

Structured Query Language (SQL)

- SQL is the most widely used commercial relational database language

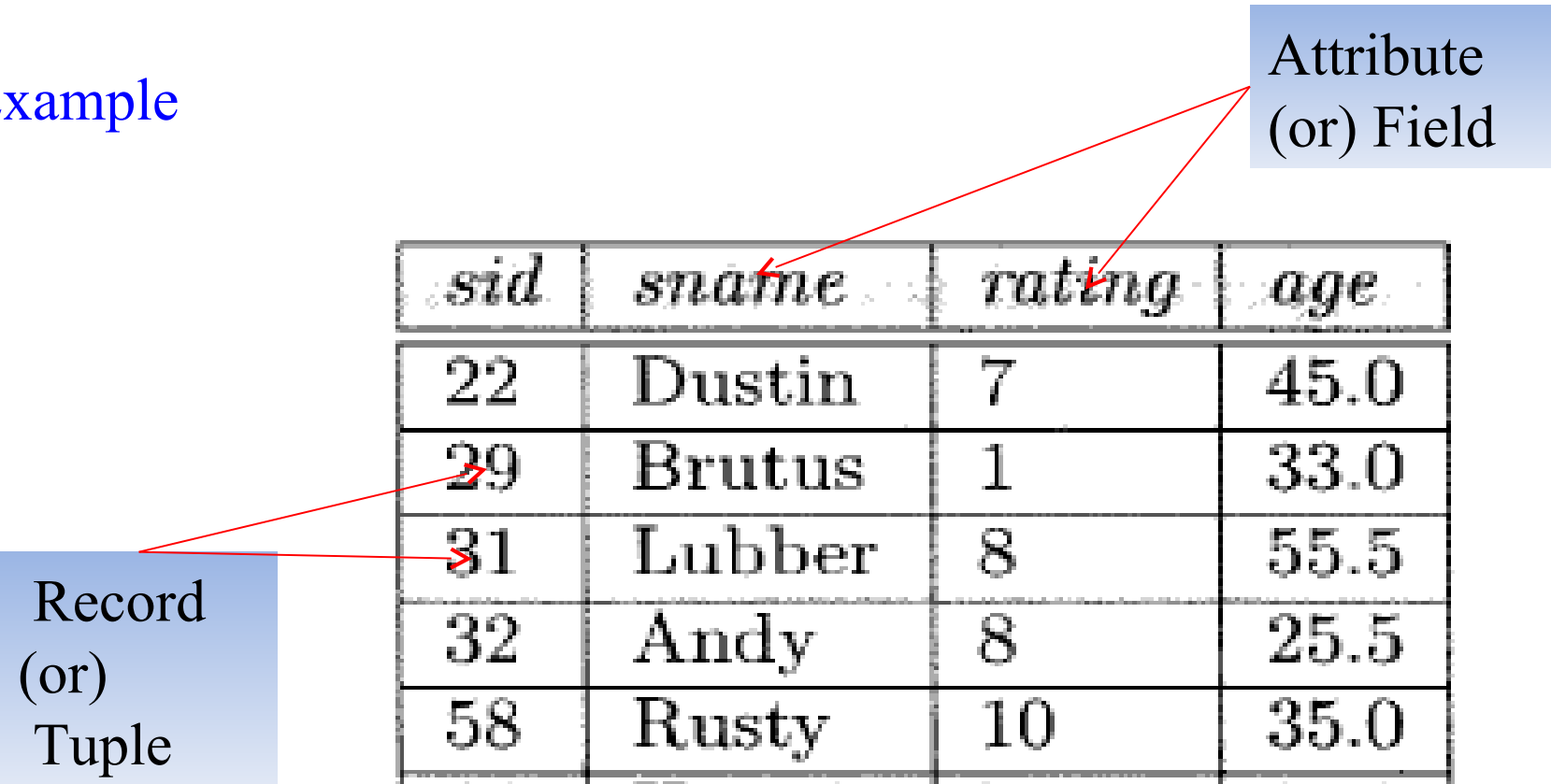
THE FORM OF A BASIC SQL QUERY

SELECT [DISTINCT] field names
FROM table names
WHERE condition

- SELECT clause contains fields to be displayed in the result
- FROM clause contains table names
- Optional WHERE clause contains conditions on the tables mentioned in the FROM clause

