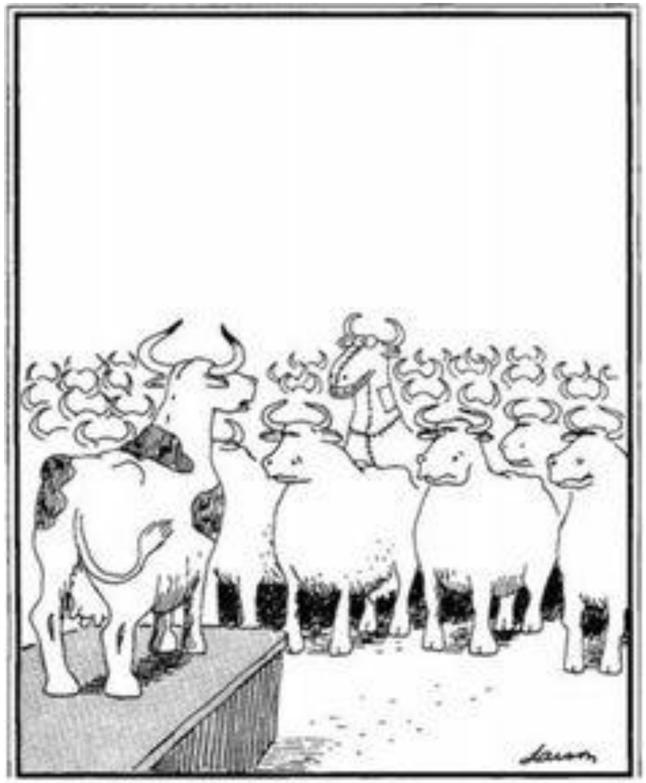
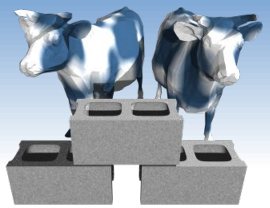
SQL: More Advanced Queries

I made a small change to problem set #1.

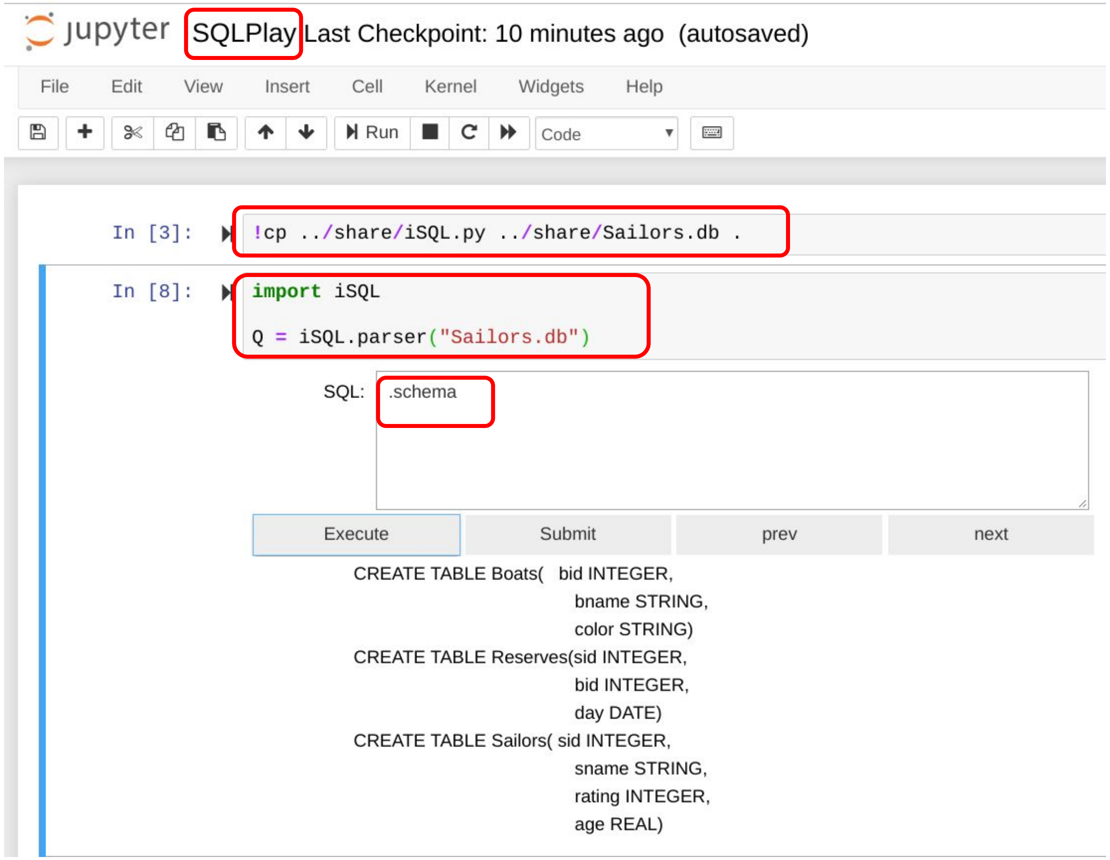
It should now show a version number of 1.1 and automaitcally load as a Python3 notebook



