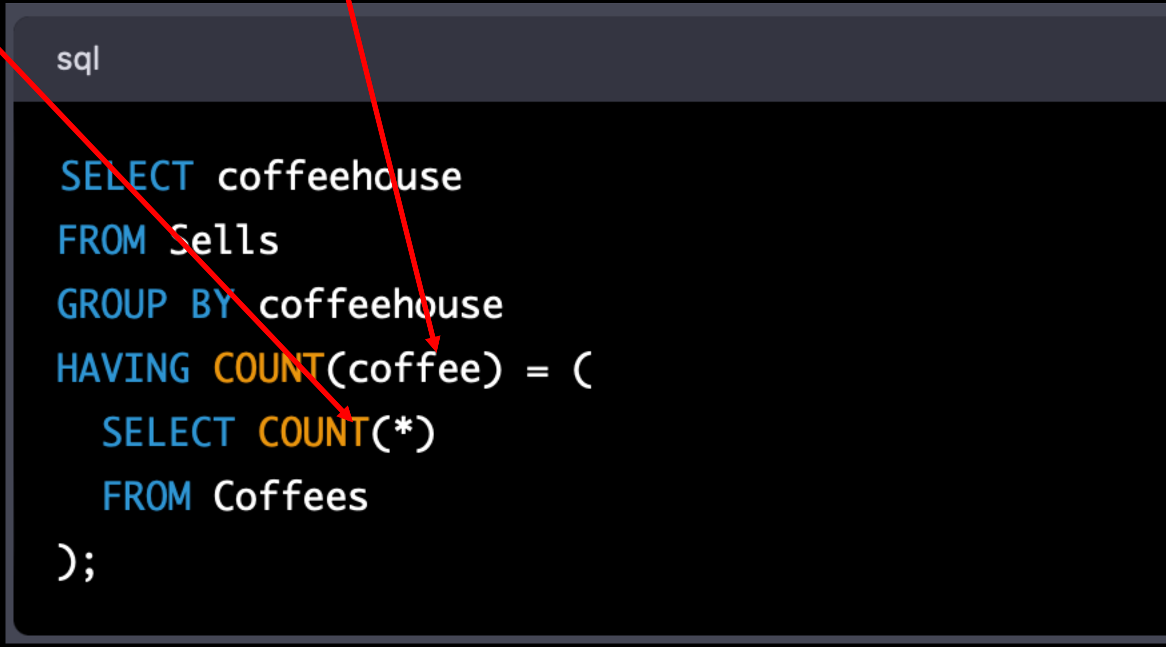
PDBM 1

What is a Division?

- R1 / R2 = records of R1 associated with all tuples of R2
- Find the students who have taken all courses in a program
- Find the airlines who land at all airports in a country/continent/the world
- Why divisions are not so simple in SQL?

Division with Counting

- Find the coffeehouses that sell all existing coffees
- **We can write division using GROUP BY, HAVING and a COUNT sub-query**
  - Step 1: Count the number of coffees
  - Step 2: For each coffeehouse, return it only if it sells that many coffee types

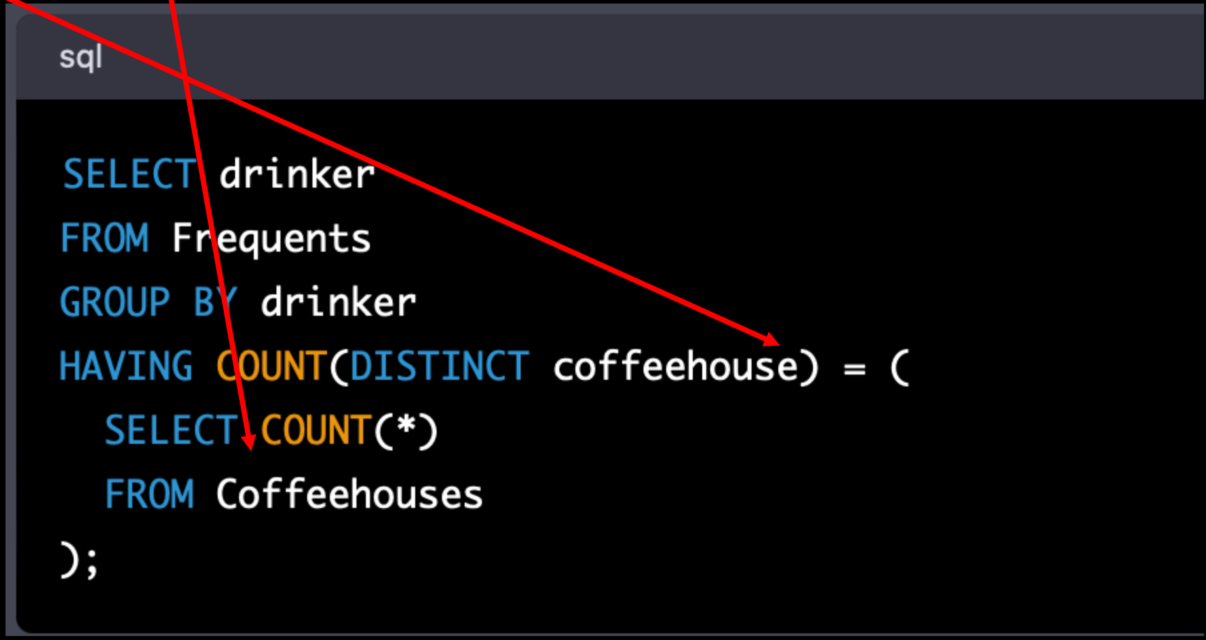


sql

```sql
SELECT coffeehouse
FROM Sells
GROUP BY coffeehouse
HAVING COUNT(coffee) = (
  SELECT COUNT(*)
  FROM Coffees
);
```

Example Query

- Names of drinkers who frequent all coffeehouses

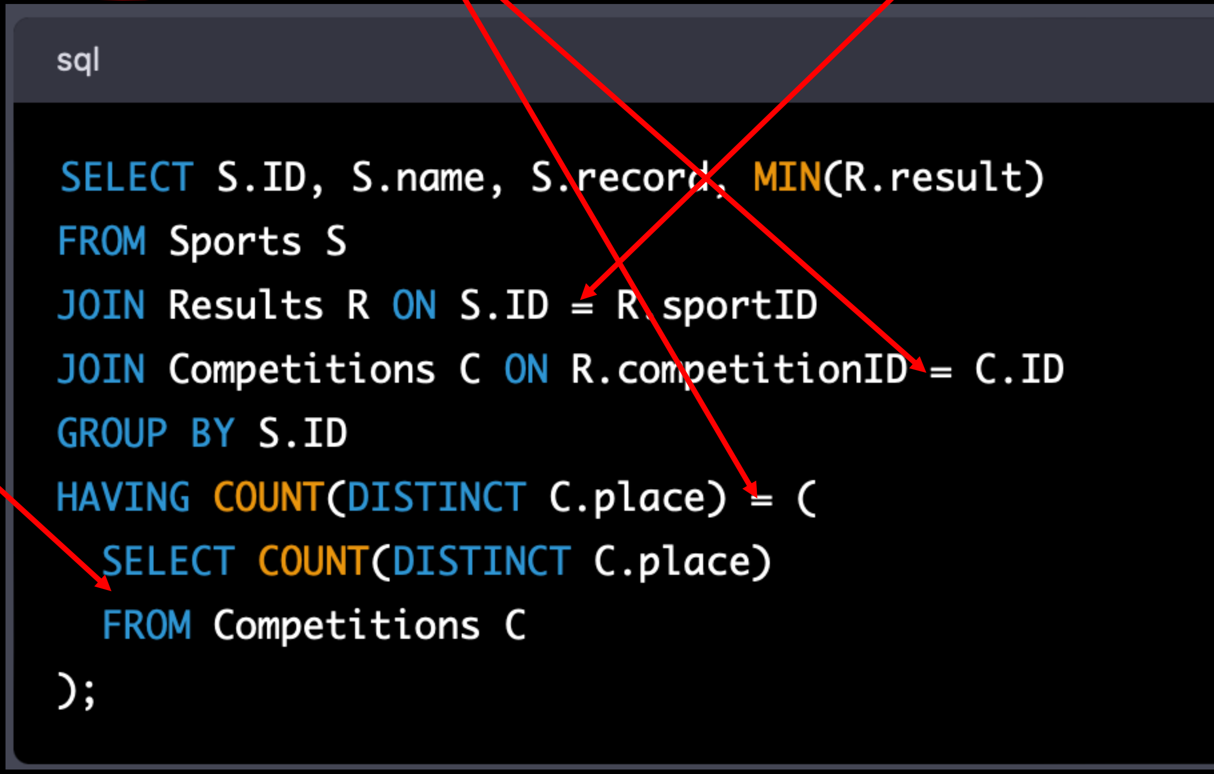


sql

```sql
SELECT drinker
FROM Frequents
GROUP BY drinker
HAVING COUNT(DISTINCT coffeehouse) = (
  SELECT COUNT(*)
  FROM Coffeehouses
);
```

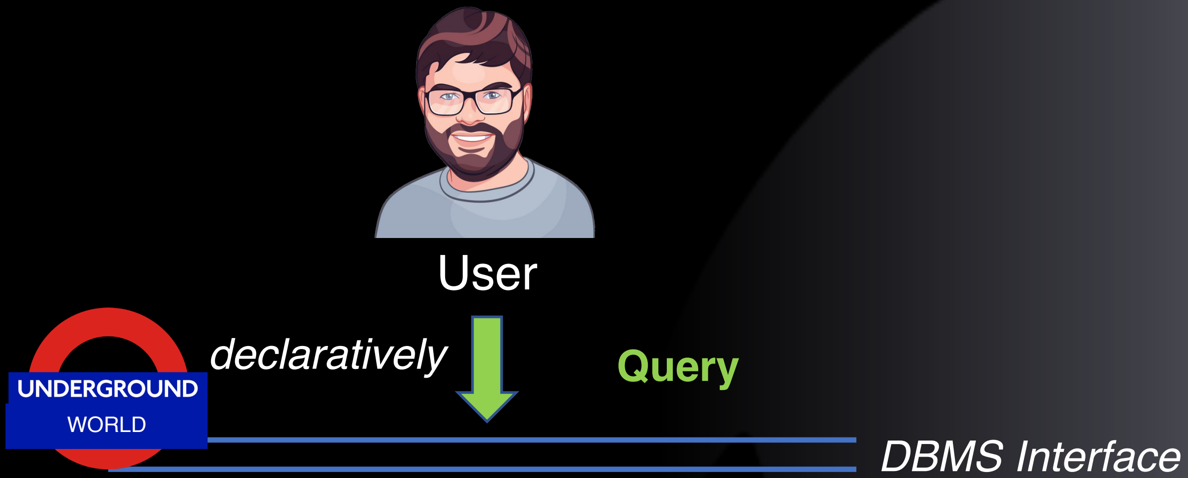
Example Complex Query

- Show the ID, name, record and worst result of all sports that have at least one result from every ‘place’ where a competition has ever been held



sql

```sql
SELECT S.ID, S.name, S.record, MIN(R.result)
FROM Sports S
JOIN Results R ON S.ID = R.sportID
JOIN Competitions C ON R.competitionID = C.ID
GROUP BY S.ID
HAVING COUNT(DISTINCT C.place) = (
  SELECT COUNT(DISTINCT C.place)
  FROM Competitions C
);
```



SQL DML -- Joins & Nulls

Readings:

PDBM 1

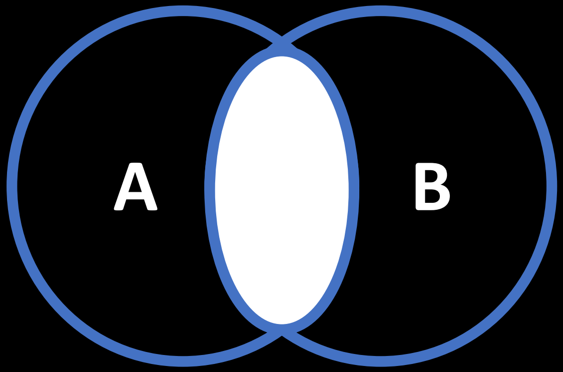
Inner Join

- **R [INNER] JOIN S ON <condition>**

- **Example: using Drinkers(name, address, phone) and Frequents(drinker, coffeehouse):**

  Drinkers JOIN Frequents ON name = drinker;

  - gives us all (d, a, p, d, b) quintuples such that drinker d lives at address a, has phone number p, and frequents bar b



Natural Join & Cross Join (Products)

Natural join: R NATURAL JOIN S;

- Assumes “=” on all column(s) of same name
  - Removes the duplicate column(s)
- Can be quite dangerous, so don’t use!
  - Example: Coffeehouses named by drinkers (or vice versa)

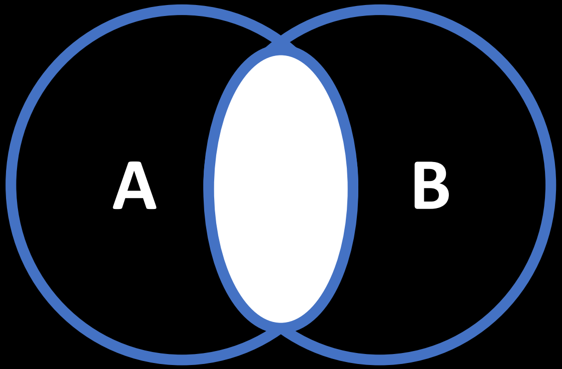
    Drinkers JOIN Coffeehouses

Cross product: R CROSS JOIN S;

- Removes no columns, has no join condition
  - Can simulate with an always-true join condition

```
SELECT *
FROM Likes L JOIN Drinkers D ON 1 = 1
```

- Rarely (but not never) the right thing to do!



