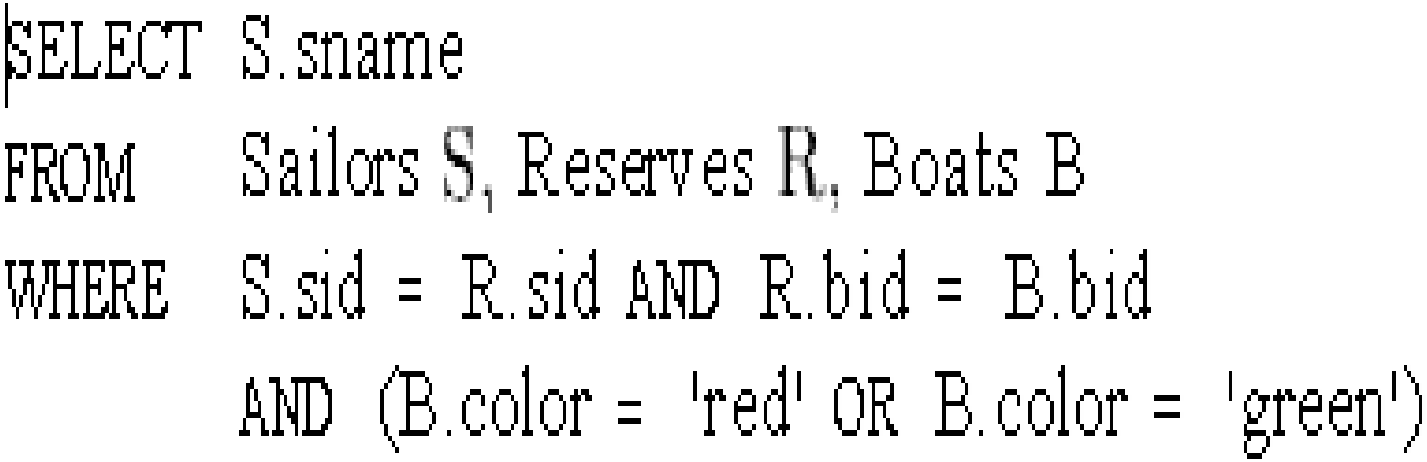
UNION, INTERSECT, AND EXCEPT

SQL supports these operations under the names UNION, INTERSECT, and EXCEPT.

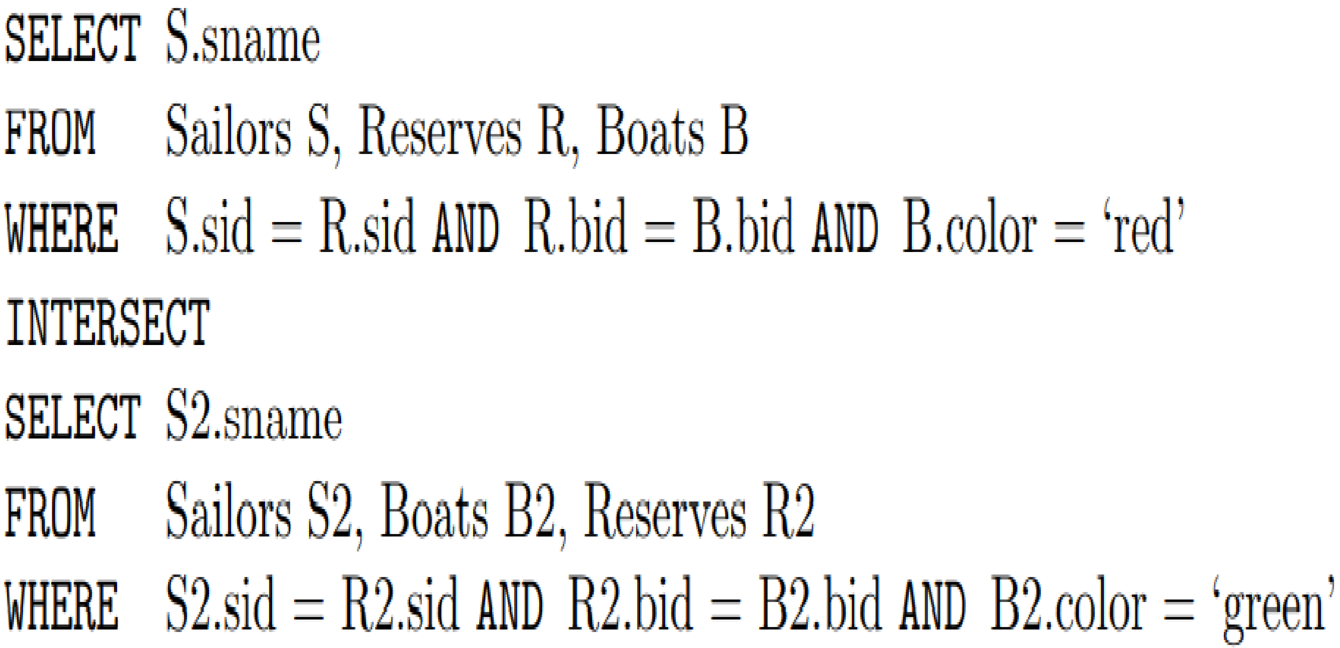
Find the names of sailors who have reserved a red or a green boat.



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

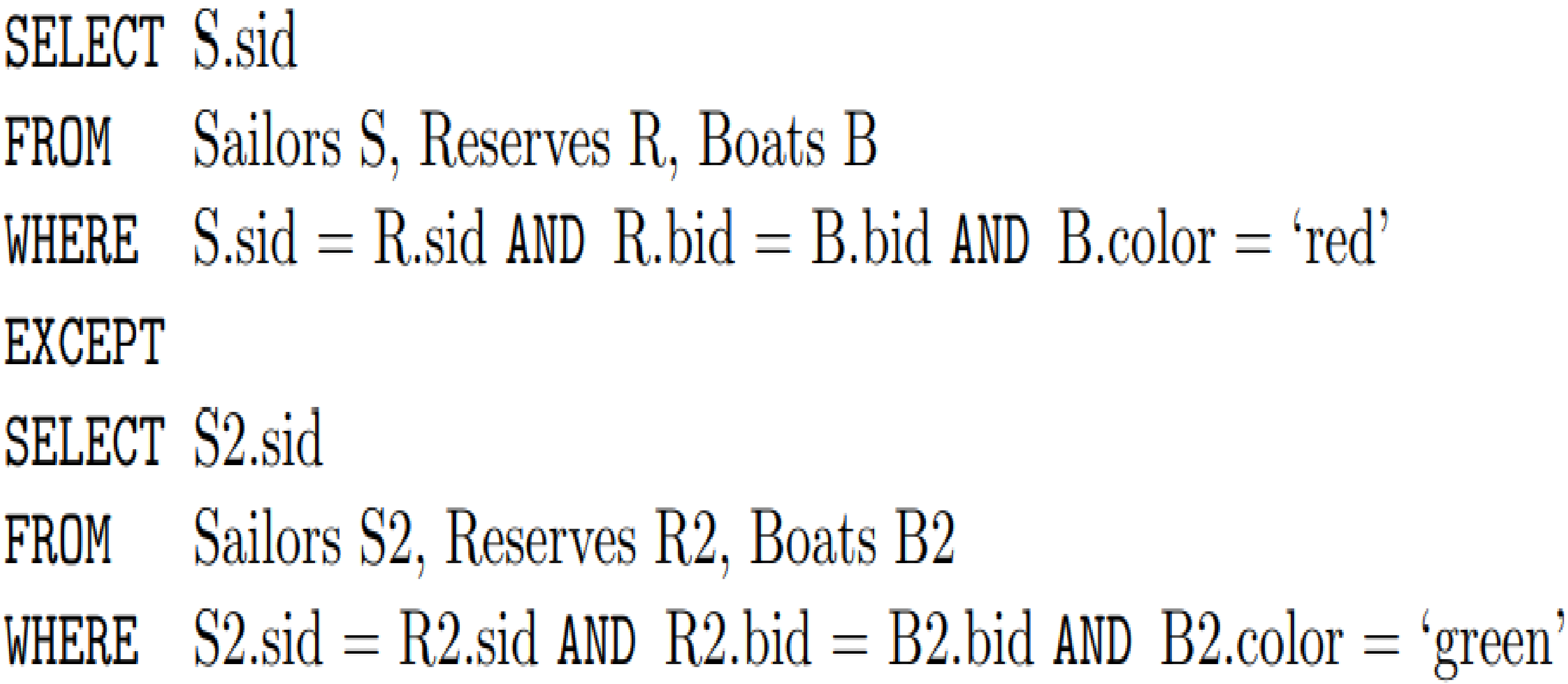
```
SELECT S.sname
FROM   Sailors S, Reserves R, Boats B
WHERE  S.sid = R.sid AND R.bid = B.bid AND B.color = ‘red’
UNION
SELECT S2.sname
FROM   Sailors S2, Boats B2, Reserves R2
WHERE  S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = ‘green’
```

Find the names of sailors who have reserved both a red and a green boat.

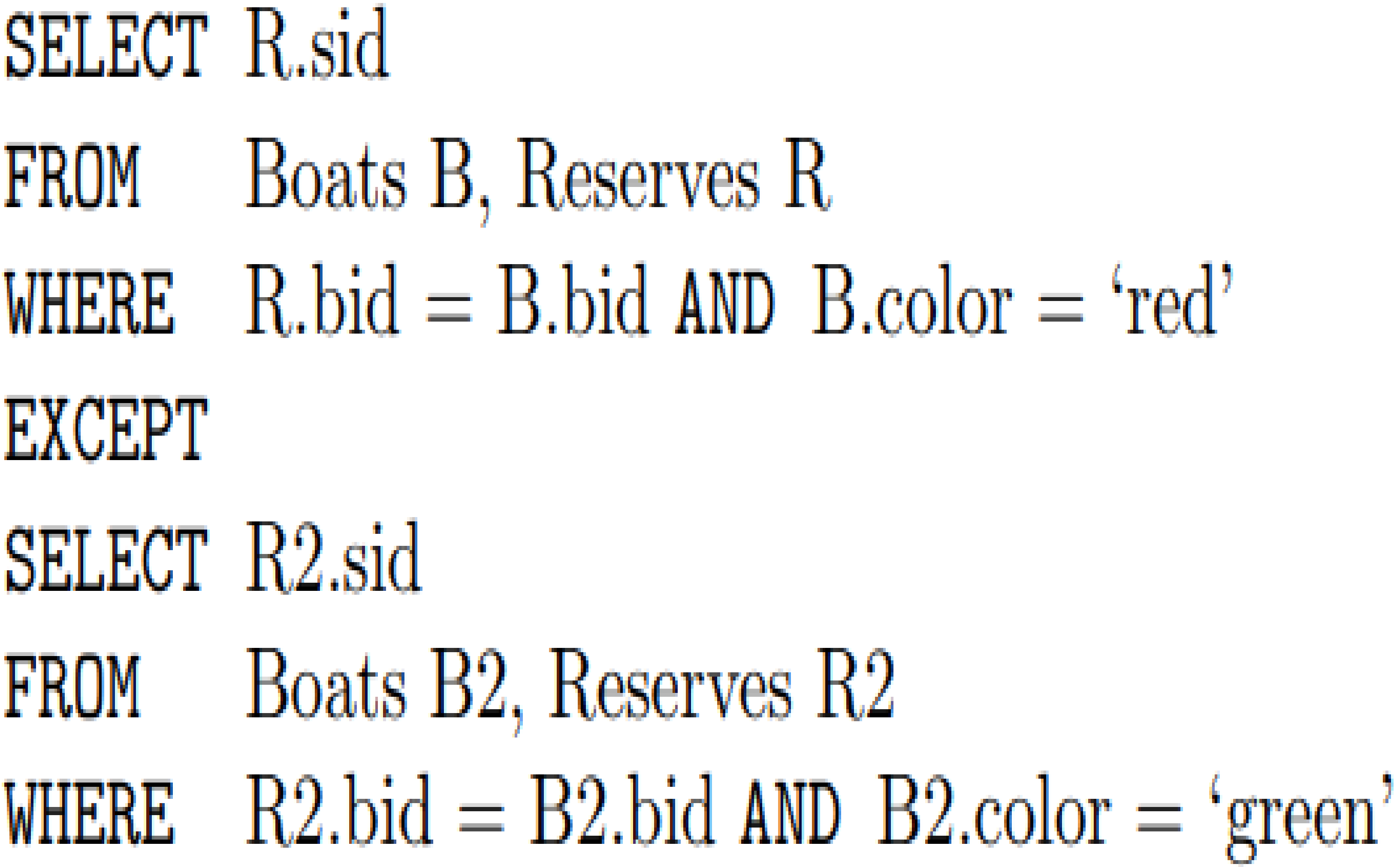


```
SELECT  S.sname
FROM    Sailors S, Reserves R, Boats B
WHERE   S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
INTERSECT
SELECT  S2.sname
FROM    Sailors S2, Boats B2, Reserves R2
WHERE   S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = 'green'
```

Find the sids of all sailors who have reserved red boats but not green boats.