"The revolution has been postponed . . . We've discovered a leak."



Let's go SQLing!



```
In [3]: !cp ../share/iSQL.py ../share/Sailors.db .
```

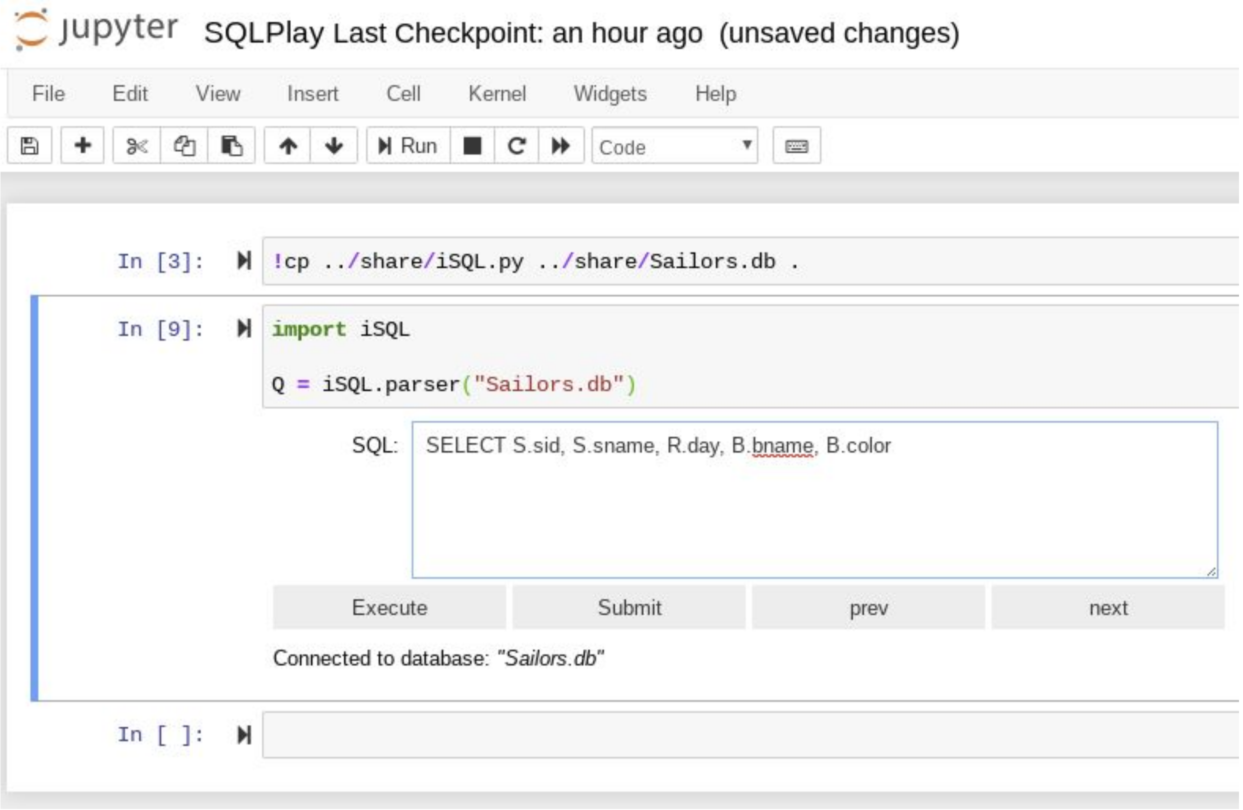
```
In [8]: import iSQL

        Q = iSQL.parser("Sailors.db")
```

SQL: .schema

```
CREATE TABLE Boats(   bid INTEGER,
                      bname STRING,
                      color STRING)
CREATE TABLE Reserves(sid INTEGER,
                      bid INTEGER,
                      day DATE)
CREATE TABLE Sailors( sid INTEGER,
                      sname STRING,
                      rating INTEGER,
                      age REAL)
```