Join Syntax

- We have used ANSI join syntax
- An alternative might be called "old-style" syntax:

```
SELECT *
FROM Likes L
    JOIN Drinkers D
    ON L.drinker = D. name;
```

```
SELECT *
FROM Likes L, Drinkers D
WHERE L.drinker = D. name;
```

- Using ANSI syntax is recommended!
  - More readable, harder to forget join conditions

```
SELECT *
FROM Likes L JOIN Drinkers D;
```

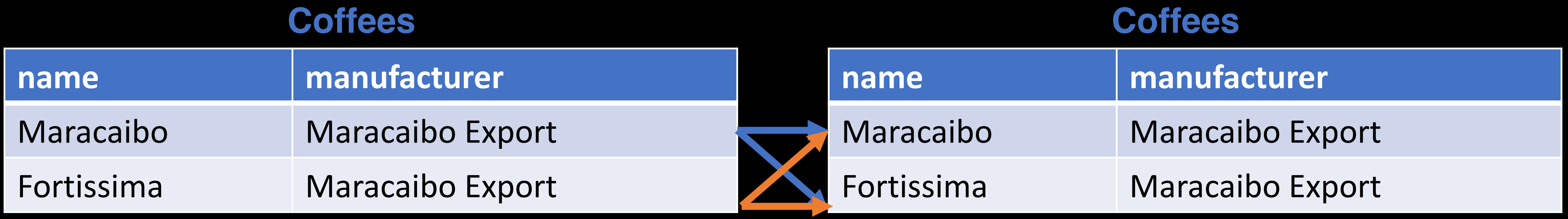
```
SELECT *
FROM Likes L, Drinkers D;
```

Advise

If you go "old-style" then put the JOIN conditions first!

Self-Join

- **From Coffees(name, manufacturer), find all pairs of coffees by the same manufacturer.**
- **This is an example of a self-join!**
  - We need to compare two coffee records with each other, so we need a ”double loop” through the same relation!



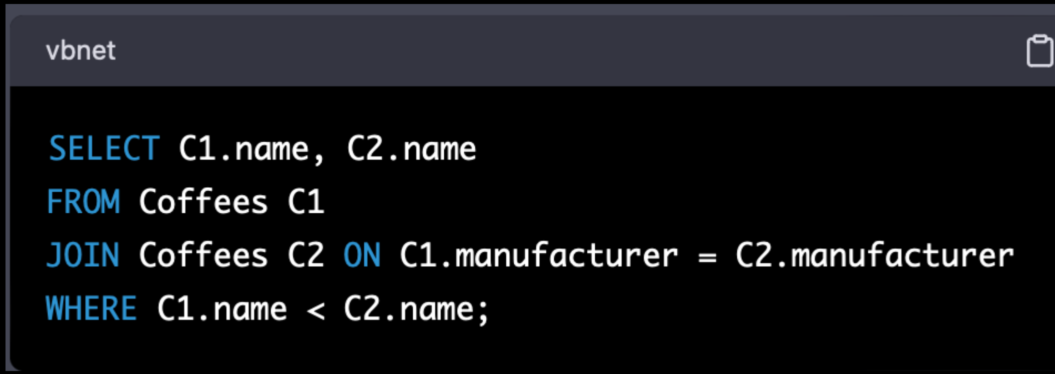
Coffees

name	manufacturer
Maracaibo	Maracaibo Export
Fortissima	Maracaibo Export

Coffees

name	manufacturer
Maracaibo	Maracaibo Export
Fortissima	Maracaibo Export

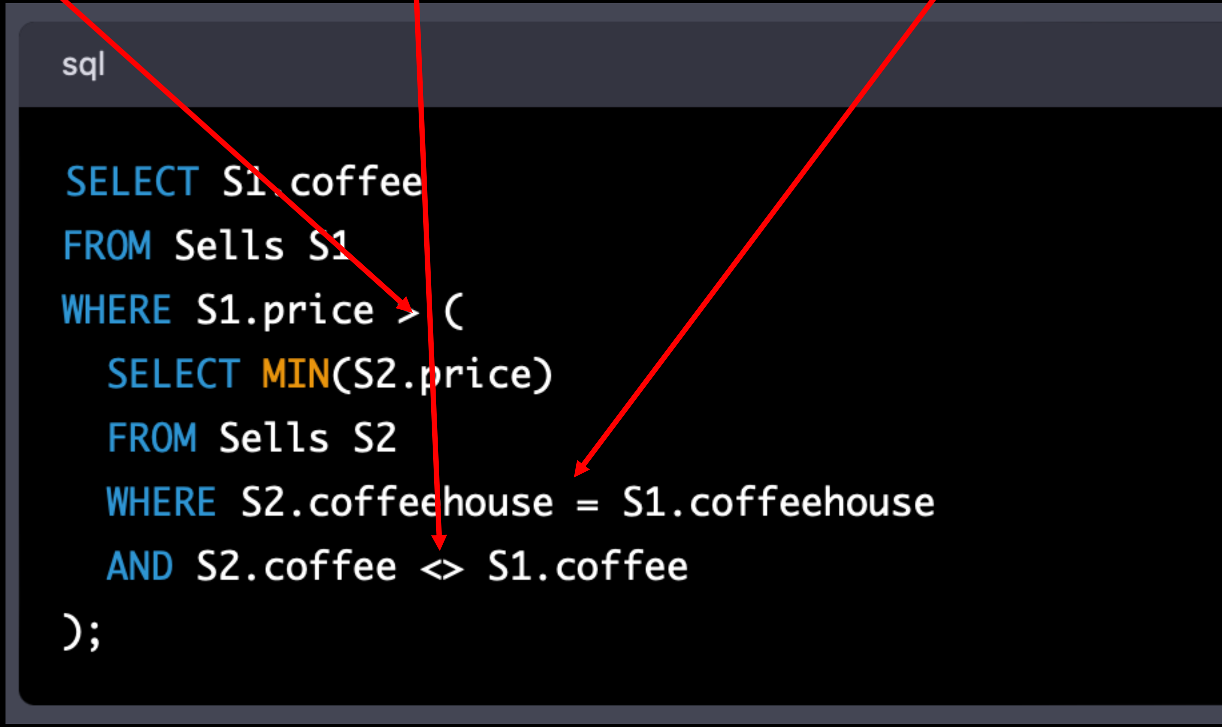
-
  - Do not produce pairs like (Maracaibo, Maracaibo).
  - Produce pairs in alphabetic order, e.g. (Fortissima, Maracaibo), not (Maracaibo, Fortissima).



```
SELECT C1.name, C2.name
FROM Coffees C1
JOIN Coffees C2 ON C1.manufacturer = C2.manufacturer
WHERE C1.name < C2.name;
```

Example Query

- Show all coffees that are more expensive than some other coffee sold at the same coffeehouse



```sql
SELECT S1.coffee
FROM Sells S1
WHERE S1.price > (
  SELECT MIN(S2.price)
  FROM Sells S2
  WHERE S2.coffeehouse = S1.coffeehouse
  AND S2.coffee <> S1.coffee
);
```

- **Add one record and run again:**
  INSERT INTO Sells(coffeehouse, coffee, price)
  VALUES('Mocha', 'Kopi luwak’, 400);

Null-Values

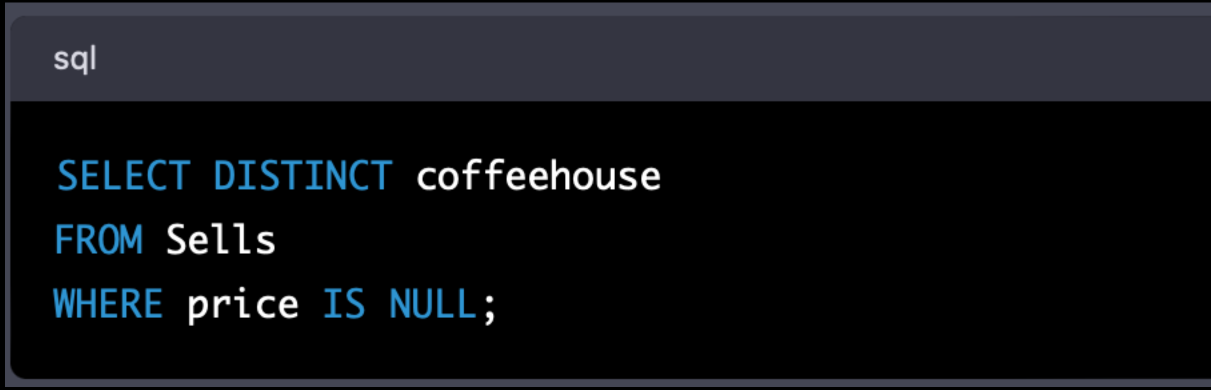
- **Tuples in SQL relations can have NULL as a value for one or more components.**
- **Meaning depends on context. Two common cases:**
  - Missing value: e.g., we know Joe’s Bar has some address, but we don’t know what it is.
    - "secret", "figure not available", "to be announced", "impossible to calculate", "partly unknown", "uncertain”, "pending"
  - Inapplicable: e.g., the value of attribute spouse for an unmarried person.
    - "undefined", "moot", "quantum uncertain", "irrelevant", "none”, "n/a"

Comparing NULL Values

- **The logic of conditions in SQL is really 3-valued logic: TRUE, FALSE, UNKNOWN.**
- **Comparing any value (including NULL itself) with NULL yields UNKNOWN.**
  - NULL != NULL
  - Must use IS NULL or IS NOT NULL
- **A record is in a query answer iff the WHERE clause is TRUE (not FALSE or UNKNOWN).**

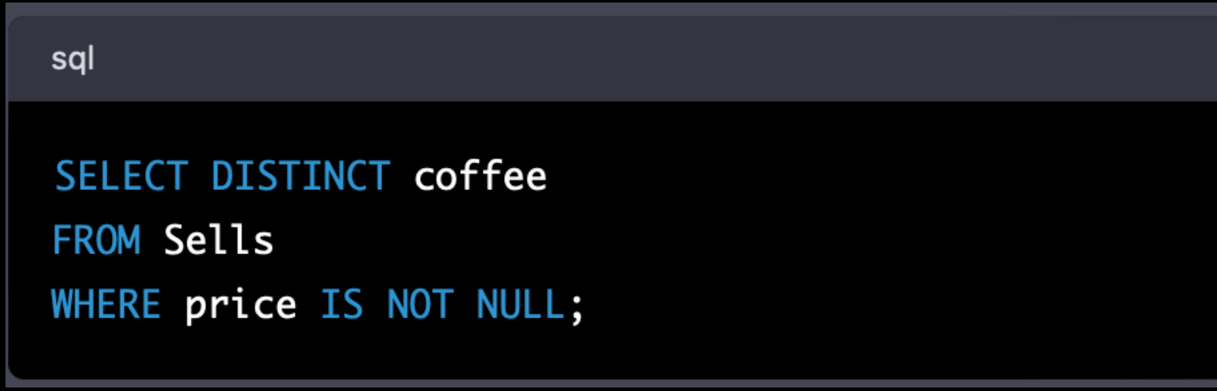
Example Queries

- Show the coffeehouses that sell a coffee at an unknown price



```sql
SELECT DISTINCT coffeehouse
FROM Sells
WHERE price IS NULL;
```

- Show the name of all coffees that are sold at a known price



```sql
SELECT DISTINCT coffee
FROM Sells
WHERE price IS NOT NULL;
```

Three-Valued Truth Tables

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

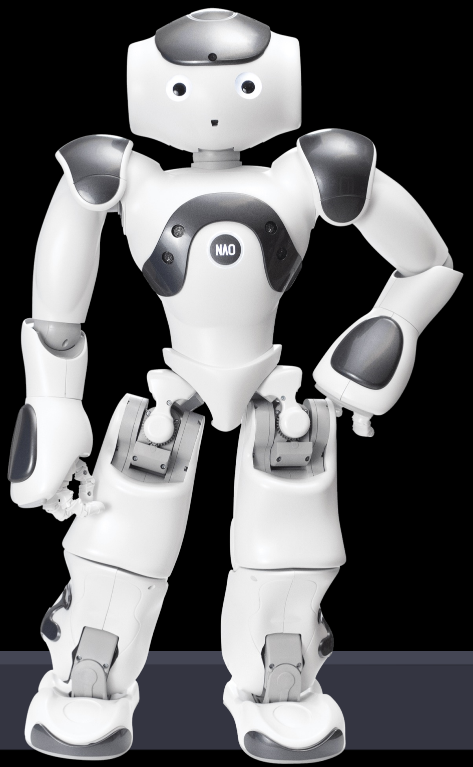
=	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	TRUE	NULL
NULL	NULL	NULL	NULL

Three-Valued Logic Example

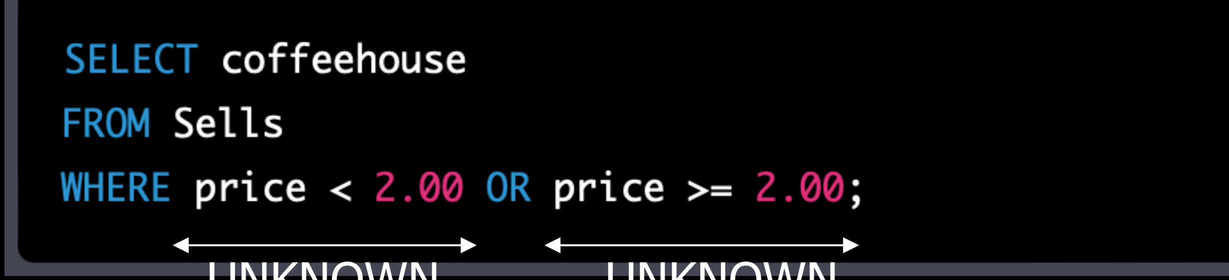
- **In two-valued logic, p OR NOT p is always TRUE**
- **Compute the truth value of p OR NOT p when p is UNKNOWN**

