CS122A: Introduction to Data Management

Lecture #10
SQL (3): Null, Outer Joins, CRUD, Integrity Constraints

Instructor: Chen Li

Null Values

- Field values in a tuple are sometimes *unknown* (e.g., a rating has not been assigned) or *inapplicable* (e.g., no spouse's name).
  - SQL provides a special value *null* for such situations.
- The presence of *null* complicates many issues. E.g.:
  - Special operators needed to check if value is/is not *null*.
  - Is *rating>8* true or false when *rating* is equal to *null*? What about AND, OR and NOT connectives?
  - We need a 3-valued logic (true, false and *unknown*).
  - Meaning of constructs must be defined carefully. (The WHERE clause eliminates rows that don't evaluate to true.)
  - New operators (in particular, *outer joins*) possible/needed.

Nulls and SQL's 3-Valued Logic

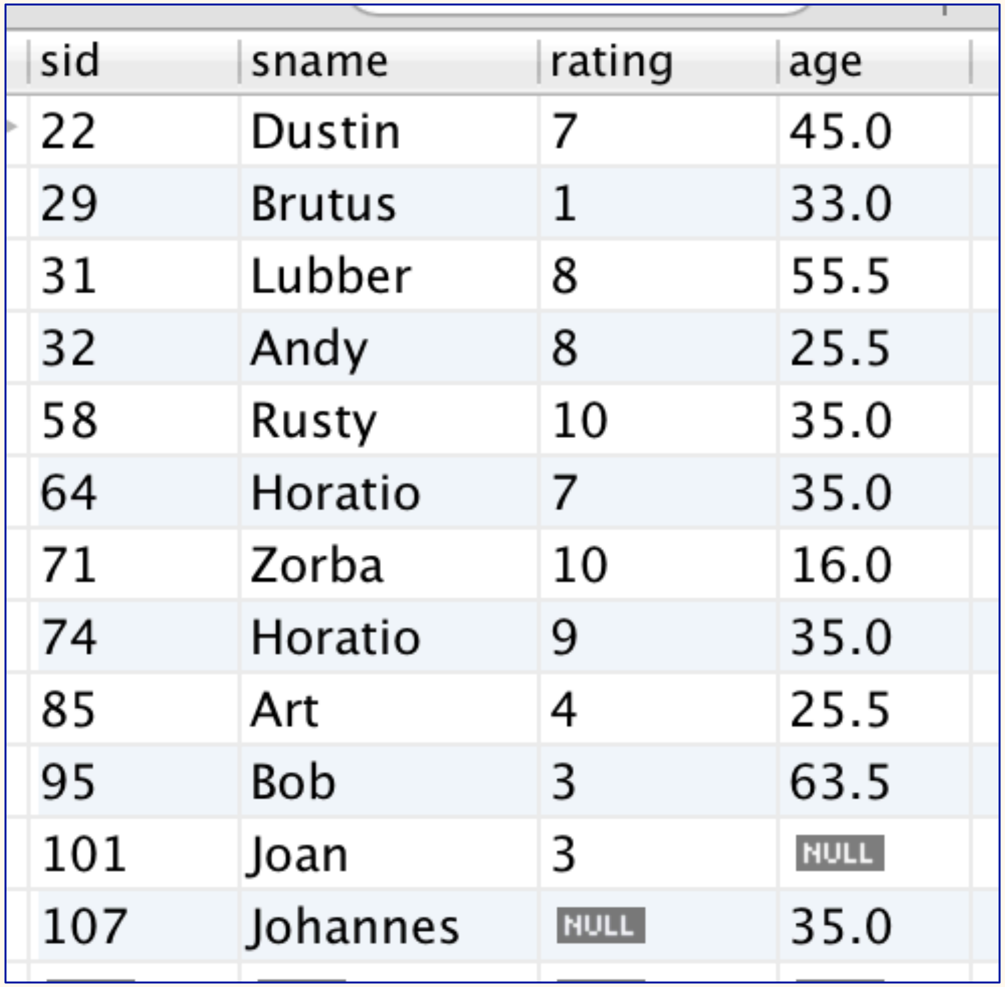
AND	**true**	**false**	**unknown**
true	true	false	unknown
false	false	false	false
unknown	unknown	false	unknown

OR	**true**	**false**	**unknown**
true	true	true	true
false	true	false	unknown
unknown	true	unknown	unknown