Example

Attribute
(or) Field



<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>
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

Record
(or)
Tuple

Fig 4.1 Sailors table

Question: Find the names and ages of all sailors

Query: SELECT DISTINCT sname, age
FROM Sailors

Output:

<i>sname</i>	<i>age</i>
Dustin	45.0
Brutus	33.0
Lubber	55.5
Andy	25.5
Rusty	35.0
Horatio	35.0
Zorba	16.0
Art	25.5
Bob	63.5

Note: The DISTINCT keyword can be used to return only distinct (different) values from the specific field

Question: Find the names and ages of all sailors

Query: SELECT sname, age
 FROM Sailors

Output:

<i>sname</i>	<i>age</i>
Dustin	45.0
Brutus	33.0
Lubber	55.5
Andy	25.5
Rusty	35.0
Horatio	35.0
Zorba	16.0
Horatio	35.0
Art	25.5
Bob	63.5

Question: Find all sailors with a rating above 7

Query: SELECT sid, sname, rating, age
FROM Sailors
WHERE rating > 7

(or)
SELECT *
FROM Sailors
WHERE rating > 7

Output:

31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0

AND, OR and NOT Operators

Question: Find the names of sailors who have reserved boat number 103

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

Output:

SNAME
Dustin
Lubber
Horatio

BETWEEN

- Used to define range limits

Question: Find all sailors whose age is in between 45.0 and 63.5

Query: SELECT *
FROM Sailors
WHERE age BETWEEN 45.0 AND 63.5

Output:

SID	SNAME	RATING	AGE
22	Dustin	7	45
31	Lubber	8	55.5
95	Bob	3	63.5

IN

- Used to check whether an attribute value matches a value contained within a set of listed values

Question: Find all sailors whose age is in the list of values(15.0,33.2,45.7,63.5)

Query: SELECT *
FROM Sailors
WHERE age IN (15.0,33.2,45.7,63.5)

Output:

SID	SNAME	RATING	AGE
95	Bob	3	63.5

STRING operators

- “%” character is used to match any substring
- “_” character is used to match any character
- Expresses patterns by using the ‘like’ comparison operator

Example1

```
SELECT *  
FROM Sailors  
WHERE sname LIKE '_u%'
```

Output:

SID	SNAME	RATING	AGE
22	Dustin	7	45
31	Lubber	8	55.5
58	Rusty	10	35

Example2

```
SELECT *  
FROM Sailors  
WHERE sname LIKE 'A_d_'
```

Output:

SID	SNAME	RATING	AGE
32	Andy	8	25.5

SET operators

- Operations such as *union*, *intersect*, *minus* and *exists* operate on relations
- Corresponding to relational-algebra operations \cup , \cap and $-$
- Relations participating in the operations must be **compatible**; i.e., must have same set of attributes

<query 1> <set operator> <query 2>

- ***union*** returns a table consisting of all rows either appearing in the result of <query1> or in the result of <query 2>

Example (union)

SELECT *
FROM Sailors

UNION

SELECT *
FROM Sailors1

Output:

<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>
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

Example (intersect)

```
SELECT *  
FROM Sailors  
INTERSECT  
SELECT *  
FROM Sailors1
```

Output:

<i>sid</i>	<i>sname</i>	<i>rating</i>	<i>age</i>
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

Example (minus)

```
SELECT *  
FROM Sailors
```

MINUS

```
SELECT *  
FROM Sailors1
```

Output: no rows selected

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 the nested queries, the inner subquery is completely independent of the outer query

Introduction to Nested Queries

Question: Find the names of sailors who have reserved boat 103

Query:

```
SELECT S.sname
```



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

Output:

SNAME
Dustin
Lubber
Horatio

Question: Find the names of sailors who have reserved a blue boat

Query:

```
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='blue'))
```

Output:

SNAME
Dustin
Horatio

Correlated Nested Queries

- In Correlated Nested Queries, inner subquery could depend on the row that is currently being examined in the outer query

Question: Find the names of sailors who have reserved boat 103

Query:

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

Output:

SNAME
Dustin
Lubber
Horatio

- The EXISTS operator is another set comparison operator, such as IN
- It allows us to **test whether a set is nonempty**. Thus, 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**