Sells

coffeehouse	coffee	price
Joe‘s	Maracaibo	NULL



sql



```sql
SELECT coffeehouse
FROM Sells
WHERE price < 2.00 OR price >= 2.00;
```

UNKNOWN UNKNOWN

UNKNOWN

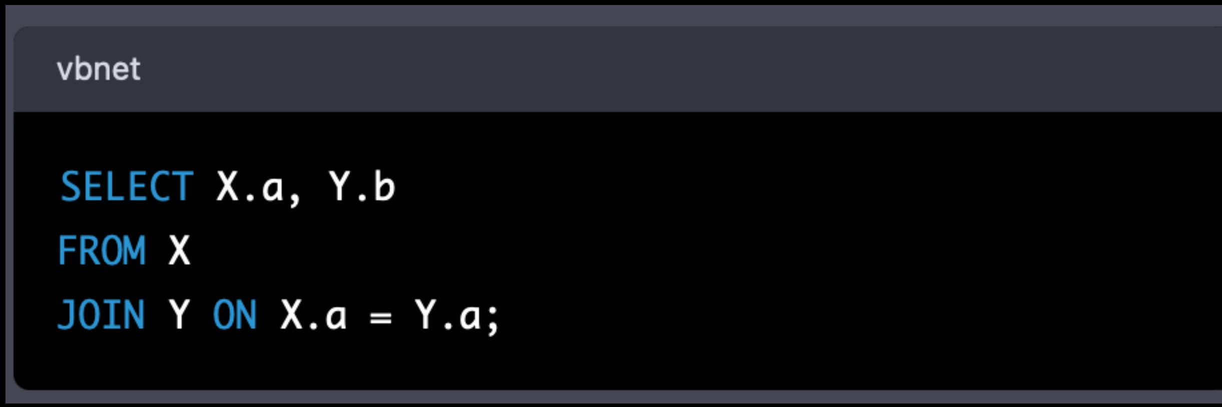
Outer Joins

- **Sometimes a coffee is sold nowhere, but we want it in our result → outer join!**

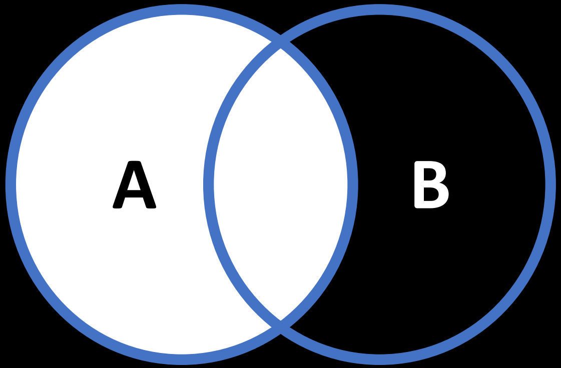
- **R OUTER JOIN S is the core of an outer join expression.**

- **It is modified by optional LEFT, RIGHT, or FULL before OUTER.**
  - LEFT = pad dangling tuples of R with NULL.
  - RIGHT = pad dangling tuples of S with NULL.
  - FULL = pad both; this choice is the default.

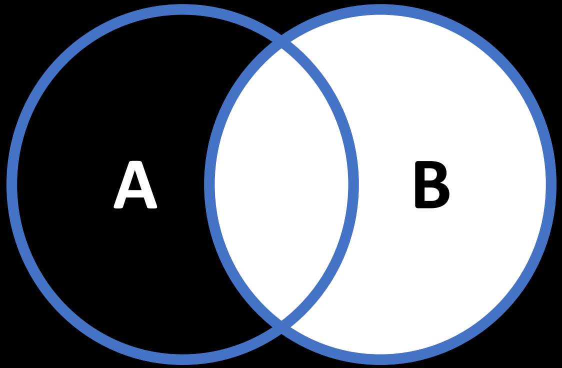
Joins in a Nutshell



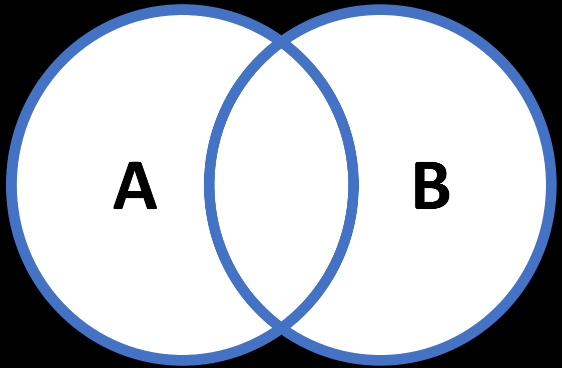
```
SELECT X.a, Y.b
FROM X
JOIN Y ON X.a = Y.a;
```



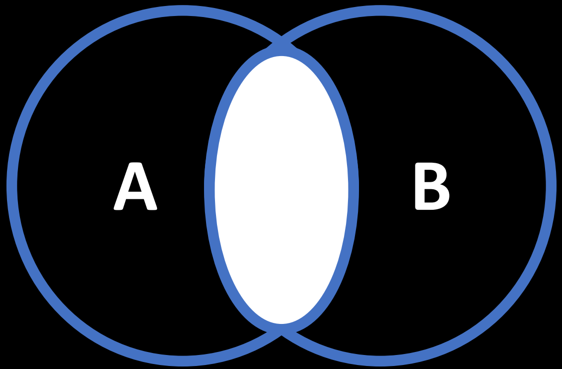
Left Outer



Right Outer



Full Outer



Inner Join

Example of an Outer Join

- **Suppose we add a new coffee:**

```
insert into Coffees values
('Bragakaffi','Ó.J. & Kaaber')
```

```
select S.coffeehouse, C.manufacturer
from Sells S full outer join Coffees C
  on S.coffeehouse = C.name
```

Then the outer join includes the new coffee

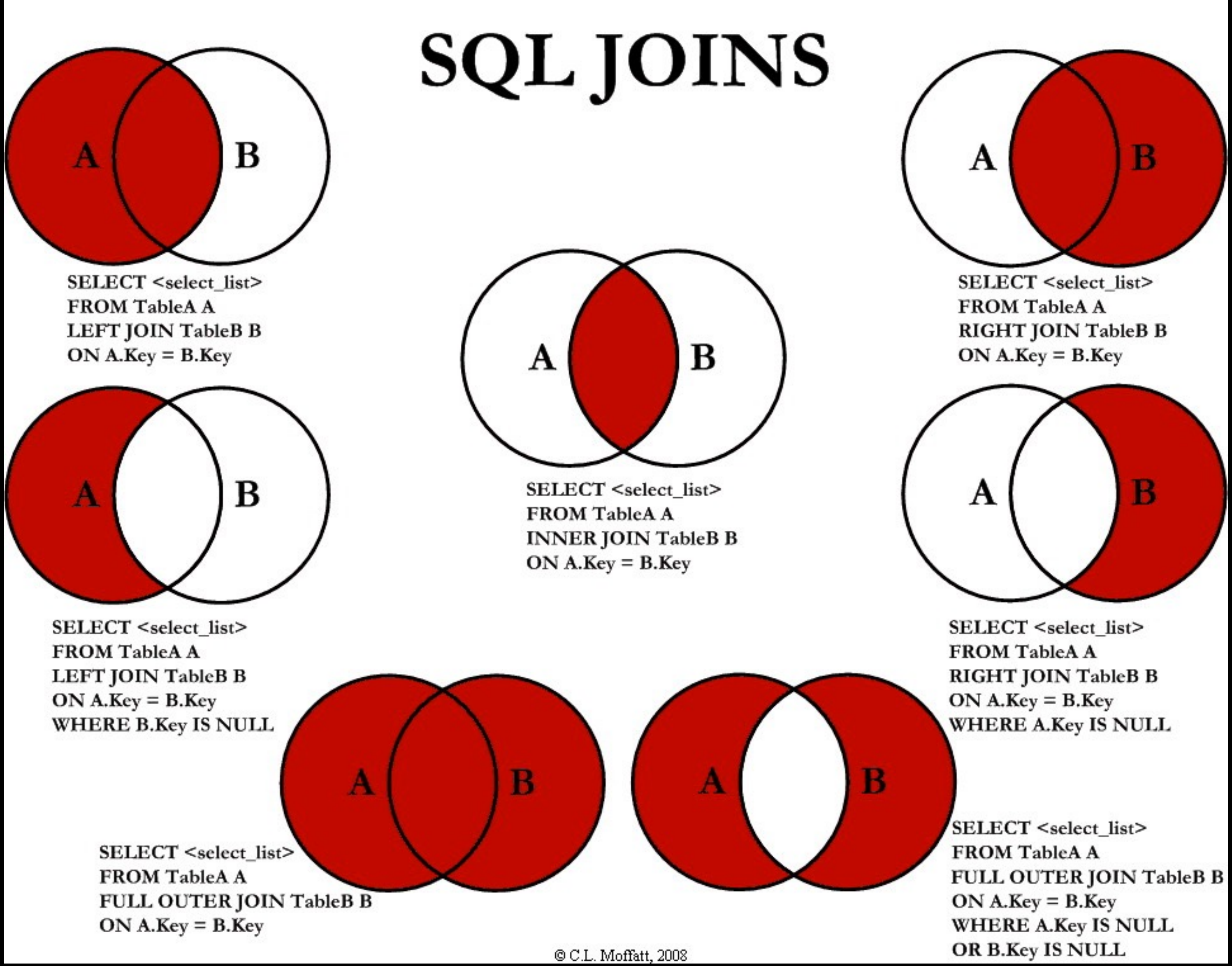
Result

coffehouse	manufacturer
Joe‘s	Ottolina
Joe‘s	Kopi Luwak Dir.
Sue‘s	Ottolina
Sue‘s	Marley Coffee
Sue‘s	Kopi Luwak Dir.
NULL	Ó.J. & Kaaber

Practice

- Show all drinkers and the coffees they like, but include drinkers that do not like any coffees
- Can you use OUTER JOIN to show ONLY drinkers who do not like any coffees?

SQL Joins in a Nutlshell



```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

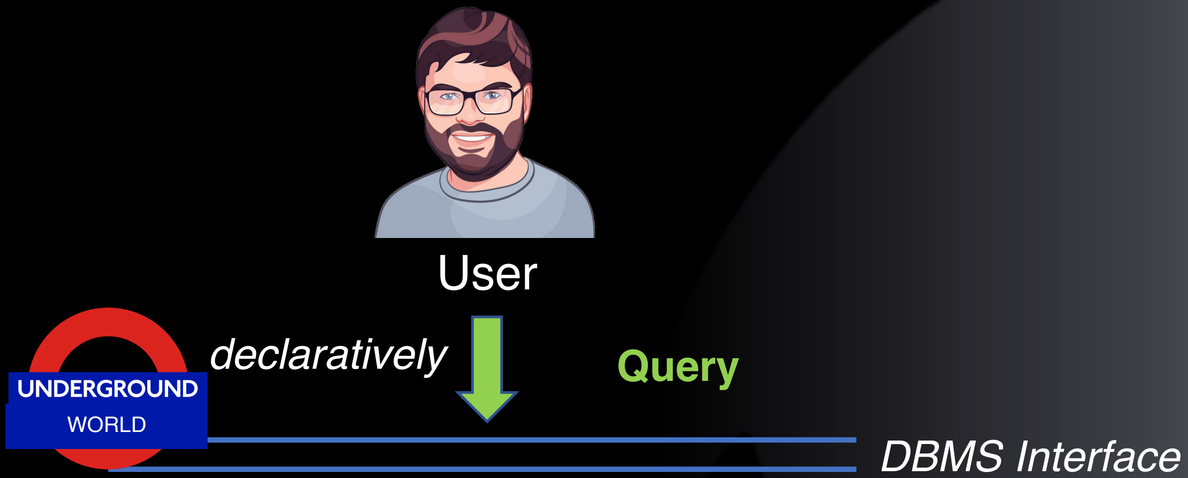
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

© C.L. Moffatt, 2008

https://www.codeproject.com/KB/database/Visual_SQL_Joins/Visual_SQL_JOINS_orig.jpg



SQL DML – Set Operations

Readings:

PDBM 1

Set Queries

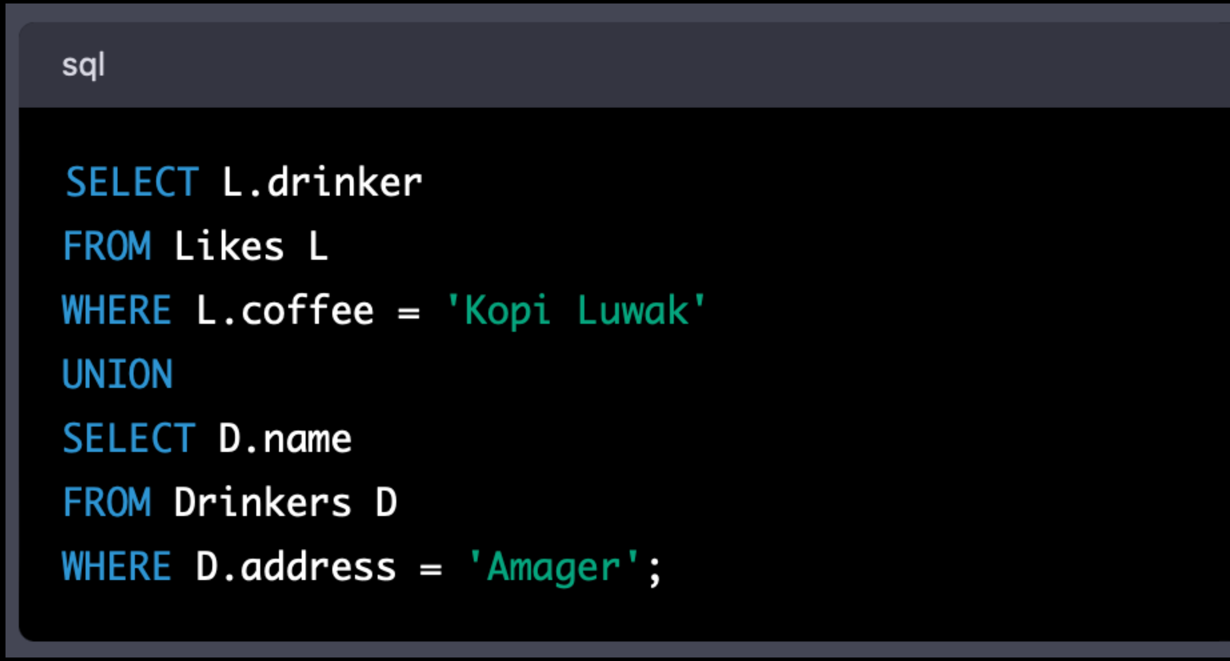
- **Syntax:**