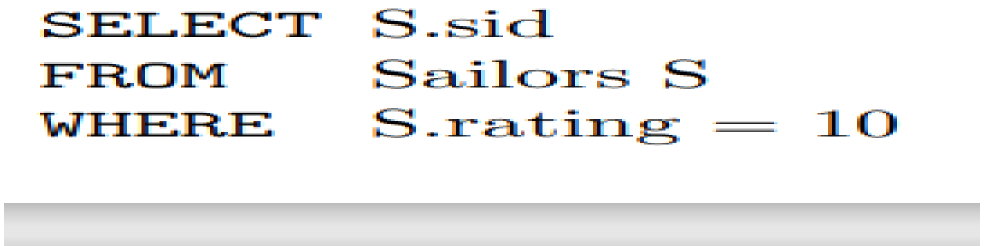


```
SELECT S.sid
FROM   Sailors S, Reserves R, Boats B
WHERE  S.sid = R.sid AND R.bid = B.bid AND B.color = ‘red’
EXCEPT
SELECT S2.sid
FROM   Sailors S2, Reserves R2, Boats B2
WHERE  S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = ‘green’
```

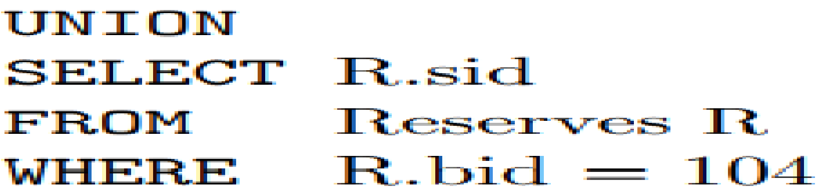


```
SELECT R.sid
FROM   Boats B, Reserves R
WHERE  R.bid = B.bid AND B.color = ‘red’
EXCEPT
SELECT R2.sid
FROM   Boats B2, Reserves R2
WHERE  R2.bid = B2.bid AND B2.color = ‘green’
```

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



```
SELECT  S.sid
FROM    Sailors S
WHERE   S.rating = 10
```

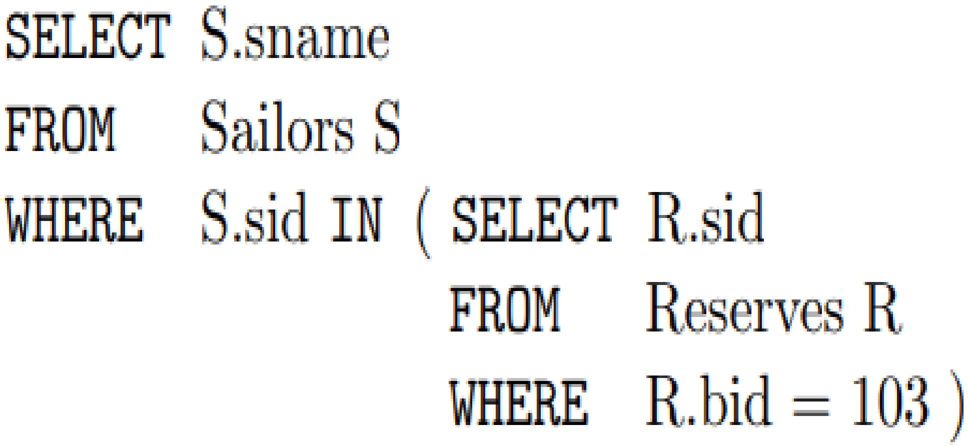


```
UNION
SELECT  R.sid
FROM    Reserves R
WHERE   R.bid = 104
```

NESTED QUERIES

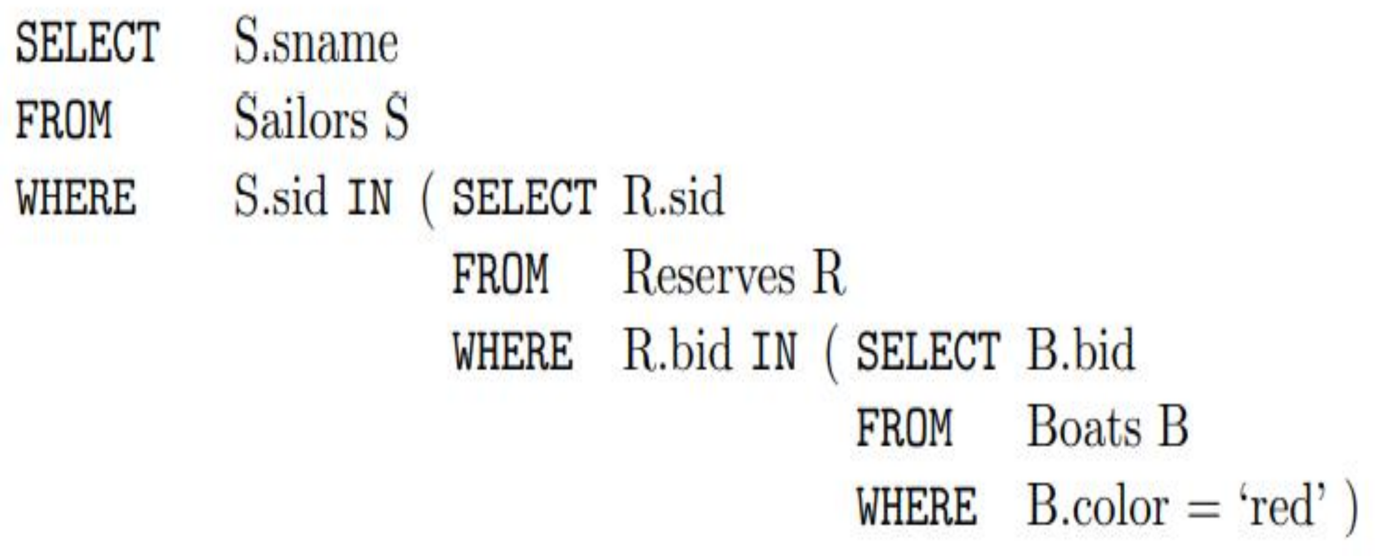
- A nested query is a query that has another query embedded within it; the embedded query is called a subquery.
- A subquery typically appears within the WHERE clause of a query. Subqueries can sometimes appear in the FROM clause or the HAVING clause.
- **IN:** To compare a list of values against a column, we have to use IN operator.

Find the names of sailors who have reserved boat 103.



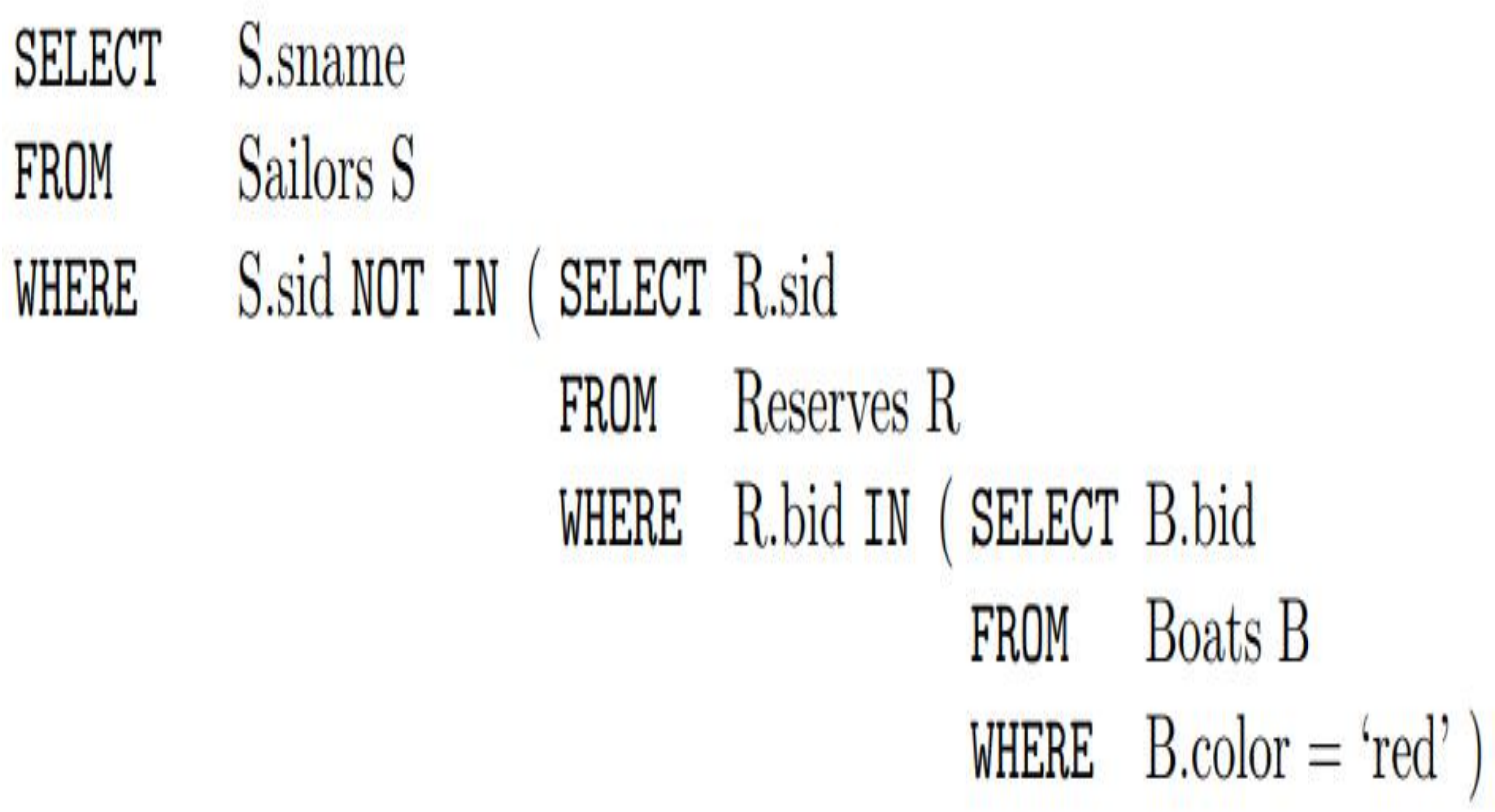
```
SELECT  S.sname
FROM    Sailors S
WHERE   S.sid IN ( SELECT  R.sid
                   FROM    Reserves R
                   WHERE   R.bid = 103 )
```

Find the names of sailors who have reserved a red boat.



```
SELECT  S.sname
FROM    Sailors S
WHERE   S.sid IN ( SELECT  R.sid
                   FROM    Reserves R
                   WHERE   R.bid IN ( SELECT  B.bid
                                      FROM    Boats B
                                      WHERE   B.color = 'red' )
```

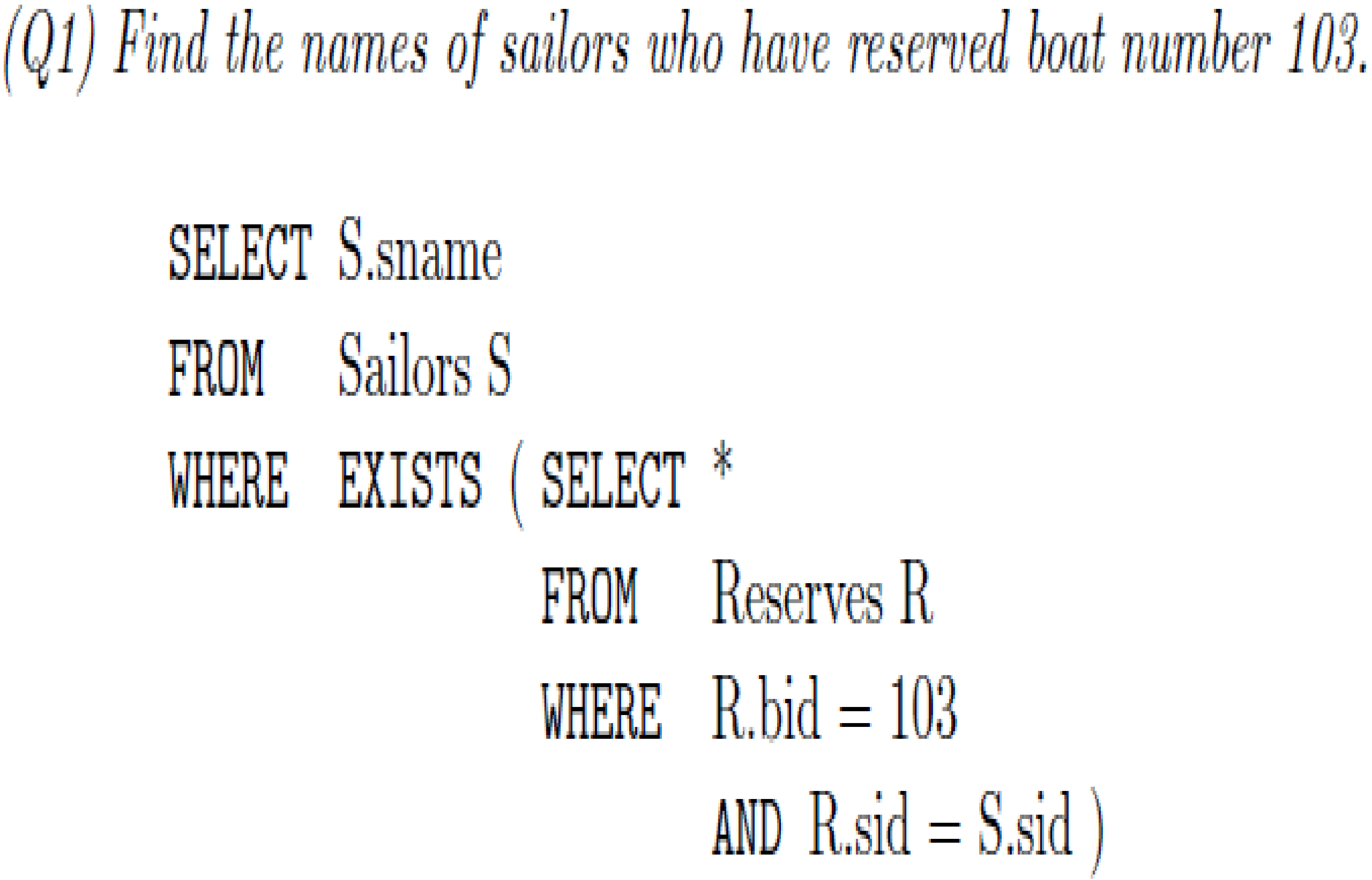
Find the names of sailors who have not reserved a red boat. To find the names of sailors who have not reserved a red boat, we replace the outermost occurrence of IN by NOT IN:



```
SELECT  S.sname
FROM    Sailors S
WHERE   S.sid NOT IN ( SELECT  R.sid
                       FROM    Reserves R
                       WHERE   R.bid IN ( SELECT  B.bid
                                          FROM    Boats B
                                          WHERE   B.color = 'red' )
```

Correlated Nested Queries

- In the nested queries that we have seen thus far, the inner subquery has been completely independent of the outer query.
- In general the inner subquery could depend on the row that is currently being examined in the outer query.
- The EXISTS operator is another set comparison operator, such as IN. It allows us to test whether a set is nonempty

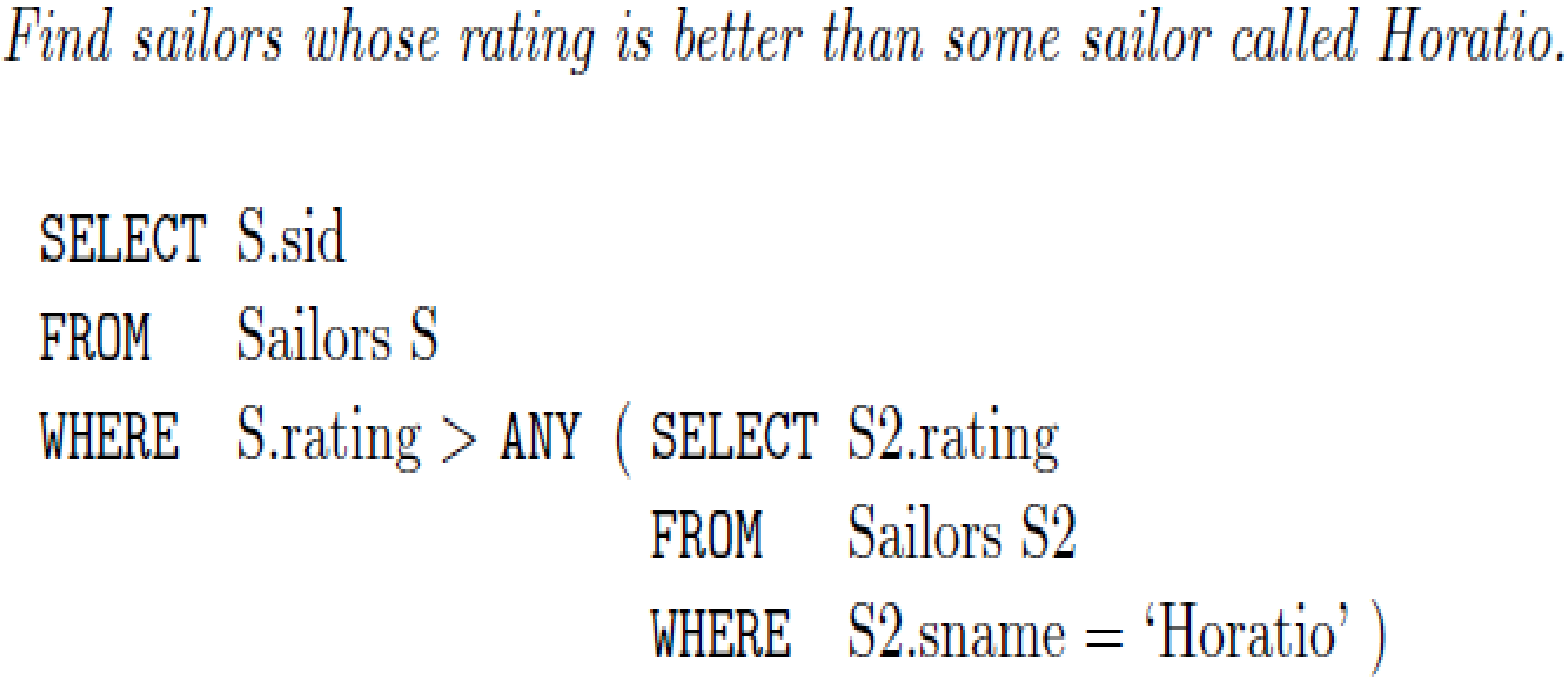


(Q1) Find the names of sailors who have reserved boat number 103.