1. Go to your Jupyter hub
2. Create a new Python3 Notebook
3. Copy iSQL.py and Sailors.db to your Notebook directory
4. import and run iSQL
5. Try ".schema"
6. Try "SELECT *
        FROM Sailors"

A little more...



jupyter SQLPlay Last Checkpoint: an hour ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code

```
In [3]:  !cp ../share/iSQL.py ../share/Sailors.db .

In [9]:  import iSQL

         Q = iSQL.parser("Sailors.db")
```

SQL: SELECT S.sid, S.sname, R.day, B.bname, B.color

Connected to database: "Sailors.db"

```
In [ ]:
```

Generate a list of reservations that includes

Sailor's id,
Sailor's name,
Reservation date,
Boat's name, and
Boat's color

How many rows are considered in the full cross product of Sailors x Reserves x Boats

What relation determines the actual number of rows?

SQL's Aggregate Operators

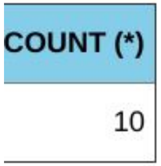
- Significant SQL extension
- Computation and summarization operations
- Appears in *target-list* of query
- Results *aggregate* rather than appear individually

COUNT (*)
COUNT ([DISTINCT] A)
SUM ([DISTINCT] A)
AVG ([DISTINCT] A)
MAX (A)
MIN (A)

single column

- E.x. How many instances in the sailor relation?

```
SELECT COUNT (*)
FROM   Sailors
```



COUNT (*)
10

More examples

- Average age of Sailors with a rating of 10?

```
SELECT  AVG(S.age)
FROM    Sailors S
WHERE   S.rating=10
```

- Names of Sailors having the maximum rating

```
SELECT S.sname, S.rating
FROM   Sailors S
WHERE  S.rating=(SELECT MAX(S2.rating)
                 FROM   Sailors S2)
```

sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

More examples (cont)

- How many distinct ratings for Sailors less than 40 years of age?

```
SELECT COUNT(DISTINCT S.rating)
FROM   Sailors S
WHERE  S.age < 40.0
```

sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

- How many reservations were made by Sailors less than 40 years old?

```
SELECT  COUNT(*)
FROM    Sailors S, Reserves R
WHERE   S.sid = R.sid AND S.age < 40
```

Find name and age of the oldest sailor(s)

- The first query is incorrect! (Switch the S.age to S.rating to see why)
- The third query is equivalent to the second query, but may not be supported in some systems.

```
SELECT S.sname, MAX(S.age)
FROM   Sailors S
```

```
SELECT S.sname, S.age
FROM   Sailors S
WHERE  S.age =
       (SELECT MAX(S2.age)
        FROM  Sailors S2)
```

```
SELECT S.sname, S.age
FROM   Sailors S
WHERE  (SELECT MAX(S2.age)
        FROM   Sailors S2)
        = S.age
```

Motivation for Grouping

- So far, we've applied aggregate operators to *all* (qualifying) tuples. Sometimes, we want to apply them to *groups*.
- Consider: *Find the age of the youngest sailor for each rating level.*
  - In general, we don't know how many rating levels exist, and what the rating values for these levels are!
  - Suppose we know that rating values go from 1 to 10; we can write 10 queries that look like this (!):

For *i* = 1, 2, ... , 10:

```
SELECT MIN(S.age)
FROM   Sailors S
WHERE  S.rating = i
```

Queries With GROUP BY *and* HAVING

```
SELECT      [DISTINCT]  target-list
FROM        relation-list
WHERE       qualification
GROUP BY    grouping-list
HAVING      group-qualification
```

- The *target-list* contains
  (i) attribute names
  (ii) terms with aggregate operations (e.g., MIN (*S.age*)).
- The attribute list (i) must be a subset of *grouping-list*. Intuitively, each answer tuple corresponds to a *group,* and these attributes must have a single value per group. (A *group* is a set of tuples that have the same value for all attributes in *grouping-list*.)

Conceptual Evaluation

- The cross-product of *relation-list* is computed, tuples that fail *qualification* are discarded, *unnecessary* fields are deleted, and the remaining tuples are partitioned into groups by the value of attributes in *grouping-list*.
- The *group-qualification* is then applied to eliminate some groups. Expressions in *group-qualification* must have a *single value per group*!
    - In effect, an attribute in *group-qualification* that is not an argument of an aggregate op also appears in *grouping-list*. (SQL does not exploit primary key semantics here!)
- One answer tuple is generated per qualifying group.

