Keys, SQL, and Views

CMPSCI 645
SQL Overview

- SQL Preliminaries
- Integrity constraints
- Query capabilities
  - SELECT-FROM-WHERE blocks,
  - Basic features, ordering, duplicates
  - Set ops (union, intersect, except)
  - Aggregation & Grouping
- Nested queries (correlation)
- Null values
- Modifying the database
- Views

Review in the textbook, Ch 5
The SQL Query Language

• Developed by IBM (system R) in the 1970s
• Need for a standard since it is used by many vendors
• Evolving standard
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision)
  - SQL-99 (major extensions)
  - SQL-2003 (minor revisions) ...
Two parts of SQL

• Data Definition Language (DDL)
  – Create/alter/delete tables and their attributes
  – establish and modify schema

• Data Manipulation Language (DML)
  – Query and modify database instance
Creating Relations in SQL

• Creates the **Student** relation. Observe that the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

• As another example, the **Takes** table holds information about courses that students take.

```sql
CREATE TABLE Student
(sid CHAR(20),
 name CHAR(20),
 login CHAR(10),
 age INTEGER,
 gpa REAL)

CREATE TABLE Takes
(sid CHAR(20),
 cid CHAR(20),
 grade CHAR(2))
```
Data Types in SQL

- **Characters:**
  - CHAR(20) -- fixed length
  - VARCHAR(40) -- variable length

- **Numbers:**
  - BIGINT, INT, SMALLINT, TINYINT
  - REAL, FLOAT -- differ in precision
  - MONEY

- **Times and dates:**
  - DATE
  - DATETIME

- **Others...**
Destroying and Altering Relations

**DROP TABLE** Student

- Destroys the relation Student. The schema information *and* the tuples are deleted.

**ALTER TABLE** Student

  **ADD COLUMN** firstYear integer

- The schema of Student is altered by adding a new field; every tuple in the current instance is extended with a *null* value in the new field.
Integrity Constraints (ICs)

- **IC**: condition that must be true for *any* instance of the database.
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.

- A *legal* instance of a relation is one that satisfies all specified ICs.
  - DBMS should only allow legal instances.

- If the DBMS checks ICs, stored data is more faithful to real-world meaning.
  - Avoids data entry errors, too!
Key Constraints

“A key constraint is a statement that a certain \textit{minimal} subset of the fields of a relation is a unique identifier for a tuple.”

Definition:

• A set of fields is a \textit{key} for a relation if:
  1. No two distinct tuples can have same values in all key fields, and...
  2. This is not true for any subset of the key.
     - If part 2 false: then fields are a \textit{superkey}.
     - If there’s more than one key for a relation, one of the keys is chosen (by DBA) to be the \textit{primary key}.

• E.g., \textit{sid} is a key for Students. (What about \textit{name}?)
  The set \{ \textit{sid}, \textit{gpa} \} is a superkey.
## Student table

### STUDENT

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>50000</td>
<td>Dave</td>
<td>dave@cs</td>
<td>19</td>
<td>3.2</td>
</tr>
<tr>
<td>53666</td>
<td>Jones</td>
<td>jones@cs</td>
<td>18</td>
<td>3.3</td>
</tr>
<tr>
<td>53688</td>
<td>Smith</td>
<td>smith@ee</td>
<td>18</td>
<td>3.2</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
<td>19</td>
<td>3.7</td>
</tr>
<tr>
<td>53831</td>
<td>Madayan</td>
<td>madayan@music</td>
<td>11</td>
<td>1.8</td>
</tr>
<tr>
<td>53832</td>
<td>Guldu</td>
<td>guldu@music</td>
<td>12</td>
<td>2.0</td>
</tr>
</tbody>
</table>
Specifying Key Constraints in SQL

```sql
CREATE TABLE Student
    (sid CHAR(20),
     name CHAR(20),
     login CHAR(10),
     age INTEGER,
     gpa REAL,
     UNIQUE (name, age),
     PRIMARY KEY (sid) )
```
Primary and Candidate Keys in SQL

- Possibly many candidate keys (specified using UNIQUE), one of which is chosen as the primary key.

- “For a given student and course, there is a single grade.” vs. “Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade.”