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

- For each Sailor row *S*, we test whether the set of Reserves rows *R* such that *R.bid = 103* AND *S.sid = R.sid* is nonempty.
- If so, sailor *S* has reserved boat 103, and we retrieve the name. The subquery clearly depends on the current row *S* and must be re-evaluated for each row in Sailors.
- The occurrence of *S* in the subquery (in the form of the literal *S.sid*) is called a *correlation*, and such queries are called *correlated queries*.
- By using NOT EXISTS instead of EXISTS, we can compute the names of sailors who have not reserved a red boat.

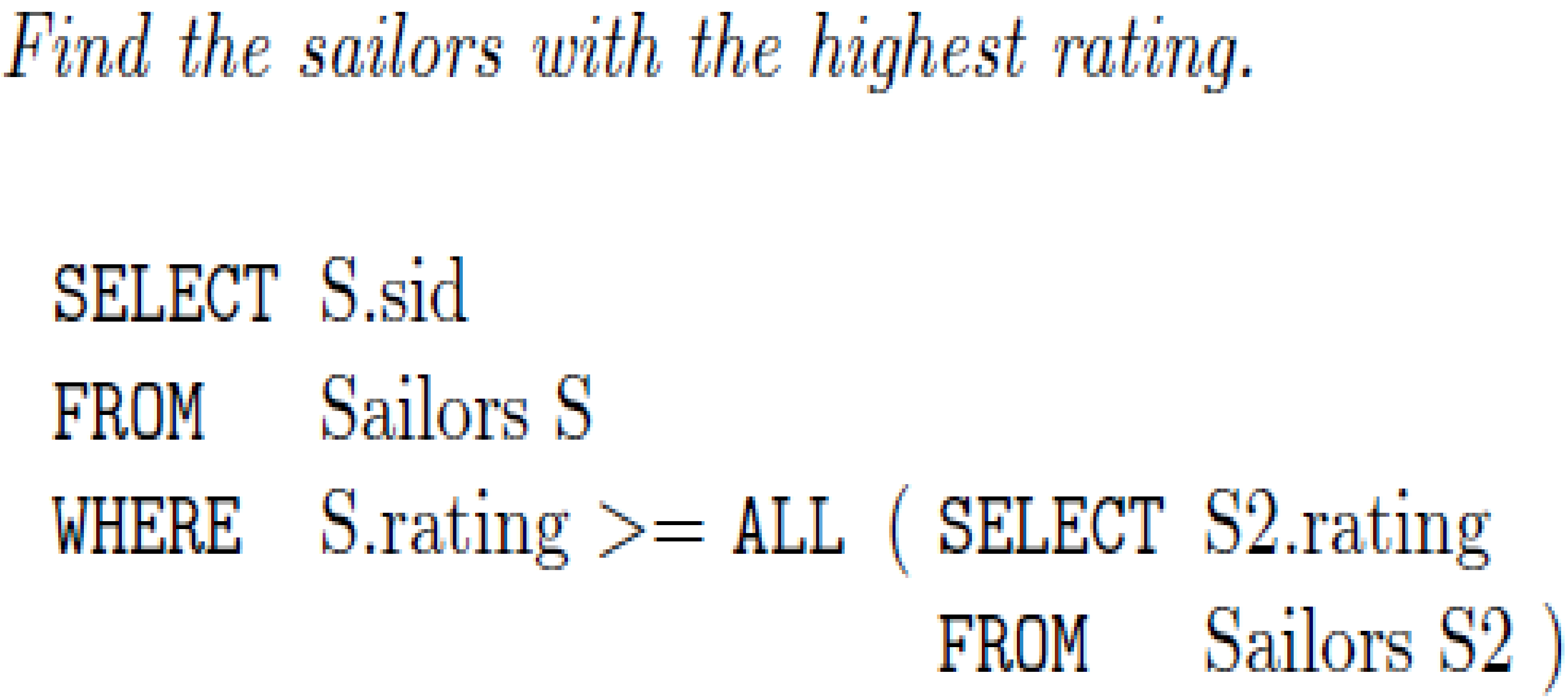
- SQL also supports op ANY and op ALL, where op is one of the arithmetic comparison operators (<;<=;=; <>;>=;>).



Find sailors whose rating is better than some sailor called Horatio.

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

- On instance *S*3, this computes the *sid*s 31, 32, 58, 71, and 74.
- Just replace ANY with ALL in the WHERE clause of the outer query. On instance *S*3, we would get the *sid*s 58 and 71.
- What will be the output if Horatio is not present in the table?



Find the sailors with the highest rating.

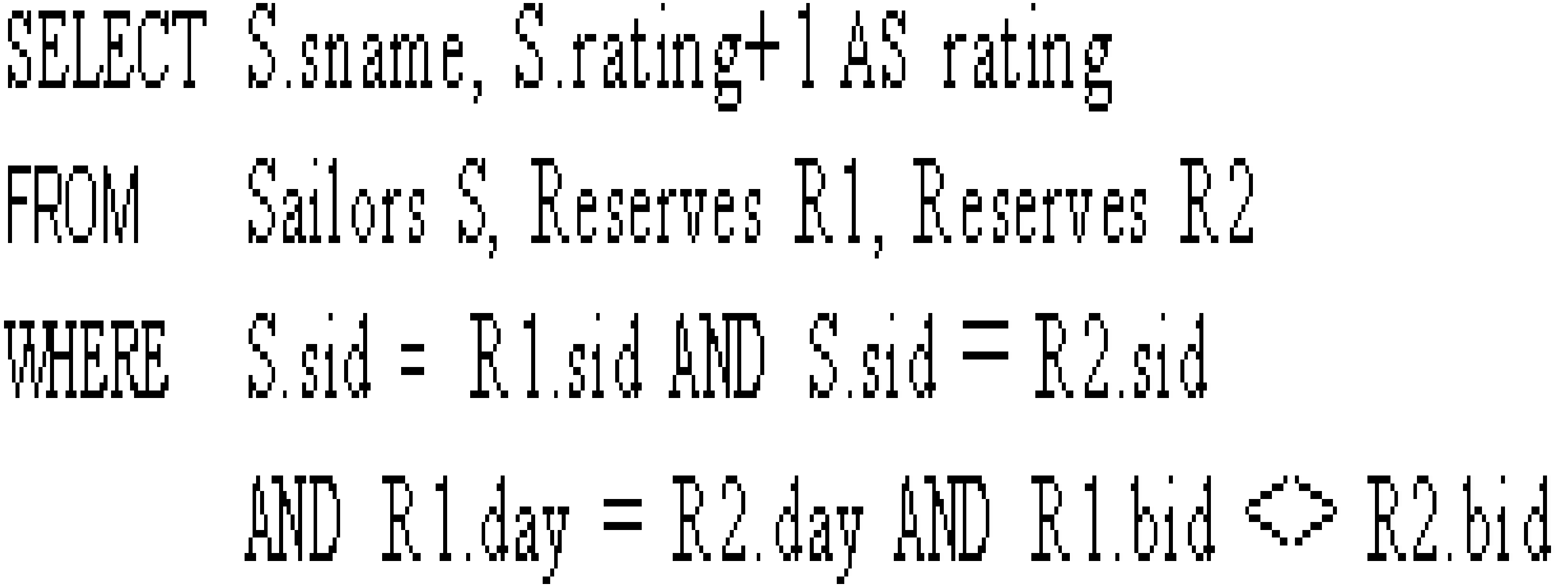
```
SELECT  S.sid
FROM    Sailors S
WHERE   S.rating >= ALL ( SELECT  S2.rating
                          FROM    Sailors S2 )
```

Relational Operators

=	**Equal to**
< or >	Less than or greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to

Expressions and Strings in the SELECT Command.

Compute increments for the ratings of persons who have sailed two different boats on the same day



```
SELECT  S.sname, S.rating+1 AS rating
FROM    Sailors S, Reserves R1, Reserves R2
WHERE   S.sid =  R1.sid AND  S.sid = R2.sid
        AND  R1.day = R2.day AND  R1.bid <> R2.bid
```

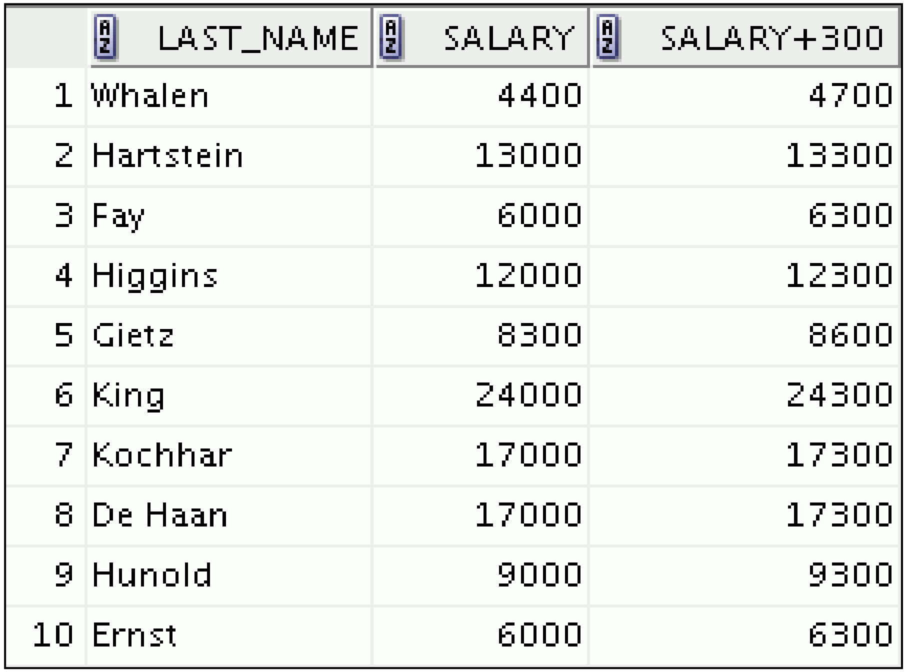
Arithmetic Operators

- Create expressions with number and date data by using arithmetic operators.

Operator	**Description**
+	Add
-	Subtract
*****	Multiply
/	Divide

Using Arithmetic Operators

```
SELECT last_name, salary, salary + 300
FROM   employees;
```



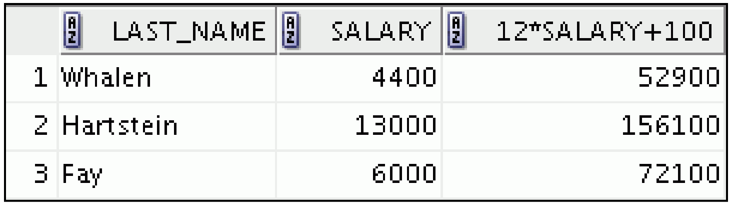
	LAST_NAME	SALARY	SALARY+300
1	Whalen	4400	4700
2	Hartstein	13000	13300
3	Fay	6000	6300
4	Higgins	12000	12300
5	Gietz	8300	8600
6	King	24000	24300
7	Kochhar	17000	17300
8	De Haan	17000	17300
9	Hunold	9000	9300
10	Ernst	6000	6300

…

Operator Precedence

```
SELECT last_name, salary, 12*salary+100
FROM   employees;
```

1

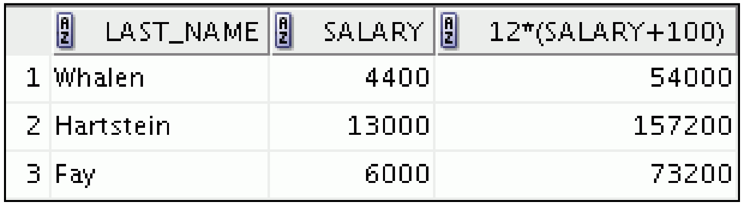


	LAST_NAME	SALARY	12*SALARY+100
1	Whalen	4400	52900
2	Hartstein	13000	156100
3	Fay	6000	72100

…

```
SELECT last_name, salary, 12*(salary+100)
FROM   employees;
```

2



	LAST_NAME	SALARY	12*(SALARY+100)
1	Whalen	4400	54000
2	Hartstein	13000	157200
3	Fay	6000	73200

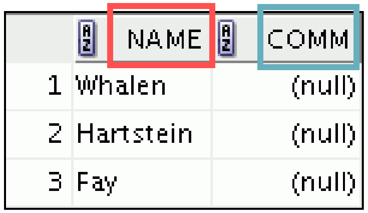
…

Defining a Column Alias

- A column alias:
  - Renames a column heading
  - Is useful with calculations
  - Immediately follows the column name (There can also be the optional `AS` keyword between the column name and alias.)
  - Requires double quotation marks if it contains spaces or special characters, or if it is case-sensitive

Using Column Aliases

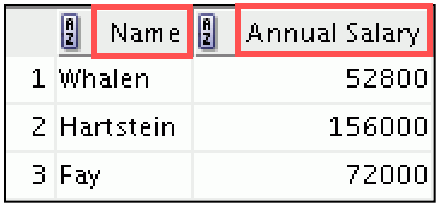
```
SELECT last_name AS name,
commission_pct comm
FROM   employees;
```



	NAME	COMM
1	Whalen	(null)
2	Hartstein	(null)
3	Fay	(null)

…

```
SELECT last_name "Name" , salary*12 "Annual
Salary"
FROM   employees;
```



	Name	Annual Salary
1	Whalen	52800
2	Hartstein	156000
3	Fay	72000

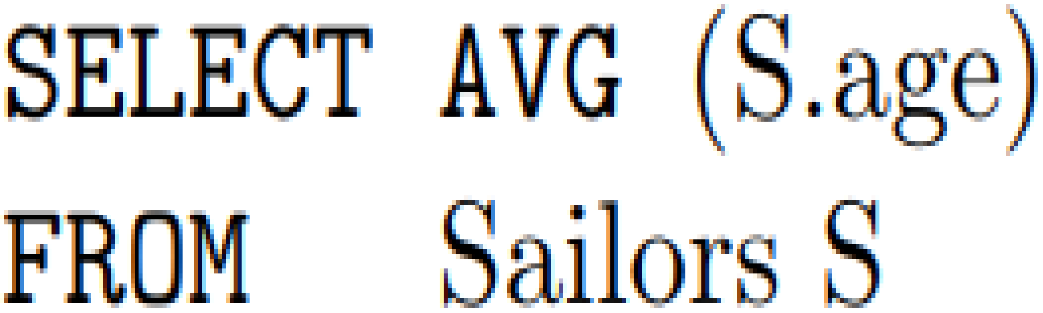
…

AGGREGATE OPERATORS:

Aggregation operators calculate the average, total, minimum, or maximum value of the numeric attributes in a collection of objects, or the number of objects in a collection. Aggregation operators compute a value from a collection of values.