Find age of the youngest sailor with age ≥ 18, for each rating having at least 2 such sailors

```
SELECT    S.rating, MIN(S.age) AS minage
FROM      Sailors S
WHERE     S.age >= 18
GROUP BY  S.rating
HAVING    COUNT(*) > 1
```

Sailors instance:

sid	sname	rating	age
22	dustin	7	45.0
29	brutus	1	33.0
31	lubber	8	55.5
32	andy	8	25.5
58	rusty	10	35.0
64	horatio	7	35.0
71	zorba	10	16.0
74	horatio	9	35.0
85	art	3	25.5
95	bob	3	63.5
96	frodo	3	25.5

Answer relation:

rating	minage
3	25.5
7	35.0
8	25.5

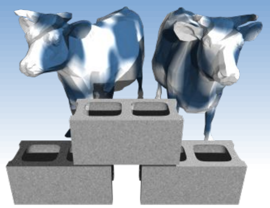
Find age of the youngest sailor with age ≥ 18, for each rating with at least 2 such sailors

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

rating	age
1	33.0
3	25.5
3	63.5
3	25.5
7	45.0
7	35.0
8	55.5
8	25.5
9	35.0
10	35.0

rating	minage
3	25.5
7	35.0
8	25.5

Find age of the youngest sailor with age ≥ 18, for each rating level with at least 2 such sailors, and where every sailor is under 60.



HAVING COUNT (*) > 1 AND MAX(S.age) < 60

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

→

rating	age
1	33.0
3	25.5
3	63.5
3	25.5
7	45.0
7	35.0
8	55.5
8	25.5
9	35.0
10	35.0

→

rating	minage
7	35.0
8	25.5

Find age of the youngest sailor with age ≥ 18, for each rating with at least 2 sailors between 18 and 60.

```
SELECT   S.rating, MIN(S.age) AS minage
FROM     Sailors S
WHERE    S.age >= 18 AND S.age <= 60
GROUP BY S.rating
HAVING   COUNT(*) > 1
```

Sailors instance:

sid	sname	rating	age
22	dustin	7	45.0
29	brutus	1	33.0
31	lubber	8	55.5
32	andy	8	25.5
58	rusty	10	35.0
64	horatio	7	35.0
71	zorba	10	16.0
74	horatio	9	35.0
85	art	3	25.5
95	bob	3	63.5
96	frodo	3	25.5

Answer relation:

rating	minage
7	35.0
8	25.5

For each red boat, find the number of times it has been reserved

```
SELECT   B.bid, COUNT(*) AS scount
FROM     Sailors S, Boats B, Reserves R
WHERE    S.sid=R.sid AND R.bid=B.bid AND B.color='red'
GROUP BY B.bid
```

- Grouping over a join of three relations.
- What do we get if we remove *B.color=‘red’* from the WHERE clause and add a HAVING clause with this condition?
- What if we drop Sailors and the condition involving S.sid?

Find age of the youngest sailor with age > 18, for each rating with at least 2 sailors (of any age)

```
SELECT    S.rating, MIN(S.age)
FROM      Sailors S
WHERE     S.age >= 18
GROUP BY S.rating
HAVING  1 < (SELECT COUNT(*)
             FROM   Sailors S2
             WHERE  S.rating=S2.rating)
```

- Shows HAVING clause can also contain a subquery.
- Compare this with the query where we considered only ratings with 2 sailors over 18!
- What if HAVING clause is replaced by:
  - HAVING COUNT(*) >1

Find the rating for which the average age is the minimum over all ratings

- Aggregate operations cannot be nested! WRONG:

```
SELECT S.rating
FROM   Sailors S
WHERE  S.age = (SELECT MIN(AVG(S2.age)) FROM Sailors S2)
```

- Correct solution:

```
SELECT rating
FROM   (SELECT S.rating, AVG(S.age) AS aveage
        FROM     Sailors S
        GROUP BY S.rating)
WHERE aveage = (SELECT MIN(aveage)
                FROM (SELECT S.rating, AVG(S.age) AS aveage
                      FROM   Sailors S
                      GROUP BY S.rating))
```

Summary

- SQL was an important factor in the early acceptance of the relational model; more natural than earlier, procedural query languages.
- Many alternative ways to write a query; optimizer should look for most efficient evaluation plan.
  - In practice, users need to be aware of how queries are optimized and evaluated for best results.