NOT	
true	false
false	true
unknown	unknown

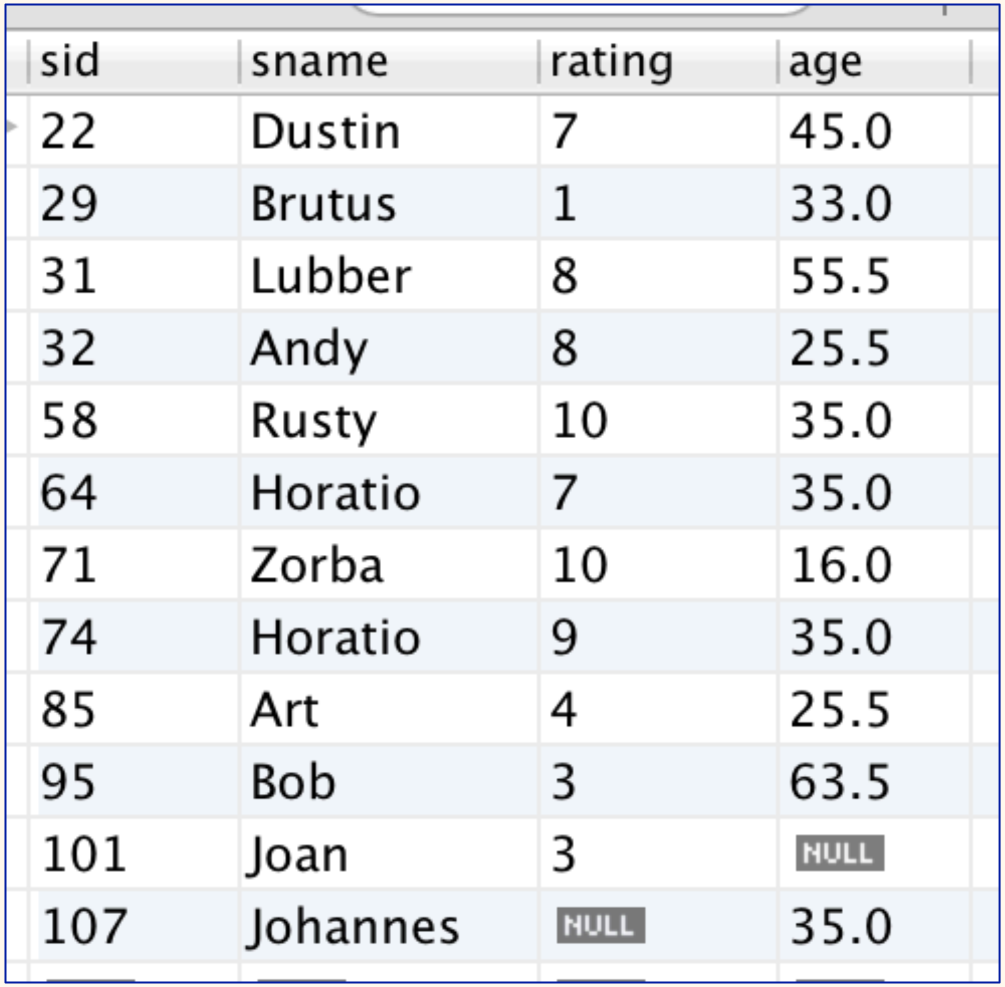
Note: SQL arithmetic expressions involving **null** values will yield **null** values (*Ex:* EMP.sal + EMP.bonus)

Ex: Sailors With Some Null Values



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	4	25.5
95	Bob	3	63.5
101	Joan	3	NULL
107	Johannes	NULL	35.0

Ex: SPJ Queries on Sailors w/Nulls



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	4	25.5
95	Bob	3	63.5
101	Joan	3	NULL
107	Johannes	NULL	35.0

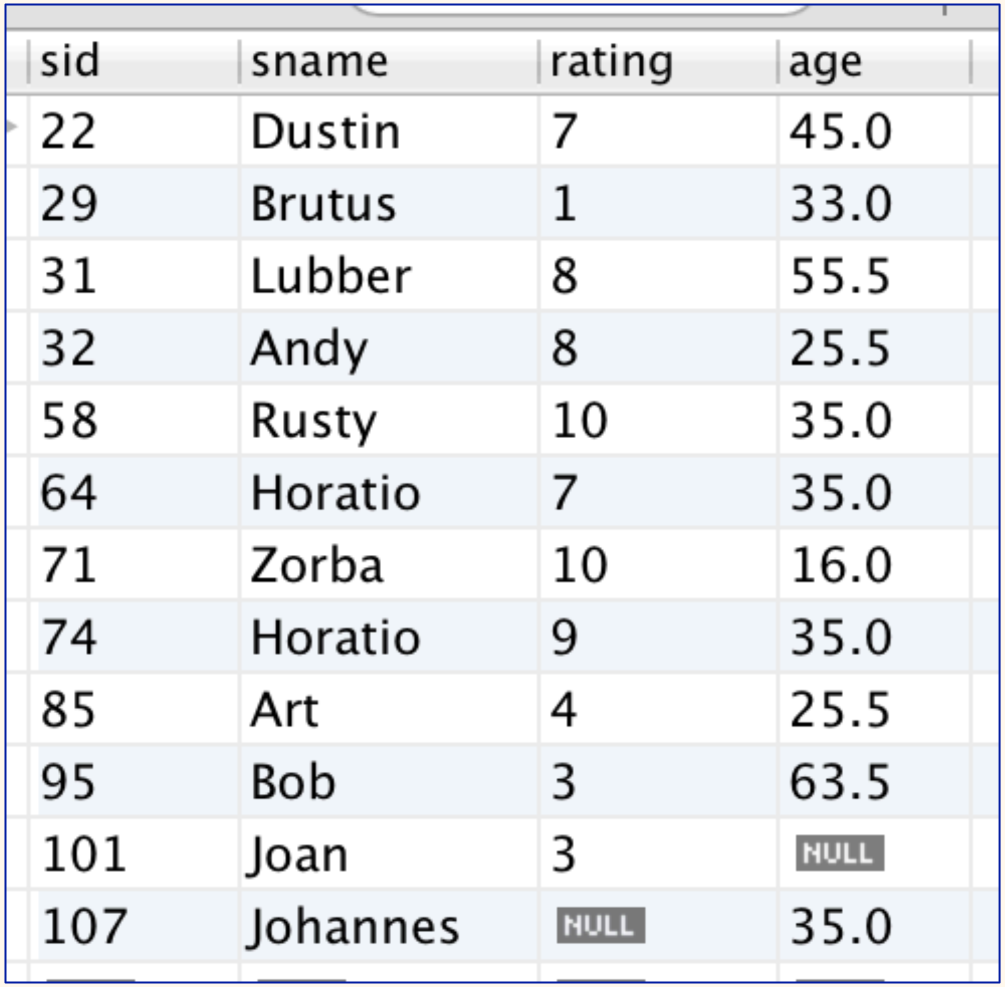
```
SELECT  *
FROM  Sailors S
WHERE age > 35.0
```

```
SELECT  *
FROM  Sailors S
WHERE age <= 35.0
```

```
SELECT COUNT(*)
FROM  Sailors S
WHERE age > 35.0
        OR age <= 35.0
        OR age IS NULL
```