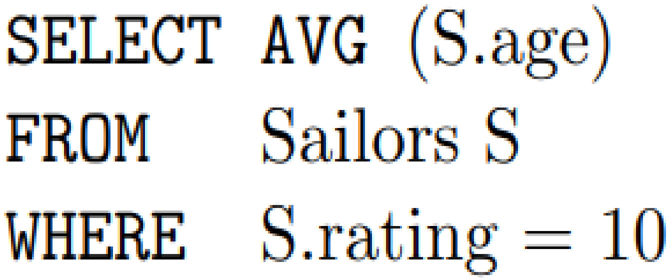
1. COUNT ([DISTINCT] A): The number of (unique) values in the A column.
2. SUM ([DISTINCT] A): The sum of all (unique) values in the A column.
3. AVG ([DISTINCT] A): The average of all (unique) values in the A column.
4. MAX (A): The maximum value in the A column.
5. MIN (A): The minimum value in the A column.

Find the average age of all sailors.



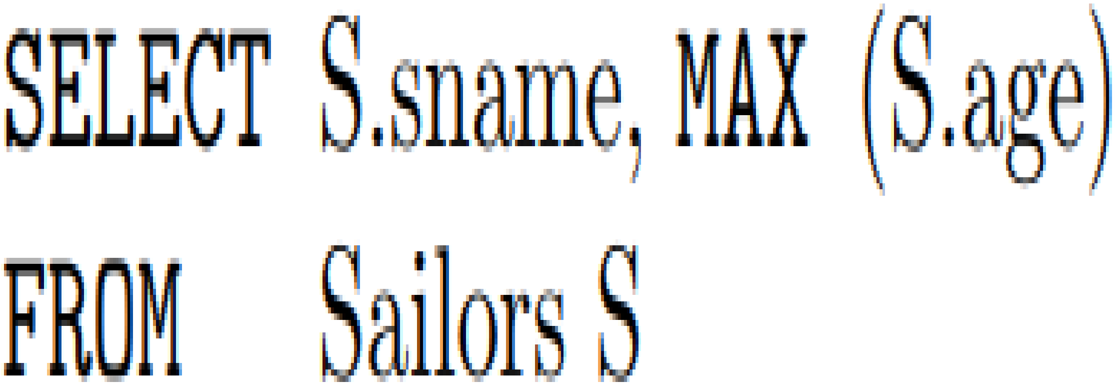
```
SELECT  AVG (S.age)
FROM    Sailors S
```

Find the average age of sailors with a rating of 10.



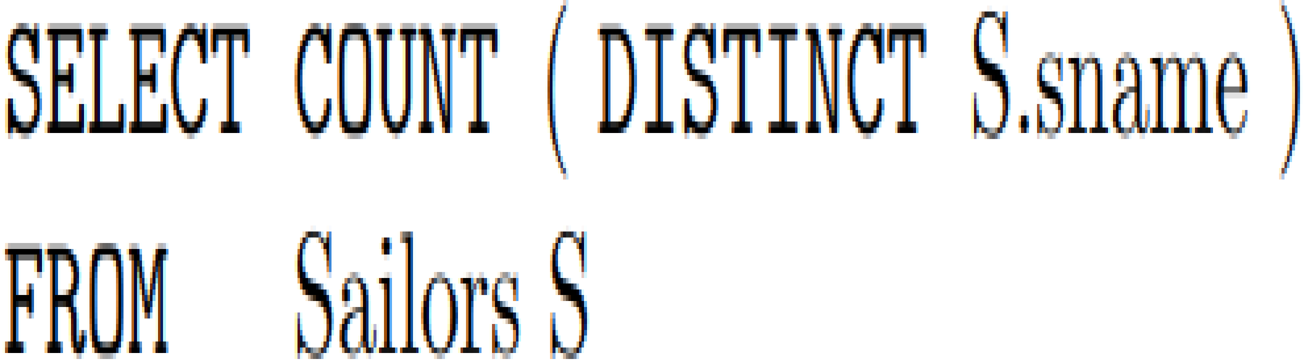
```
SELECT  AVG (S.age)
FROM    Sailors S
WHERE   S.rating = 10
```

Find the name and age of the oldest sailor.



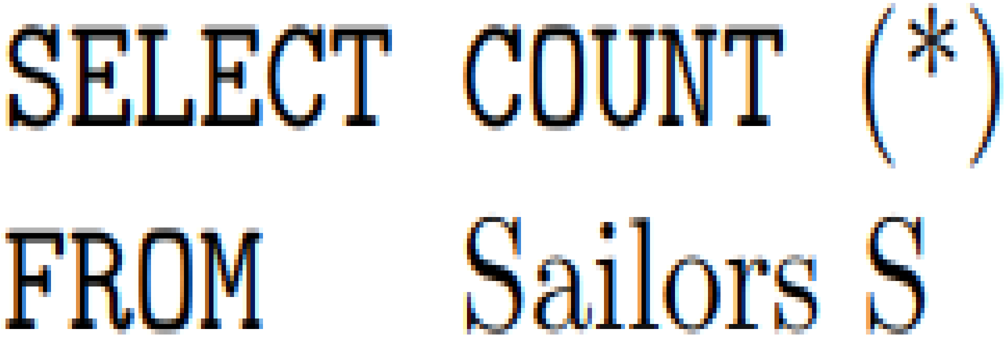
```
SELECT  S.sname, MAX  (S.age)
FROM    Sailors S
```

Count the number of different sailor names.



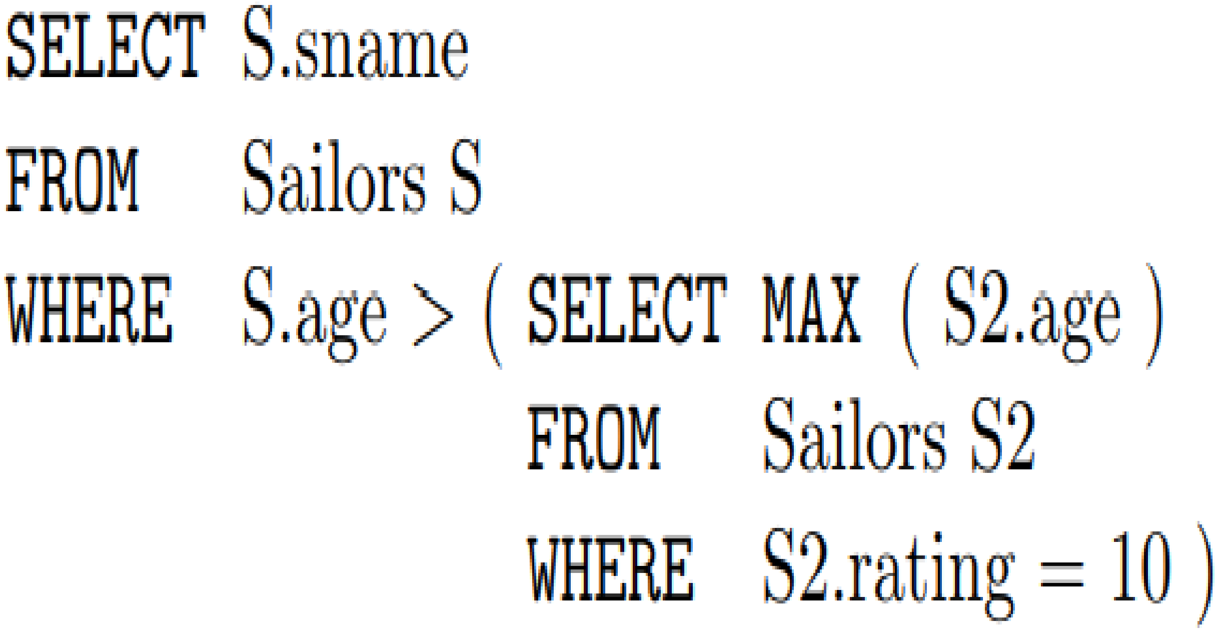
```
SELECT  COUNT ( DISTINCT  S.sname )
FROM    Sailors S
```

Count the number of sailors.



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

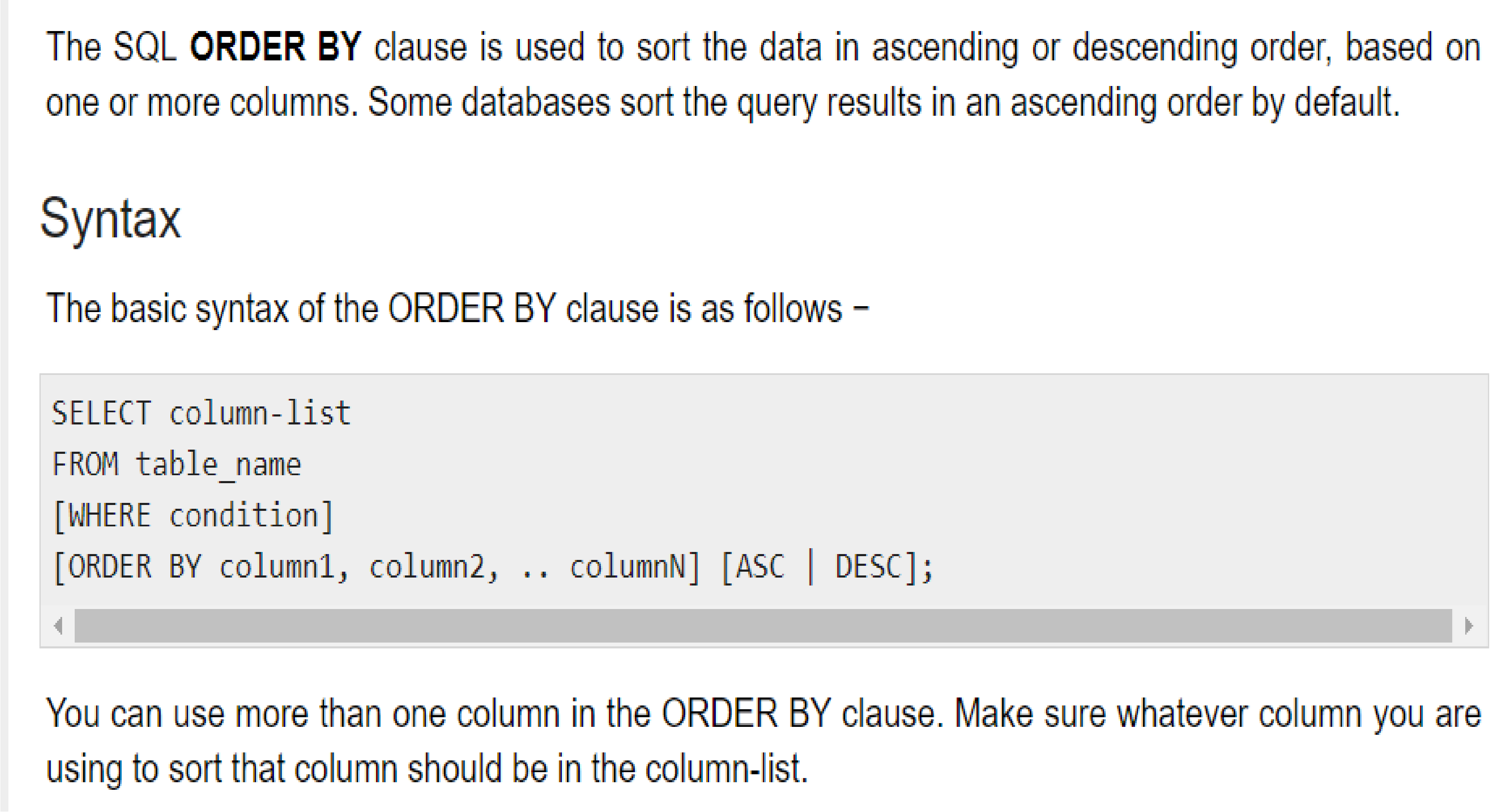
Find the names of sailors who are older than the oldest sailor with a rating of 10.



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

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

SQL ORDER BY



The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. Some databases sort the query results in an ascending order by default.

