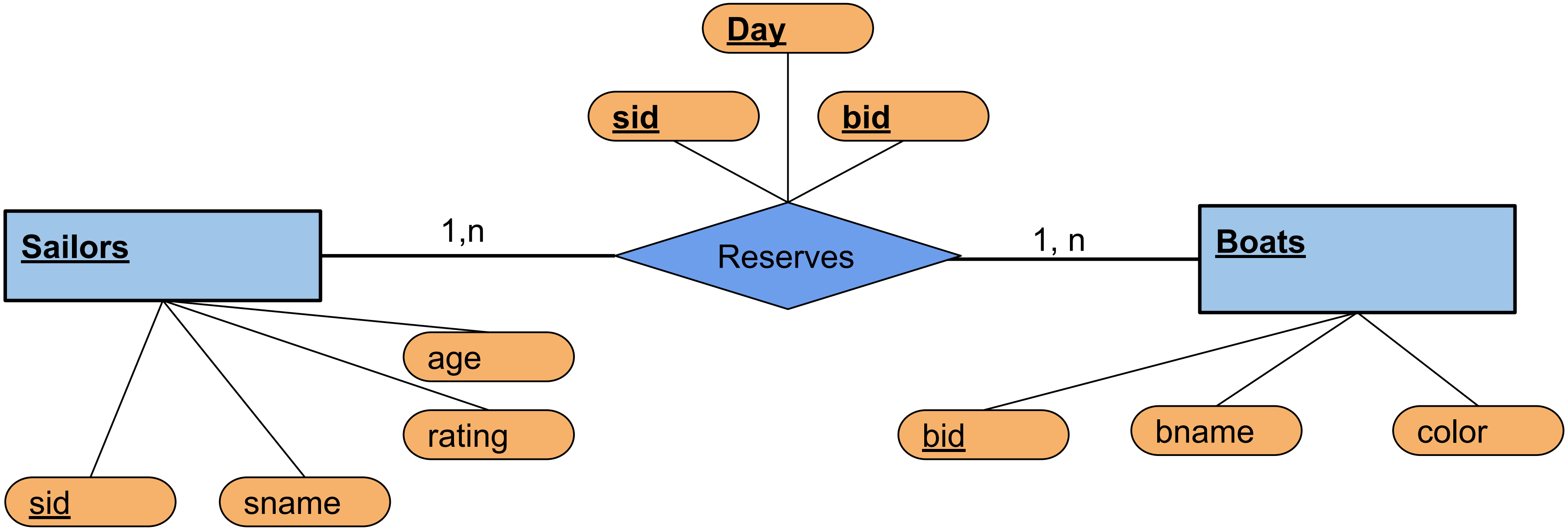
SQL: Queries, Constraints

Our Database

- Sailors(sid: integer, *sname: string, rating: integer, age: real)*
- Boats(*bid: integer, bname: string, color: string)*
- Reserves *(sid: integer, bid: integer, day: date)*

Our Model: Schema



Basic SQL Query

```
SELECT      [DISTINCT]  target-list
FROM         relation-list
WHERE        qualification
```

- *target-list* A list of attributes of relations in *relation-list*
- *relation-list* A list of relation names.
- *qualification* Comparisons (Attr *op* const or Attr1 *op* Attr2, where *op* is one of $<, >, =, \leq, \geq, \neq$) combined using AND, OR and NOT.
- DISTINCT is an optional keyword indicating that the answer should not contain duplicates. Default is that duplicates are *not* eliminated!

Examples

- Produce the names of sailors:

```
SELECT sname
        FROM    Sailors
```

SNAME
dustin
lubber
rusty

- Change the column heading:

```
SELECT sname  as "Sailor Name"
        FROM    Sailors
```

Sailor Name
dustin
lubber
rusty

- Can rename table names as well :

```
SELECT S.sname
        FROM    Sailors S
```

Examples (2)

- Find names and ages of all sailors.

```
SELECT    DISTINCT S.sname, S.age
          FROM  Sailors S;

SELECT    DISTINCT Sailors.sname, Sailors.age
          FROM  Sailors;

SELECT    DISTINCT sname, age
          FROM  Sailors;
```

Examples (3)

- Find sailors with rating above 7.

```
SELECT      S.sid, S.sname, S.rating, S.age
       FROM Sailors  S
       WHERE    S.rating > 7;

SELECT      *
       FROM Sailors
       WHERE    rating > 7;
```

LIMIT and ORDER BY

❖ LIMIT: If you want to get limit the number of results

Get the 10 first Sailors

```
SELECT * FROM Sailors LIMIT 10
```

❖ ORDER BY: order results by an attribute

```
SELECT * FROM Sailors ORDER BY age
```

❖ ORDER BY and LIMIT: get TOP 10 Youngest Sailors

```
SELECT * FROM Sailors ORDER BY age LIMIT 10
```

Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of *relation-list*.
  - Discard resulting tuples if they fail *qualifications*.
  - Delete attributes that are not in *target-list*.
  - If DISTINCT is specified, eliminate duplicate rows.
- This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute *the same answers*.

Example Instances

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

- We will use these instances of the Sailors and Reserves relations in our examples.

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
28	yuppy	9	35.0
44	guppy	5	35.0

Using multiple tables

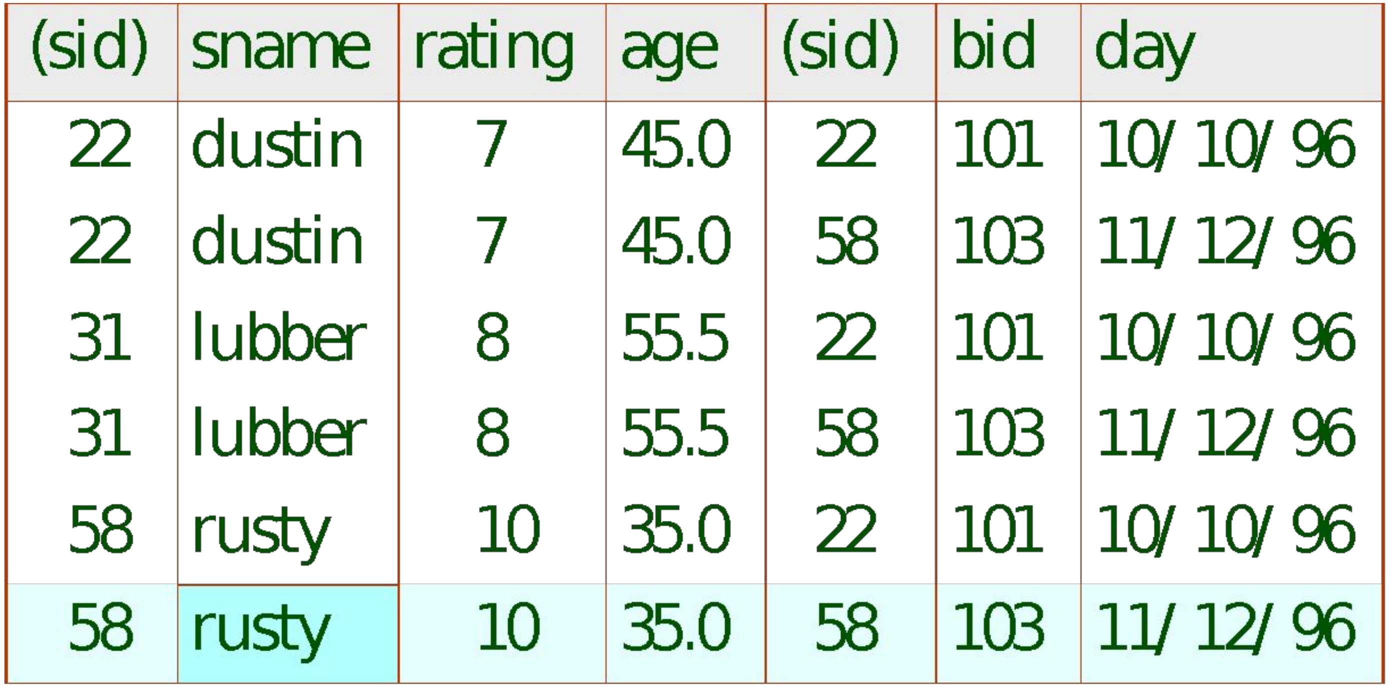
- Exercice: Find names of sailors who have reserved boat number 103.