```
<Query 1>
UNION / INTERSECT / EXCEPT
<Query 2>
```

- **Queries must be "union compatible" = results have matching schema:**
  - Same number of attributes
  - Attributes i of both tables have same (matching) type

Example Query -- UNION

- Show all drinkers that like “Kopi luwak” **or** live in “Amager”

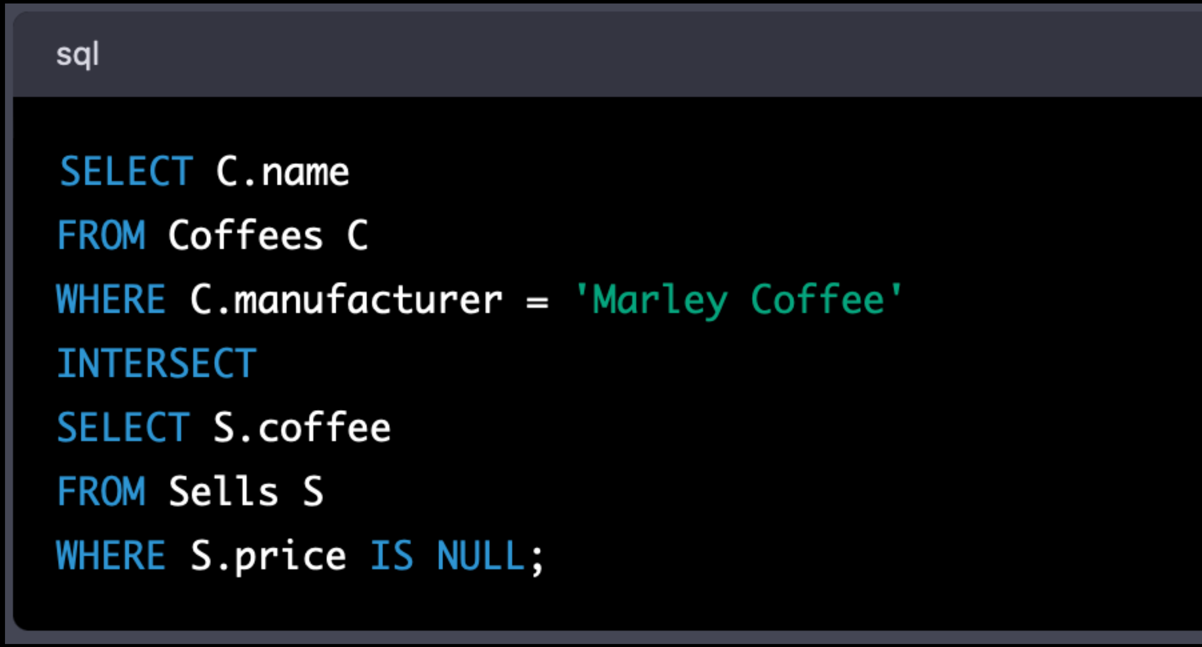


sql

```sql
SELECT L.drinker
FROM Likes L
WHERE L.coffee = 'Kopi Luwak'
UNION
SELECT D.name
FROM Drinkers D
WHERE D.address = 'Amager';
```

Example Query -- INTERSECT

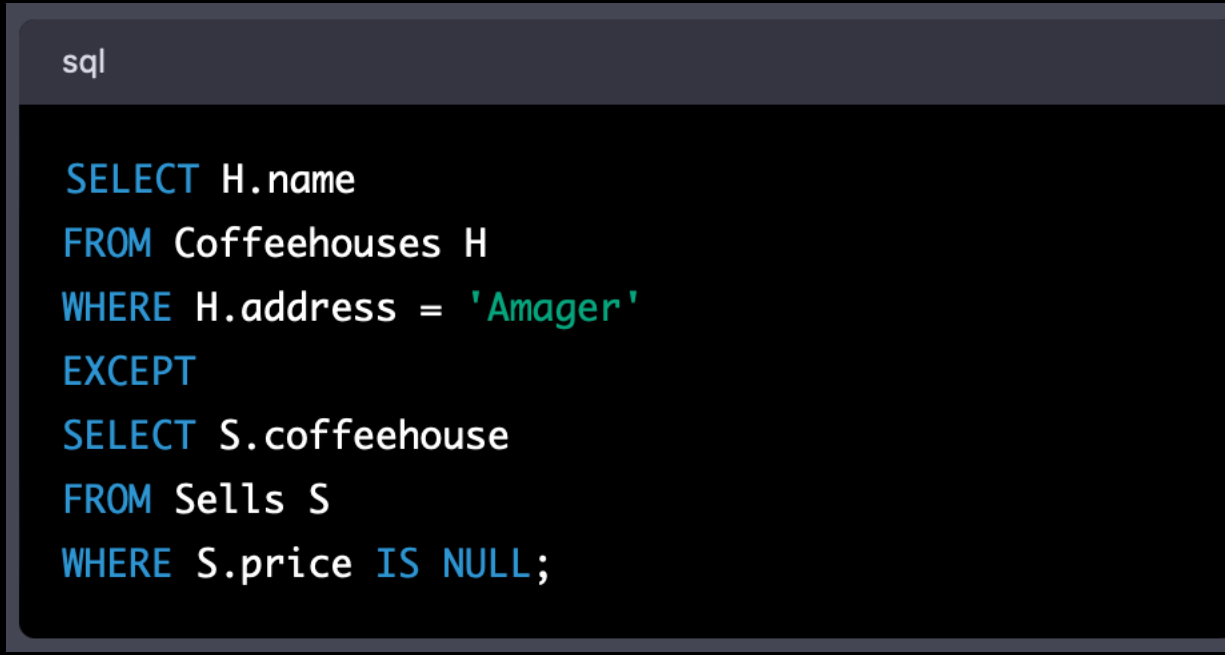
- **Show all coffees that are manufactured by “Marley Coffee” and sold at an unknown price**



```sql
SELECT C.name
FROM Coffees C
WHERE C.manufacturer = 'Marley Coffee'
INTERSECT
SELECT S.coffee
FROM Sells S
WHERE S.price IS NULL;
```

Example Query – EXCEPT (occasionally MINUS)

- Show all coffeehouses that are at ‘Amager’ **but** which do not sell a coffee with an unknown price



```sql
SELECT H.name
FROM Coffeehouses H
WHERE H.address = 'Amager'
EXCEPT
SELECT S.coffeehouse
FROM Sells S
WHERE S.price IS NULL;
```

Duplicates in Set Queries

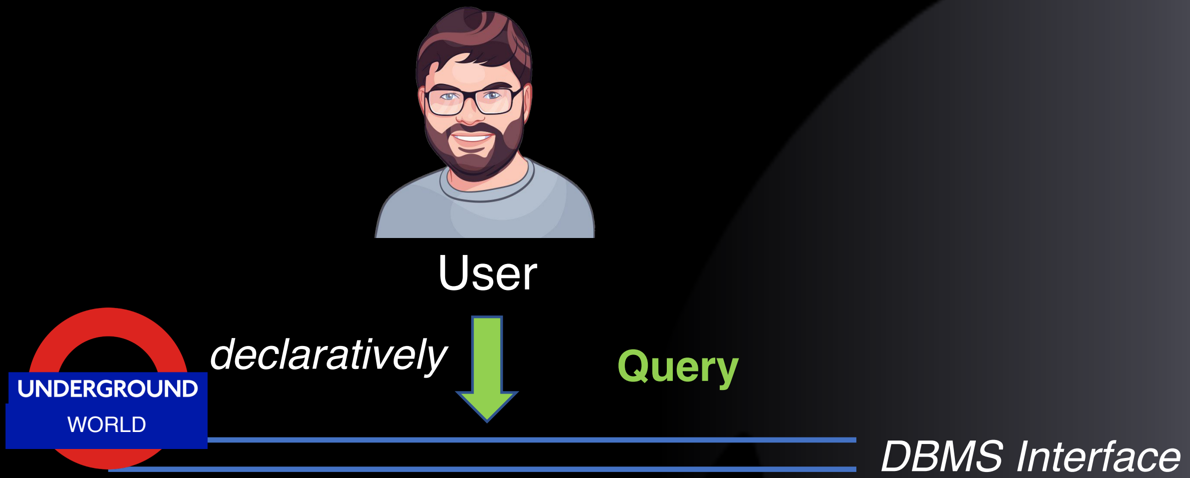
- **Set operators remove duplicates**

- **To keep duplicates use:**
  - R UNION ALL S – (0, 1, 5x2, 3x3, 4, 5)
  - R INTERSECT ALL S – (2x2, 1x3)
  - R EXCEPT ALL S – (1, 0x2, 1x3, 4)

R	S
1	0
2	2
2	2
3	2
3	3
4	5

- **Assume R has m rows and S has n rows with some values, how many rows could each at most contain?**

- **The most practical use:**
  UNION ALL when R and S are known to be disjoint!



SQL DML -- Subqueries

Readings:

PDBM 1

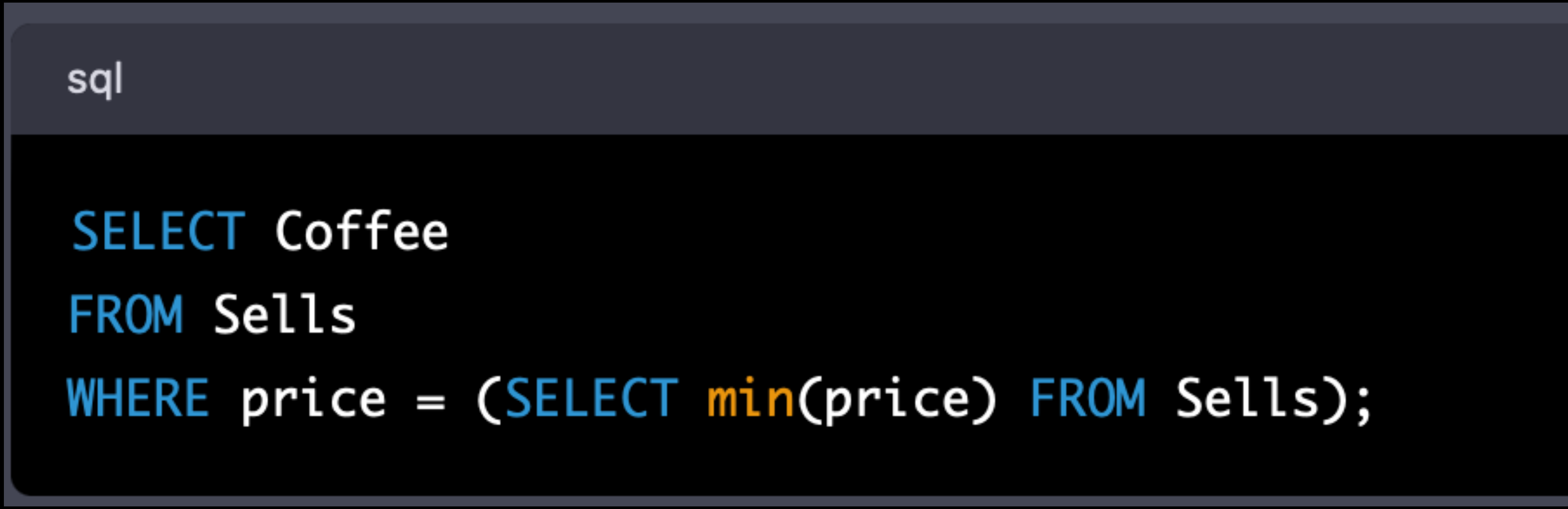
What is a Subquery?

- **A parenthesized SELECT-FROM-WHERE statement (*subquery*) can be used as a value in a number of places, including WHERE and FROM clauses.**

- **If a sub-query is guaranteed to produce one tuple, then the sub-query can be used as a value**
  - Usually, the tuple has only one attribute
  - An implicit type cast is performed, and a run-time error occurs if there is no tuple or more than one tuple

- **A bit UGLY but practical!**
  - Finding records with highest values
  - Division queries

Example of a Single-Tuple Subquery