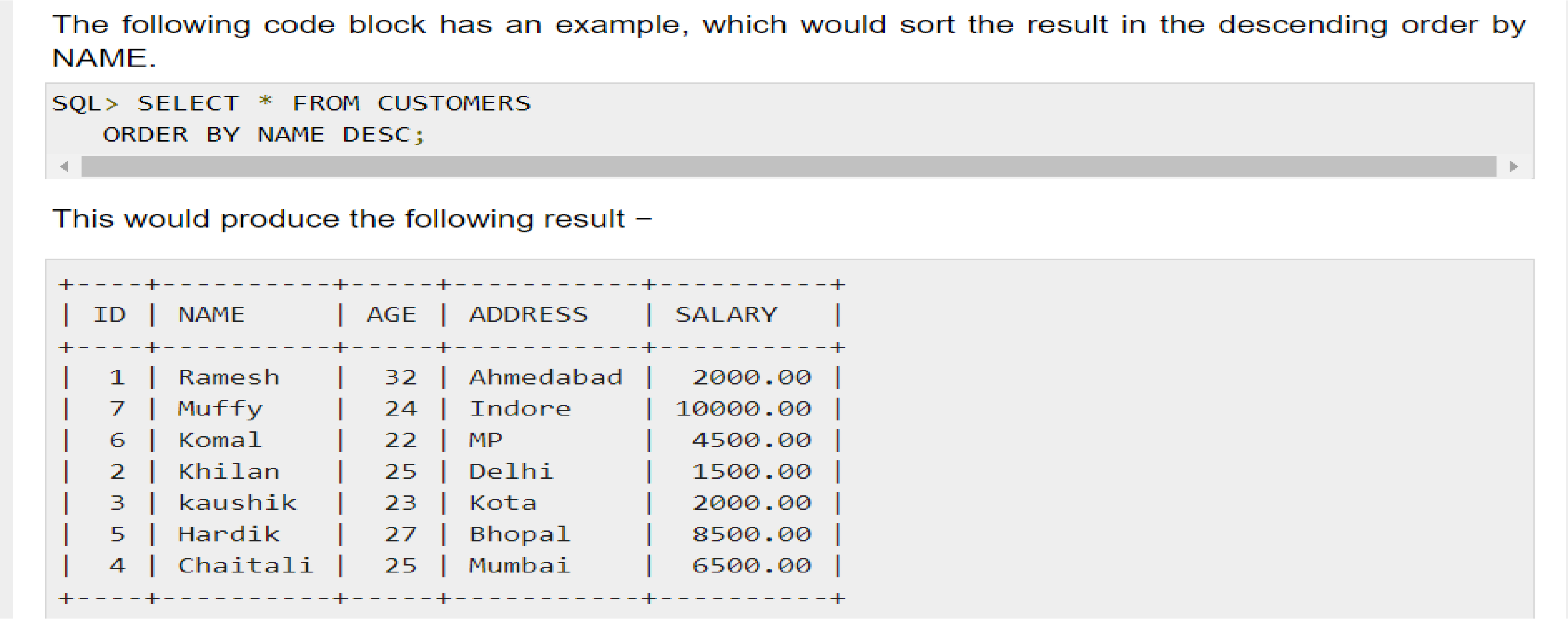
Syntax

The basic syntax of the ORDER BY clause is as follows −

```
SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

You can use more than one column in the ORDER BY clause. Make sure whatever column you are using to sort that column should be in the column-list.

SQL ORDER BY



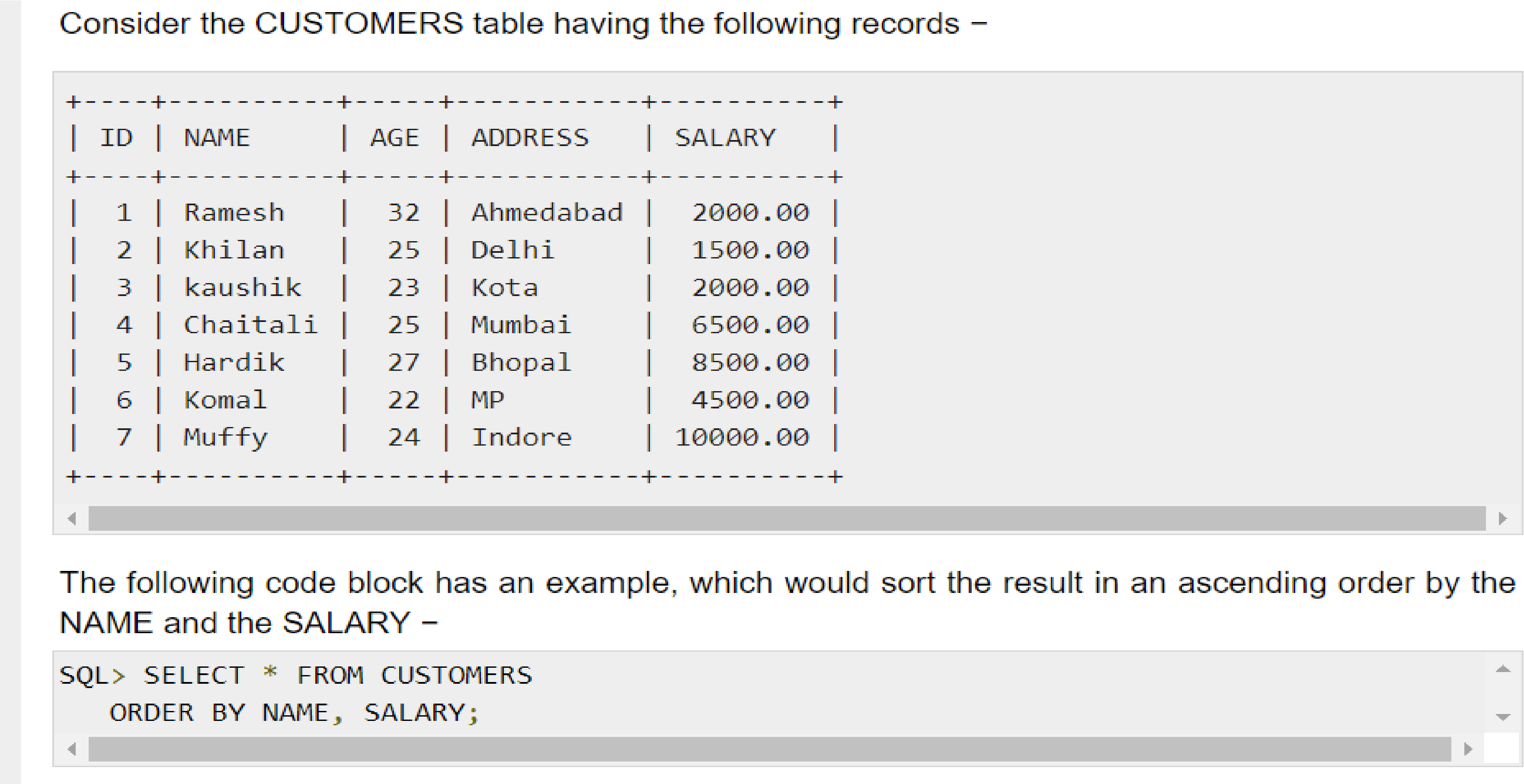
The following code block has an example, which would sort the result in the descending order by NAME.

```
SQL> SELECT * FROM CUSTOMERS
   ORDER BY NAME DESC;
```

This would produce the following result –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
+----+----------+-----+-----------+----------+
```

SQL ORDER BY



Consider the CUSTOMERS table having the following records −

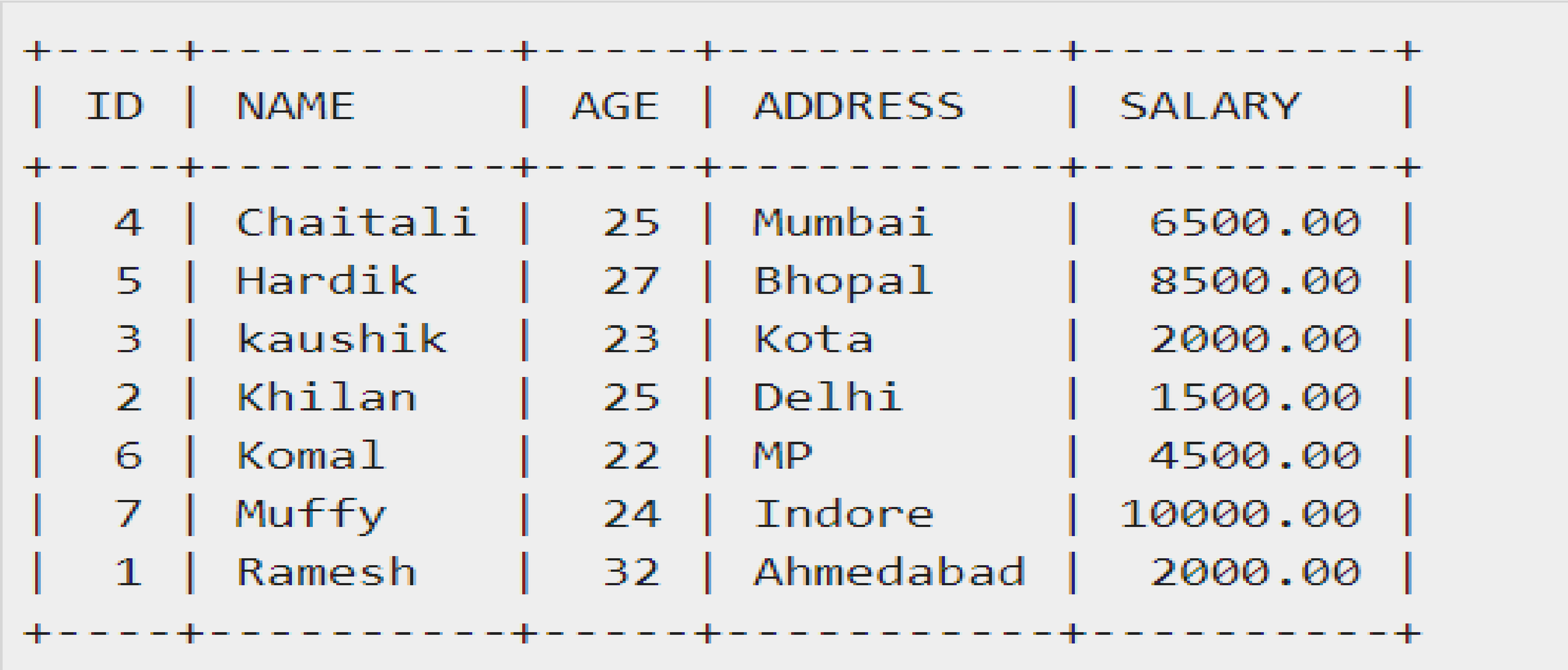
```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following code block has an example, which would sort the result in an ascending order by the NAME and the SALARY −

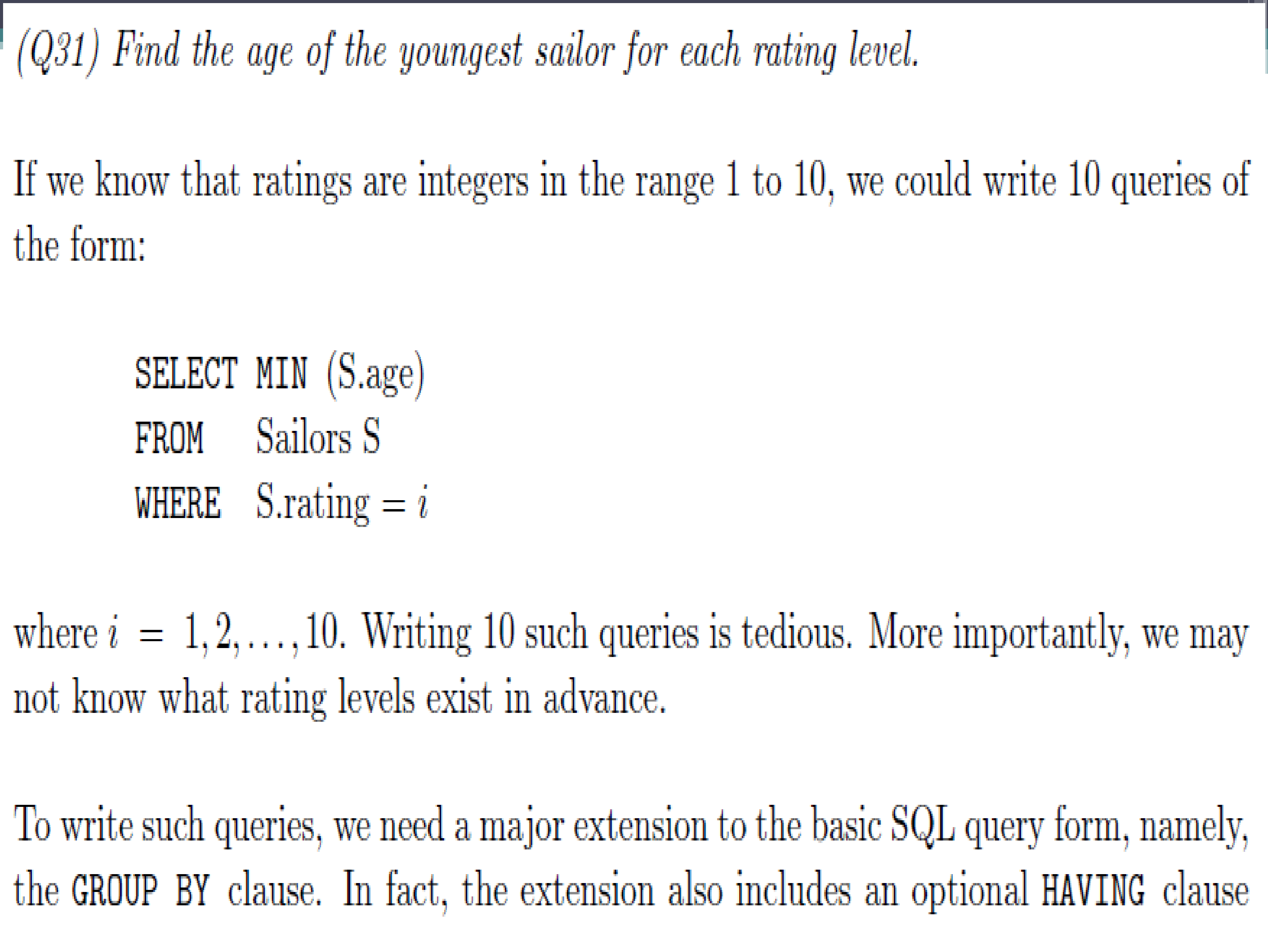
```
SQL> SELECT * FROM CUSTOMERS
   ORDER BY NAME, SALARY;
```

SQL ORDER BY

This would produce the following result –