Note 1: Sailor names only available at Sailors table

Note 2: We need reservation info as well…

Example using R1, S1 instances

```
SELECT  S.sname
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid AND R.bid=103
```



(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Note on Renaming

- Really needed only if the same relation appears twice in the FROM clause. The previous query can also be written as:

```
SELECT  S.sname
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid AND bid=103
```

OR

```
SELECT  sname
FROM    Sailors, Reserves
WHERE   Sailors.sid=Reserves.sid
        AND bid=103
```

It is good style, however, to use range variables always!

Natural Join

- SQL:99 has a "natural join" clause:

"NATURAL" means equi-join for each pair of attributes with the same name

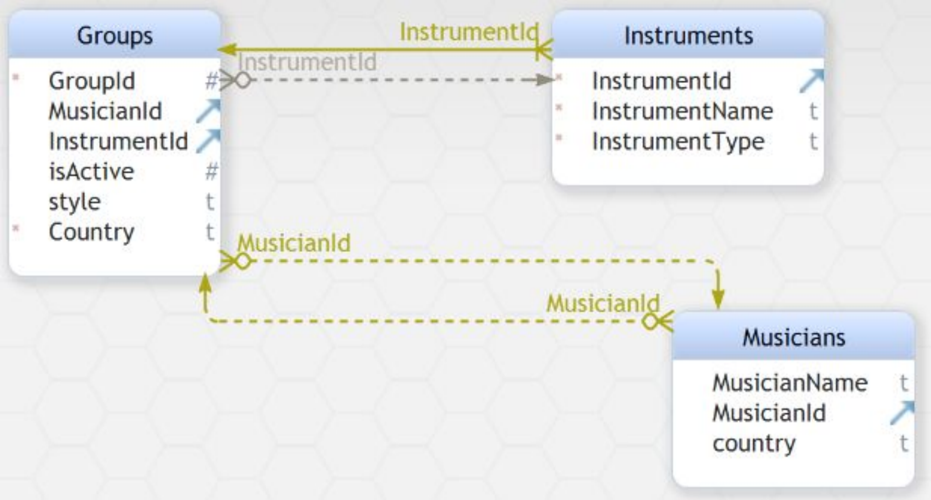
```
SELECT  *
FROM    Sailors
natural join Reserves
WHERE  bid=103
```

SID	SNAME	RATIN G	AGE	BID	DAY
22	dustin	7	45	103	08-OCT-98
31	lubber	8	55.5	103	06-NOV-98
58	rusty	10	35	103	12-NOV-98

Natural Join is evil

Sometimes you don’t expect that 2 tables have 2 columns with the same name and that are different from primary key.

Example with this data model:



http://gplivna.blogspot.fr/2007/10/natural-joins-are-evil-motto-if-you.html

Natural Join is evil - Example

```
Musicians
MusicianId         MusicianNa  Age    Country
---------------    ----------  -----  ----------
1                  Jimmy       17     US
2                  Paul        20     EN
3                  Gustavo     60     BR
```

```
Groups
GroupId            MusicianId  Instr    style       Country
---------------    ----------  -----    ----------  --
1                  1           1        Blues       EN
2                  3           2        Bossa Nova  BR
3                  3           2        Bossa Nova  US
```

<u>Question:</u> I want to JOIN all Musicians with groups:

SELECT * FROM Groups JOIN Musicians WHERE Groups.MusicianId = Musicians.MusicianId

```
Groups
MusicianName   GroupId    Musicians.Country Groups.Country
----------     -------    ----------------- --------------
Jimmy          1          US                EN
Gustavo        2          BR                US
Gustavo        3          BR                BR
```

✓

SELECT * FROM Groups NATURAL JOIN Musicians

```
Groups
MusicianName   GroupId    Musicians.Country Groups.Country
----------     -------    ----------------- --------------
Gustavo        3          BR                BR
```

✗

- Exercice: Find sids of sailors who have reserved a red boat.

  - Answer with manual Join:

    ```
    SELECT    R.sid
    FROM  Boats B, Reserves R
    WHERE    B.bid = R.bid AND B.color = ‘red’;
    ```

  - Answer with Natural Join:

    ```
    SELECT    sid
    FROM Boats  NATURAL JOIN Reserves
    WHERE color = 'red'
    ```

Expressions and Strings

- AS and = are two ways to name fields in result.
- LIKE is used for string matching. `_' stands for any one character and `%' stands for 0 or more arbitrary characters.
- Example:

```
SELECT  S.age FROM  Sailors S
      WHERE  S.sname LIKE 'B_%B'
```

Illustrates use of arithmetic expressions and string pattern matching: *Find records (age) for sailors whose names begin and end with B and contain at least three characters.*

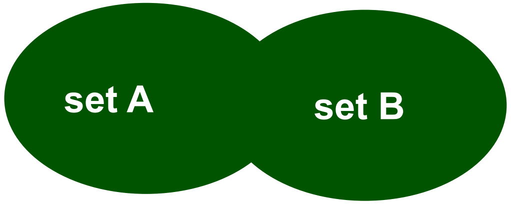
Expressions and Strings (Example)

- Find sailors that have the character ‘u’ in their names

```
SELECT  sname from Sailors
where sname like '%u%'
```

SNAME
dustin
dustin
lubber
rusty
Brutus

Union (Addition)



- Goal:Find sid's of sailors who've reserved a red or a green boat
- UNION: Can be used to compute the union of any two *union-compatible* sets of tuples (which are themselves the result of SQL queries).
- If we replace OR by AND in the first version, what do we get?
- **Also available: EXCEPT (What do we get if we replace UNION by EXCEPT?)**

```
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
  AND (B.color='red' OR B.color='green')
```

```
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
            AND B.color='red'
UNION
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid
            AND B.color='green'
```

Union (Addition) (2)

- Find sids of sailors who have rating of 10 or reserved boat 104.

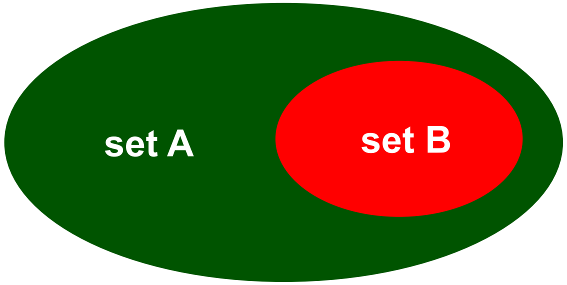
One attempt:

```
SELECT S.sid FROM Sailors S, Reserves R
WHERE S.sid = R.sid and (S.rating = 10 or R.bid = 104)
```

- with UNION:

```
SELECT   S.sid    FROM Sailors S
      WHERE    S.rating = 10
UNION
SELECT   R.sid    FROM Reserves R
      WHERE    R.bid = 104;
```

Except / minus

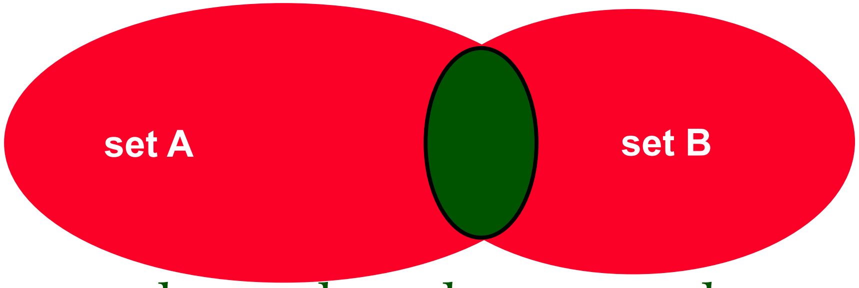


- To substract a set from another, use:
  - EXCEPT : PostgreSQL
  - MINUS : MySQL et Oracle
- Find sids of sailors who have reserved green boats but not red boats.

```
SELECT   R.sid
         FROM     Reserves R, Boats B
         WHERE  R.bid = B.bid AND B.color = ’green’
MINUS
SELECT   R2.sid
         FROM     Boats B2, Reserves R2
         WHERE  R2.bid = B2.bid AND B2.color = ‘red’;
```

