Intermediate SQL (1)

NULL in SQL

Null is a special value in SQL to represent **missing data**

- Null means **attribute missing**
- the results of comparing null with something is *unexpected*
    - i.e. it is not known whethere two things that are unknown are equal or not

A simple solution is to declare *everything* as NOT NULL

- **using a higher normal form (basically anything above third NF) and then NULL attributes disappear almost entirely**
    - otherwise you have to memorise a bunch of (silly) *special* comparators:

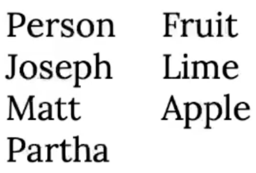
```
SELECT * FROM fruit WHERE fruit IS NULL; // if one person in



SELECT * FROM fruit WHERE fruit IS NOT NULL; // would return
```

Clearly testuing for equality when something is NULL is problematic…

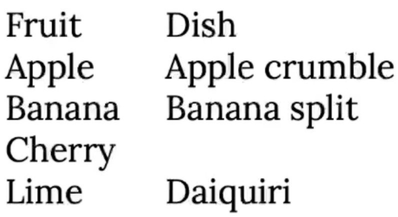
Joining NULL Tables

What happens when you try to join two tables together which contain NULL values?

for example:



Person	Fruit
Joseph	Lime
Matt	Apple
Partha	



Fruit	Dish
Apple	Apple crumble
Banana	Banana split
Cherry	
Lime	Daiquiri

- left database table is people matched to their favoruite fruit, with Partha having a NULL entry
- right table suggests the best thing to make using each fruit, with the cherry having an NULL value

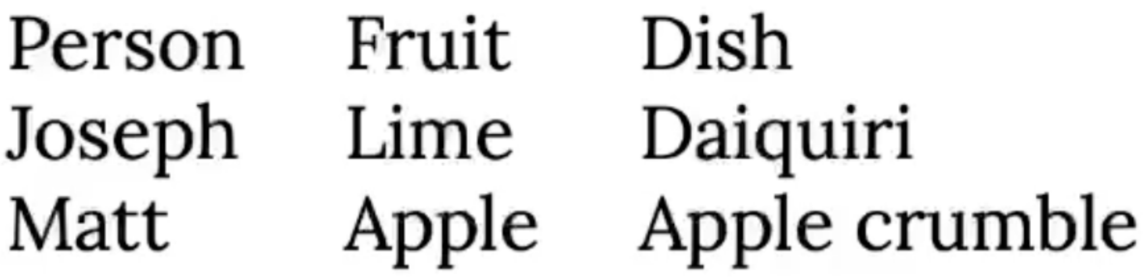
So say you wanted to use both these tables to decide what to make for a specific person based on their fruit preferance?

- to do this you could use a **natural join**

Natural Joins

- similar to regular joins but it **assumes the samed named columns ought to be equal**

applying this we get this table:



Person	Fruit	Dish
Joseph	Lime	Daiquiri
Matt	Apple	Apple crumble

- But Partha has been excluded
- how do we get him in the table? - even if we dont know his favoruite fruit and in turn dish

LEFT & RIGHT JOIN

When doing the previous JOIN we only wanted rows that matched

- this is technically known as an **inner join**

Sometimes we’re okay with the database sticking NULL in if we want to keep columns where a join *can’t* be made…

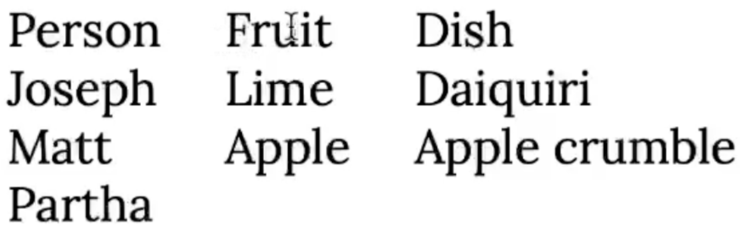
Two versions of this: left and right

Left JOIN

- returns all records from the left table and the **matched** records from the right table.
    - The result is NULL from the right side if there is no match
- basically says whatever the table is on the left side of the join, if you cant make a join with the thing on the right, then it is okay to stick in NULLs

- display the **entire table on the left**, then if there is any matching data based on joining the primary and foreign keys, pull in any data from the table on the right and join them together
    - e.g.

```
SELECT person, fruit.fruit, dish
FROM fruit
LEFT JOIN recipes
ON fruit.fruit = recipes.fruit;
```



