# More SQL

INFO/CS 2300:
Intermediate Web Design and Programming

1 sheet on half wall

# Grades – P2 and HW1

As soon as we can

# Coming up

Homework 2

- Given queries in English, translate them into SQL.
- Due Tuesday March 7 at 5pm
- Don't share SQL

Friday quiz and activity:

databases / SQL – 3 lectures

# No script

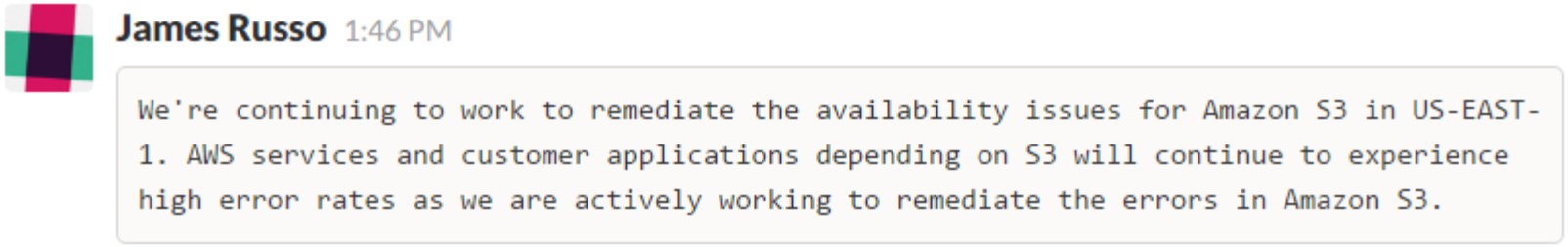
Or What does one do when Piazza goes down on the day 2 assignments are due

Help the decision-maker(s)

- Provide documentation of the problem
- Clearly state what needs to be solved
- Find a working communications channel

# What was useful


"Students are trying to access the assignment files"



# Click In!

# Click In!

Sailors( sailorId, sailorName, rating, age )

Boats( boatId, boatName, color )

Reserves( sailorId, boatId, day )

Which table(s) have foreign keys?

A. Sailors

B. Boats

C. Reserves

D. Sailors and Boats

E. Sailors, Boats and Reserves

# Click In!

Sailors( sailorId, sailorName, rating, age )

Boats( boatId, boatName, color )

Reserves( sailorId, boatId, day )

Which table(s) have foreign keys?

A. Sailors

B. Boats

C. Reserves

D. Sailors and Boats

E. Sailors, Boats and Reserves

# Click In!

We can connect information from two tables using:

A. JOIN

B. COMBINE

C. DISTINCT

D. SELECT

E. WHERE

# Click In!

We can connect information from two tables using:

A. JOIN

B. COMBINE

C. DISTINCT

D. SELECT

E. WHERE

# Basic SQL

# Selection

Recall the basic form of a SQL statement:

```
SELECT Fields
FROM Table
WHERE Condition;
```

| title | year | length |
|---|---|---|
| Gladiator | 2000 | 155 |
| A Beautiful Mind | 2001 | 135 |
| Chicago | 2002 | 113 |
| The Return of the King | 2003 | 201 |
| Million Dollar Baby | 2004 | 132 |

```
SELECT title, length
FROM movies
WHERE length > 150 AND title LIKE '%King%';
```

# Inline calculation

We can modify output of table on the fly.

```
SELECT
  title,
  year,
  length/60 AS Hours
FROM movies;
```

# INNER JOIN 2 tables

students( NetID, FirstName, LastName)

courses( CourseID, Dept, Number, Time, Semester)

registrations( NetID, CourseID)

```
SELECT
  students.NetID,
  students.FirstName,
FROM registrations
  INNER JOIN students
    ON registrations.NetID = students.NetID
WHERE registrations.CourseID = 12345;
```

# INNER JOIN 3 tables

students( NetID, FirstName, LastName)

courses( CourseID, Dept, Number, Time, Semester)

registrations( NetID, CourseID)

```
SELECT
  students.NetID,
  students.FirstName,
  courses.Number
FROM registrations
  INNER JOIN students
    ON registrations.NetID = students.NetID
  INNER JOIN courses
    ON registrations.CourseID = courses.CourseID
WHERE courses.Dept = 'Computer Science';
```