```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
+----+----------+-----+-----------+----------+
```



(Q31) Find the age of the youngest sailor for each rating level.

If we know that ratings are integers in the range 1 to 10, we could write 10 queries of the form:

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

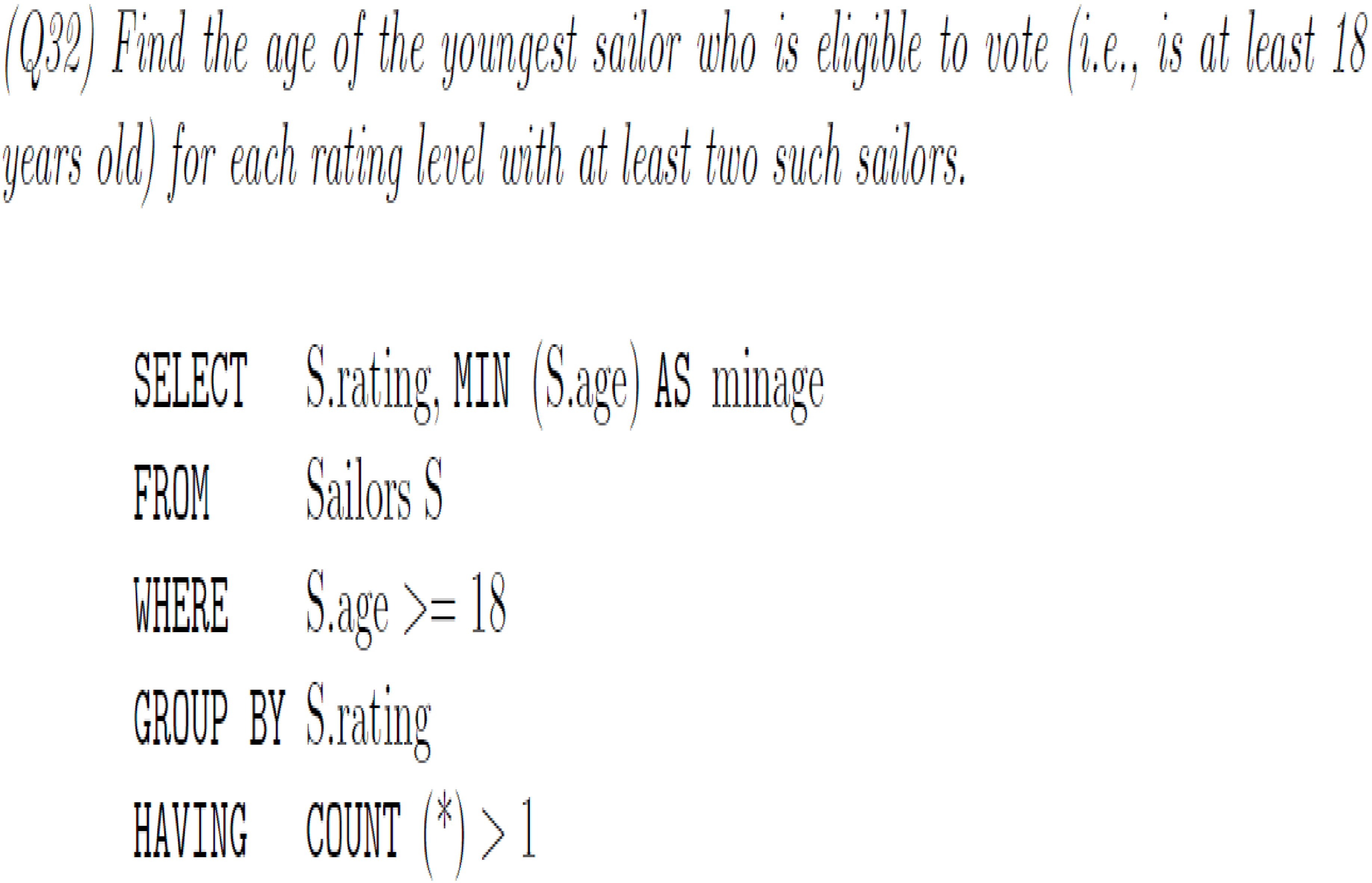
where $i = 1, 2, \ldots, 10$. Writing 10 such queries is tedious. More importantly, we may not know what rating levels exist in advance.

To write such queries, we need a major extension to the basic SQL query form, namely, the GROUP BY clause. In fact, the extension also includes an optional HAVING clause

```
SELECT    [ DISTINCT ] select-list
FROM      from-list
WHERE     qualification
GROUP BY  grouping-list
HAVING    group-qualification
```

Using the GROUP BY clause, we can write Q31 as follows:

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

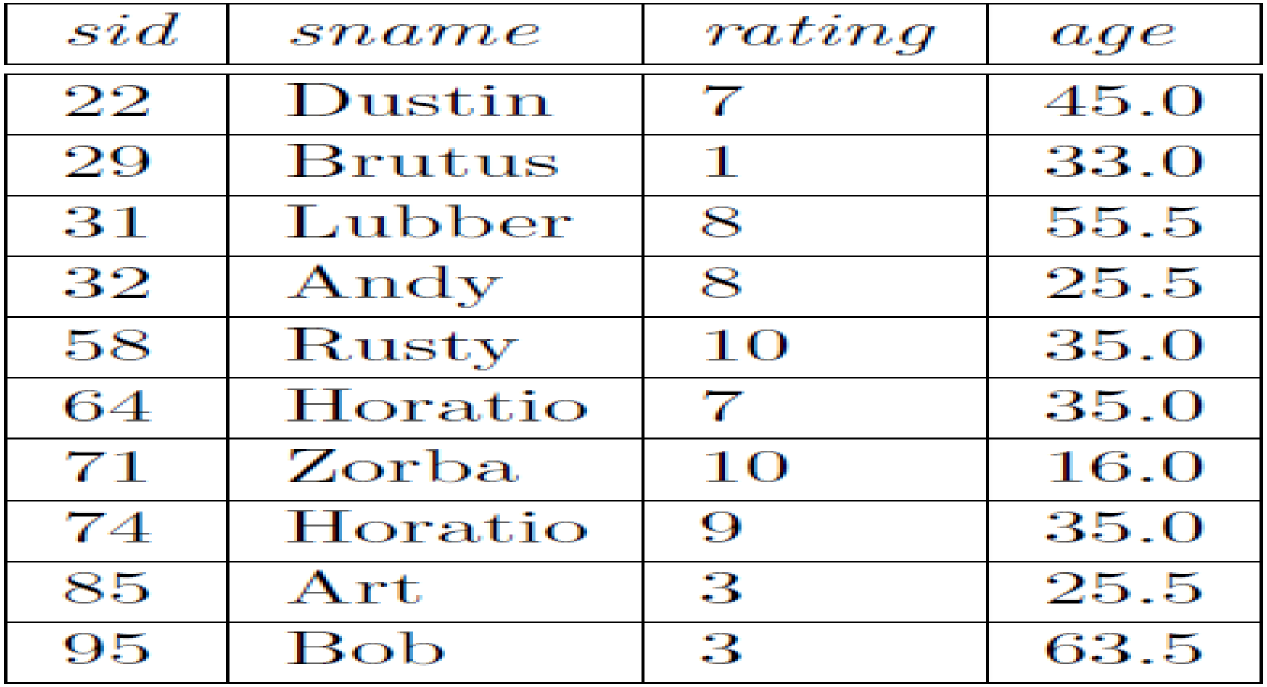


(Q32) Find the age of the youngest sailor who is eligible to vote (i.e., is at least 18 years old) for each rating level with at least two such sailors.

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

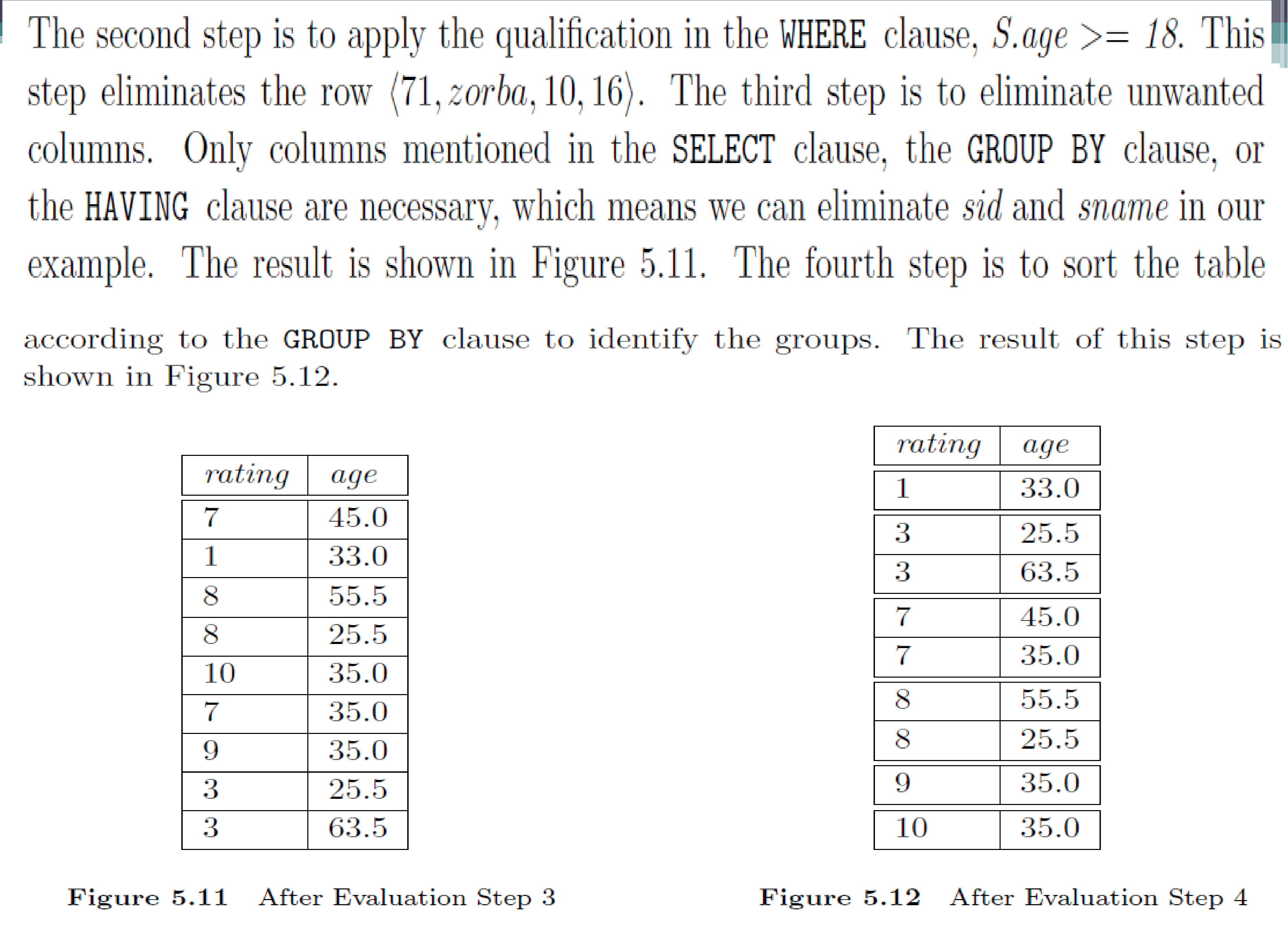
Evaluation Steps:

- The first step is to construct the cross-product of tables in the **from-list**. Because the only relation in the from-list in Query is Sailors result is just the instance shown in Figure



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 5.10 **Instance** $S3$ **of Sailors**



The second step is to apply the qualification in the WHERE clause, *S.age* >= *18*. This step eliminates the row ⟨71, *zorba*, 10, 16⟩. The third step is to eliminate unwanted columns. Only columns mentioned in the SELECT clause, the GROUP BY clause, or the HAVING clause are necessary, which means we can eliminate *sid* and *sname* in our example. The result is shown in Figure 5.11. The fourth step is to sort the table according to the GROUP BY clause to identify the groups. The result of this step is shown in Figure 5.12.

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

Figure 5.11 After Evaluation Step 3

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

Figure 5.12 After Evaluation Step 4

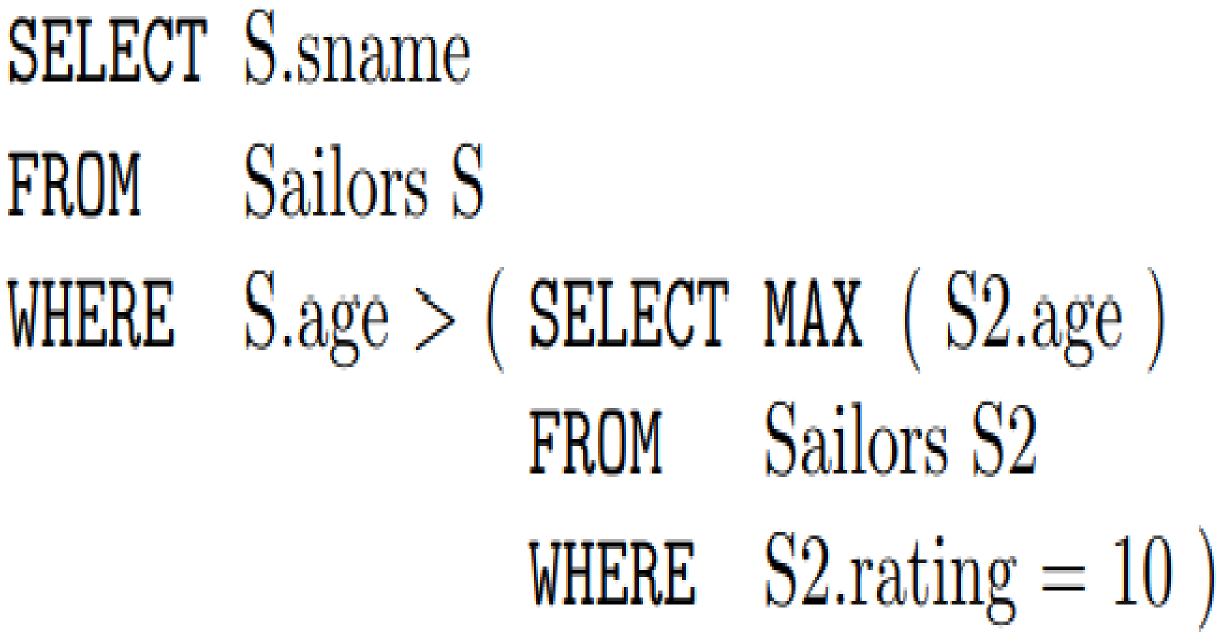
The fifth step is to apply the group-qualification in the `HAVING` clause, that is, the condition `COUNT (*)` > 1. This step eliminates the groups with *rating* equal to 1, 9, and 10. Observe that the order in which the `WHERE` and `GROUP BY` clauses are considered is significant: If the `WHERE` clause were not considered first, the group with *rating=10* would have met the group-qualification in the `HAVING` clause. The sixth step is to generate one answer row for each remaining group. The answer row corresponding to a group consists of a subset of the grouping columns, plus one or more columns generated by applying an aggregation operator. In our example, each answer row has a *rating* column and a *minage* column, which is computed by applying `MIN` to the values in the *age* column of the corresponding group. The result of this step is shown in Figure 5.13.

rating	*minage*
3	25.5
7	35.0
8	25.5

Figure 5.13 Final Result in Sample Evaluation

If the query contains `DISTINCT` in the `SELECT` clause, duplicates are eliminated in an additional, and final, step.

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



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

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

```
SELECT B.bid, COUNT (*) AS  NOR

 FROM Boats B, Reserves R

WHERE R.bid = B.bid AND B.color = ‘red’