- Used carelessly, an IC can prevent the storage of database instances that arise in practice!

```
CREATE TABLE Takes
  (sid CHAR(20)
  cid CHAR(20),
  grade CHAR(2),
  PRIMARY KEY (sid,cid) )
```

```
CREATE TABLE Takes
  (sid CHAR(20)
  cid CHAR(20),
  grade CHAR(2),
  PRIMARY KEY (sid),
  UNIQUE (cid, grade) )
```
Foreign Keys, Referential Integrity

• **Foreign key**: Set of fields in one relation that is used to `refer` to a tuple in another relation. (Must correspond to primary key of the second relation.) Like a `logical pointer`.

• E.g. *sid* is a foreign key referring to *Students*:
  - **Takes**(*sid*: string, *cid*: string, *grade*: string)
  - If all foreign key constraints are enforced, *referential integrity* is achieved, i.e., no dangling references.
  - Can you think of a data model without referential integrity?
Foreign Keys in SQL

- Only students listed in the Students relation should be allowed to enroll for courses.

```sql
CREATE TABLE Takes
    (sid CHAR(20), cid CHAR(20), grade CHAR(2),
     PRIMARY KEY (sid,cid),
     FOREIGN KEY (sid) REFERENCES Students )
```

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>50000</td>
<td>445</td>
<td>A</td>
</tr>
<tr>
<td>53688</td>
<td>483</td>
<td>C</td>
</tr>
<tr>
<td>53666</td>
<td>435</td>
<td>B</td>
</tr>
</tbody>
</table>

**STUDENT**

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
</tr>
</thead>
<tbody>
<tr>
<td>50000</td>
<td>Dave</td>
<td>dave@cs</td>
</tr>
<tr>
<td>53688</td>
<td>Jones</td>
<td>jones@cs</td>
</tr>
<tr>
<td>53666</td>
<td>Smith</td>
<td>smith@ee</td>
</tr>
<tr>
<td>53650</td>
<td>Smith</td>
<td>smith@math</td>
</tr>
</tbody>
</table>
Enforcing Referential Integrity

- Consider **Student** and **Takes**; \( sid \) in **Takes** is a foreign key that references **Student**.
- What should be done if a **Takes** tuple with a non-existent student id is inserted? *(Reject it!)*
- What should be done if a **Student** tuple is deleted?
  - Also delete all Takes tuples that refer to it.
  - Disallow deletion of a Students tuple that is referred to.
  - Set \( sid \) in Takes tuples that refer to it to a *default *\( sid \).
  - *(In SQL, also: Set \( sid \) in Takes tuples that refer to it to a special value *null*, denoting *unknown* or *inapplicable*.)
- Similar if primary key of Students tuple is updated.
Referential Integrity 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 deleted tuple)
  - SET NULL / SET DEFAULT (sets foreign key value of referencing tuple)

CREATE TABLE Takes
(sid CHAR(20),
cid CHAR(20),
grade CHAR(2),
PRIMARY KEY (sid,cid),
FOREIGN KEY (sid)
  REFERENCES Students
    ON DELETE CASCADE
    ON UPDATE SET DEFAULT )
Where do ICs Come From?

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.
- We can check a database instance to see if an IC is violated, but we can **NEVER** infer that an IC is true by looking at an instance.
  - An IC is a statement about *all possible* instances!
  - From example, we know *name* is not a key, but the assertion that *sid* is a key is given to us.
- Key and foreign key ICs are the most common; more general ICs supported too.
SQL Overview

• Query capabilities
  – SELECT-FROM-WHERE blocks,
  – Basic features, ordering, duplicates
  – Set operations (union, intersect, except)
  – Aggregation & Grouping
  – Nested queries (correlation)
  – Null values
Example database

Sailors (sid, sname, rating, age)
Boats (bid, bname, color)
Reserves (sid, bid, day)

Key for each table indicated by underlined attributes.
### Sailors

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>brutus</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>85</td>
<td>art</td>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>95</td>
<td>bob</td>
<td>3</td>
<td>63.5</td>
</tr>
<tr>
<td>96</td>
<td>frodo</td>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>64</td>
<td>horatio</td>
<td>7</td>
<td>35</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>32</td>
<td>andy</td>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>74</td>
<td>horatio</td>
<td>9</td>
<td>35</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
<tr>
<td>71</td>
<td>zorba</td>
<td>10</td>
<td>16</td>
</tr>
</tbody>
</table>