Note: don’t forget to indent for lisibility

Intersect



- Find sid's of sailors who've reserved a red and a green boat

~~SELECT S.sid~~
~~FROM Sailors S, Boats B1, Reserves R1~~
~~WHERE S.sid=R1.sid AND R1.bid=B1.bid~~
~~AND (B1.color='red' AND B1.color='green')~~

```
SELECT  S.sid
      FROM  Sailors S, Boats B, Reserves R
      WHERE  S.sid=R.sid AND R.bid=B.bid
             AND B.color='red'
INTERSECT
SELECT  S.sid
      FROM  Sailors S, Boats B, Reserves R
      WHERE  S.sid=R.sid AND R.bid=B.bid
            AND B.color='green'
```

- INTERSECT: Can be used to compute the intersection of any two *union-compatible* sets of tuples.
- Included in the SQL/92 standard, but some systems don't support it.

Nested Queries

- *Find names of sailors who've reserved boat #103:*

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.sid IN  (SELECT  R.sid
                  FROM  Reserves R
                  WHERE  R.bid=103)
```

```
SELECT  S.sname
    FROM  Sailors S,
    Reserves R
    WHERE  S.sid = R.siD
    AND  R.bid=103)
```

- A very powerful feature of SQL: a WHERE clause can itself contain an SQL query
- To find sailors who've *not* reserved #103, use NOT IN.
- To understand semantics of nested queries, think of a <u>*nested loops*</u> evaluation: *For each Sailors tuple, check the qualification by computing the subquery.*

Nested Queries with Correlation

Find names of sailors who've reserved boat #103:

```
SELECT  S.sname
FROM  Sailors S
WHERE   EXISTS  (SELECT  *
                 FROM  Reserves R
                 WHERE  R.bid=103 AND S.sid=R.sid)
```

- EXISTS is another set comparison operator, like IN.
  - IN: Returns true if a specified value matches any value in a subquery or a list.
  - Exists: Returns true if a subquery contains any rows.

More on Set-Comparison Operators

- We've already seen IN, EXISTS. Can also use NOT IN, NOT EXISTS.
- Also available: *op* ANY, *op* ALL, *op* IN $>,<,=,\geq,\leq,\neq$
- Find sailors whose rating is greater than any ratings of some sailor called Horatio:

```
SELECT  *
FROM  Sailors S
WHERE  S.rating > ANY  (SELECT  S2.rating
                        FROM  Sailors S2
                        WHERE S2.sname='Horatio')
```

Rewriting INTERSECT *Queries Using* IN

Find sid’s of sailors who’ve reserved both a red and a green boat:

```
SELECT  S.sid
FROM  Sailors S, Boats B, Reserves R
WHERE  S.sid=R.sid AND R.bid=B.bid AND B.color=‘red’
          AND S.sid IN  (SELECT  S2.sid
                          FROM  Sailors S2, Boats B2, Reserves R2
                          WHERE  S2.sid=R2.sid AND R2.bid=B2.bid
                                  AND  B2.color=‘green’)
```

- Similarly, EXCEPT queries re-written using NOT IN.
- To find *names* (not *sid*’s) of Sailors who’ve reserved both red and green boats, just replace *S.sid* by *S.sname* in SELECT clause. (What about INTERSECT query? See next slide)

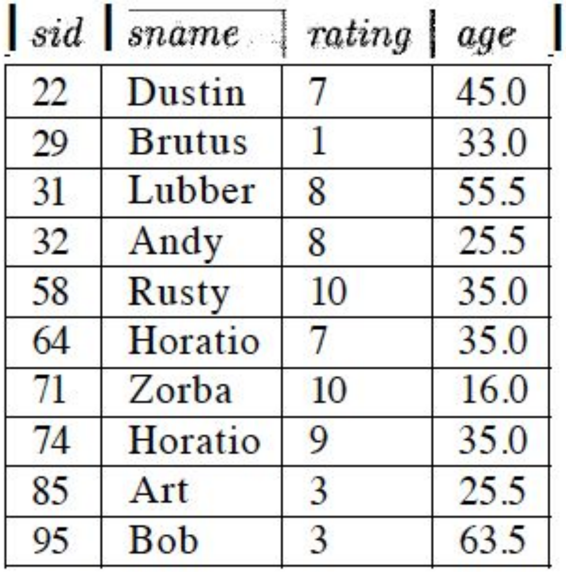
Intersect: Replacing sid by sname (1)

- The query will like this:

```
SELECT  S.sname
      FROM  Sailors S, Boats B, Reserves R
      WHERE  S.sid=R.sid AND R.bid=B.bid AND B.color=‘red’
INTERSECT
SELECT  S.sname
      FROM  Sailors S, Boats B, Reserves R
      WHERE  S.sid=R.sid AND R.bid=B.bid AND B.color=‘green’
```

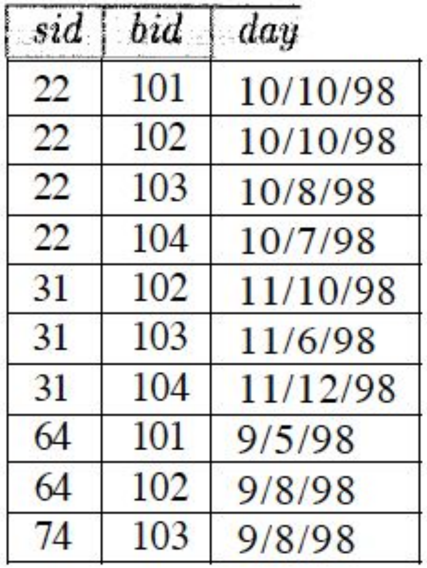
Does it work ?

Intersect: Replacing sid by sname (2)



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

Figure 5.1 An Instance *S3* of Sailors

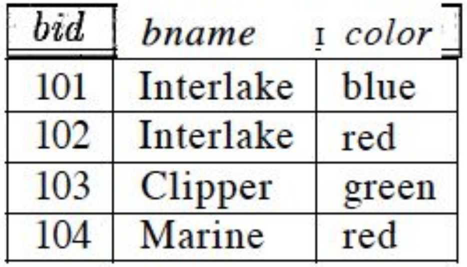


sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

red

green

Figure 5.2 An Instance *R2* of Reserves



bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 5.3 An Instance *B1* of Boats

Bug: Horatio will be in the result and shouldn't be.

Intersect: Replacing sid by sname (3)

To resolve earlier version that used INTERSECT and had small bug:

```
SELECT     S.sname
FROM  Sailors S WHERE      S.sid IN (
      (SELECT   R.sid
            FROM  Boats B, Reserves R
            WHERE    R.bid = B.bid AND B.color = ‘red’)
      INTERSECT
      (SELECT   R2.sid
            FROM  Boats B2, Reserves R2
            WHERE    R2.bid = B2.bid AND B2.color = ‘green’));
```

Aggregate Operators

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

single column

- Significant extension of relational algebra.

- Example: Minimum rating (worst) of the sailors:

```
SELECT  MIN (S.rating)
      FROM  Sailors S
```

Aggregate Operators - Example

- Example: Find average age of sailors with a rating of 10.

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

- Example: Count the number of sailors.

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

- Example: Count number of different sailors' names.