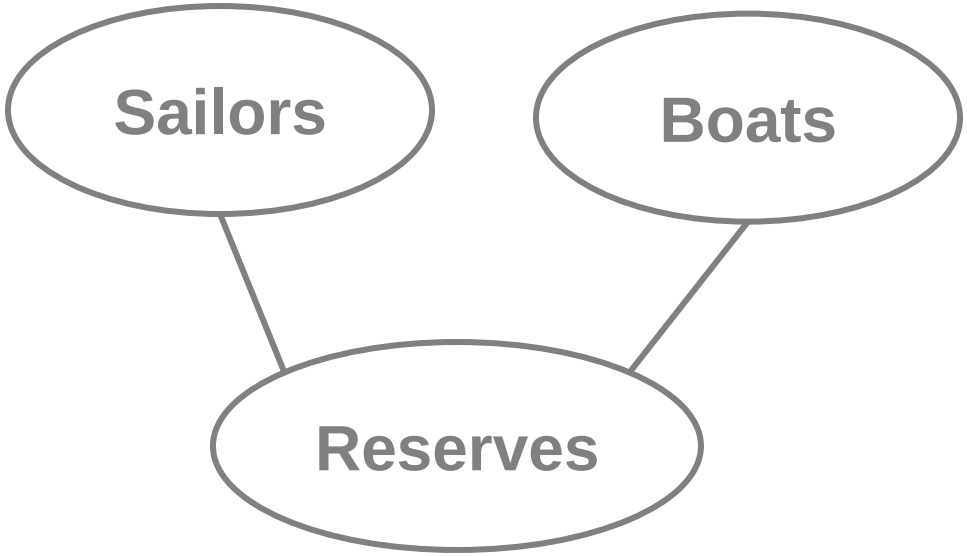
# Sailors, Boats and Reserves

Sailors(
sailorId: integer,
sailorName: string,
rating: integer,
age: integer)

Boats(
boatId: integer,
boatName: string,
color: string)

Reserves(
sailorId: integer,
boatId: integer,
day: date)

What do we notice about this schema?