"age IS NOT NULL"

Other Illogical results!



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	4	25.5
95	Bob	3	63.5
101	Joan	3	NULL
107	Johannes	NULL	35.0

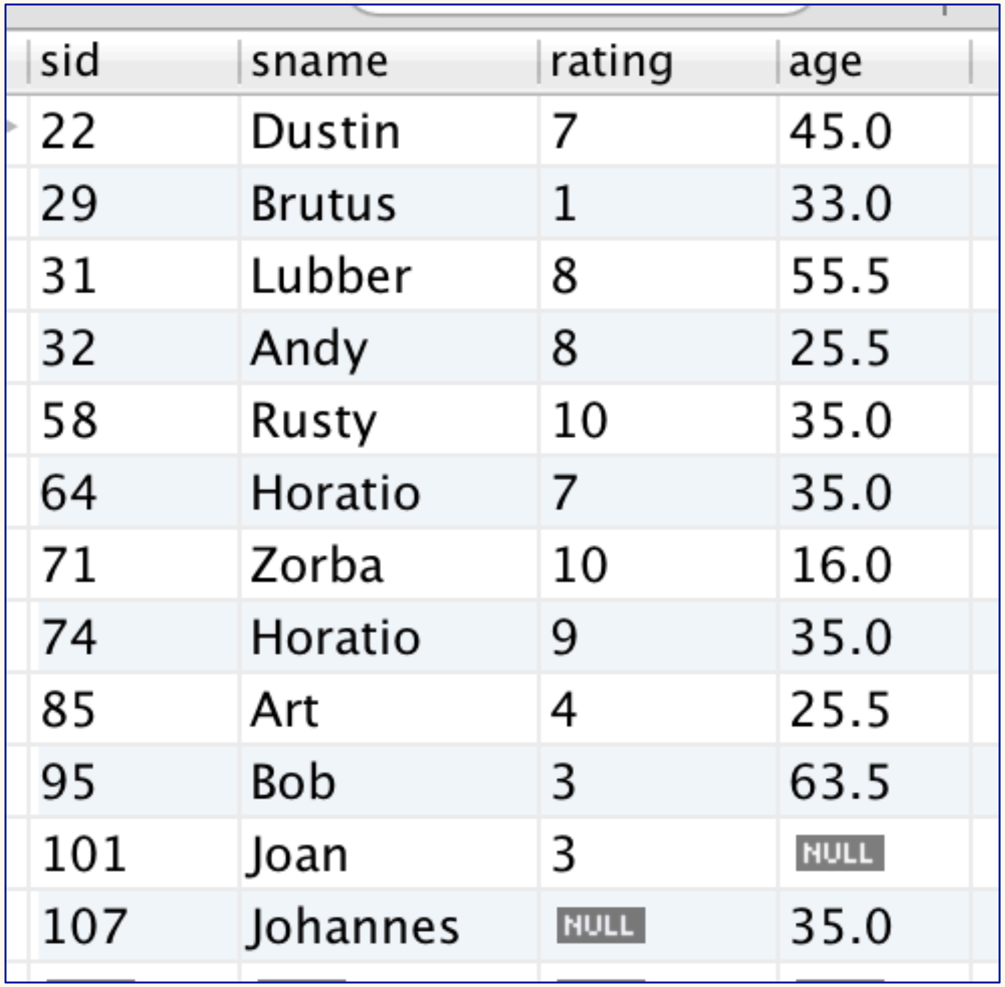
```
SELECT  *
FROM  Sailors S
WHERE age – age = 0
```

```
SELECT  *
FROM  Sailors S
WHERE age * 0 = 0
```