```
SELECT   COUNT(DISTINCT S.sname)
         FROM Sailors S;
```

Aggregate Operators - Example (2)

- Example: Find name and age of the oldest sailor(s)

```
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
```

- The first query has a bug! (We'll look into the reason a bit later, when we discuss GROUP BY.)

- The third query is equivalent to the second query, and is allowed in the SQL/92 standard, but is not supported in some systems.

Aggregate Operators - Example (3)

- Example: Find names of sailors who are older than oldest sailor with rating of 10.

- Solution using MAX:

```
SELECT S.sname FROM Sailors S
     WHERE S.age > (SELECT   MAX(S2.age)
               FROM Sailors S2 WHERE    S2.rating = 10);
```

- Solution using ALL:

```
SELECT S.sname FROM Sailors S
          WHERE S.age > ALL (
                    SELECT   S2.age
                    FROM Sailors S2
                    WHERE    S2.rating = 10);
```

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*!
- One answer tuple is generated per qualifying group.

For each red boat, find the number of reservations for this boat

```
SELECT  B.bid,  COUNT (*) AS scount
    FROM  Boats B, Reserves R
    WHERE  R.bid=B.bid AND B.color=‘red’
    GROUP BY  B.bid
```

Aggregate Operators - Example (4)

- Select * from sailors order by rating

SID	SNAME	RATING	AGE
10	dustin	1	45
29	Brutus	1	33
85	Art	3	25.5
95	Bob	3	63.3
96	Frodo	3	25.5
22	dustin	7	45
64	Horatio	7	35
31	lubber	8	55.5
32	Andy	8	25.5
74	Horatio	9	35
71	Zorba	10	16
58	rusty	10	35

- Example: Number of all sailors
  select count(*) from sailors ➔ 12

- select rating, count(*) as count from sailors **group by rating** order by rating

RATING	COUNT
1	2
3	3
7	2
8	2
9	1
10	2

Aggregate Operators - Example (4)

- Select * from sailors order by rating

SID	SNAME	RATING	AGE
10	dustin	1	45
29	Brutus	1	33
85	Art	3	25.5
95	Bob	3	63.3
96	Frodo	3	25.5
22	dustin	7	45
64	Horatio	7	35
31	lubber	8	55.5
32	Andy	8	25.5
74	Horatio	9	35
71	Zorba	10	16
58	rusty	10	35

- If we want to add sname to results we try this but it won’t work:

select sname, rating, count(*) from sailors **group by rating** order by rating

SNAME	RATING	COUNT
Dustin, Brutus ??	1	2
Art, Bob, Frodo ??	3	3
Dustin, Horatio??	7	2
Lubber, Andy ??	8	2
Horatio	9	1
Zorba, rusty ??	10	2

Column List is Restricted

- Only one value is allowed per attribute, therefore the columns must either be in the group by list, or summarization (aggregate) functions like sum, count, avg, min, max

- sname cannot be put into column list
- Some database will accept it but will display only the first sname in the result table.

Aggregate Operators - Example (5)

❖ Select * from reserves order by bid =>

SID	BID	DAY
64	101	05-SEP-98
22	101	10-OCT-98
22	102	10-OCT-98
31	102	10-NOV-98
64	102	08-SEP-98
31	103	06-NOV-98
22	103	08-OCT-98
74	103	08-SEP-98
31	104	12-NOV-98
22	104	07-OCT-98

❖ Example: Find the number of past reservations for each boat ??

```
SELECT bid, COUNT(*)
      FROM reserves GROUP BY bid
```

BID	COUNT(*)
102	3
101	2
104	2
103	3

❖ Can we list boat names in the list as well?

```
SELECT r.bid, b.bname, count(*)
        FROM reserves r,boats b
        WHERE r.bid = b.bid
        GROUP BY r.bid, b.bname
```

Aggregate Operators – Example (6)

- List the ratings > 5 for which the average age is less than 40:

```
SELECT rating, AVG(age), COUNT(*)
        FROM sailors
        WHERE rating > 5
        GROUP BY rating
        HAVING AVG(age) < 40
        ORDER BY rating
```

RATING	AVG(AGE)	COUNT(*)
9	35	1
10	25.5	2

Let’s detail this query in next slides

Aggregate Operators - Example (6)

- Step 1: SELECT rating, age FROM sailors

RATING	AGE
1	45
1	33
3	25.5
3	63.3
3	25.5
7	45
7	35
8	55.5
8	25.5
9	35
10	16
10	35

- Step 2: WHERE rating > 5

RATING	AGE
7	45
7	35
8	55.5
8	25.5
9	35
10	16
10	35

- Step 3: GROUP BY rating (it apply aggregators: avg and count)

RATING	AVG(AGE)	COUNT(
7	(45+35)/2 = 40	2
8	(55.5+25.5)/2=40.5	2
9	35	1
10	(16+35)/2=25.5	2

Aggregate Operators - Example (6)

- Step 4: HAVING AVG(AGE) < 40

RATING	AVG(AGE)	COUNT(
7	40	2
8	40.5	2
9	35	1
10	33.5	2

- Final result is:

RATING	AVG(AGE)	COUNT(*)
9	35	1
10	25.5	2

Find age of the youngest sailor with at least age 18, for each rating with 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 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
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 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.
- What if HAVING clause is replaced by:
  - HAVING COUNT(*) >1

Summary on SET operators

- UNION: addition
- INTERSECT: subtraction
- CROSS JOIN: Multiplication

- Where is the Division ?
  - It doesn't exist in SQL.
  - But First what is the definition ?

Division in SQL: Definition

The general idea is the division of the dividend from a table which is divided with a divider table to obtain a quotient (a table result). The result is calculated from the values of a column to which the second column of the table dividend have all the values of the divider.

```
SELECT sname
   FROM Reserves R JOIN Boat B ON R.bid = B.bid
   WHERE boat_id IN (SELECT bid FROM boats)
   GROUP BY sailor_name
   HAVING
       COUNT(DISTINCT(bid))
        =
       (SELECT count(*) FROM Boats)
```

Reserves

sailor_name		boat_id
Peter	5	pedalo
Peter		**Titanic**
Peter		catamaran
Peter		jet-ski
Peter		pedalo
Anna	4	jet-ski
Anna		catamaran
Anna		catamaran
Anna		Yatch
Bob	1	Yatch

Boats

boat_id
jet-ski
Yatch
catamaran

Goal: Find sailors who've reserved all boats:

Division in SQL : 1st Method : Count

For sailors example:

```
SELECT sailor_name FROM Reserves
          WHERE boat_id IN (SELECT boat_id FROM Boats)
GROUP BY sailor_name
HAVING COUNT(DISTINCT(boat_id)) = (SELECT COUNT(boat_id) FROM Boats)
```

Explanation:

Gets all sailors, count the number of different boats they have sailed and check if this number is equal to the number total of boat to sail.

Generic Formula:

```
SELECT group_id FROM Dividend
   WHERE item_name IN (SELECT item_name FROM Divisor)
GROUP BY group_id
HAVING COUNT(DISTINCT(item_name)) = (SELECT COUNT(item_name) FROM Divisor)
```

Division in SQL : 2nd Method : Double Negation

The Query for sailors database:

```
SELECT DISTINCT(sailor_name) FROM Sailors AS S1
    WHERE NOT EXISTS
        (SELECT * FROM Boats AS B
            WHERE NOT EXISTS
                (SELECT * FROM RESERVES AS R
                    WHERE R.sid=S1.sid AND R.bid = B.bid
                )
        )
```

Explanation:

Each sailor look at the boats and say:

    There is no boat here
        That I haven't sail/reserved

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? (5 < null => *unknown*)
  - We need a 3-valued logic (true, false and *unknown*).
  - Meaning of constructs must be defined carefully. (e.g., WHERE clause eliminates rows that don't evaluate to true.)
  - New operators (in particular, *outer joins*) possible/needed.

Three-Valued Logic

OR	Unknown	True	False
Unknown	**Unknown**	**True**	**Unknown**
True	True	True	True
False	Unknown	True	False

AND	Unknown	True	False
Unknown	**Unknown**	**Unknown**	**False**
True	Unknown	True	False
False	False	False	False

SQL and NULL Values

WHERE clause is unknown? SQL query eliminates those rows (it evaluates to false)

DISTINCT - two rows are duplicates if corresponding columns are equal or both are null

SQL and NULL Values (continued)

Arithmetic operators - return NULL if argument is null

COUNT(*) - includes null values

All other aggregate operators discard null values.

Integrity Constraints (Review)

- An IC describes conditions 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), or prevent inconsistencies (e.g., *sname* has to be a string, *age* must be < 200)
- *Types of IC's*: Domain constraints, primary key constraints, foreign key constraints, general constraints.
  - *Domain constraints*: Field values must be of right type. Always enforced.

General Constraints

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

- Useful when more general ICs than keys are involved.
- Can use queries to express constraint.
- Constraints can be named.