```sql
SELECT Coffee
FROM Sells
WHERE price = (SELECT min(price) FROM Sells);
```

Practice

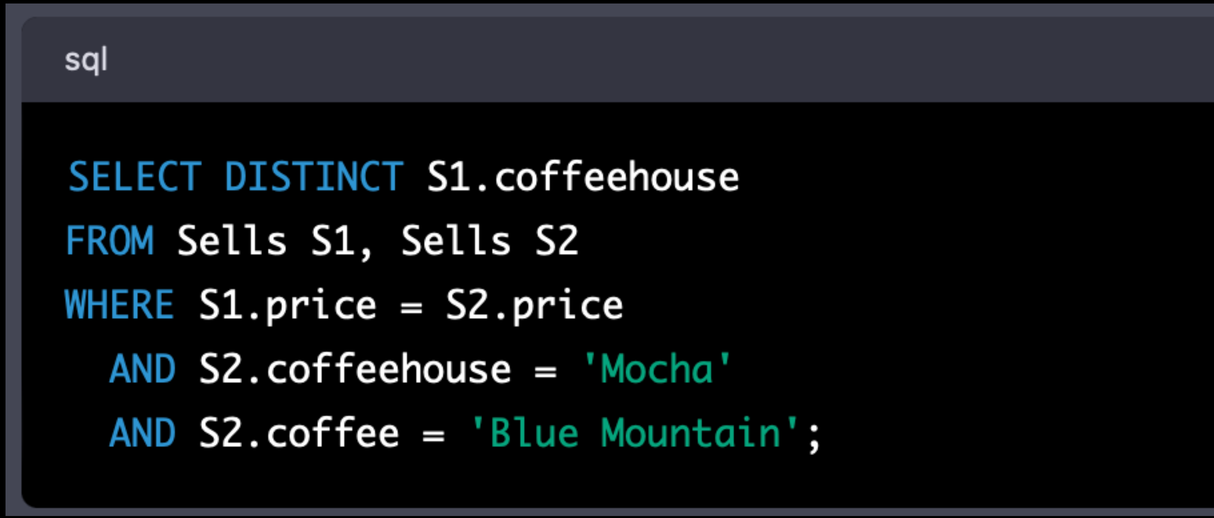
- **Find the coffeehouses that serve some coffee for the same price Mocha charges for Blue Mountain**

- **Makes the following two queries a single-query:**
  - Find the price Mocha charges for Blue Mountain
  - Find the coffeehouses that serve a coffee at that price

Join or Subquery?

Using a self join, find the coffeehouses that serve any coffee for the same price as Mocha charges for Blue Mountain

- Harder to read!
- Self-join is better for producing all pairs…



```sql
SELECT DISTINCT S1.coffeehouse
FROM Sells S1, Sells S2
WHERE S1.price = S2.price
  AND S2.coffeehouse = 'Mocha'
  AND S2.coffee = 'Blue Mountain';
```

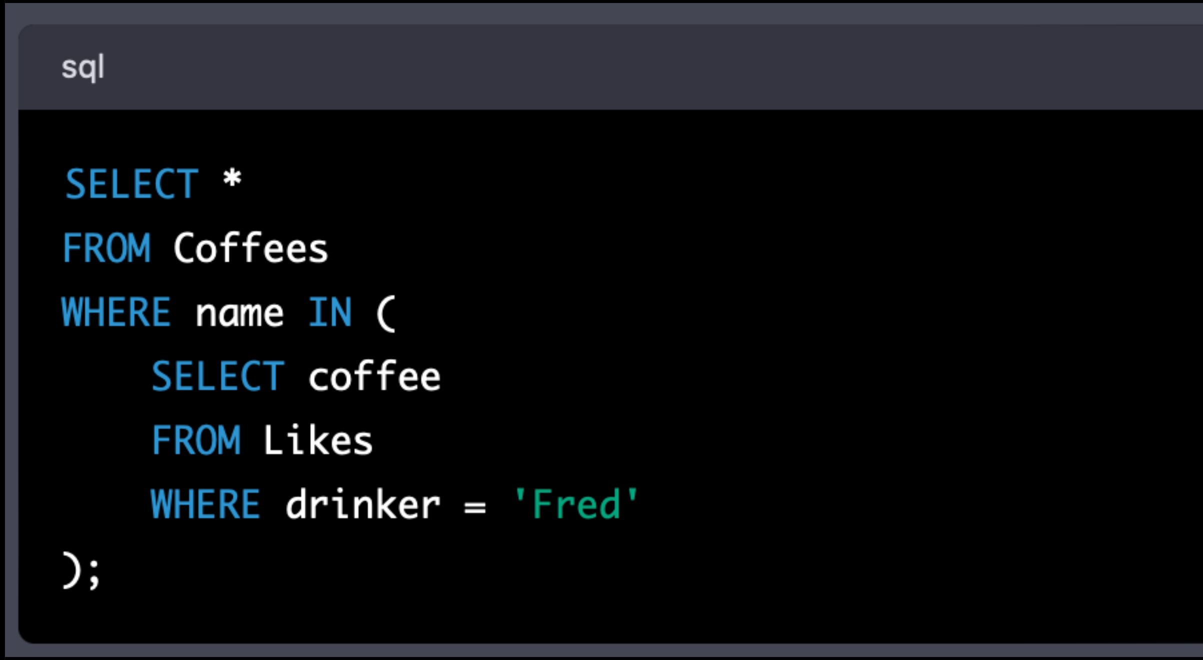
The IN Operator

- **<tuple> IN (<subquery>) is true if and only if the tuple is a member of the relation produced by the subquery.**
  - Opposite: <tuple> NOT IN (<subquery>)

- **IN-expressions can appear in WHERE clauses.**

Examples: IN

- **Using Coffees(name, manufacturer) and Likes(drinker, coffee), find the name and manufacturer of each coffee that Fred likes.**

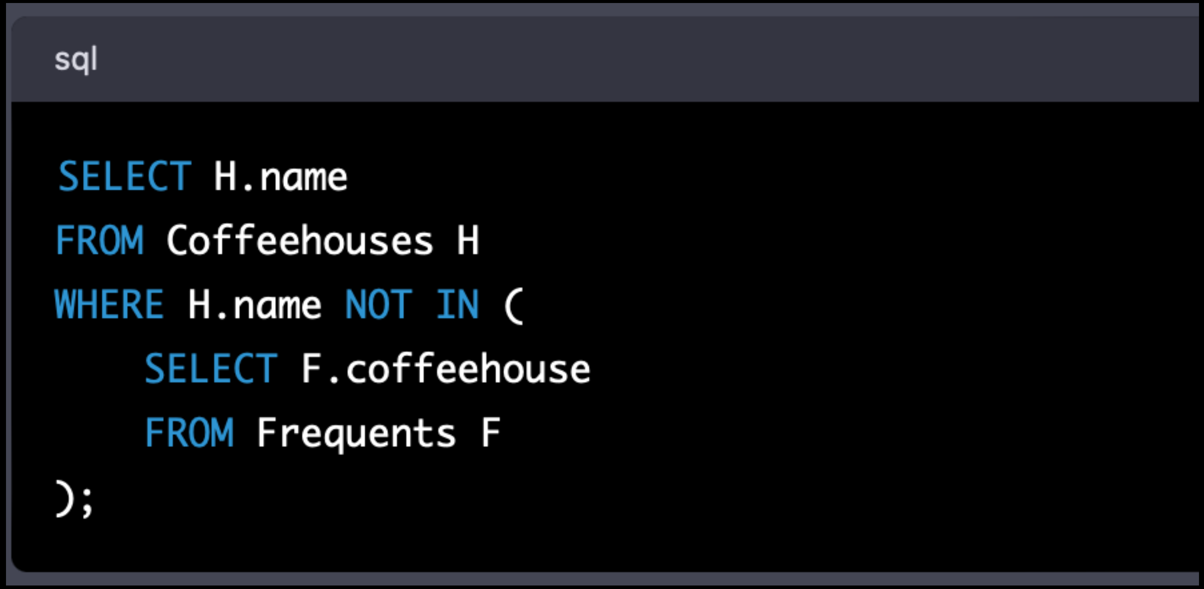


sql

```sql
SELECT *
FROM Coffees
WHERE name IN (
    SELECT coffee
    FROM Likes
    WHERE drinker = 'Fred'
);
```

Examples: NOT IN

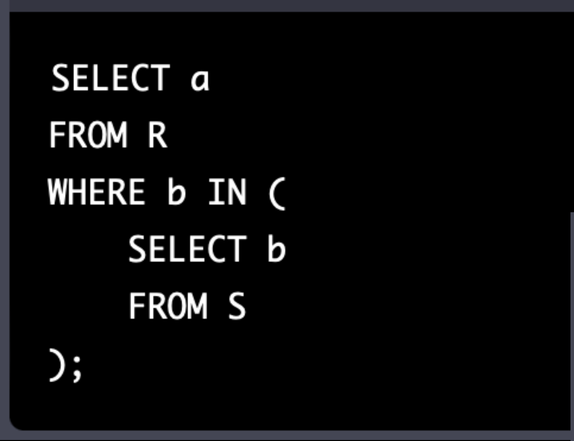
- Show the name of all coffeehouses that no one frequents!



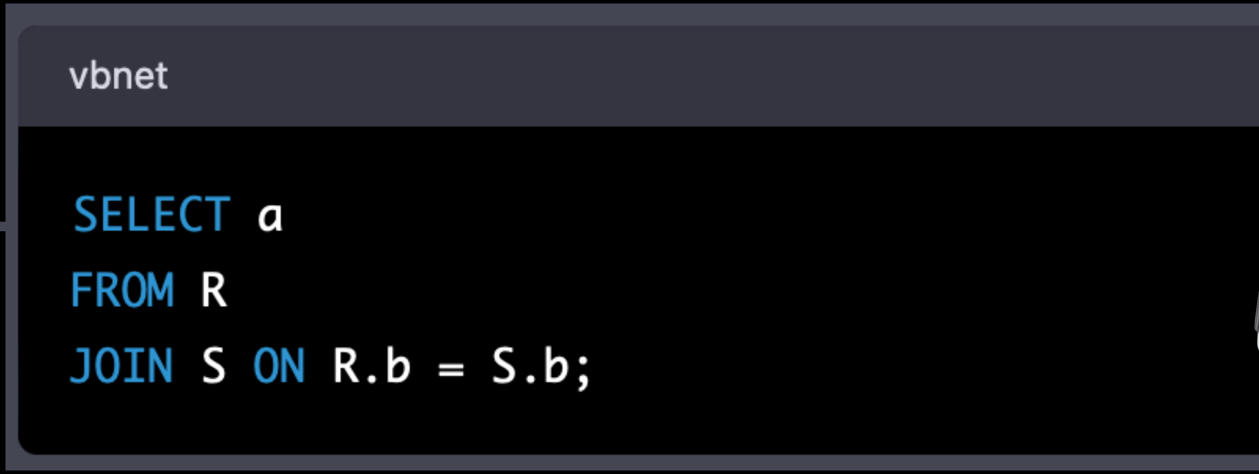
```sql
SELECT H.name
FROM Coffeehouses H
WHERE H.name NOT IN (
    SELECT F.coffeehouse
    FROM Frequents F
);
```

Practice

- Show all drinkers that like “Kopi luwak” or live in “Amager”
- Show all coffees thatare manufactured by “Marley Coffee” and sold at an unknown price
- Show all coffeehouses at ‘Amager’ which don‘t sell a coffee with an unknown price



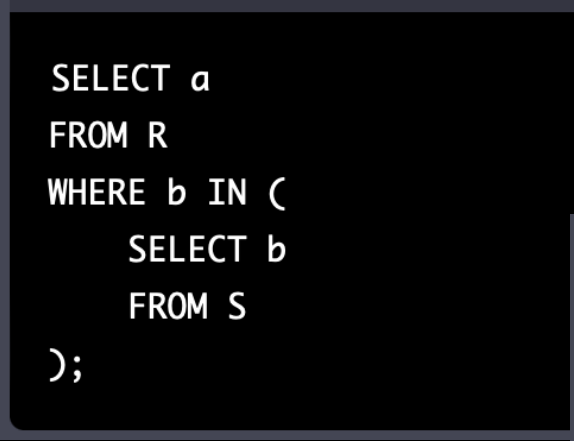
```
SELECT a
FROM R
WHERE b IN (
    SELECT b
    FROM S
);
```



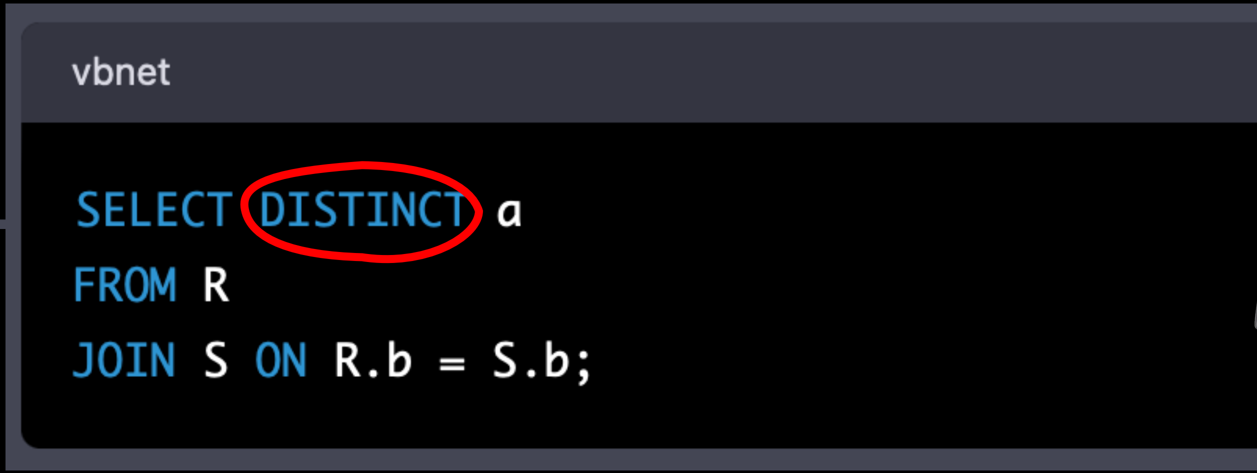
vbnet

```
SELECT a
FROM R
JOIN S ON R.b = S.b;
```

What is the Difference?