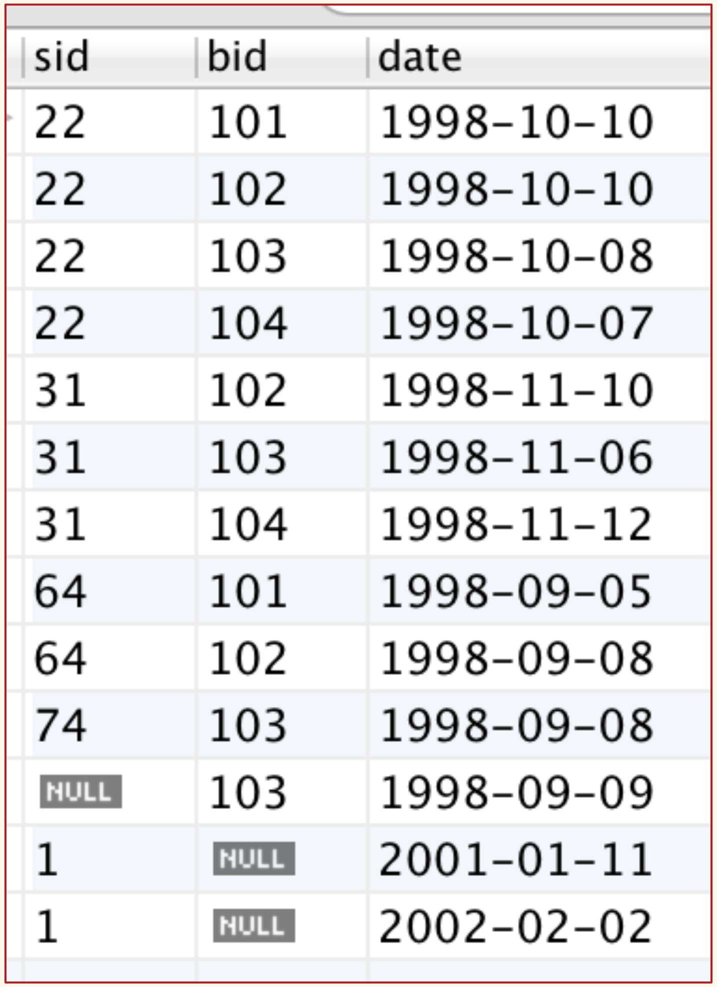
```
SELECT COUNT(*)
FROM  Sailors S
WHERE age > 35.0
        OR age <= 35.0
```

The “Joan” record will not be returned!