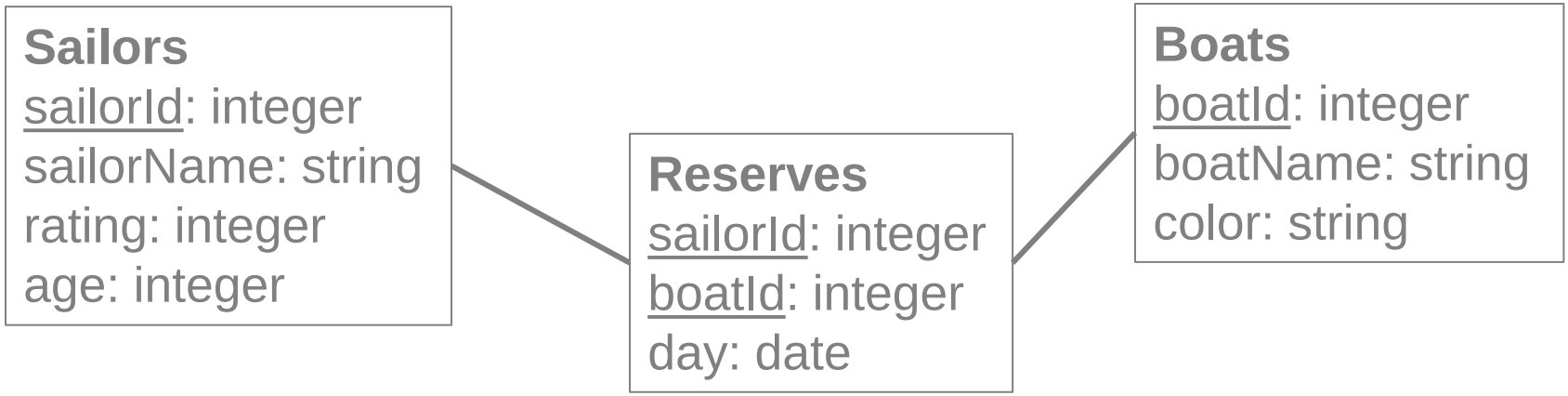


# Exercises 1 - 3

1. Find the names and ages of all sailors with a rating above 7.

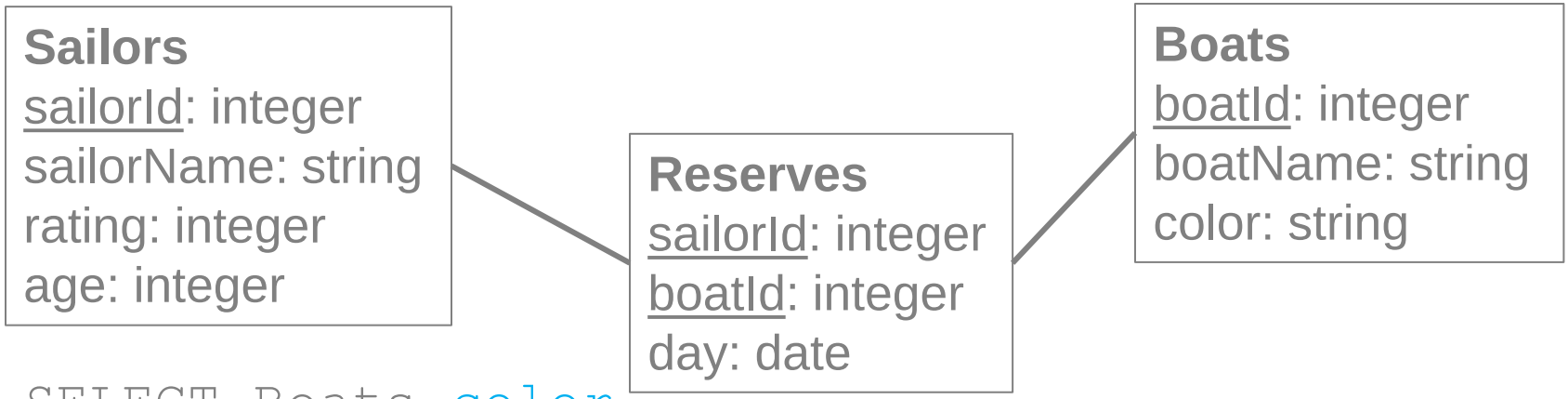
```
SELECT sailorName, age
FROM Sailors
WHERE rating > 7;
```

# 2. Find the names of the sailors who have reserved boat number 103



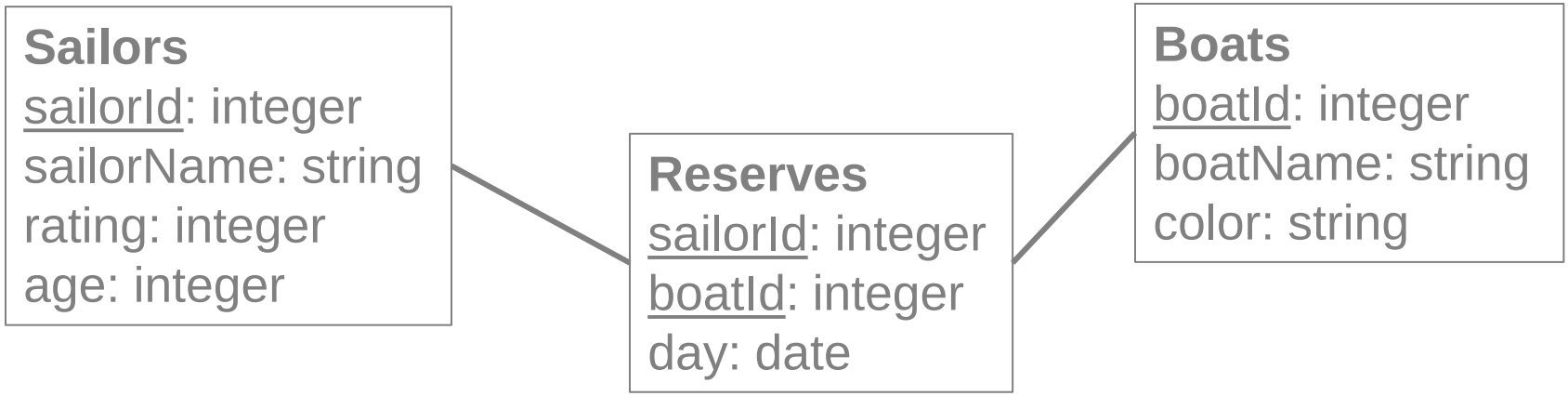

```
SELECT sailorName
 FROM Sailors INNER JOIN Reserves
   ON Sailors.sailorId = Reserves.sailorId
WHERE
 boatID = 103;
```