COMPARISON OPERATORS

- These operators can be used in 'WHERE' clause and 'HAVING' clause

SYMBOL	MEANING
=	Equal to
<	Less than
<=	Less than or equal to
>	Grater than
>=	Greater than or equal to
<> or != or ^=	Not equal to

Example

Question: Find sailors whose rating is better than some sailor called Horatio

Query:

```
SELECT S1.sname, S1.rating  
FROM Sailors S1  
WHERE S1.rating > ANY (SELECT S2.rating FROM  
Sailors S2 WHERE S2.sname='Horatio' )
```

Output:

SNAME	RATING
Rusty	10
Zorba	10
Horatio	9
Lubber	8
Andy	8

Question: Find sailors whose rating is better than some sailor called Horatio

Query:

```
SELECT S1.sname, S1.rating  
FROM Sailors S1  
WHERE S1.rating > ALL ( SELECT S2.rating FROM  
Sailors S2 WHERE S2.sname='Horatio' )
```

Output:

SNAME	RATING
Rusty	10
Zorba	10

AGGREGATE OPERATORS

- In addition to simply retrieving data, we often want to perform some computation or summarization

- SQL supports the following aggregate operators which can be applied on any column, say A, of a relation(table):

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

Note: not specify DISTINCT in conjunction with MIN or MAX

Examples:

Question: Find the average age of all sailors

Query:

```
SELECT AVG (age)
```

```
FROM Sailors
```

Output:

AVG(AGE)
36.9

Question: Find the name and age of the oldest sailor

Query:

```
SELECT S1.sname, S1.age  
FROM Sailors S1  
WHERE S1.age = ( SELECT MAX (S2.age) FROM Sailors  
S2 )
```

Output:

SNAME	AGE
Bob	63.5

Question: Count the number of sailors

Query:

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

Output:

COUNT(*)
10

The GROUP BY and HAVING Clauses

- We have applied aggregate operators to all (qualifying) rows in a relation(table)
- GROUP BY used to apply aggregate operators to each of a number of groups of rows in a relation
- HAVING is used to place a condition, which is applied on the groups of rows

general form:

SELECT [DISTINCT] fieldname

FROM table names

WHERE condition

GROUP BY fieldname

HAVING group-condition

Examples

Question: Find the number of sailors belongs to each rating level

Query:

```
SELECT rating, COUNT(rating)
FROM Sailors
GROUP BY rating
```

Output.

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

Question: Find the age of the youngest sailor for each rating level

Query:

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

Output:

RATING	MIN(AGE)
1	33
3	25.5
7	35
8	25.5
9	35
10	16

Question: Find the age of the youngest sailor for each rating level, which is greater than 7

Query:

```
SELECT rating, MIN(age)
FROM Sailors
GROUP BY rating
HAVING rating>7
```

Output:

RATING	MIN(AGE)
8	25.5
9	35
10	16

ORDER BY

- The order by clause is used to sort the tuples in a query result based on the values of some attributes

Example

Question: display the sailors table in the ascending order of sname

Query:

```
SELECT *  
FROM Sailors  
ORDER BY sname
```

Output:

SID	SNAME	RATING	AGE
32	Andy	8	25.5
85	Art	3	25.5
95	Bob	3	63.5
29	Brutus	1	33
22	Dustin	7	45
64	Horatio	7	35
74	Horatio	9	35
31	Lubber	8	55.5
58	Rusty	10	35
71	Zorba	10	16

Question: display the sailors table in the descending order of sname

Query:

```
SELECT *  
FROM Sailors  
ORDER BY sname DESC
```

Output:

SID	SNAME	RATING	AGE
71	Zorba	10	16
58	Rusty	10	35
31	Lubber	8	55.5
64	Horatio	7	35
74	Horatio	9	35
22	Dustin	7	45
29	Brutus	1	33
95	Bob	3	63.5
85	Art	3	25.5
32	Andy	8	25.5

NULL VALUES

- Thus far, we have assumed that column values in a row are always known. In practice column values can be unknown
- We use *null* when the column value is either unknown

Example

- Insert the row (98,Dan,null,39) to represent Dan into sailors table

Query: INSERT INTO Sailors VALUES(98,'Dan',null,39)