```
SELECT a
FROM R
WHERE b IN (
    SELECT b
    FROM S
);
```

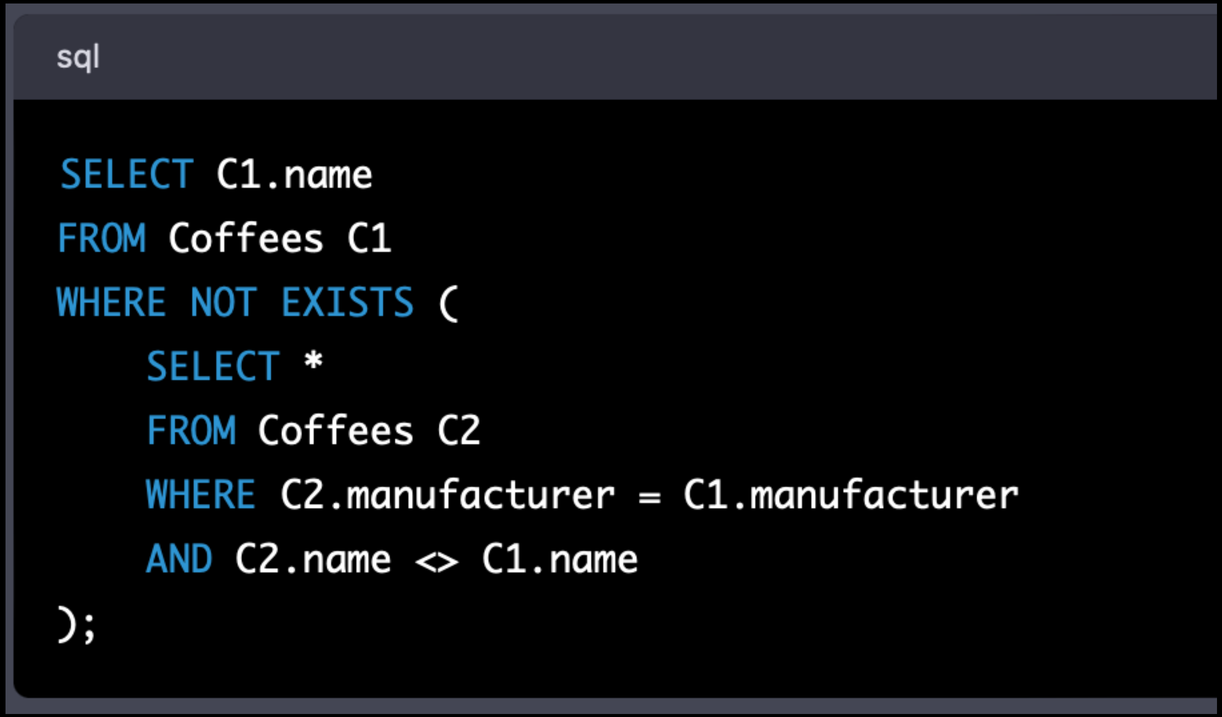


```
SELECT DISTINCT a
FROM R
JOIN S ON R.b = S.b;
```

Now There is no Difference!

EXIST Operator

- EXISTS(<subquery>) is true if and only if the sub-query result is not empty.
- Example: From Coffees(name, manufacturer), find those coffees that are the only coffee by their manufacturer.



```sql
SELECT C1.name
FROM Coffees C1
WHERE NOT EXISTS (
    SELECT *
    FROM Coffees C2
    WHERE C2.manufacturer = C1.manufacturer
    AND C2.name <> C1.name
);
```

Note

Unlike IN, the sub-query is normally correlated with the outer query!

Division in Tuple Relational Calculus (TRC)

The best method: counting!

- Already did this – strongly recommended!!!

In TRC – for some, the most logical method:

- Universal quantifier – simple but does not translate to SQL
- Double negation – "medium" complex but does translate to SQL

Names of coffeehouses that sell ~~at least one~~ coffees

all

{ R | ∃ H ∈ Coffeehouses ~~∃~~ ∀ C ∈ Coffees ∃ S ∈ Sells (
    H.name = S.coffeehouse ∧
    C.name = S.coffee ∧
    R.name = H.name) }

Division in TRC with Double Negation

- **Use a property of quanitifiers**
  - For all x there exists y = There is no x such that there is no y

$$\forall X \exists Y = \neg\exists X \neg\exists Y$$

Division in TRC with Double Negation

Names of coffeehouses that sell all coffees

- All coffeehouses B such that there is no coffee E such that there is no tuple in Sells for B and E

$$
\begin{aligned}
\{\, R \mid\ & \exists H \in \text{Coffeehouses}\ \forall C \in \text{Coffees}\ \exists S \in \text{Sells}\ (\\
& \quad H.\text{name} = S.\text{coffeehouse} \land \\
& \quad C.\text{name} = S.\text{coffee} \land \\
& \quad R.\text{name} = H.\text{name}\ )\ \}
\end{aligned}
$$

$$\Downarrow$$

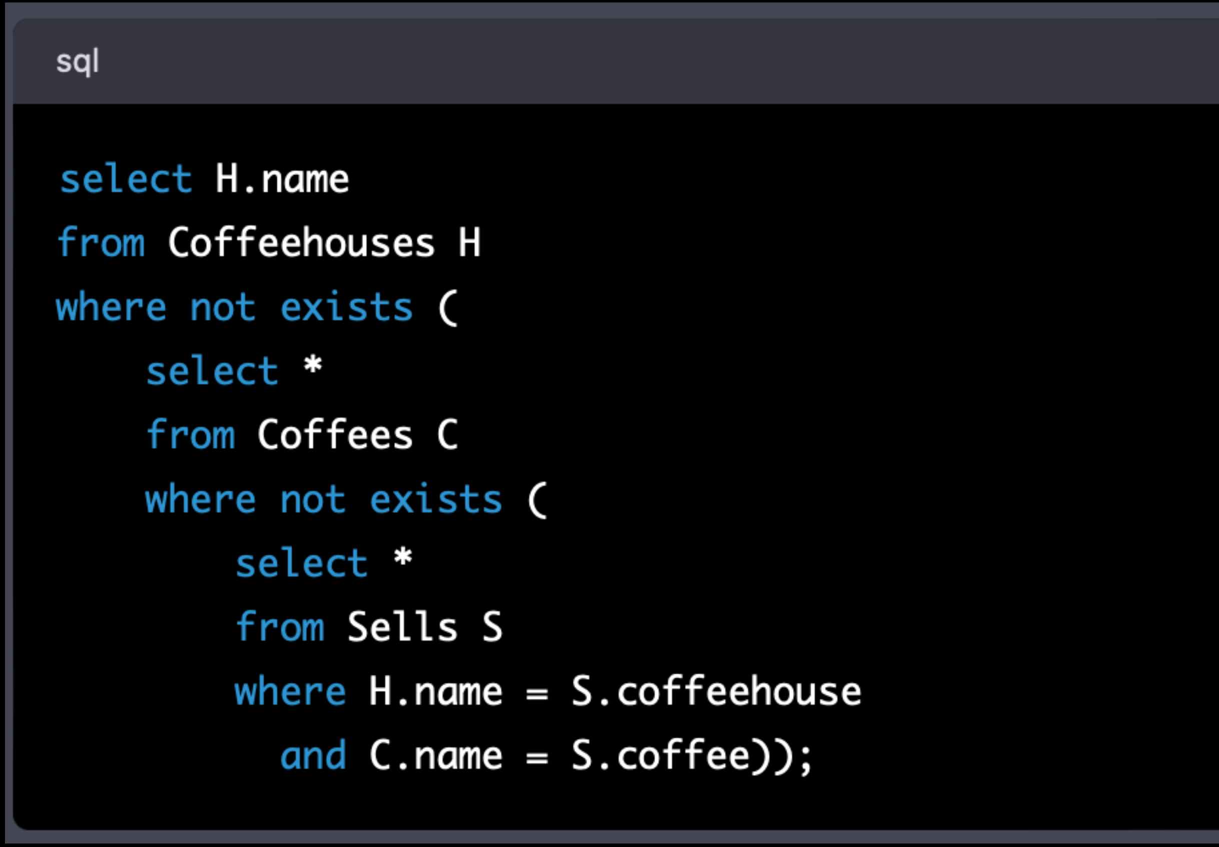
$$
\begin{aligned}
\{\, R \mid\ & \exists H \in \text{Coffeehouses}\ \neg\exists C \in \text{Coffees}\ \neg\exists S \in \text{Sells}\ (\\
& \quad H.\text{name} = S.\text{coffeehouse} \land \\
& \quad C.\text{name} = S.\text{coffee} \land \\
& \quad R.\text{name} = H.\text{name}\ )\ \}
\end{aligned}
$$

Division in TRC with Double Negation Translated into SQL

Names of coffeehouses that sell all coffees

- All coffeehouses B such that there is no coffee E such that there is no tuple in Sells for B and E

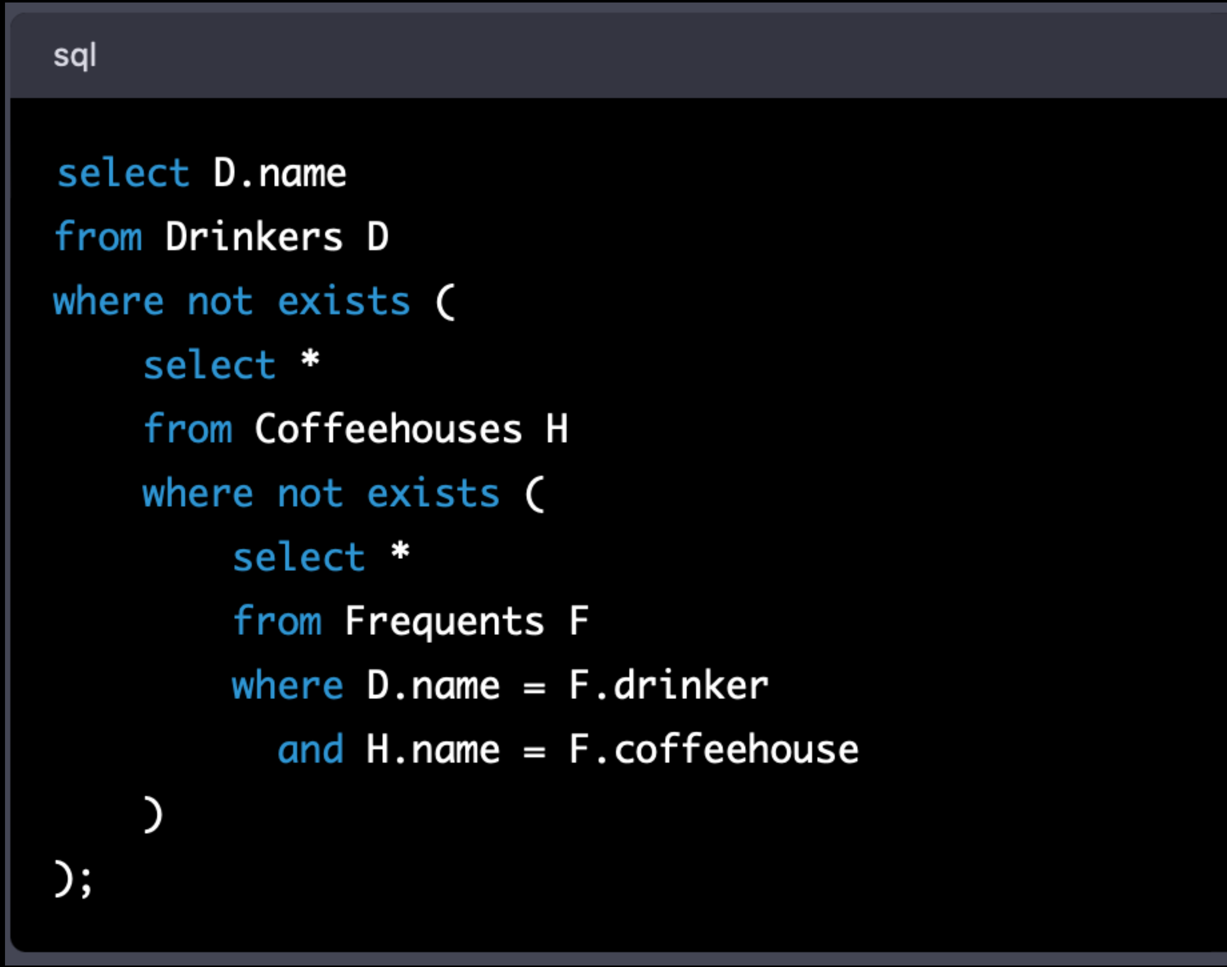
$$
\begin{aligned}
\{ R \mid\ & \exists H \in \text{Coffeehouses}\ \neg\exists C \in \text{Coffees}\ \neg\exists S \in \text{Sells}\ (\\
& \quad H.\text{name} = S.\text{coffeehouse} \land \\
& \quad C.\text{name} = S.\text{coffee} \land \\
& \quad R.\text{name} = H.\text{name}\ ) \}
\end{aligned}
$$