# 3. Find the colors of the boats reserved by the sailor named ‘Lubber’.




```
SELECT Boats.color
 FROM
    INNER JOIN
      ON
    INNER JOIN
      ON
WHERE
 Sailor.sailorName = ‘Lubber’;
```

# 3. Find the colors of the boats reserved by the sailor named ‘Lubber’.




```
SELECT Boats.color
 FROM Reserves
   INNER JOIN Sailors
     ON Reserves.sailorID = Sailors.sailorID
   INNER JOIN Boats
     ON Reserves.boatID = Boats.boatID
WHERE
 sailorName = ‘Lubber’;
```

# 3. Find the colors of the boats reserved by the sailor named ‘Lubber’.

```
SELECT Boats.color
 FROM Reserves
   INNER JOIN Sailors
     ON Reserves.sailorID = Sailors.sailorID
   INNER JOIN Boats
     ON Reserves.boatID = Boats.boatID
WHERE
 sailorName = ‘Lubber’;
```

Need "DISTINCT" here to prevent duplicates

Don’t need table name because sailorName is unique among field names

Table name required because boatID is not unique among field names

# INNER JOIN

| Title | Year | Length |
|---|---|---|
| Gladiator | 2000 | 155 |
| A Beautiful Mind | 2001 | 135 |
| Chicago | 2002 | 113 |
| The Return of the King | 2003 | 201 |
| Million Dollar Baby | 2004 | 132 |

| Name | Title | Year |
|---|---|---|
| Russell Crowe | Gladiator | 2000 |
| Russell Crowe | A Beautiful Mind | 2001 |
| Viggo Mortensen | Return of the King | 2003 |
| Hillary Swank | Million Dollar Baby | 2004 |

```
SELECT movies.title
FROM movies
   INNER JOIN starsIn
      ON movies.title = starsIn.title
      AND movies.year = starsIn.year
```

| Title |
|---|
| Gladiator |
| A Beautiful Mind |
| The Return of the King |
| Million Dollar Baby |

# Outer joins

| Title | Year | Length |
|---|---|---|
| Gladiator | 2000 | 155 |
| A Beautiful Mind | 2001 | 135 |
| Chicago | 2002 | 113 |
| The Return of the King | 2003 | 201 |
| Million Dollar Baby | 2004 | 132 |

| Name | Title | Year |
|---|---|---|
| Russell Crowe | Gladiator | 2000 |
| Russell Crowe | A Beautiful Mind | 2001 |
| Viggo Mortensen | Return of the King | 2003 |
| Hillary Swank | Million Dollar Baby | 2004 |

What if we want all the movies whether or not there is a record in StarsIn and we want the Name from StarsIn if there is one?