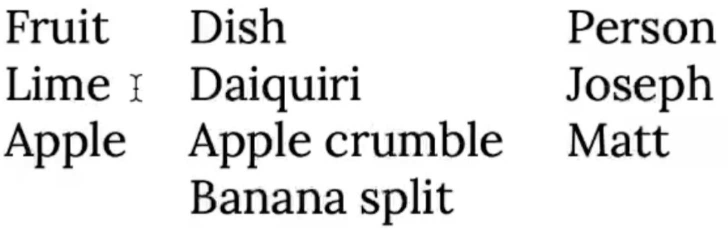
Person	Fruit	Dish
Joseph	Lime	Daiquiri
Matt	Apple	Apple crumble
Partha		

- as Partha didnt have a fave fruit, we stick in a NULL value to the Dish column as we dont know what to join

Right JOIN

- exactly the opposite of a left join
- display the **entire table on the right**
    - If there are any matches, then we pull in any matches from the left
- returns all records from the right table and the matched records from the left table, if no match exists, the result is NULL on both sides

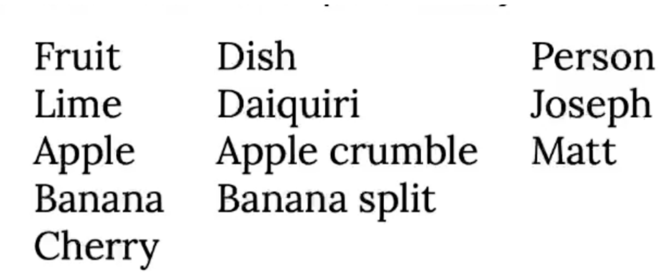
```
SELECT person, fruit.fruit, dish
FROM fruit
RIGHT JOIN recipes
ON fruit.fruit = recipes.fruit;
```



Fruit	Dish	Person
Lime	Daiquiri	Joseph
Apple	Apple crumble	Matt
	Banana split	

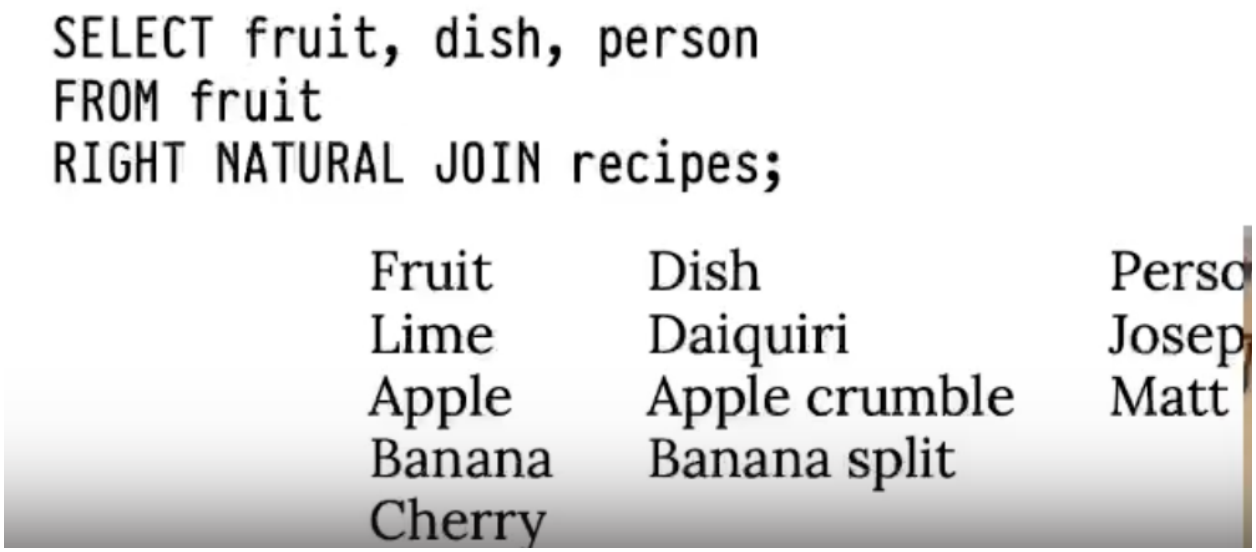
or to select the fruit colum from the recipes table

```
SELECT recipes.fruit, dish, person
FROM fruit
RIGHT JOIN recipes
ON fruit.fruit = recipes.fruit;
```



Fruit	Dish	Person
Lime	Daiquiri	Joseph
Apple	Apple crumble	Matt
Banana	Banana split	
Cherry		

Alternatively you could also do a natural join here whic would usually take care of it:



```
SELECT fruit, dish, person
FROM fruit
RIGHT NATURAL JOIN recipes;
```

