$$\Pi_{S.name}\left[\left(\sigma_{b.color=Red}(Sailors \bowtie Reservations \bowtie Boats)\right) \cup \left(\sigma_{b.color=Blue}(Sailors \bowtie Reservations \bowtie Boats)\right)\right]$$

↓

```
SELECT s.name
FROM sailors s, reservations r, boats b
WHERE s.sid=r.sid and r.bid = b.bid and b.color=
                                              Red
UNION
SELECT s.name
FROM sailors s, reservations r, boats b
WHERE s.sid=r.sid and r.bid = b.bid and b.color=
                                              Green
```

↓

```
SELECT s.name
FROM sailors s, reservations r, boats b
WHERE s.sid = r.sid AND r.bid = b.bid
    AND (b.color = Red or b.color = Green);
```

But what if we want names of sailors who

have reserved a Red boat and a Green boat

Can we just replace OR with AND?

↳ NO. Because we are only doing a query on one boat at a time...

{

```
SELECT s.sname
FROM   sailors s, reserv r1, boats b1, reserv r2, boats b2
WHERE  s.sid = r1.sid and s.sid = r2.sid and r1.bid = b1.bid and
       r2.bid = b2.bid and b1.color = "Red" and
                           b2.color = "Green"
```

⇓

UGLY

Q: Sailors who have reserved a red or green boat

↓

∪ (UNION)

Q: " " red and green boat

↓

∩ (INTERSECT)

Q: " "red but not green boats"

↓

— (EXCEPT)

★ INTERSECT and EXCEPT are not implemented in MySQL
(we will see how to do these operations later)

Q: Find all sids of sailors who have a rating > 50 or have reserved boat #11?

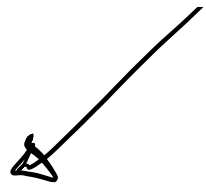
Find sids of sailors who have rating > 50?

↳
```
SELECT sid
FROM sailors
WHERE rating > 50;
```

Find sids of sailors who have reserved boat #11?

↳
```
SELECT sid
FROM reservations
WHERE bid = 11;
```
{

)



```
SELECT sid
FROM sailors
WHERE rating > 50
UNION
SELECT sid
FROM reservations
WHERE bid = 103
```

SQL Joins

- efficiency-wise for simple commands these are the same as implicit joins (using FROM and WHERE)
- but they are more clear
  - they explicitly state that a join is happening
  - they keep the σ or Θ part of the join together w/ the join syntax
- they also, for more complex queries, let you control the order that operations occur in
  - ex. allow filtering before a JOIN

- you will not be tested explicitly on JOINs in SQL

- JOINs become necessary when dealing with NULL values in foreign keys.

INNER JOIN

- same as $\bowtie_\theta$, same as using FROM, WHERE

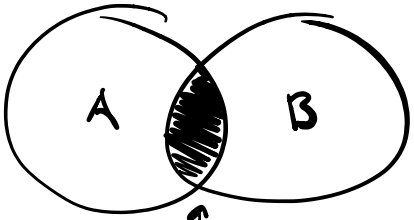
```
SELECT *
FROM table_a a
INNER JOIN table_b b
ON a.key = b.key;
```

⇓

same as

⇓

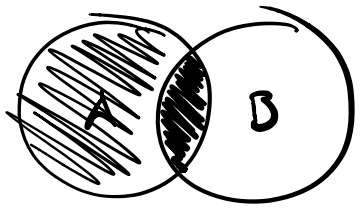
```
SELECT *
FROM table_a a, table_b b
WHERE a.key = b.key;
```



↑ tuples that match on θ

LEFT JOIN

```
SELECT *
FROM table_a a
 LEFT JOIN table_b b
ON a.key = b.key;
```



RIGHT JOIN

```
SELECT *
FROM table_a a
RIGHT JOIN table_b b
ON a.key = b.key
```