# (Left) Outer joins

```
SELECT
  movies.title,
  movies.year,
  movies.length,
  starsin.name
FROM movies
  LEFT OUTER JOIN starsin
    ON movies.title = starsin.title
     AND movies.year = starsin.year;
```

With Outer Joins the order of the tables matters

Equivalent:
starsin RIGHT OUTER JOIN movies

| movies.title | movies.year | movies.length | starsin.name |
|---|---|---|---|
| A Beautiful Mind | 2001 | 135 | Russell Crowe |
| Chicago | 2002 | 113 | (null) |
| Gladiator | 2000 | 155 | Russell Crowe |
| Million Dollar Baby | 2004 | 132 | Hillary Swank |
| The Return of the King | 2003 | 201 | Viggo Mortensen |

# IS NULL

What movies don't have an actor in the starsIn table?

```
SELECT
  movies.title,
FROM movies
  LEFT OUTER JOIN starsIn
    ON movies.year = starsin.year
     AND movies.title = starsin.title
WHERE starsIn.title IS NULL;
```

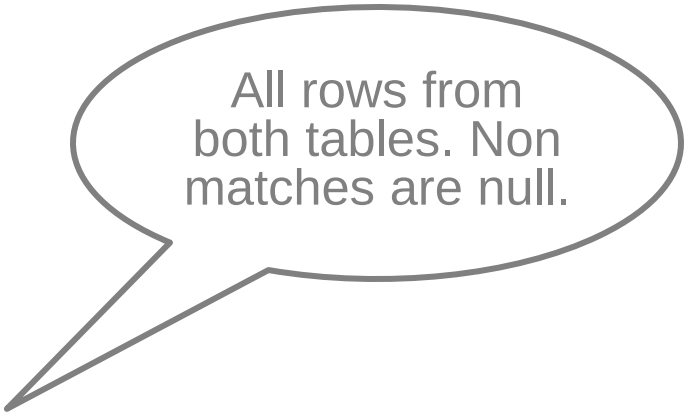
| movies.title |
| --- |
| Chicago |

# FULL JOIN

| title | year | length |
|---|---|---|
| Gladiator | 2000 | 155 |
| A Beautiful Mind | 2001 | 135 |
| Chicago | 2002 | 113 |

| name | title | year |
|---|---|---|
| Russell Crowe | Gladiator | 2000 |
| Russell Crowe | A Beautiful Mind | 2001 |
| Tom Hanks | Big | 1988 |

```
SELECT *
FROM movies m
  FULL JOIN starsIn s
    ON m.year = s.year
     AND m.title = s.title;
```




| m.title | m.year | m.length | s.name | s.title | s.year |
|---|---|---|---|---|---|
| A Beautiful Mind | 2001 | 135 | Russell Crowe | A Beautiful Mind | 2001 |
| Chicago | 2002 | 113 | (null) | (null) | (null) |
| Gladiator | 2000 | 155 | Russell Crowe | Gladiator | 2000 |
| (null) | (null) | (null) | Tom Hanks | Big | 1988 |

# Aggregation and Grouping

# Aggregation

We can aggregate results of a given field across all records of a table.

SUM – sums a field with numerical value

AVG – averages a field with numerical values

MIN, MAX – produces minimum, maximum of a field (either for numbers or strings)

COUNT – counts the number of records

| title | year | length |
|---|---|---|
| Gladiator | 2000 | 155 |
| A Beautiful Mind | 2001 | 135 |
| Chicago | 2002 | 113 |
| The Return of the King | 2003 | 201 |
| Million Dollar Baby | 2004 | 132 |

```
SELECT MAX(length)
  FROM Movies;
```

| MAX(length) |
|---|
| 201 |

```
SELECT AVG(length) as average
  FROM Movies;
```

| average |
|---|
| 147.2 |

```
SELECT COUNT(*)
  FROM Movies
  WHERE Year >= 2002;
```

| Count(*) |
|---|
| 3 |

# Grouping

```
SELECT AVG(length)
FROM Movies
WHERE year = 2000;
```

| title | year | length |
|---|---|---|
| Gladiator | 2000 | 155 |
| Crouching Tiger, Hidden Dragon | 2000 | 120 |
| Moulin Rouge | 2001 | 127 |
| A Beautiful Mind | 2001 | 135 |
| Chicago | 2002 | 113 |
| Lost in Translation | 2003 | 102 |
| The Return of the King | 2003 | 201 |

If we want to know the average length for each year, do we have to run a query for every year?