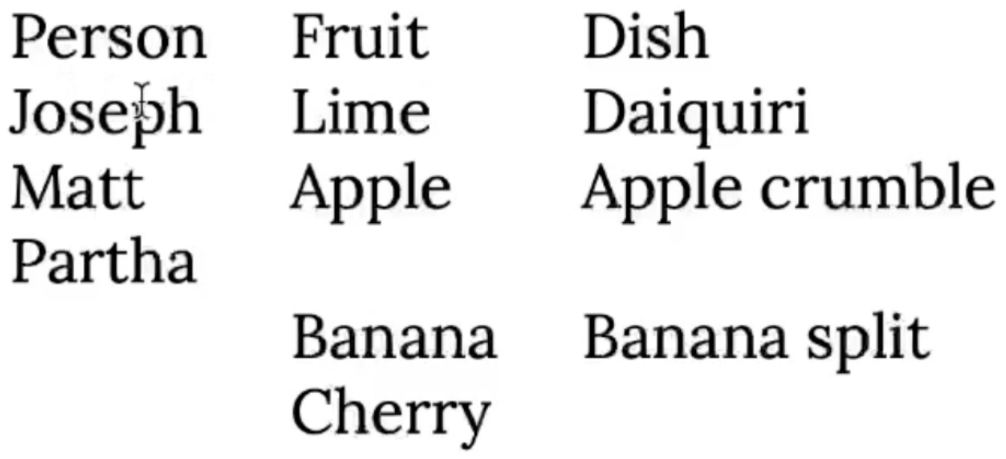
Fruit	Dish	Perso
Lime	Daiquiri	Josep
Apple	Apple crumble	Matt
Banana	Banana split	
Cherry		

Full Outer JOIN

- **what if we want to do a LEFT and a RIGHT JOIN at the same time?**

- combines the results of both left and right join, returning all records when there is a match **in either table or both**
    - if there is no match, the result set will still include a row for each unmathced record from both tables, filling it with NULL values of the side of the table where the **match is missing**

```
SELECT *
FROM fruit
FULL OUTER NATURAL JOIN recipes;
```



Person	Fruit	Dish
Joseph	Lime	Daiquiri
Matt	Apple	Apple crumble
Partha		
	Banana	Banana split
	Cherry	

- e.g. partha we know nothing about them
- bananas a we klnow we can make a banana split from them but we dont know who likes them
- cherry we know nothing about

In practise you wont really need joins other than a natural as you shouldnt usually have NULLs in the data

Statistical Functions

- in SQL Basics, `COUNT` was introduced a s a way to count how many things exist
- what does COUNT do with NULL?
  - it **ignores it**
- other stats functions do not however

Lets say we wanted to make a new table with personal rankings of fruit preference:

Fruit	Stars
Apple	0
Banana	4
Cherry	NULL
Lime	5

If we wanted to find the average stars from the whole table:

```
SELECT AVG(stars) AS Average FROM ranking;
```

Average
3.0

- average calculates average ranking but also **ignores the nulls**

what about a way of finding the **mean** stars:

```
SELECT SUM(starts)/COUNT(fruit) AS Average
FROM ranking;
```

Average

2

- 4 + 5 + 0 = 9
    - 9 / 5 (total number of fruit)
- so why did we end up with 2? - rounding errors due to computation
- SQL isnt good at mathematics...

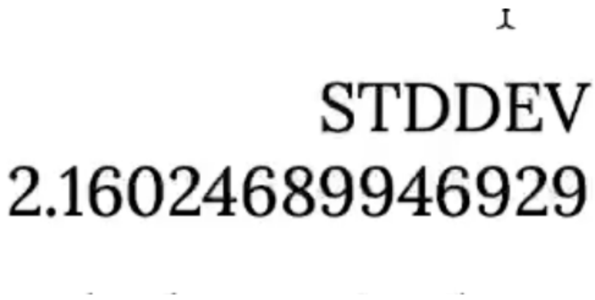
- as this is *ordinal data* to find the average we shoudlnt be finding the mean anyway

Joe recommends not using SQL for this sort of work, use SQL for storing the data, then export it into a programming language for data analysis

Finding the Standard Deviation

- **how far ON AVERAGE something deviates from the mean**
    - gives idea of how spread out the values are

```
SELECT SQRT(AVG(Deviation)) AS STDDEV
FROM (
        SELECT Fruit, Stars, Mean,
                (Stars-mean)*(Stars-Mean) AS Deviation
        FROM ranking JOIN (
                SELECT AVG(stars) AS Mean
                FROM ranking
        )
        WHERE starts IS NOT NULL   // doesnt include null values
);
```



```
STDDEV
2.16024689946929
```

- this highlights how you can nest SQL queries inside one another, these are known as **subqueries**
- this however makes SQL **SLOW**
  - if this is done too much

So generally it is best to use SQL for data retrieval and leave complex statistical analysis to programming langauges such as R or Python