```
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)))
```

JOINS

```
SELECT (column_list)
FROM  table_name
 [INNER | {LEFT |RIGHT | FULL } OUTER] JOIN table_name
   ON qualification_list
WHERE …
```

Explicit join semantics needed unless it is an INNER join

(INNER is default)

Inner Join

Only the rows that match the search conditions are returned.

```
    SELECT s.sid, s.name, r.bid
          FROM Sailors s INNER JOIN Reserves r
          ON s.sid = r.sid
```

Returns only those sailors who have reserved boats

SQL-92 also allows:

```
    SELECT s.sid, s.name, r.bid
          FROM Sailors s NATURAL JOIN Reserves r
```

Inner Join : Example

SELECT s.sid, s.name, r.bid
FROM Sailors s INNER JOIN Reserves r
ON s.sid = r.sid

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
95	Bob	3	63.5

sid	bid	day
22	101	10/10/96
95	103	11/12/96

Result:

s.sid	s.name	r.bid
22	Dustin	101
95	Bob	103

Left Outer Join

Left Outer Join returns all matched rows, plus all unmatched rows from the table on the left of the join clause.

(use nulls in fields of non-matching tuples)

```
SELECT s.sid, s.name, r.bid
    FROM Sailors s LEFT OUTER JOIN Reserves r
    ON s.sid = r.sid
```

Returns all sailors & information on whether they have reserved boats

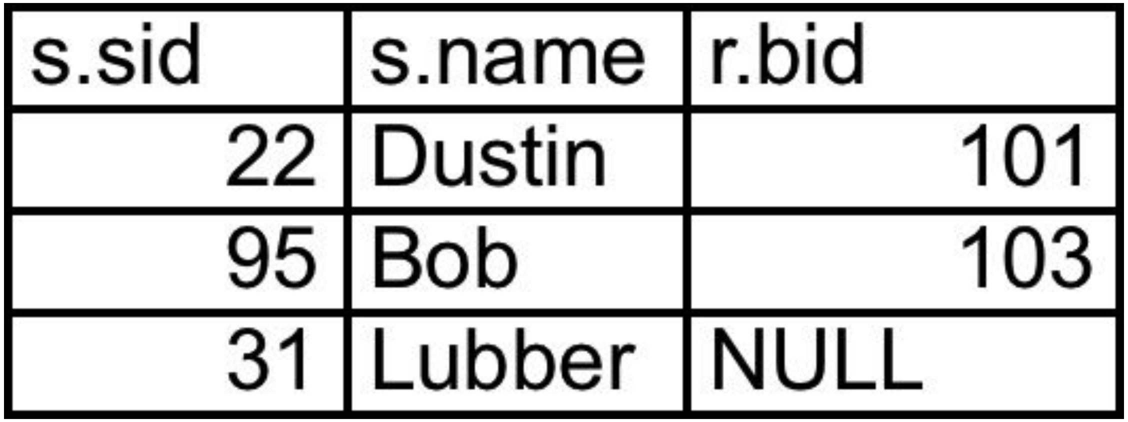
Left Outer Join : Example

```
SELECT s.sid, s.name, r.bid
    FROM Sailors s LEFT OUTER JOIN Reserves r
    ON s.sid = r.sid
```

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
95	Bob	3	63.5

sid	bid	day
22	101	10/10/96
95	103	11/12/96

Result:



s.sid	s.name	r.bid
22	Dustin	101
95	Bob	103
31	Lubber	NULL

Right Outer Join

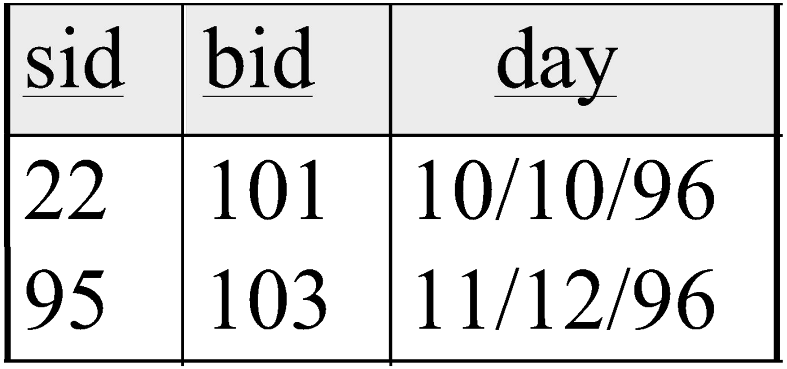
Right Outer Join returns all matched rows, plus all unmatched rows from the table on the right of the join clause

```
SELECT r.sid, b.bid, b.name
    FROM Reserves r RIGHT OUTER JOIN Boats b
    ON r.bid = b.bid
```

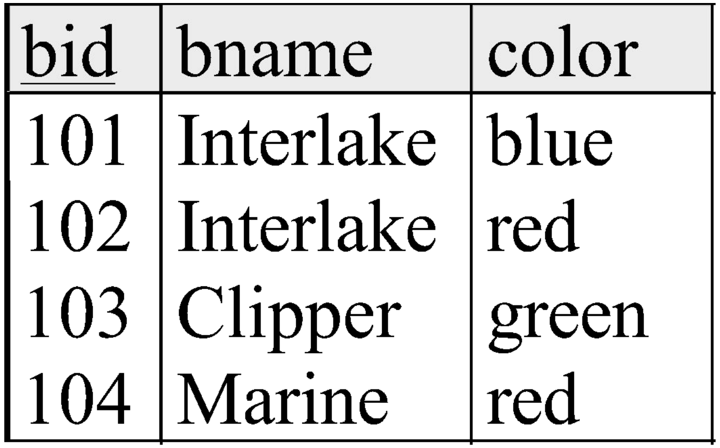
Returns all boats & information on which ones are reserved.

Right Outer Join: Example

```
SELECT r.sid, b.bid, b.name
    FROM Reserves r RIGHT OUTER JOIN Boats b
    ON r.bid = b.bid
```

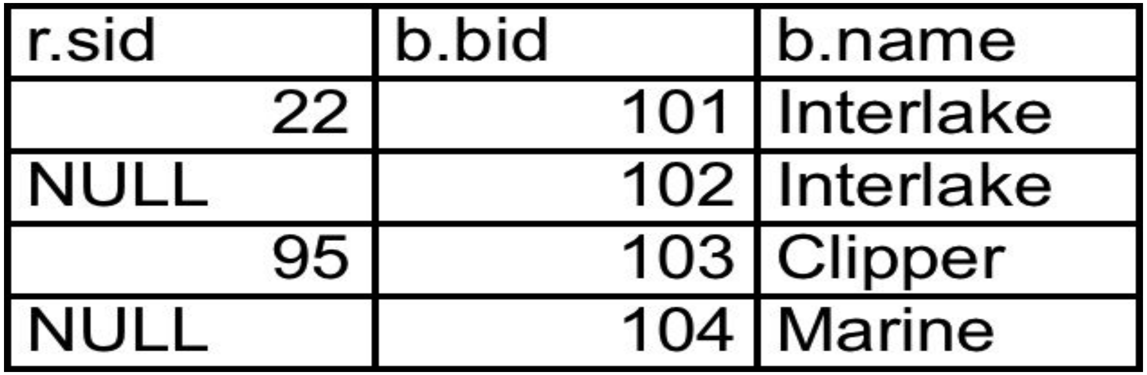


sid	bid	day
22	101	10/10/96
95	103	11/12/96



bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Result:



r.sid	b.bid	b.name
22	101	Interlake
NULL	102	Interlake
95	103	Clipper
NULL	104	Marine

Full Outer Join

Full Outer Join returns all (matched or unmatched) rows from the tables on both sides of the join clause