No

# GROUP BY

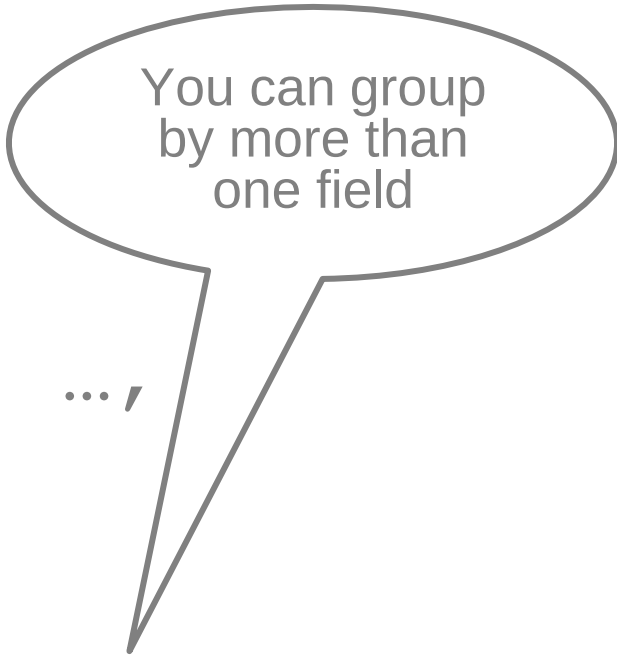
| Title | Year | Length |
|---|---|---|
| Gladiator | 2000 | 155 |
| Crouching Tiger, Hidden Dragon | 2000 | 120 |
| Moulin Rouge | 2001 | 127 |
| A Beautiful Mind | 2001 | 135 |
| Chicago | 2002 | 113 |
| Lost in Translation | 2003 | 102 |
| The Return of the King | 2003 | 201 |

```
SELECT
  Year,
  AVG(Length) AS AvgLength
FROM Movies
GROUP BY Year;
```

| Year | AvgLength |
|---|---|
| 2000 | 137.5 |
| 2001 | 131 |
| 2002 | 113 |
| 2003 | 151.5 |

# Grouping generalized

```
SELECT
  Field1, …,
  Fieldk,
  Aggregate1(B1) AS C1, …,
  AggregateN(Bn) AS Cn
FROM Table
GROUP BY Field1, …, Fieldk;
```




Each field either must be named in the GROUP BY clause or include an aggregate function.

# Selecting groups

| Title | Year | Length |
|---|---|---|
| Gladiator | 2000 | 155 |
| Crouching Tiger, Hidden Dragon | 2000 | 120 |
| Moulin Rouge | 2001 | 127 |
| A Beautiful Mind | 2001 | 135 |
| ~~Chicago~~ | ~~2002~~ | ~~113~~ |
| Lost in Translation | 2003 | 102 |
| The Return of the King | 2003 | 201 |

Suppose we want average length by year but only for years with more than one movie.

Can't use "WHERE" because the criteria aren't known until after the grouping

# Criteria for groups

| Title | Year | Length |
|---|---|---|
| Gladiator | 2000 | 155 |
| Crouching Tiger, Hidden Dragon | 2000 | 120 |
| Moulin Rouge | 2001 | 127 |
| A Beautiful Mind | 2001 | 135 |
| ~~Chicago~~ | ~~2002~~ | ~~113~~ |
| Lost in Translation | 2003 | 102 |
| The Return of the King | 2003 | 201 |

```
SELECT Year, AVG(Length) AS AvgLength
FROM Movies
GROUP BY Year
HAVING COUNT(Title) > 1;
```

| Year | AvgLength |
|---|---|
| 2000 | 137.5 |
| 2001 | 131 |
| 2003 | 151.5 |

# HAVING vs WHERE

‘HAVING’ allows for conditions on *groups*.

‘WHERE’ allows for conditions on *rows*.