```sql
select H.name
from Coffeehouses H
where not exists (
    select *
    from Coffees C
    where not exists (
        select *
        from Sells S
        where H.name = S.coffeehouse
          and C.name = S.coffee));
```

Example Query

- Names of drinkers who frequent all coffeehouses



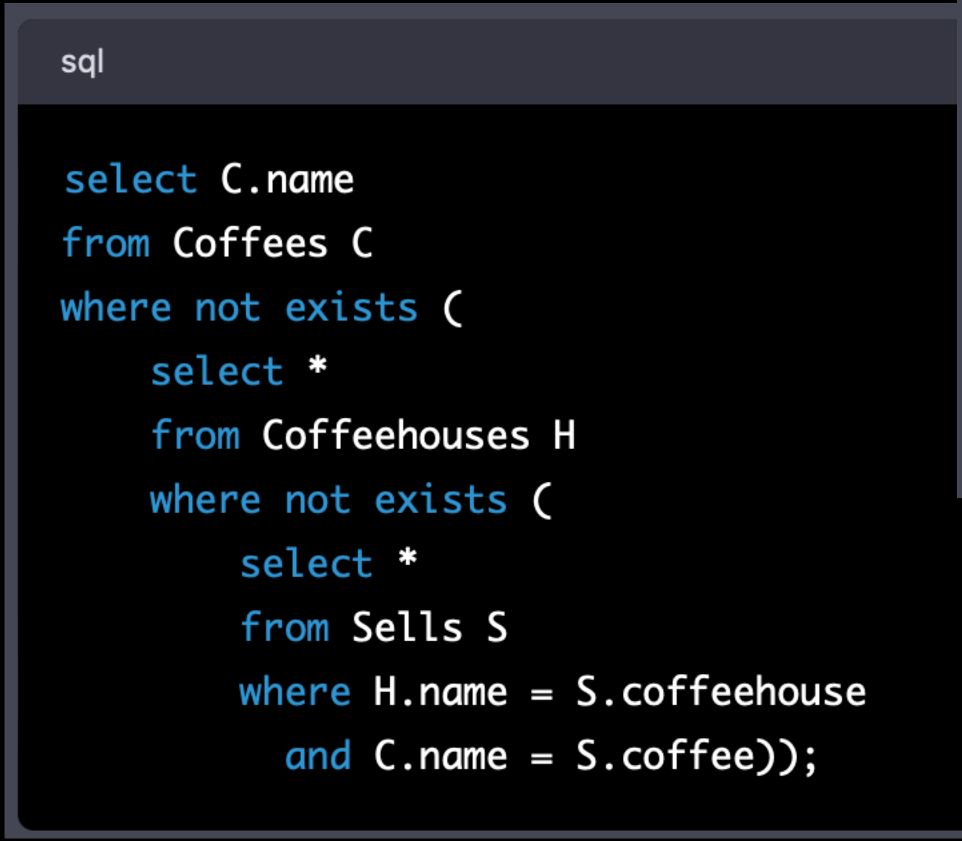
```sql
select D.name
from Drinkers D
where not exists (
    select *
    from Coffeehouses H
    where not exists (
        select *
        from Frequents F
        where D.name = F.drinker
          and H.name = F.coffeehouse
    )
);
```

Practice

- Names of coffees that are sold at all coffeehouses

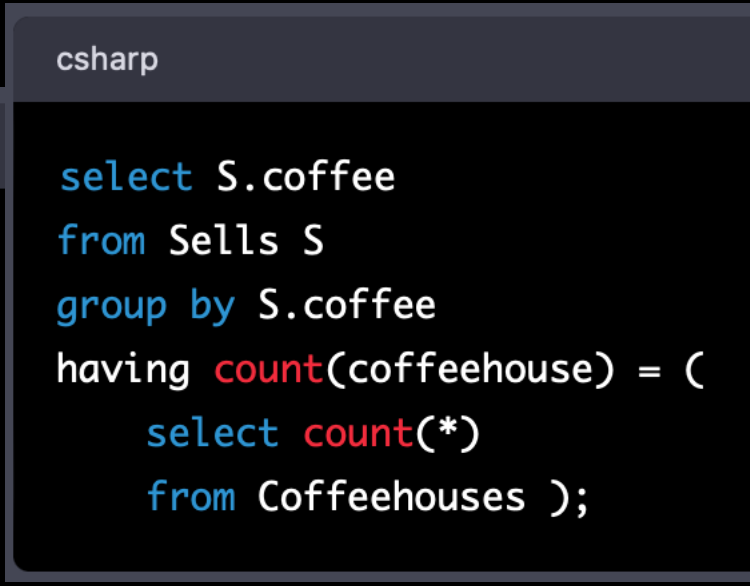
Double Negation vs. Counting

- **Are they completely equivalent?**



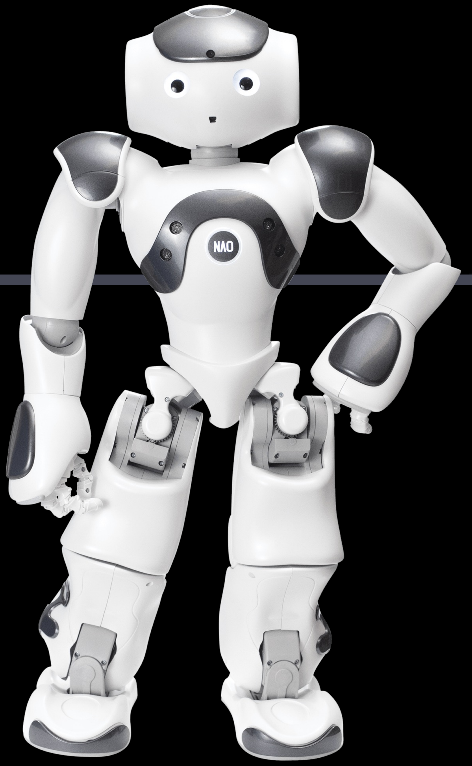
sql

```sql
select C.name
from Coffees C
where not exists (
    select *
    from Coffeehouses H
    where not exists (
        select *
        from Sells S
        where H.name = S.coffeehouse
          and C.name = S.coffee));
```



csharp

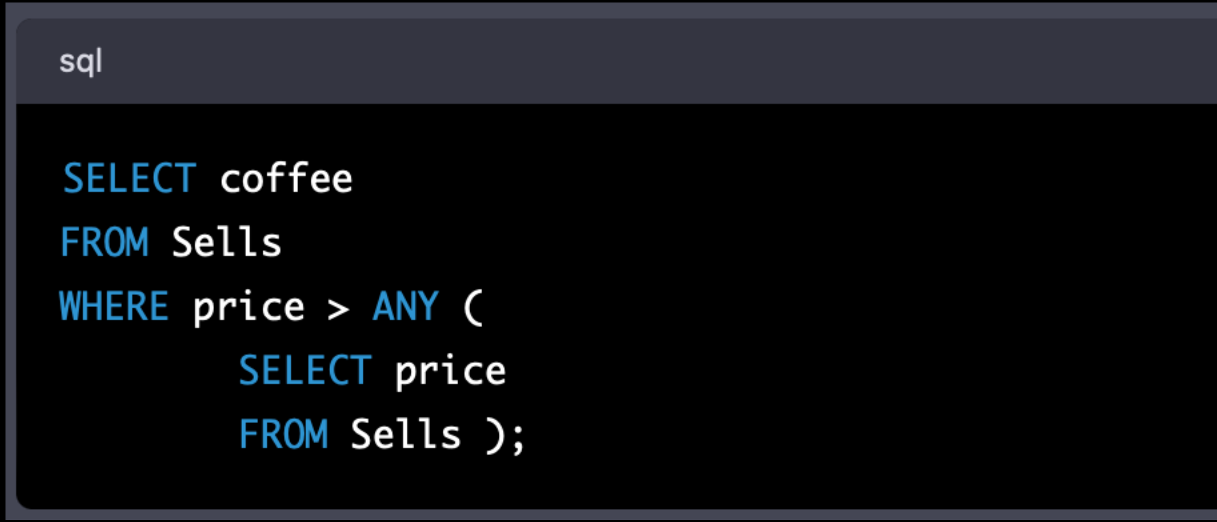
```csharp
select S.coffee
from Sells S
group by S.coffee
having count(coffeehouse) = (
    select count(*)
    from Coffeehouses );
```



The Operator ANY

- **x = ANY(<subquery>) is a boolean condition that is true iff x equals at least one tuple in the subquery result.**
  - = could be any comparison operator.

- **Example: x > ANY(<subquery>) means x is not the uniquely smallest tuple produced by the subquery (greater than at least one).**
  - Note tuples must have one component only.



```sql
SELECT coffee
FROM Sells
WHERE price > ANY (
        SELECT price
        FROM Sells );
```

Practice

- For each coffeehouse, show all coffees that are more expensive than some other coffee sold at that bar

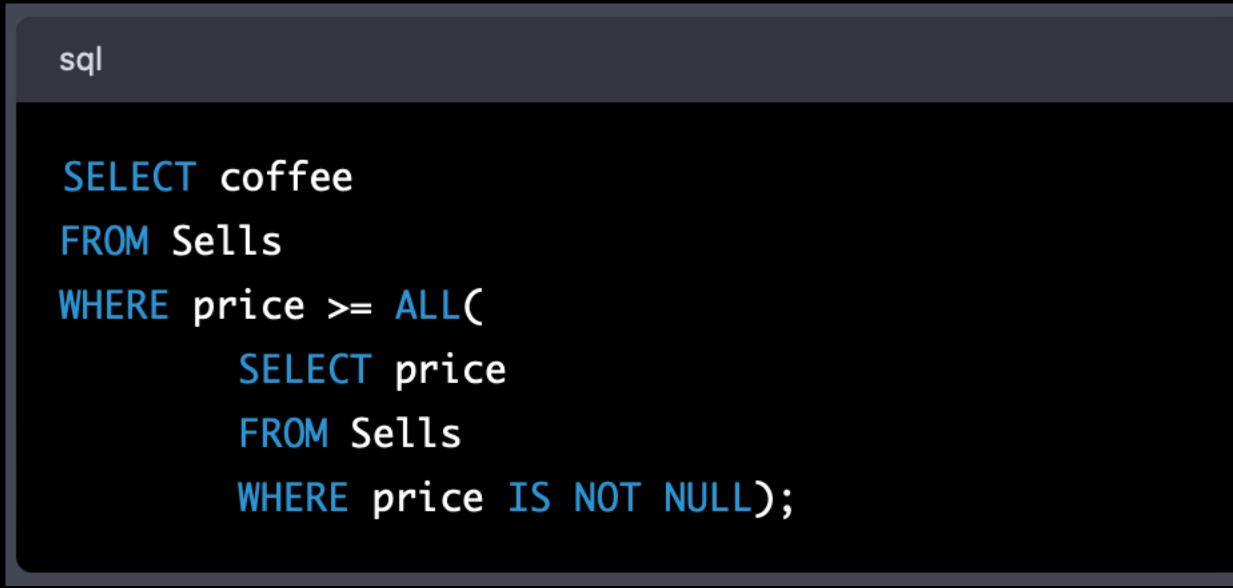
The Operator ALL

- **x <> ALL(<subquery>) is true iff for every tuple t in the relation, x is not equal to t.**
  - That is, x is not in the subquery result.

- **<> can be any comparison operator.**

- **Example: x >= ALL(<subquery>) means there is no tuple larger than x in the subquery result.**

- **From Sells(coffeehouse, coffee, price), find the coffee(s) sold for the highest price.**



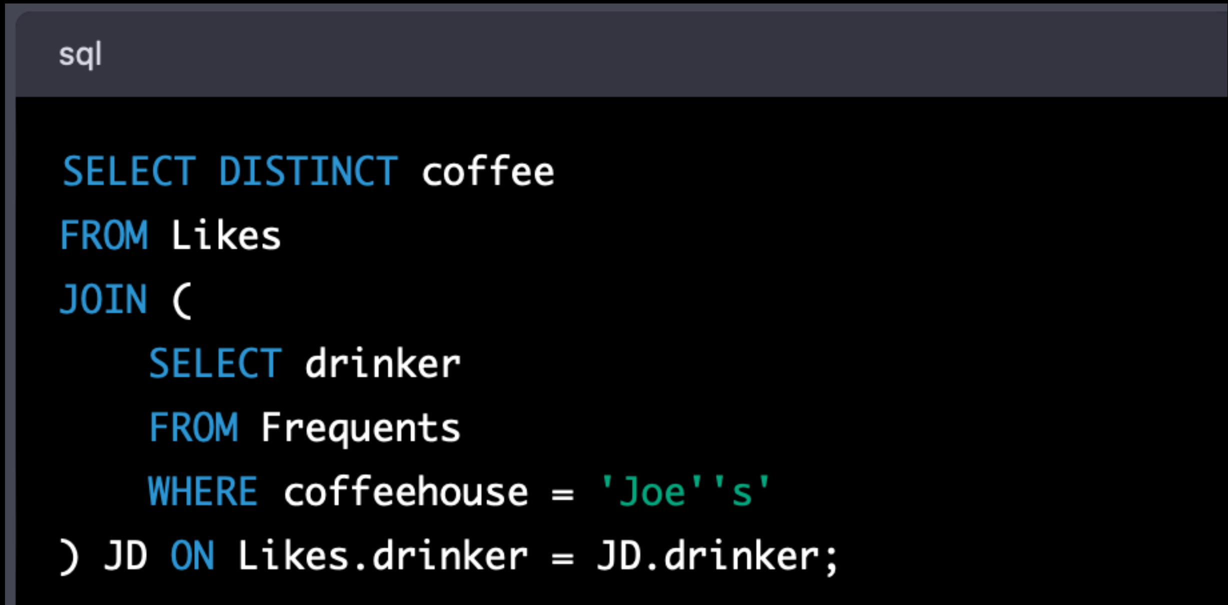
```sql
SELECT coffee
FROM Sells
WHERE price >= ALL(
        SELECT price
        FROM Sells
        WHERE price IS NOT NULL);
```

Practice

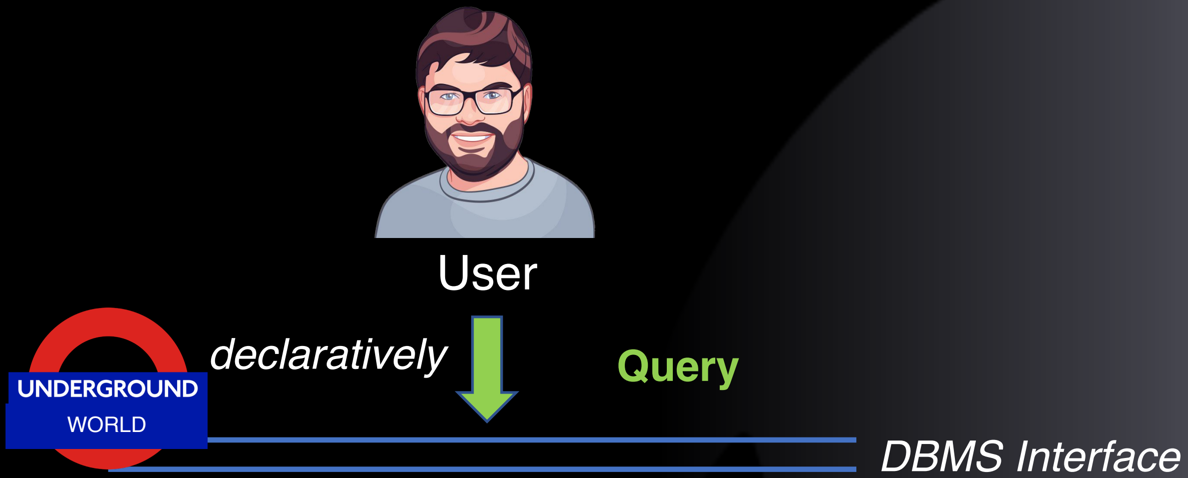
- For each coffeehouse, show the most expensive coffee(s) sold at that coffeehouse

Subqueries in FROM

- A parenthesized SELECT-FROM-WHERE statement (subquery) can be used as a value in a number of places, including WHERE and FROM clauses.
- In place of a relation in the FROM clause, we can use a subquery and then query its result.
- Must use a tuple-variable to name tuples of the result.
- Find the coffees liked by at least one person who frequents Joe’s.