Ex: Sailors and Reserves w/Nulls

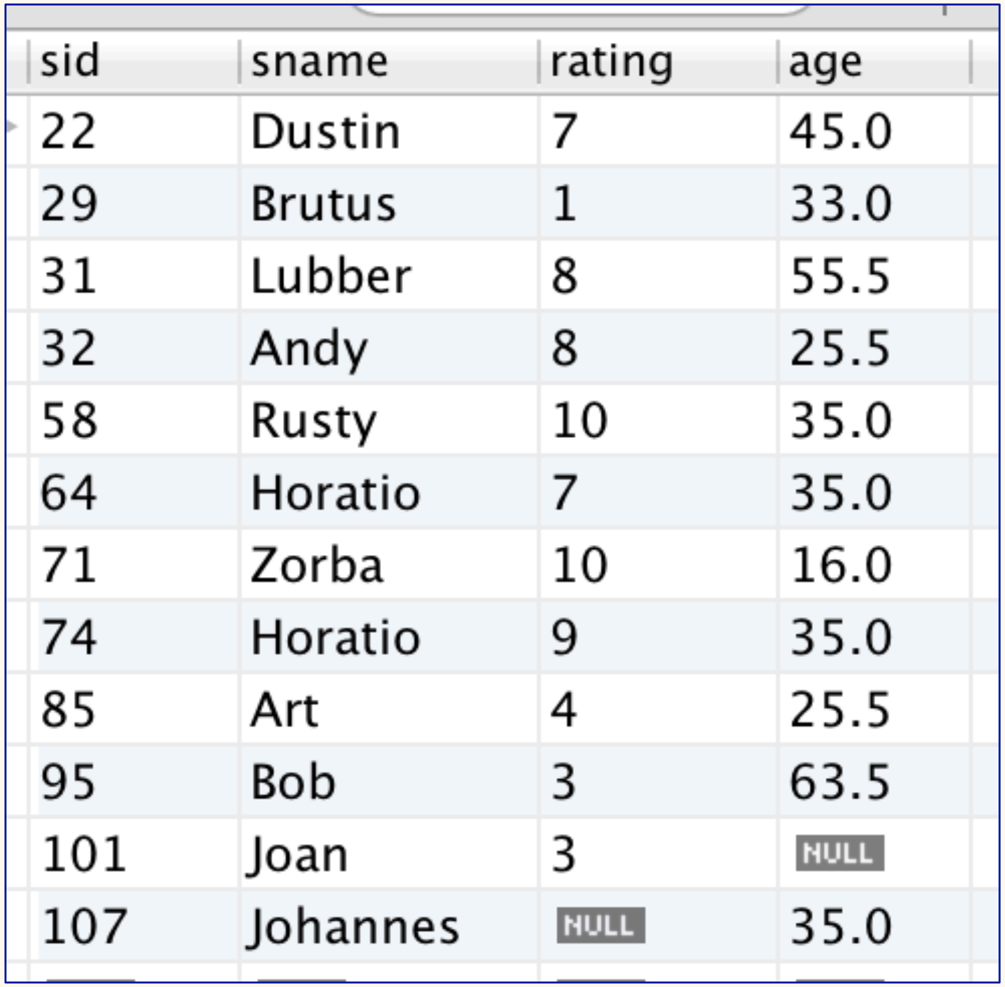


sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	4	25.5
95	Bob	3	63.5
101	Joan	3	NULL
107	Johannes	NULL	35.0



sid	bid	date
22	101	1998-10-10
22	102	1998-10-10
22	103	1998-10-08
22	104	1998-10-07
31	102	1998-11-10
31	103	1998-11-06
31	104	1998-11-12
64	101	1998-09-05
64	102	1998-09-08
74	103	1998-09-08
NULL	103	1998-09-09
1	NULL	2001-01-11
1	NULL	2002-02-02

Nulls w/Aggregates & Grouping



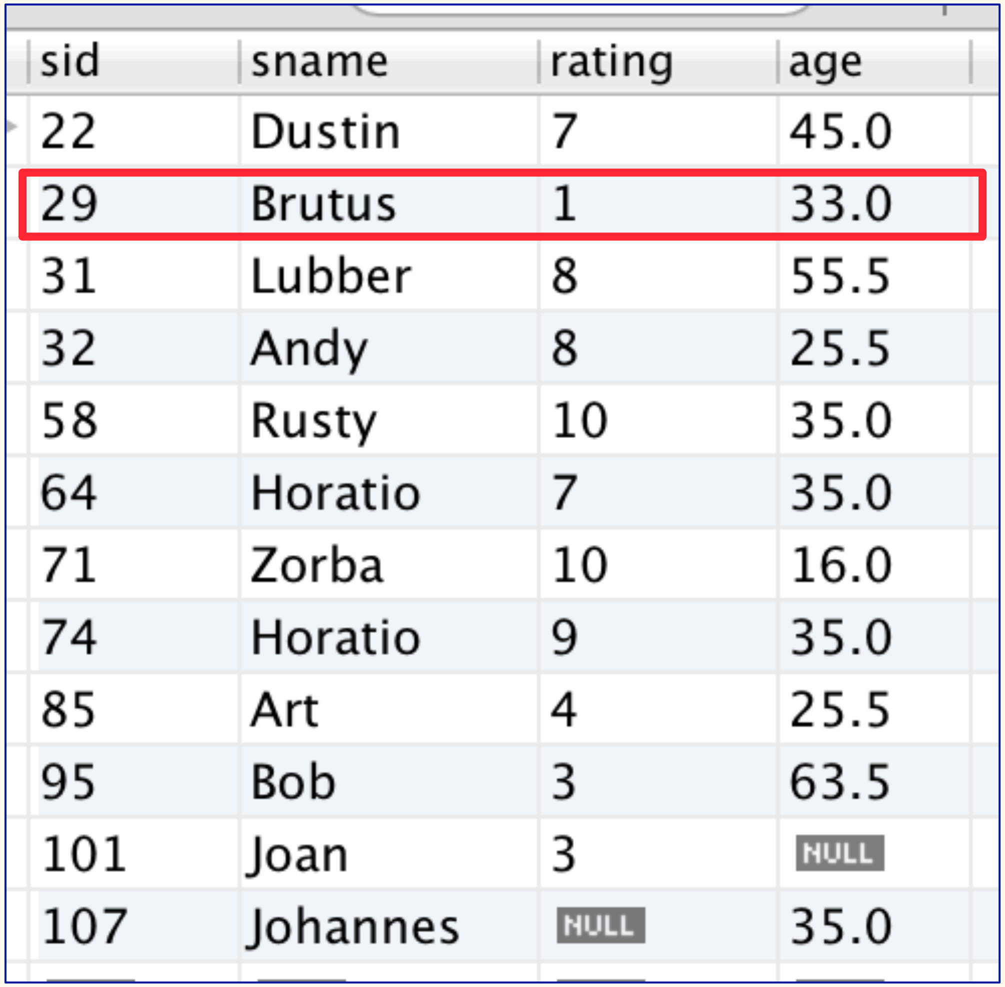
sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	4	25.5
95	Bob	3	63.5
101	Joan	3	NULL
107	Johannes	NULL	35.0

```
SELECT COUNT(rating)
FROM Sailors          (11)
```