GROUP BY B.bid;
```

SELECT queries

SQL SELECT statement syntax- It is the most frequently used SQL command and has the following general syntax
SELECT [DISTINCT| ALL] { * | [fieldExpression [AS newName]}
FROM tableName [alias]
[WHERE condition]
[GROUP BY fieldName(s)]
[HAVING condition] ORDER BY fieldName(s)
Here

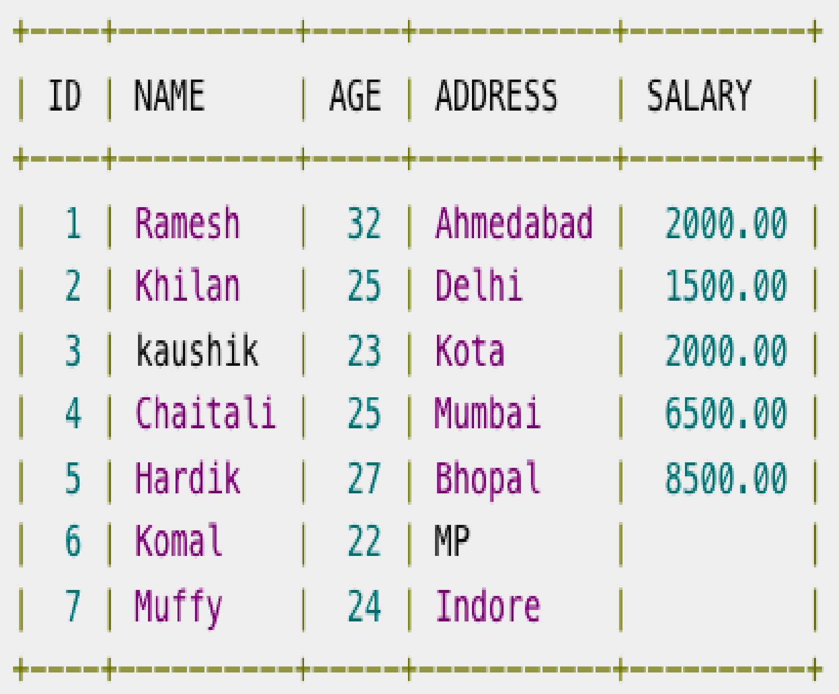
- ✓ **SELECT** is the SQL keyword that lets the database know that we want to retrieve data.

- **[DISTINCT | ALL]** are optional keywords that can be used to fine tune the results returned from the SQL SELECT statement. If nothing is specified then ALL is assumed as the default.
- **{*| [fieldExpression [AS newName]}** at least one part must be specified, "*" selected all the fields from the specified table name, field Expression performs some computations on the specified fields such as adding numbers or putting together two string fields into one.
- **FROM** tableName is mandatory and must contain at least one table, multiple tables must be separated using commas or joined using the JOIN keyword.

- **WHERE** condition is optional, it can be used to specify criteria in the result set returned from the query.
- **GROUP BY** is used to put together records that have the same field values.
- **HAVING** condition is used to specify criteria when working using the GROUP BY keyword.
- **ORDER BY** is used to specify the sort order of the result set.

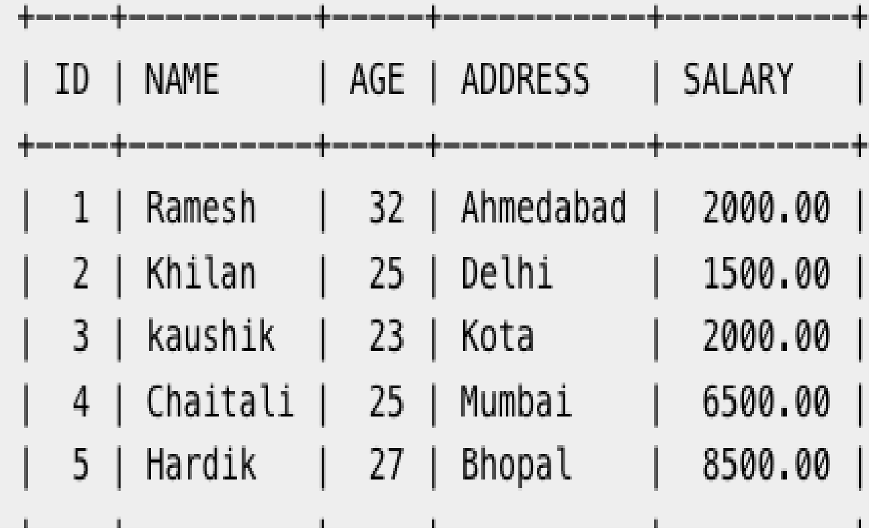
Null Values

- We use null when the column value is either unknown or inapplicable
- If we compare two null values using , =, and so on, the result is always unknown. For example, if we have null in two distinct rows of the sailor relation, any comparison returns unknown.
- SQL also provides a special comparison operator IS NULL to test whether a column value is null and also IS NOT NULL.



```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |          |
|  7 | Muffy    |  24 | Indore    |          |
+----+----------+-----+-----------+----------+
```

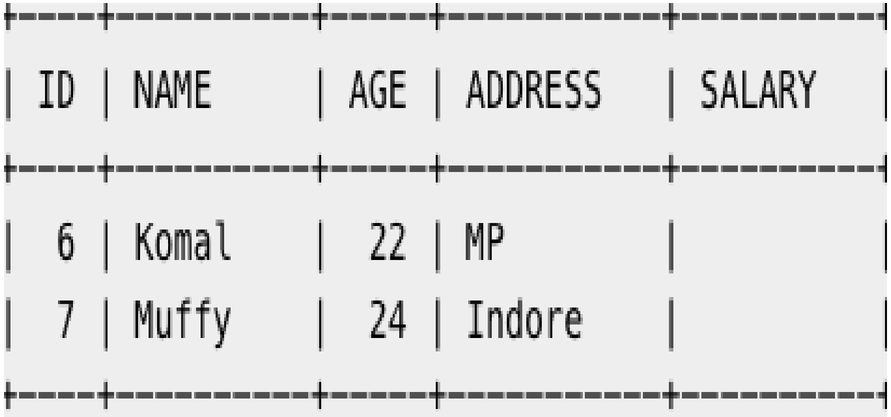
SELECT *
FROM CUSTOMERS
WHERE SALARY IS NOT NULL;



```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
```

SELECT * FROM CUSTOMERS
WHERE SALARY IS NULL;

This would produce the following result



```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  6 | Komal    |  22 | MP        |          |
|  7 | Muffy    |  24 | Indore    |          |
+----+----------+-----+-----------+----------+
```

Logical operators AND, OR, and NOT with NULL

- Once we have null values, we must define the logical operators AND, OR, and NOT using a three-valued logic in which expressions evaluate to true, false, or unknown.
- The expression NOT unknown is defined to be unknown.
- OR of two arguments evaluates to true if either argument evaluates to true, and to unknown if one argument evaluates to false and the other evaluates to unknown.
- AND of two arguments evaluates to false if either argument evaluates to false, and to unknown if one argument evaluates to unknown and the other evaluates to true or unknown.

NULL in SQL

- In SQL,the qualification in the WHERE clause eliminates rows (in the cross-product of tables named in the FROM clause) for which the qualification does not evaluate to true. Therefore, in the presence of null values, any row that evaluates to false or to unknown is eliminated.
- We can disallow null values by specifying NOT NULL as part of the field definition, for example, sname CHAR(20) NOT NULL.
- In addition, the fields in a primary key are not allowed to take on null values. Thus, there is an implicit NOT NULL constraint for every field listed in a PRIMARY KEY constraint.

Joins

- An SQL **join clause combines columns from one or more tables in a relational database.**

- There are 4 different types of SQL joins:
- SQL INNER JOIN (simple join)
- SQL LEFT OUTER JOIN (LEFT JOIN)
- SQL RIGHT OUTER JOIN (RIGHT JOIN)
- SQL FULL OUTER JOIN (FULL JOIN)

Inner join

- It is the most common type of join. SQL Server INNER JOINS return all rows from multiple tables where the join condition is met.

- **Syntax:**

  ```
  SELECT columns
  FROM table1 INNER JOIN table2
  ON table1.column = table2.column;
  ```

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

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

```
Select * from S1 INNER JOIN S2 ON S1.sid = S2.sid ;
```

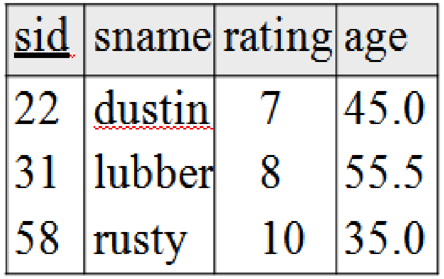
LEFT OUTER JOIN

- This type of join returns all rows from the LEFT-hand table specified in the ON condition and only those rows from the other table where the joined fields are equal (join condition is met).

- **Syntax:**

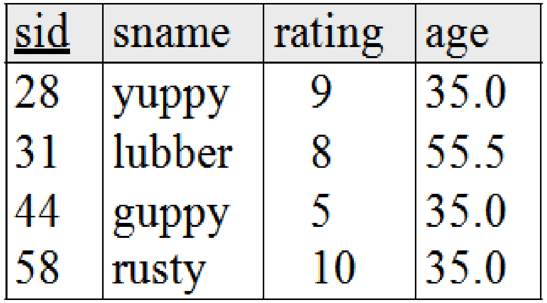
```
SELECT columns
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```

S1



sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2



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

Select * from S1 LEFT OUTER JOIN S2 ON S1.sid = S2.sid ;

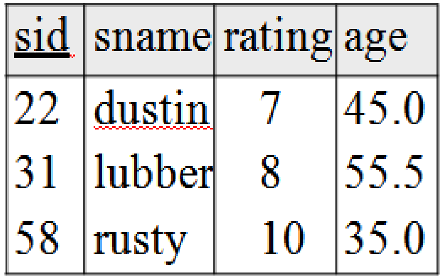
RIGHT OUTER JOIN

- This type of join returns all rows from the RIGHT-hand table specified in the ON condition and **only those rows from the other table where** the joined fields are equal (join condition is met).

- Syntax:

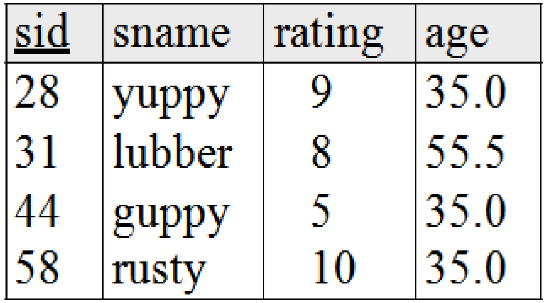
```
SELECT columns
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;
```

S1



sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2



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

Select * from S1 RIGHT OUTER JOIN S2 ON S1.sid = S2.sid ;

Full outer join

- This type of join returns all rows from the LEFT-hand table and RIGHT-hand table with nulls in place where the join condition is not met.

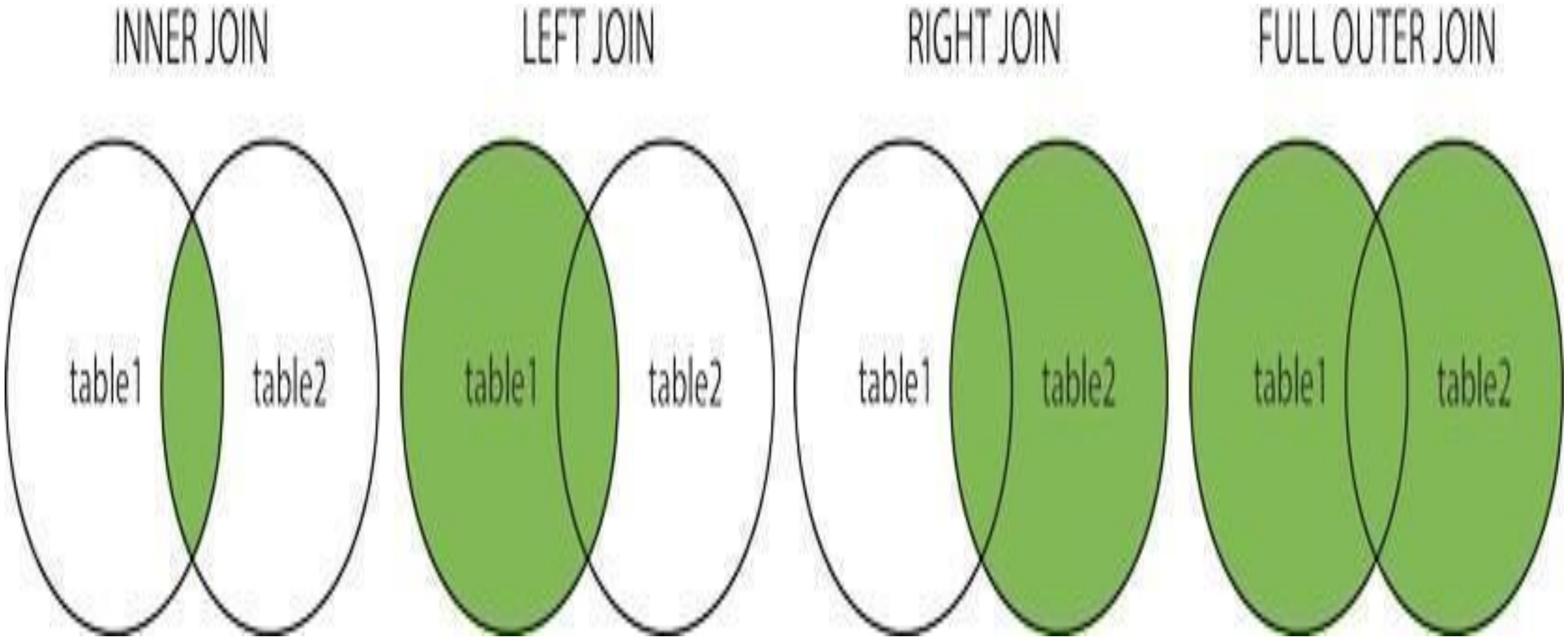
- **Syntax:** SELECT columns

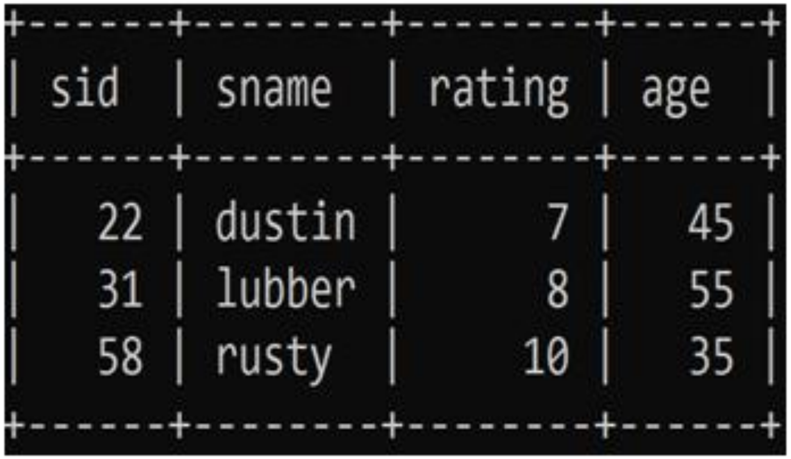
```
FROM table1
FULL [OUTER] JOIN table2
ON table1.column = table2.column;
```