Query: SELECT *
FROM Sailors

Output:

SID	SNAME	RATING	AGE
22	Dustin	7	45
29	Brutus	1	33
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35
64	Horatio	7	35
71	Zorba	10	16
74	Horatio	9	35
85	Art	3	25.5
95	Bob	3	63.5
98	Dan		39

Comparisons Using Null Values

- Consider a comparison such as *rating* = 8
- *If this is applied to the row for Dan, is this condition true or false? Since Dan's rating is unknown, it is evaluated to the value unknown*
- This is the case for the comparisons *rating* > 8 and *rating* < 8 as well
- SQL also provides a special comparison operator IS NULL to test whether a column value is *null*

- for example, we can say rating IS NULL, which would evaluate to true on the row representing Dan
- We can also say rating IS NOT NULL, which would evaluate to false on the row for Dan

Example

Query:

```
SELECT *  
FROM sailors  
WHERE rating IS NULL
```

Output:

SID	SNAME	RATING	AGE
98	Dan		39

Logical Connectives AND, OR, and NOT

- what about Boolean expressions such as

rating = 8 OR age < 40

rating = 8 AND age < 40?

- *Considering the row for Dan again, because age < 40, the first expression evaluates to true regardless of the value of rating, but what about the second? We can only say 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*

Impact on SQL Constructs

- In the presence of *null values*, any row that evaluates to *false* or to *unknown* is eliminated

- If we compare two *null* values using $=$, the result is unknown! In the context of duplicates, this comparison is implicitly treated as true, which is an anomaly
- The arithmetic operations $+$, $-$, $*$, $/$ and $=$ all return *null* if one of their arguments is null

Example

Query:

```
SELECT sid, rating, sid+rating  
FROM Sailors
```

Output:

SID	RATING	SID + RATING
22	7	29
29	1	30
31	8	39
32	8	40
58	10	68
64	7	71
71	10	81
74	9	83
85	3	88
95	3	98
98	-	-

- nulls can cause some unexpected behavior with aggregate operators
- COUNT(*) handles *null values just like other values*, that is, they get counted

Example

Query:

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

Output:

COUNT(*)
11

- All the other aggregate operators (COUNT, SUM, AVG, MIN, MAX, and variations using DISTINCT) simply discard *null values*

Outer Joins

- join operation that rely on *null values*, called **outer joins**
- Consider the join of two tables, say Sailors \bowtie_c Reserves
- In a *full outer join*, ‘matching rows’ plus ‘Sailors rows without a matching Reserves rows’ (columns inherited from Reserves assigned *null* values) plus ‘Reserves rows without a matching Sailors rows’ (columns inherited from Sailors assigned *null* values) appear in the result

- In a *left outer join*, ‘matching rows’ plus ‘Sailors rows without a matching Reserves rows’ (columns inherited from Reserves assigned *null* values) appear in the result
- In a *right outer join*, ‘matching rows’ plus ‘Reserves rows without a matching Sailors rows’ (columns inherited from Sailors assigned *null* values) appear in the result
- *Note:* In *inner join* only matching rows appear in the result

Example

Query:

```
SELECT S.sid,S.sname,R.bid,R.day  
FROM Sailors S LEFT OUTER JOIN Reserves R ON  
S.sid=R.sid
```

Output:

SID	SNAME	BID	DAY
22	Dustin	101	10-OCT-98
22	Dustin	102	10-OCT-98
22	Dustin	103	10-AUG-98
22	Dustin	104	10-JUL-98
31	Lubber	102	11-OCT-98
31	Lubber	103	11-JUN-98
31	Lubber	104	11-DEC-98
64	Horatio	101	09-MAY-98
64	Horatio	102	09-AUG-98
74	Horatio	103	09-AUG-98
71	Zorba	-	-
85	Art	-	-
58	Rusty	-	-
32	Andy	-	-
29	Brutus	-	-
95	Bob	-	-

Disallowing Null Values

- We can disallow *null* values by specifying NOT NULL as part of the field definition, for example,

sname VARCHAR2(20) NOT NULL

- The fields in a primary key are not allowed to take on *null* values
- There is an implicit NOT NULL constraint for every field listed in a PRIMARY KEY constraint