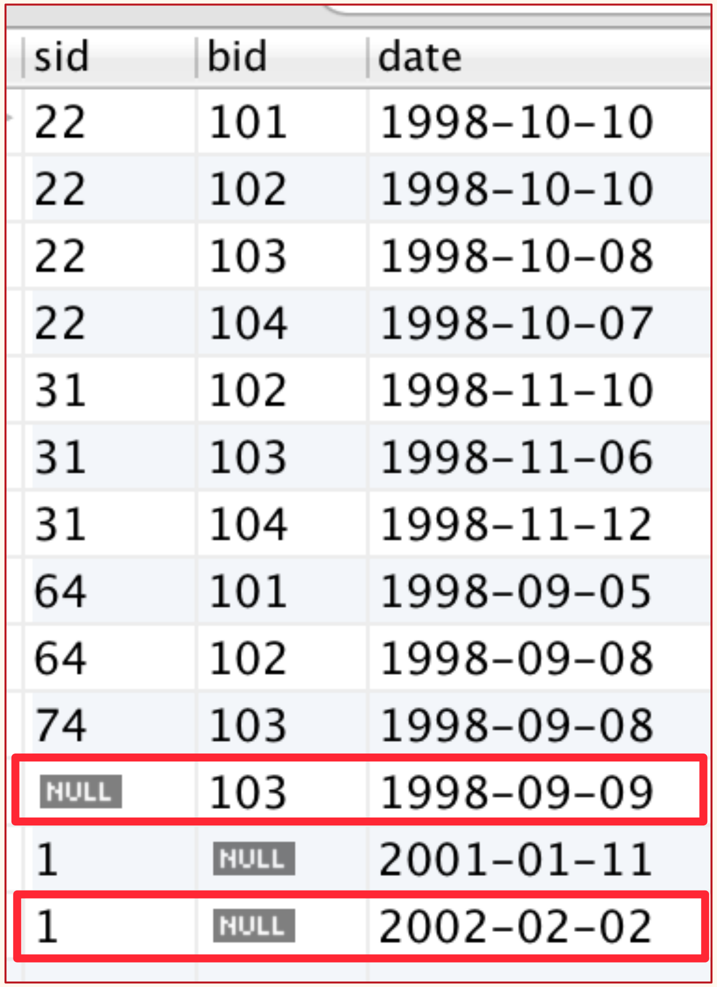
```
SELECT
COUNT (DISTINCT rating)
FROM Sailors          (7)
```

```
SELECT SUM(rating),
          COUNT(rating),
          AVG(rating)
FROM Sailors
          (70, 11, 6.3636)
```

Nulls w/Joins → Inner/Outer Joins



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	4	25.5
95	Bob	3	63.5
101	Joan	3	NULL
107	Johannes	NULL	35.0

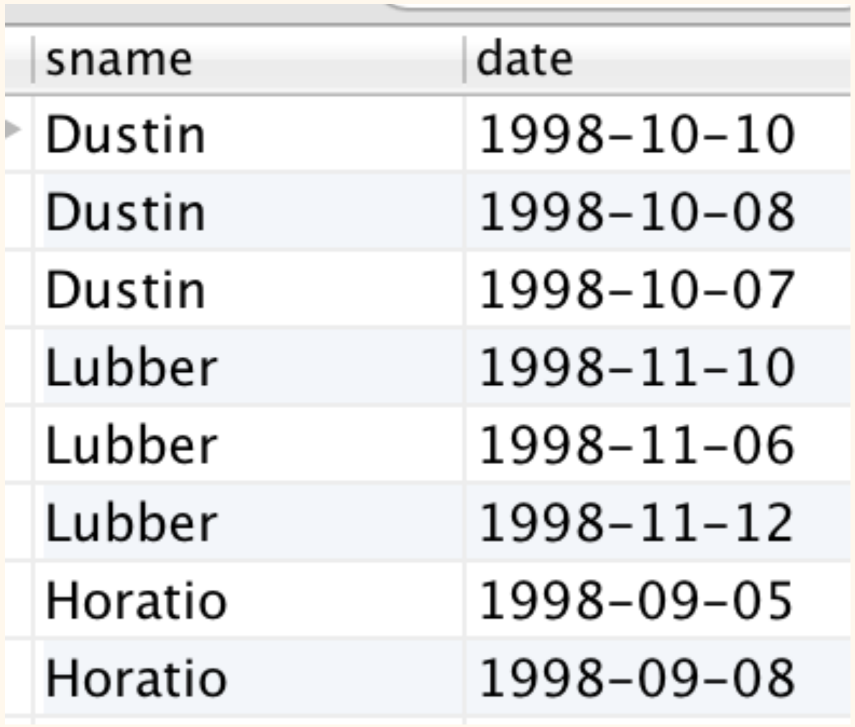


sid	bid	date
22	101	1998-10-10
22	102	1998-10-10
22	103	1998-10-08
22	104	1998-10-07
31	102	1998-11-10
31	103	1998-11-06
31	104	1998-11-12
64	101	1998-09-05
64	102	1998-09-08
74	103	1998-09-08
NULL	103	1998-09-09
1	NULL	2001-01-11
1	NULL	2002-02-02