```
SELECT r.sid, b.bid, b.name
    FROM Reserves r FULL OUTER JOIN Boats b
    ON r.bid = b.bid
```

Returns all boats & all sailors & all links between reservation if there are

Full Outer Join

Find sids of sailors who have rating of 10 or reserved boat 104

What about sailors who have not reserved any boats?

What if we have some old reservations in the database for which the Sailor info is missing?

Use full outer join that includes rows with no match:

```
SELECT S.sid
    FROM Sailors S full outer join Reserves R on (S.sid = R.sid)
    WHERE S.rating = 10 or R.bid = 104
```

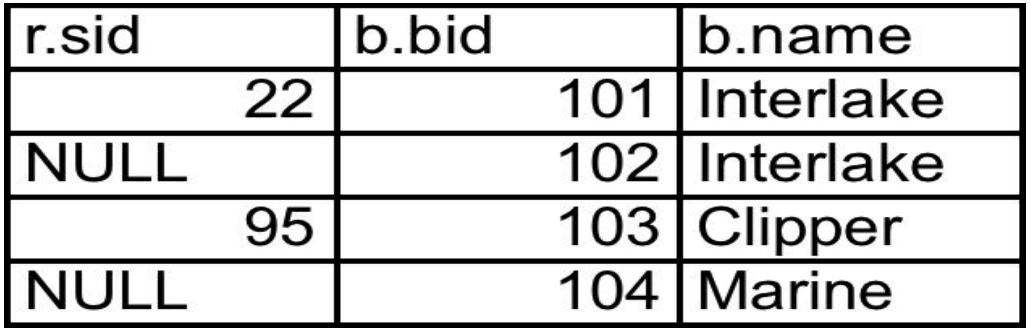
Full Outer Join : Example

SELECT r.sid, b.bid, b.name
FROM Reserves r FULL OUTER JOIN Boats b
ON r.bid = b.bid

sid	bid	day
22	101	10/10/96
95	103	11/12/96

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Result:

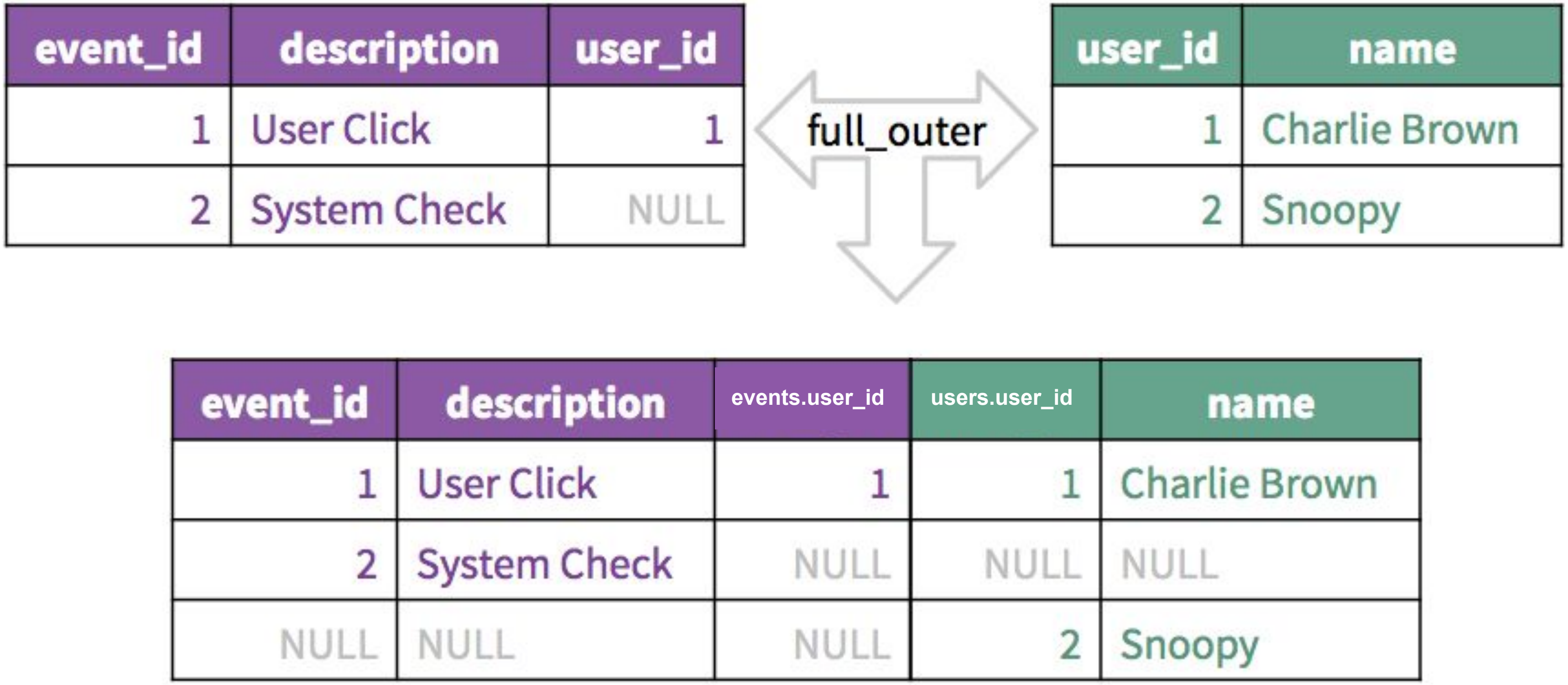


r.sid	b.bid	b.name
22	101	Interlake
NULL	102	Interlake
95	103	Clipper
NULL	104	Marine

Note: in this case it is the same as the ROJ because
bid is a foreign key in reserves, so all reservations must
have a corresponding tuple in boats.

Full Outer Join : Example (2)

- *Example will Null on left and right tables:*

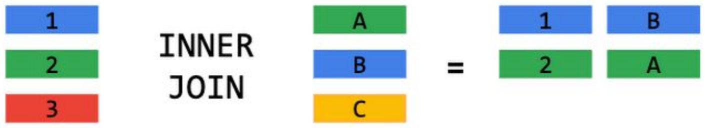
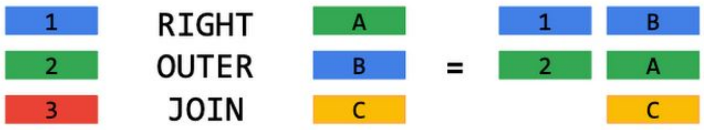
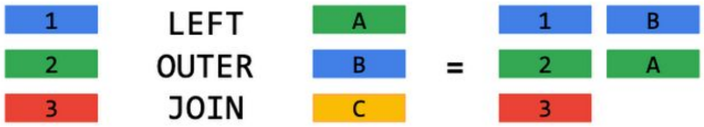
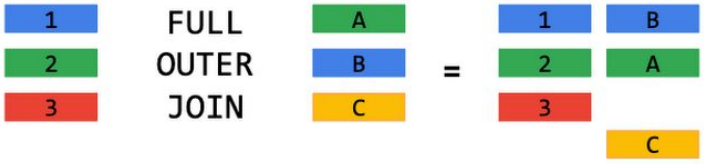
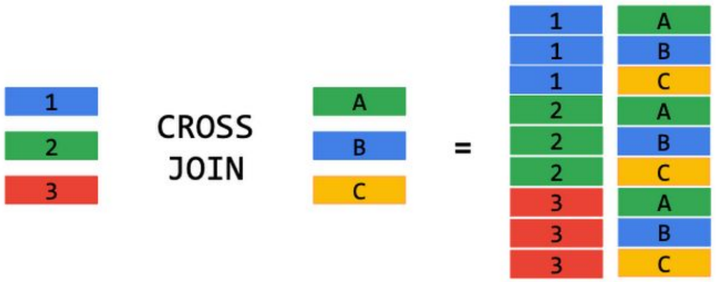


event_id	description	user_id
1	User Click	1
2	System Check	NULL

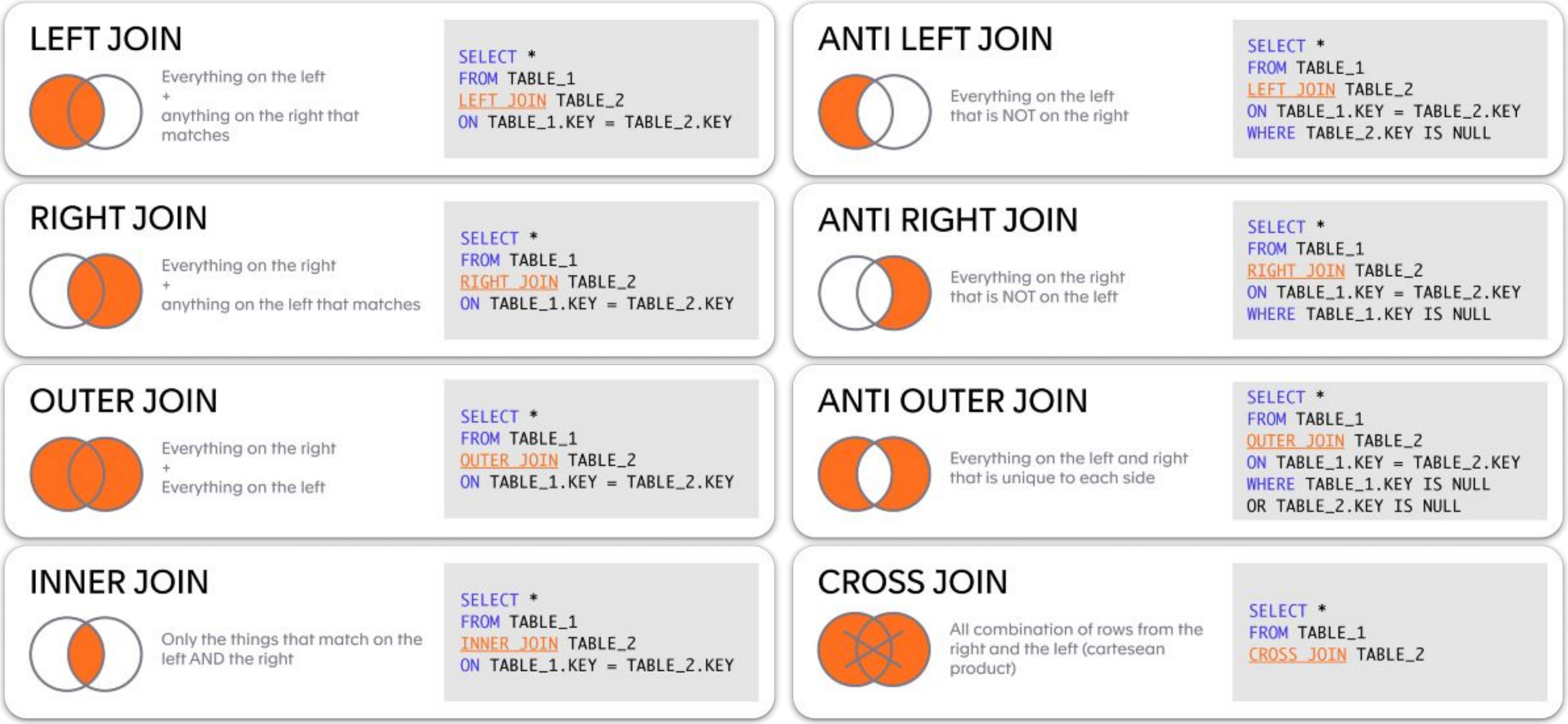
user_id	name
1	Charlie Brown
2	Snoopy

event_id	description	events.user_id	users.user_id	name
1	User Click	1	1	Charlie Brown
2	System Check	NULL	NULL	NULL
NULL	NULL	NULL	2	Snoopy

Summary

INNER JOIN		Only returns rows that meet the join condition
RIGHT OUTER JOIN	 1 2 3 RIGHT OUTER JOIN A B C = 1 B 2 A C	Returns all rows from the table on the right side of JOIN and matched rows from the left side of the JOIN
LEFT OUTER JOIN	 1 2 3 LEFT OUTER JOIN A B C = 1 B 2 A 3	Returns all rows from the table on the left side of JOIN and matched rows from the right side of the JOIN
FULL OUTER JOIN	 1 2 3 FULL OUTER JOIN A B C = 1 B 2 A 3 C	Returns all rows from both sides even if join condition is not met
CROSS JOIN	 1 2 3 CROSS JOIN A B C = 1 A 1 B 1 C 2 A 2 B 2 C 3 A 3 B 3 C	Cartesian product between the two sides is a join but without a join condition. Returns all rows joined from both sides

Advanced Summary



LEFT JOIN

Everything on the left
+
anything on the right that matches