Different Types of SQL JOINs

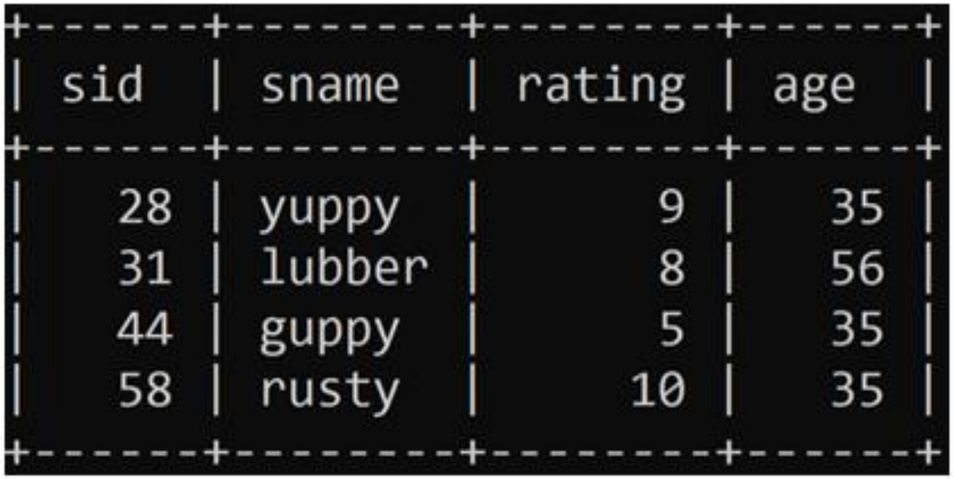
The different types of the JOINs in SQL:

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table


INNER JOIN
table1
table2
LEFT JOIN
table1
table2
RIGHT JOIN
table1
table2
FULL OUTER JOIN
table1
table2



```
+-----+--------+--------+-----+
| sid | sname  | rating | age |
+-----+--------+--------+-----+
|  22 | dustin |      7 |  45 |
|  31 | lubber |      8 |  55 |
|  58 | rusty  |     10 |  35 |
+-----+--------+--------+-----+
```



```
+------+--------+--------+------+
| sid  | sname  | rating | age  |
+------+--------+--------+------+
|   28 | yuppy  |      9 |   35 |
|   31 | lubber |      8 |   56 |
|   44 | guppy  |      5 |   35 |
|   58 | rusty  |     10 |   35 |
+------+--------+--------+------+
```

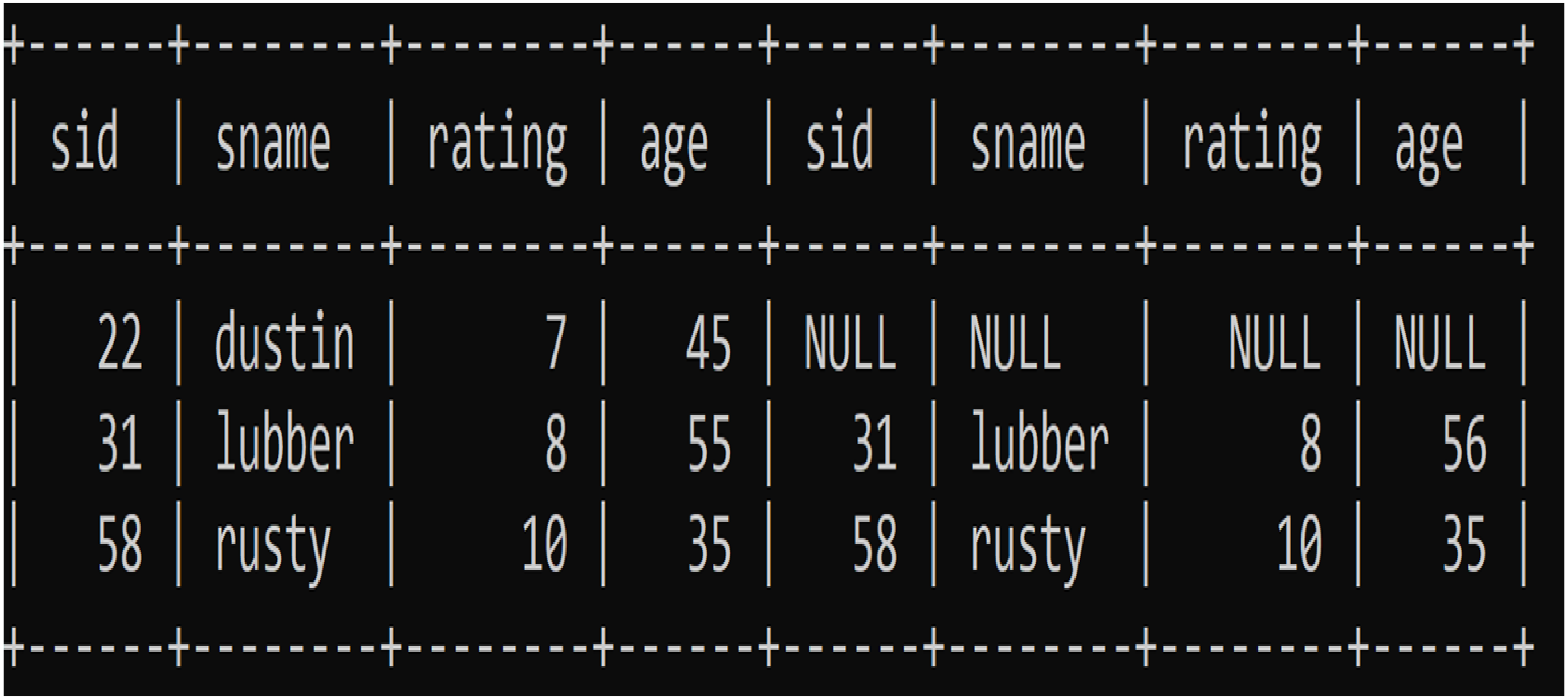
LEFT OUTER JOIN

- This type of join returns all rows from the LEFT-hand table specified in the ON condition and only those rows from the other table where the joined fields are equal (join condition is met).

- **Syntax:**

```
SELECT columns
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```

Select *from sailors1 s1 left outer join sailors2 s2 on s1.sid=s2.sid;



```
+-----+--------+--------+-----+-----+--------+--------+-----+
| sid | sname  | rating | age | sid | sname  | rating | age |
+-----+--------+--------+-----+-----+--------+--------+-----+
|  22 | dustin |      7 |  45 | NULL| NULL   |    NULL| NULL|
|  31 | lubber |      8 |  55 |   31| lubber |       8|   56|
|  58 | rusty  |     10 |  35 |   58| rusty  |      10|   35|
+-----+--------+--------+-----+-----+--------+--------+-----+
```

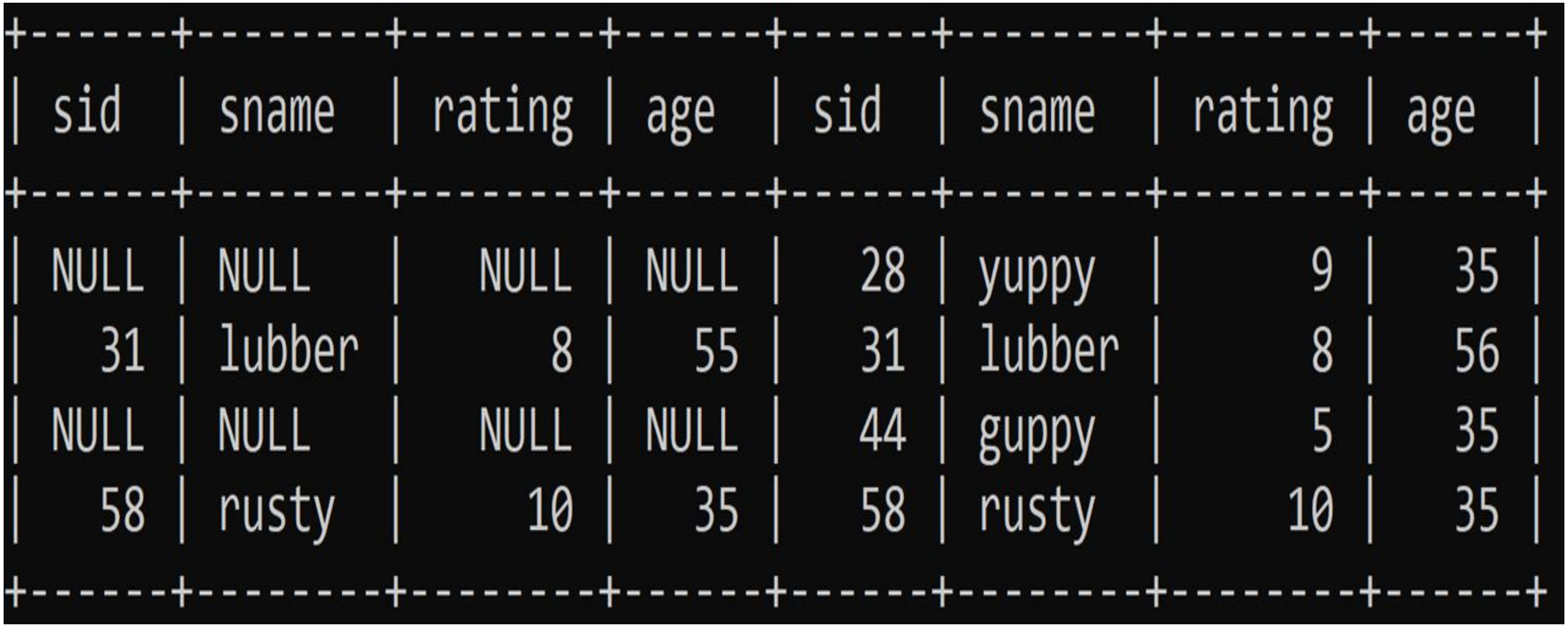
RIGHT OUTER JOIN

- This type of join returns all rows from the RIGHT-hand table specified in the ON condition and **only those rows from the other table where** the joined fields are equal (join condition is met).

- Syntax:

```
SELECT columns
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;
```

Select *from sailors1 s1 right outer join sailors2 s2 on s1.sid=s2.sid;



```
+------+--------+--------+------+------+--------+--------+------+
| sid  | sname  | rating | age  | sid  | sname  | rating | age  |
+------+--------+--------+------+------+--------+--------+------+
| NULL | NULL   |   NULL | NULL |   28 | yuppy  |      9 |   35 |
|   31 | lubber |      8 |   55 |   31 | lubber |      8 |   56 |
| NULL | NULL   |   NULL | NULL |   44 | guppy  |      5 |   35 |
|   58 | rusty  |     10 |   35 |   58 | rusty  |     10 |   35 |
+------+--------+--------+------+------+--------+--------+------+
```

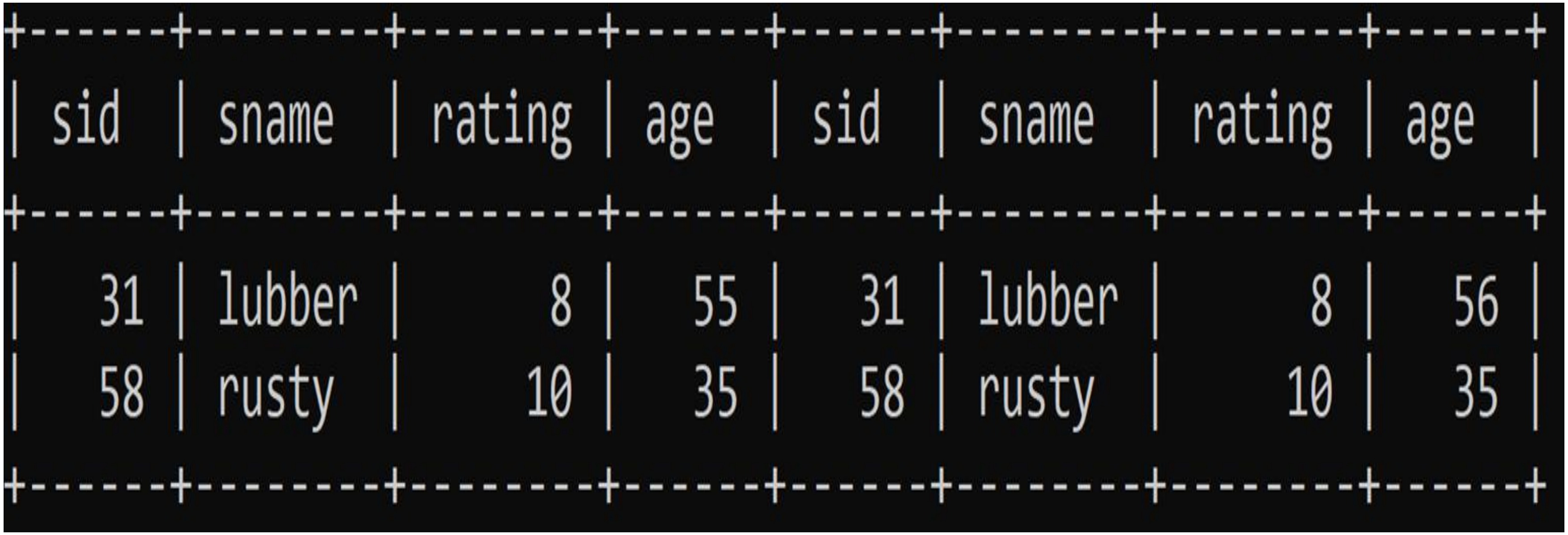
Inner join

- It is the most common type of join. SQL Server INNER JOINS return all rows from multiple tables where the join condition is met.

- **Syntax:**

  ```
  SELECT columns
  FROM table1 INNER JOIN table2
  ON table1.column = table2.column;
  ```

Select *from sailors1 s1 inner join sailors2 s2 on s1.sid=s2.sid;



```
+------+--------+--------+------+------+--------+--------+------+
| sid  | sname  | rating | age  | sid  | sname  | rating | age  |
+------+--------+--------+------+------+--------+--------+------+
|   31 | lubber |      8 |   55 |   31 | lubber |      8 |   56 |
|   58 | rusty  |     10 |   35 |   58 | rusty  |     10 |   35 |
+------+--------+--------+------+------+--------+--------+------+
```