"Dangling" tuple examples

Inner *vs. Outer Joins in SQL*

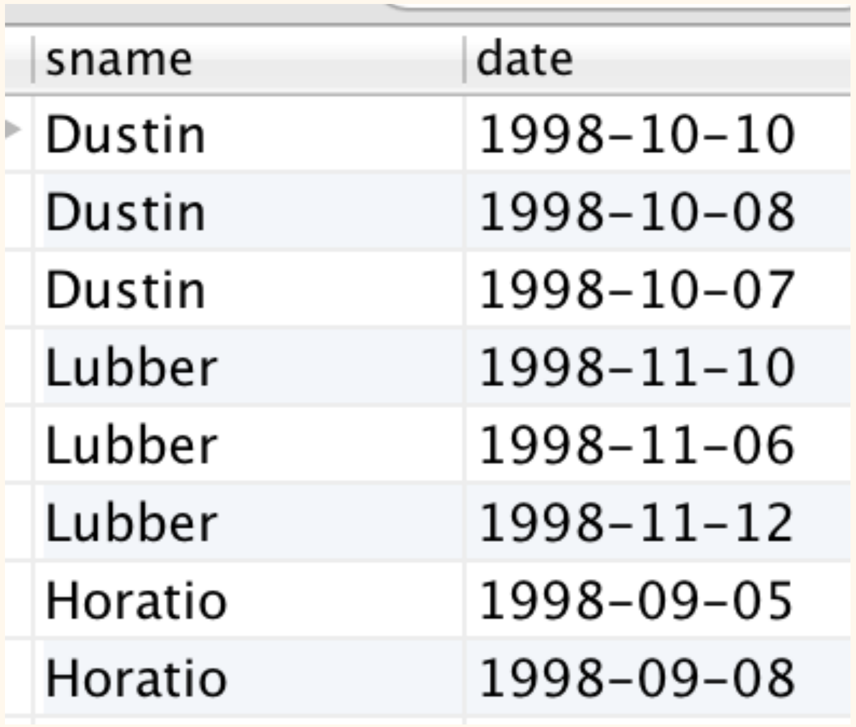
```
SELECT DISTINCT s.sname, r.date
FROM Sailors s, Reserves r
WHERE s.sid = r.sid
```



sname	date
Dustin	1998-10-10
Dustin	1998-10-08
Dustin	1998-10-07
Lubber	1998-11-10
Lubber	1998-11-06
Lubber	1998-11-12
Horatio	1998-09-05
Horatio	1998-09-08

Inner *vs. Outer Joins in SQL (2)*

```
SELECT DISTINCT s.sname, r.date
FROM Sailors s INNER JOIN Reserves r ON s.sid = r.sid
```



sname	date
Dustin	1998-10-10
Dustin	1998-10-08
Dustin	1998-10-07
Lubber	1998-11-10
Lubber	1998-11-06
Lubber	1998-11-12
Horatio	1998-09-05
Horatio	1998-09-08

*Inner vs. **Outer** Joins in SQL (3)*

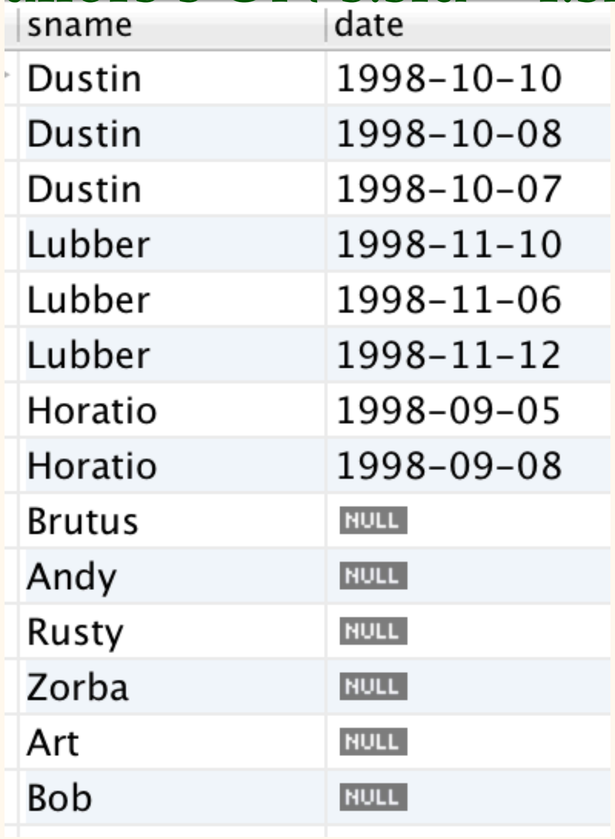
(1) SELECT DISTINCT s.sname, r.date
FROM Sailors s LEFT OUTER JOIN Reserves r ON s.sid = r.sid