```
SELECT *
FROM TABLE_1
LEFT JOIN TABLE_2
ON TABLE_1.KEY = TABLE_2.KEY
```

ANTI LEFT JOIN

Everything on the left that is NOT on the right

```
SELECT *
FROM TABLE_1
LEFT JOIN TABLE_2
ON TABLE_1.KEY = TABLE_2.KEY
WHERE TABLE_2.KEY IS NULL
```

RIGHT JOIN

Everything on the right
+
anything on the left that matches

```
SELECT *
FROM TABLE_1
RIGHT JOIN TABLE_2
ON TABLE_1.KEY = TABLE_2.KEY
```

ANTI RIGHT JOIN

Everything on the right that is NOT on the left

```
SELECT *
FROM TABLE_1
RIGHT JOIN TABLE_2
ON TABLE_1.KEY = TABLE_2.KEY
WHERE TABLE_1.KEY IS NULL
```

OUTER JOIN

Everything on the right
+
Everything on the left

```
SELECT *
FROM TABLE_1
OUTER JOIN TABLE_2
ON TABLE_1.KEY = TABLE_2.KEY
```

ANTI OUTER JOIN

Everything on the left and right that is unique to each side

```
SELECT *
FROM TABLE_1
OUTER JOIN TABLE_2
ON TABLE_1.KEY = TABLE_2.KEY
WHERE TABLE_1.KEY IS NULL
OR TABLE_2.KEY IS NULL
```

INNER JOIN

Only the things that match on the left AND the right

```
SELECT *
FROM TABLE_1
INNER JOIN TABLE_2
ON TABLE_1.KEY = TABLE_2.KEY
```

CROSS JOIN

All combination of rows from the right and the left (cartesean product)