### Reserves

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
<td>10/10</td>
</tr>
<tr>
<td>22</td>
<td>103</td>
<td>10/8</td>
</tr>
<tr>
<td>22</td>
<td>104</td>
<td>10/7</td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td>11/10</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>11/6</td>
</tr>
<tr>
<td>31</td>
<td>104</td>
<td>11/12</td>
</tr>
<tr>
<td>64</td>
<td>101</td>
<td>9/5</td>
</tr>
<tr>
<td>64</td>
<td>102</td>
<td>9/8</td>
</tr>
<tr>
<td>74</td>
<td>103</td>
<td>9/8</td>
</tr>
</tbody>
</table>

### Boats

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Interlake</td>
<td>blue</td>
</tr>
<tr>
<td>102</td>
<td>Interlake</td>
<td>red</td>
</tr>
<tr>
<td>103</td>
<td>Clipper</td>
<td>green</td>
</tr>
<tr>
<td>104</td>
<td>Marine</td>
<td>red</td>
</tr>
</tbody>
</table>
SQL Query

Basic form: (plus many many extensions)

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

For example:

```
SELECT sid, sname, rating, age
FROM Sailors
WHERE age > 21
```
Basic SQL Query

- **target-list** A list of attributes of relations in *relation-list*
- **relation-list** A list of relation names (possibly with a *range-variable* after each name).
- **qualification** Comparisons (Attr *op* const or Attr1 *op* Attr2, where *op* is one of <, >, =, ≤, ≥, ≠) 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!
Simple SQL Query

```
SELECT * FROM Sailors WHERE age > 21
```

Conditions in the WHERE clause are like selection: $\sigma_{\text{age} < 21}$
Selection conditions

What goes in the **WHERE** clause:

- $x = y$, $x < y$, $x \leq y$, $x \neq y$, etc
  - For number, they have the usual meanings
  - For CHAR and VARCHAR: lexicographic ordering
  - For dates and times, what you expect...

- Also, pattern matching on strings: $s \text{ LIKE } p$
The **LIKE** operator

- **s LIKE p**: pattern matching on strings
- **p** may contain two special symbols:
  - `%` = any sequence of characters
  - `_` = any single character

Find all students whose name begins and ends with ‘b’:

```sql
SELECT * FROM Sailors WHERE sname LIKE 'b%b'
```

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>brutus</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>85</td>
<td>art</td>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>95</td>
<td>bob</td>
<td>3</td>
<td>63.5</td>
</tr>
<tr>
<td>96</td>
<td>frodo</td>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>64</td>
<td>horatio</td>
<td>7</td>
<td>35</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
</tbody>
</table>
Simple SQL Query

```
SELECT sname, age
FROM Sailors
WHERE age > 21
```

Conditions in the SELECT clause are like projection: $\Pi_{sname,age}$
Note confusing terminology 😞

Conditions in the WHERE clause are like selection: $\sigma_{age<21}$

Conditions in the SELECT clause are like projection: $\Pi_{\text{sname, age}}$

```
SELECT sname, age
FROM Sailors
WHERE age > 21
```
Eliminating Duplicates

\[
\text{SELECT DISTINCT sname FROM Sailors}
\]

Compare to:

\[
\text{SELECT sname FROM Sailors}
\]

Default behavior does not eliminate duplicates.
Ordering the Results

```sql
SELECT  sname, rating, age
FROM    Sailors
WHERE   age > 18
ORDER BY rating, sname
```

Ordering is ascending, unless you specify the DESC keyword.

Ties are broken by the second attribute on the ORDER BY list, etc.
Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of a 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.

- Probably the least efficient way to compute a query -- optimizer will find more efficient plan.
Example of Conceptual Evaluation

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

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(sid)</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>(sid)</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
<td>58</td>
<td>103</td>
<td>11/12/96</td>
</tr>
</tbody>
</table>
Example

SELECT sname
FROM Sailors S, Reserves R, Boats B
    AND B.color = 'red'

What does this query compute?