(2) SELECT DISTINCT s.sname, r.date
FROM Reserves r RIGHT OUTER JOIN Sailors s ON s.sid = r.sid

- Variations on a theme:
  - JOIN (= INNER JOIN)
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
  - FULL OUTER JOIN

  (Varies from RDBMS to RDBMS)

  (See: http://dev.mysql.com/doc/refman/5.7/en/join.html for MySQL's join syntax)



sname	date
Dustin	1998-10-10
Dustin	1998-10-08
Dustin	1998-10-07
Lubber	1998-11-10
Lubber	1998-11-06
Lubber	1998-11-12
Horatio	1998-09-05
Horatio	1998-09-08
Brutus	NULL
Andy	NULL
Rusty	NULL
Zorba	NULL
Art	NULL
Bob	NULL

*Updates: Oh **CRUD**!*
(Create, Retrieve, Update, Delete)

- Can add one or more tuples using INSERT:

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES  (53688, 'Smith', 'smith@ee', 18, 3.2)

INSERT INTO Students (sid, name, login, age, gpa)
SELECT ... (your favorite SQL query goes here) ...
```

- Can DELETE all tuples satisfying any SQL query condition:

```
DELETE FROM Students S
WHERE S.sid IN (SELECT B.sid FROM Banned B)
```

*Updates: Oh **CRUD**!*

- Can change one or more tuples using UPDATE:

```
UPDATE Sailors
SET  sname = 'Arthur',
     rating = rating + 1
WHERE sname = 'Art';
```

- A few things to note:
  - LHS of SET is column name, RHS is (any) expression
  - WHERE predicate is any SQL condition, which again means SQL subqueries are available as a tool, e.g., to search for targets based on multiple tables' content

SQL Data Integrity (Largely Review)

- An *integrity constraint* describes a condition that every *legal instance* of a relation must satisfy.
  - Inserts/deletes/updates that violate IC's are disallowed.
  - Can be used to ensure application semantics (e.g., *sid* is a key, *bid* refers to a known boat) or prevent inconsistencies (e.g., *sname* has to be a string, integer *age* must be < 120)
- *Types of IC's*: Domain constraints, primary key constraints, foreign key constraints, unique constraints, general constraints.
  - *Domain constraints*: Field values must be of the right type (i.e., per the schema specification). Always enforced!

SQL Data Integrity (Cont.)

- So far we have been making good use of:
  - PRIMARY KEY
  - UNIQUE
  - NOT NULL
  - FOREIGN KEY
- Other features for ensuring field value integrity:
  - DEFAULT
  - CHECK
- More powerful integrity features include
  - ASSERTION
  - TRIGGER (a sledge hammer to use when all else fails!)

Some Integrity Related Examples

- CHECK useful when more general ICs than just keys are involved.
- Could use SQL subqueries to express richer constraints (if supported ☺).
- Constraints can be named (to manage them)

```
CREATE TABLE  Sailors
      ( sid  INTEGER,
      sname  CHAR(10),
      rating  INTEGER,
      age  REAL DEFAULT 18.0,
      PRIMARY KEY  (sid),
      CHECK  ( rating >= 1
               AND rating <= 10 )
```

```
CREATE TABLE  Reserves
      ( sname  CHAR(10),
      bid  INTEGER,
      day  DATE,
      PRIMARY KEY  (bid,day),
      CONSTRAINT  noInterlakeRes
      CHECK  (`Interlake' <>
             ( SELECT  B.bname
             FROM  Boats B
             WHERE  B.bid=bid)))
```

Enforcing Referential Integrity (RI)

- Consider Sailors and Reserves; *sid* in Sailors is a foreign key that references Reserves.
- What should be done if a Reserves tuple with a non-existent sailor id is inserted? (**A:** *Reject it!*)
- What should be done if a Sailors tuple is deleted?
  - Also delete all Reserves tuples that refer to it.
  - Disallow deletion of a Sailors tuple that's being referred to.
  - Set sid in Reserves tuples that refer to it to a *default sid*.
  - (In SQL, also: Set sid in Reserves tuples that refer to it to *null*, denoting `*unknown*' or `*inapplicable*'.)
- Similar if primary key of Sailors tuple is updated.

RI Enforcement Options in SQL

- SQL/92 and SQL:1999 support all 4 options on deletes and updates.
  - Default is NO ACTION (delete/update is rejected)
  - CASCADE (also delete all tuples that refer to the deleted tuple)
  - SET NULL / SET DEFAULT (set foreign key value of referencing tuple)

Ex:

```
CREATE TABLE Reserves
   (sid INTEGER,
    bid INTEGER,
    date DATE,
     ....
     FOREIGN KEY (sid)
     REFERENCES Sailors
          ON DELETE CASCADE
          ON UPDATE SET NULL)
```