```
SELECT *
FROM TABLE_1
CROSS JOIN TABLE_2
```

Data dictionary

- Tables in which descriptions of the objects of the base are stored.
  - It is held up to date automatically by the DBMS.
  - These tables can be consulted by SQL language.
- Examples:
  - DICTIONARY (DICT) : dictionary view
  - USER_TABLES : tables and views created by the current user
  - USER_CATALOG (CAT) : tables and views on which the user has rights, other than the tables and views of the dictionary of the data
  - USER_TAB_COLUMNS (COLS) : columns of the tables or views created by the user
  - USER_TAB_PRIVS : objects on which the user is donor or receiver of rights

Exercise

- We have a database which manage reservation places for theatrical performance (spectacles) .

  Spectacles (NumSpec, NameSpec, DatBegSpec, DateEndSpec)

  Performance (NumRep, NumSpec, DatPerf, HourPerf)

  Place (NumRep, NumPlace, Price)

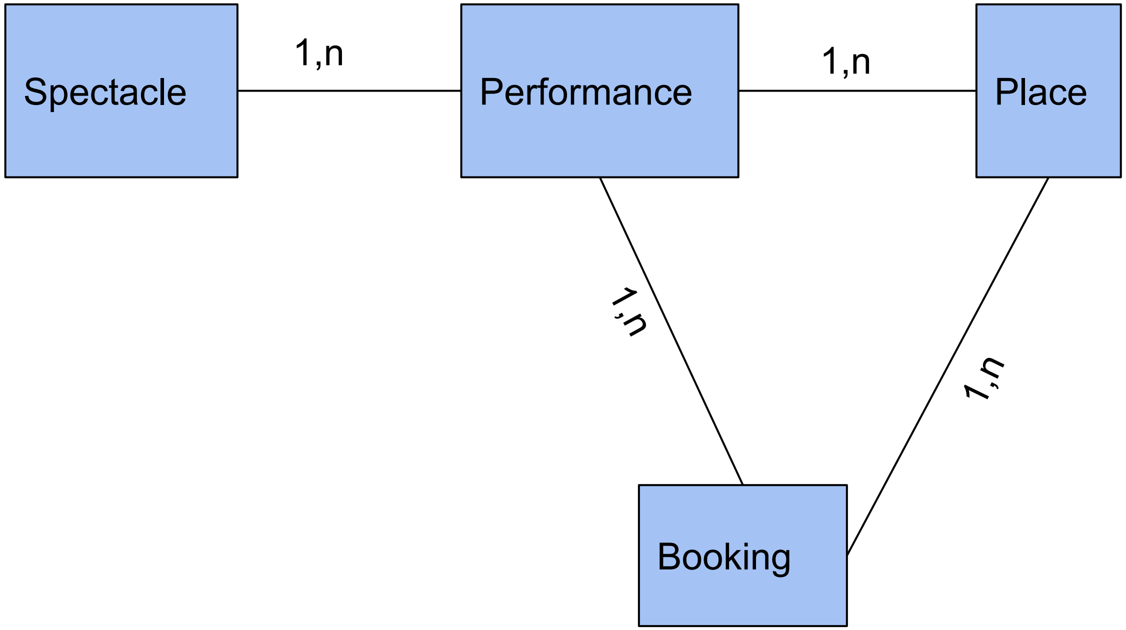
  Book (NumRes, NumRep, NumPlace, NameDem, TelDem)

- Answer following requests:
  - All performances of spectacle N°13 between 15 of september 2005 and 15 of January 2006 or between, 15 of Fébruary 2006 and 15 of May 2006.
  - All performances of the spectacle “Revisor” between 15 September 2005 and 15 January 2006.
  - Find all persons that have booked a place for the last performance of 2005.
  - Find all persons that have booked a place for the performance N°1 and N°2.

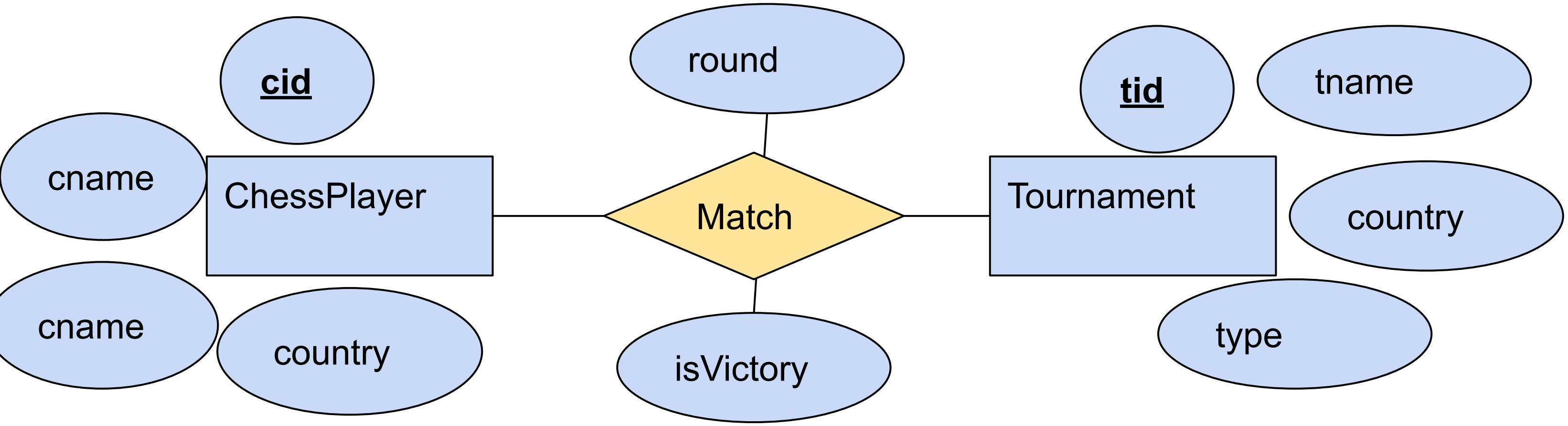
Exercise

- We have a database which manage reservation places for theatrical performance (spectacles) .

  Spectacles (NumSpec, NameSpec, DatBegSpec, DateEndSpec)
  Performance (NumRep, NumSpec, DatPerf, HourPerf)
  Place (NumRep, NumPlace, Price)
  Booking (NumRes, NumRep, NumPlace, NameDem, TelDem)



- ChessPlayer (cid: Integer, cname: String, age: Integer, country: String)
- Match (cid: Integer, tid: Integer, isVictory: Boolean, round: String)
- Tournament (tid: Integer, tname: String, country: String, type: String)



- **All performances of spectacle N°13 between 15 of september 2005 and 15 of January 2006 or between, 15 of Fébruary 2006 and 15 of May 2006:**

```
SELECT *
FROM Performance
WHERE DatPerf BETWEEN ‘15-SEP-2005’ AND ‘15-JAN-2006’
OR DatPerf BETWEEN ‘15-FEV-2006’ AND ‘15-MAI-2006’
AND NumSpec = 13;
```

- **All performances of the spectacle “Revisor” between 15 September 2005 and 15 January 2006:**

```
SELECT DatPerf, HeurPerf
FROM Performance R, Spectacle S
WHERE DatPerf BETWEEN ′15-SEP-05′ AND ′15-JAN-06′
      AND NameSpec = ′Revisor′
      AND S.NumSpec = R.NumSpec ;
```

- **Find all persons that have booked a place for the last performance of 2005:**

```
SELECT NameDem
FROM Booked B, Performance P
WHERE  P.NumRep = B.NumRep
AND P.DatPerf = (   SELECT MAX(DatPerf)
                  FROM Performance
                  WHERE DatPerf<=’31-12-2005’ );
```

- **Find all persons that have booked a place for the performance N°1 and N°2:**

```
SELECT DISTINCT NameDem, TelDem
FROM Performance
WHERE NumRep = 1
INTERSECT
SELECT DISTINCT NameDem, TelDem
FROM Performance
WHERE NumRep = 2 ;
```

❖ **Find all persons that have booked a place for the last performance of 2005:**

```
SELECT NameDem
FROM Booked B, Performance P
WHERE  P.NumRep = B.NumRep
AND P.DatPerf = (   SELECT MAX(DatPerf)
                   FROM Performance
                   WHERE DatPerf<='31-12-2005' );
```

❖ **Find all persons that have booked a place for the performance N°1 and N°2:**

```
SELECT DISTINCT NameDem, TelDem
FROM Performance
WHERE NumRep = 1
INTERSECT
SELECT DISTINCT NameDem, TelDem
FROM Performance
WHERE NumRep = 2 ;
```

Games to learn SQL

Here are some games to learn/practice SQL:

https://datalemur.com/blog/games-to-learn-sql