A little more on Group By

http://www.tutorialspoint.com/sql/sql-group-by.htm

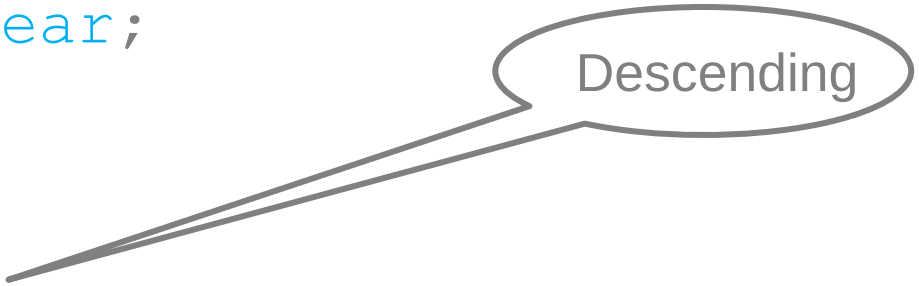
# Sorting and Limits

# Output in sorted order

```
SELECT *
FROM Movies
ORDER BY Length;

SELECT *
FROM Movies
ORDER BY Length, Year;

SELECT *
FROM Movies
ORDER BY Length DESC, Year;
```




Assume but not guaranteed: records are returned in random order unless ORDER BY is used.

# Limit

```
SELECT *
FROM Movies
ORDER BY Length
LIMIT 3;
```

It is particularly important to use ORDER BY when using LIMIT.

# Click In!

# Click In!

We can find the average length of movies by year with the query

A. SELECT Year, AVG(Length) FROM Movies
B. SELECT Year, AVG(Length) FROM Movies GROUP BY Year
C. SELECT AVG(Length) FROM Movies
D. AVG(Length) BY Year

# Click In!

We can find the average length of movies by year with the query

A. SELECT Year, AVG(Length) FROM Movies

B. SELECT Year, AVG(Length) FROM Movies GROUP BY Year

C. SELECT AVG(Length) FROM Movies

D. AVG(Length) BY Year

# Exercises 4 - 5

# 4. Find the age of the youngest sailor for each rating level.

**Sailors**
sailorId: integer
sailorName: string
rating: integer
age: integer

```
SELECT rating, MIN(age)
FROM Sailors
GROUP BY rating;
```

| Rating | MIN(age) |
|---|---|
| 1 | 14 |
| 2 | 18 |
| 3 | 21 |

# 5. Find the average age of sailors for each rating level that has at least two sailors.

**Sailors**
sailorId: integer
sailorName: string
rating: integer
age: integer

```
SELECT rating, AVG(age)
FROM Sailors
GROUP BY rating
HAVING COUNT(*) >=2;
```

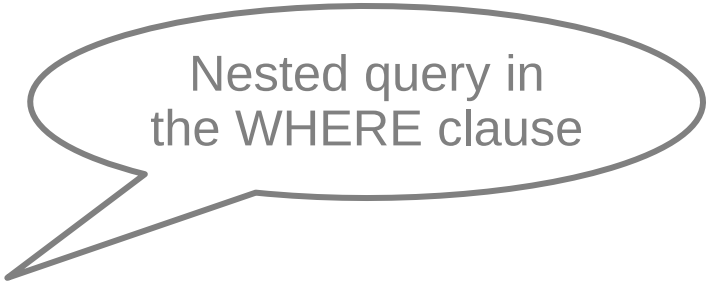
# More SQL: Nested queries

When you know the rules but not the values

# Nested Query

| title | year | length |
|---|---|---|
| Gladiator | 2000 | 155 |
| A Beautiful Mind | 2001 | 135 |
| Chicago | 2002 | 113 |
| The Return of the King | 2003 | 201 |
| Million Dollar Baby | 2004 | 132 |

List the title and year of movies that are longer than average.




```
SELECT title, year
FROM movies
WHERE length > (SELECT AVG(length)
                FROM movies);
```

# Nested values IN

Using a subquery to generate a set of values.

```
SELECT name
FROM starsIn
WHERE ( title, year ) IN
     (SELECT title, year
       FROM movies
       WHERE
         length > (SELECT AVG(length)
                     FROM movies)
     );
```