*Find the names of sailors who have reserved a red boat*
Please write in SQL

*Find the colors of boats reserved by ‘Lubber’*

```
SELECT B.color
FROM Sailors S, Reserves R, Boats B
  AND S.sname = ‘Lubber’
```
Renaming Columns

SELECT `sname` AS name, `age` AS `x` FROM `Sailors` WHERE `age` > 21

<table>
<thead>
<tr>
<th>sid</th>
<th><code>sname</code></th>
<th>rating</th>
<th><code>age</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
<tr>
<td>71</td>
<td>zorba</td>
<td>10</td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th><code>x</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>dustin</td>
<td>45</td>
</tr>
<tr>
<td>lubber</td>
<td>55.5</td>
</tr>
<tr>
<td>rusty</td>
<td>35</td>
</tr>
</tbody>
</table>
Disambiguating Attributes

- Sometimes two relations have the same attr:
  Person(pname, address, worksfor)
  Company(cname, address)

```
SELECT DISTINCT pname, address
FROM Person, Company
WHERE worksfor = cname
```

Which address?
Range Variables in SQL

Find all stores that sold at least one product that was sold at ‘BestBuy’:

```
SELECT DISTINCT x.store
FROM Purchase AS x, Purchase AS y
WHERE x.product = y.product AND y.store = 'BestBuy'
```
Self-join on Flights:
The departure and arrival cities of trips consisting of two direct flights.

SELECT F1.depart, F2.arrive
FROM Flights as F1, Flights as F2
WHERE F1.arrive = F2.depart
SQL Overview

• Query capabilities
  – SELECT-FROM-WHERE blocks,
  – Basic features, ordering, duplicates
  – Set operations (union, intersect, except)
  – Aggregation & Grouping
  – Nested queries (correlation)
  – Null values
Set operations

- UNION
- INTERSECTION
- EXCEPT (sometimes called MINUS)

- Recall: schemas must match for these operations.
UNION example

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

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

UNION

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

• Duplicates ARE NOT eliminated by default in basic SELECT-FROM-WHERE queries
• Duplicate ARE eliminated by default for UNION queries.
• To preserve duplicates in UNION, you must use UNION ALL
Find the names of sailors who have reserved a red or a green boat.

```
SELECT DISTINCT sname
FROM Sailors S, Reserves R, Boats B
  AND (B.color = 'red' OR B.color = 'green')
```
Find the names of sailors who have reserved a red or a green boat.

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

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

SELECT sname
FROM Sailors S, Reserves R, Boats B
AND (B.color = 'red' AND B.color = 'green')

This doesn’t work! What does this query return?
Find the names of sailors who have reserved a red and a green boat.

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

• Query capabilities
  – SELECT-FROM-WHERE blocks,
  – Basic features, ordering, duplicates
  – Set ops (union, intersect, except)
  – Aggregation & Grouping
Aggregation

\[
\text{SELECT} \quad \text{Avg}(S.\text{age}) \\
\text{FROM} \quad \text{Sailors} \\
\text{WHERE} \quad S.\text{rating} = 10
\]

SQL supports several aggregation operations:

- \text{COUNT} (*)
- \text{COUNT} ([\text{DISTINCT}] A)
- \text{SUM} ([\text{DISTINCT}] A)
- \text{AVG} ([\text{DISTINCT}] A)
- \text{MAX} (A)
- \text{MIN} (A)
Aggregation: Count

SELECT  Count(*)
FROM     Sailors
WHERE   rating > 5

Except for COUNT, all aggregations apply to a single attribute.
Aggregation: Count

COUNT applies to duplicates, unless otherwise stated:

```sql
SELECT Count(category)
FROM Product
WHERE year > 1995
```

Better:

```sql
SELECT Count(DISTINCT category)
FROM Product
WHERE year > 1995
```
Simple Aggregation

Purchase(product, date, price, quantity)

Example 1: find total sales for the entire database

```
SELECT Sum(price * quantity)
FROM Purchase
```

Example 1’: find total sales of bagels

```
SELECT Sum(price * quantity)
FROM Purchase
WHERE product = ‘bagel’
```
GROUP BY and HAVING clauses

- We often want to apply aggregates to each of a number of groups of rows in a relation.

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

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

For i = 1, 2, ... 10
```
### Sailors

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>brutus</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>85</td>
<td>art</td>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>95</td>
<td>bob</td>
<td>3</td>
<td>63.5</td>
</tr>
<tr>
<td>96</td>
<td>frodo</td>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
<td>7</td>
<td>45</td>
</tr>
<tr>
<td>64</td>
<td>horatio</td>
<td>7</td>
<td>35</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>32</td>
<td>andy</td>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>74</td>
<td>horatio</td>
<td>9</td>
<td>35</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35</td>
</tr>
<tr>
<td>71</td>
<td>zorba</td>
<td>10</td>
<td>16</td>
</tr>
</tbody>
</table>

The SQL query is:

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

### New Table

<table>
<thead>
<tr>
<th>rating</th>
<th>age?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>