```sql
SELECT DISTINCT coffee
FROM Likes
JOIN (
    SELECT drinker
    FROM Frequents
    WHERE coffeehouse = 'Joe''s'
) JD ON Likes.drinker = JD.drinker;
```



SQL DML -- Views

Readings:

PDBM 1

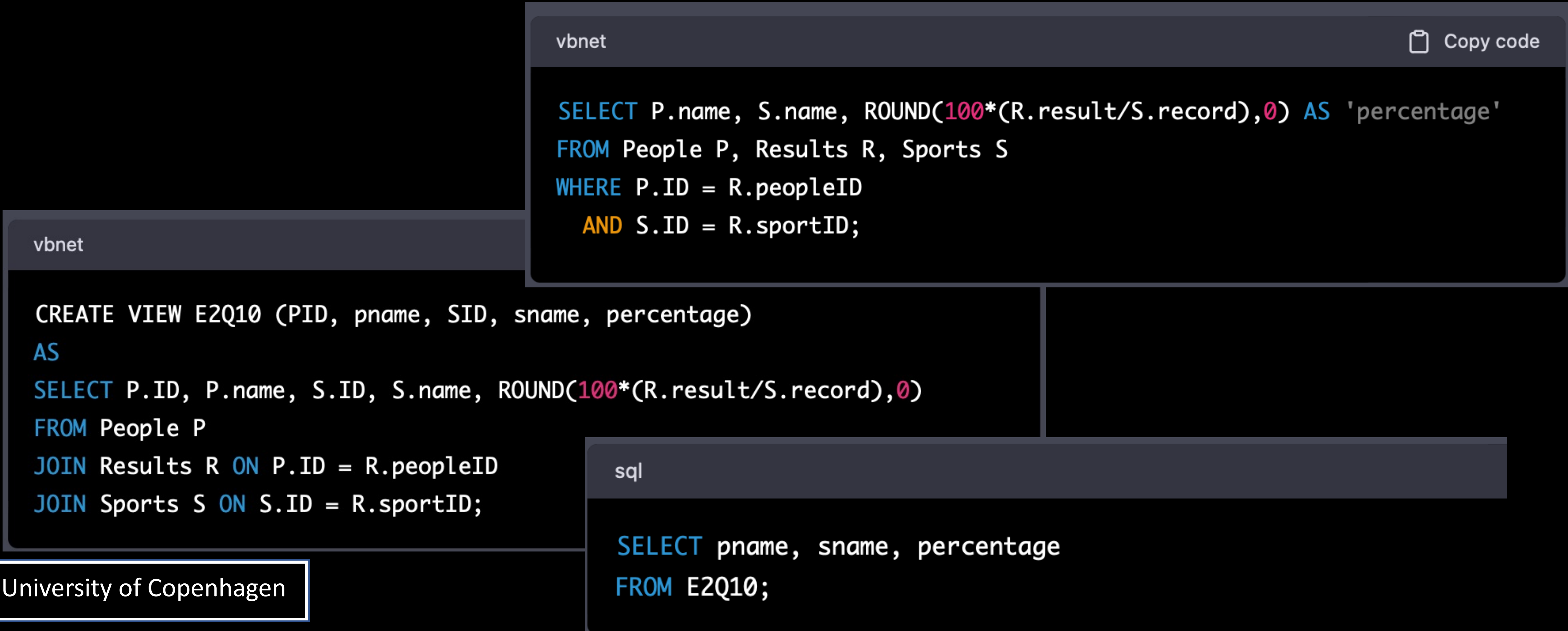
What is a View?

- **A virtual table constructed from actual tables on the fly**
  - Can be accessed in queries like any other table
  - Not materialized, constructed when accessed
  - Similar to a subroutine in ordinary programming
  - Is a schema element

```
CREATE VIEW Name( Columns, ... )
AS
    SQL QUERY
```

Example from the Sports DB

- For each result, the name of the athlete, the name of the sport, and the percentage of the record achieved by the result (a result that is a record should therefore appear as 100; this column should be named “percentage”).



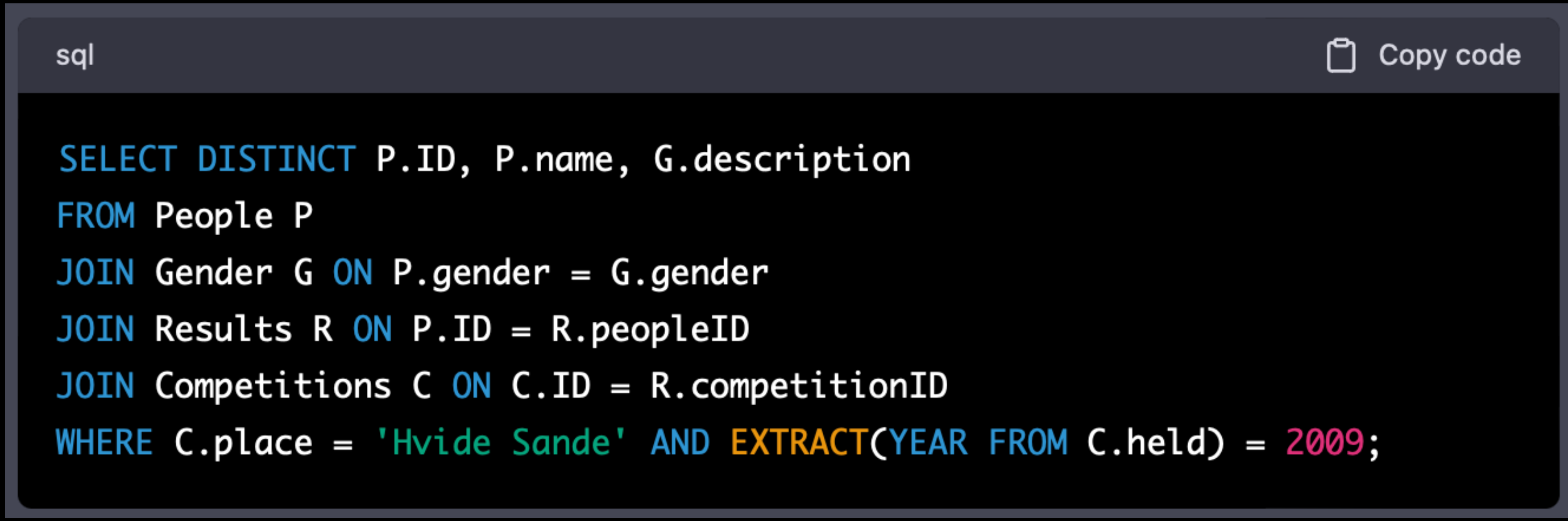
```
SELECT P.name, S.name, ROUND(100*(R.result/S.record),0) AS 'percentage'
FROM People P, Results R, Sports S
WHERE P.ID = R.peopleID
  AND S.ID = R.sportID;
```

```
CREATE VIEW E2Q10 (PID, pname, SID, sname, percentage)
AS
SELECT P.ID, P.name, S.ID, S.name, ROUND(100*(R.result/S.record),0)
FROM People P
JOIN Results R ON P.ID = R.peopleID
JOIN Sports S ON S.ID = R.sportID;
```

```
SELECT pname, sname, percentage
FROM E2Q10;
```

Practice from the Sports DB

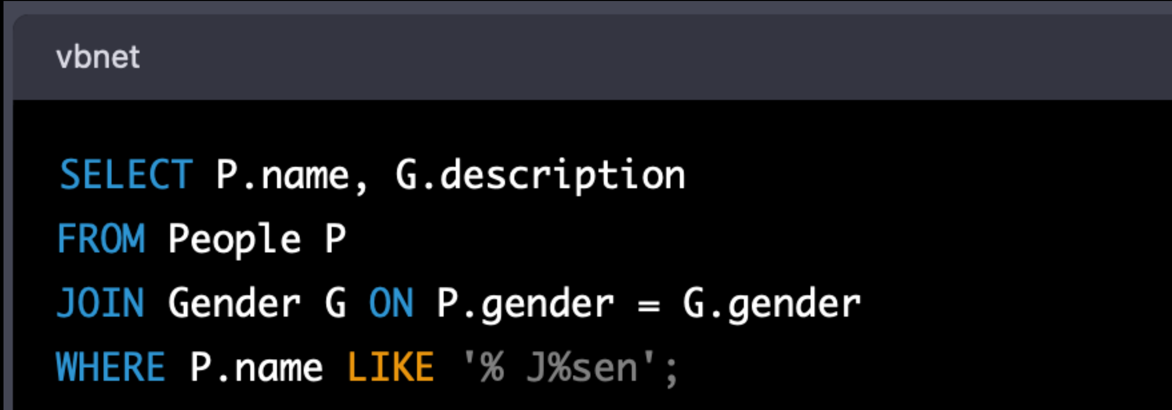
- **Create a view for this query and write the query using the view: The ID, name and gender of all athletes who participated in the competition held in Hvide Sande in 2009.**



sql Copy code

```sql
SELECT DISTINCT P.ID, P.name, G.description
FROM People P
JOIN Gender G ON P.gender = G.gender
JOIN Results R ON P.ID = R.peopleID
JOIN Competitions C ON C.ID = R.competitionID
WHERE C.place = 'Hvide Sande' AND EXTRACT(YEAR FROM C.held) = 2009;
```

- **Write the query using the view for: The name and gender of all people with last name that starts with a “J” and ends with a “sen”.**

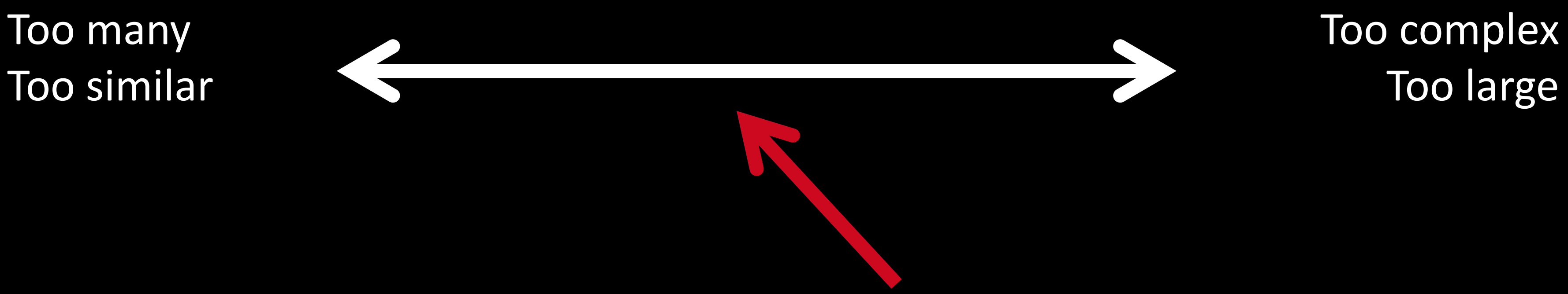


vbnet

```
SELECT P.name, G.description
FROM People P
JOIN Gender G ON P.gender = G.gender
WHERE P.name LIKE '% J%sen';
```

Why Views?

- **Simpler queries – for development, maintenance!**
- **Code re-use – view used across applications**
- **Access management – grant access to views only**
- **Physical data independence – views can mask changes**
- **Can be overused!**



Too many
Too similar

Too complex
Too large

Takeaways

1 Block

- select, from, where, group by, having, sort by clauses

Many query blocks

- Multi-block queries
  - Connected by UNION, EXCEPT, INTERSECT
- Sub-queries in FROM or WHERE clause
  - Blocks connected by =, IN, EXISTS, ANY, ALL
  - Opposites (usually) with NOT
  - BUT: Try to use JOIN when possible!
- Division is one important type of queries with >1 query block!

What is next?

Next week: Triggers, Functions, Transactions, SQL & DBMS programming in Java

- One single lecture for both B.Sc. and M.Sc.

Homework 1 is out

- learnIT: instructions and quiz (homework)
- Help from Tas today at 14:15
- Deadline: 24.02.2023 at 23:59:59
- Remember: 3 out of 4 MAs to sit on the exam