# Temporary Table

Using a subquery to generate the table for the FROM statement.


```
SELECT Title, Year
FROM (SELECT *
        FROM StarsIn
        WHERE Name <> 'Russell Crowe')
     AS NotRuss
  INNER JOIN Movies
    ON NotRuss.Title = Movies.Title
      AND NotRuss.Year = Movies.Year;
```

# Multi-level nesting

```
SELECT name
FROM starsIn
INNER JOIN
   (SELECT title, year
     FROM movies
     WHERE length > (SELECT AVG(length)
                     FROM movies)
   ) AS longMovies
     ON starsIn.title = longMovies.title
       AND starsIn.year = longMovies.year;
```

# SOME and ALL

For numeric values, we can compare against SOME or ALL of the values from a subquery.

```
SELECT Title, Year
FROM Movies
WHERE Length >= ALL (SELECT Length
                     FROM Movies);
```

# What does this do?

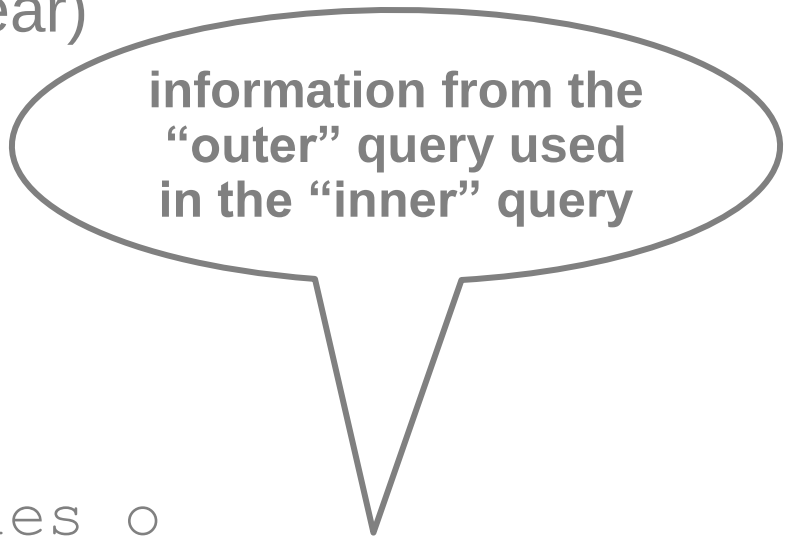
```
SELECT Title, Year
FROM Movies
WHERE Length > SOME (SELECT Length
                     FROM Movies);
```

# EXISTS

What movies are remakes?
(released in more than one year)

E.g.

```
SELECT DISTINCT Title
FROM Movies m
WHERE EXISTS (SELECT *
                FROM Movies o
                WHERE o.Title = m.Title AND
                    o.Year <> m.Year);
```



# Exercises 6 - 9

# More examples

6. Find the names of the sailors with the highest rating.

```
SELECT sailorName
FROM Sailors
WHERE
 rating = ( SELECT MAX(rating)FROM Sailors);

SELECT Sailors.sailorName
FROM Sailors
WHERE
 rating >= ALL (SELECT rating FROM Sailors);
```

7. Find the names of sailors whose rating is better than every sailor named ‘Horatio’.

```
SELECT sailorName
FROM Sailors
WHERE
 rating > ALL ( SELECT rating
                FROM Sailors
                WHERE sailorName = ‘Horatio’);
```

## 8. Find the names of sailors who are older than the oldest sailor with a rating of 10.

```
SELECT sailorName
FROM Sailors
WHERE
 age > ( SELECT MAX(age)
          FROM Sailors
          WHERE rating = 10);
```

Alternatively

```
WHERE
 age > ALL (SELECT age
          FROM Sailors
          WHERE rating = 10);
```

9. Find the names of every sailor who has not reserved a boat.

```
SELECT sailorName
FROM Sailors
 LEFT OUTER JOIN Reserves
   ON Sailors.sailorID = Reserves.sailorID
WHERE
 sailorID IS NULL;
```

Alternatively

```
WHERE
 sailorID NOT IN (SELECT DISTINCT sailorID
                  FROM Reserves);
```

# Review

- Group By allows aggregate functions to run on groups of rows
- We can use the results of SQL queries as part of another SQL query.