SQL

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

Structured 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 set of fields is a 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 superkey.
     - If there’s more than one key for a relation, one of the keys is chosen (by DBA) to be the primary key.

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

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!

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

```sql
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 name a data model w/o referential integrity?
    - Links in HTML!
Foreign Keys in SQL

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

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

<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>

<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

<table>
<thead>
<tr>
<th>SELECT</th>
<th>[DISTINCT] target-list</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROM</td>
<td>relation-list</td>
</tr>
<tr>
<td>WHERE</td>
<td>qualification</td>
</tr>
</tbody>
</table>

- **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

<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</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>

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: $s$ **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’:

```
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>
</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: $\Pi_{sname, age}$
Eliminating Duplicates

```sql
SELECT DISTINCT sname
FROM Sailors
```

Compare to:

```sql
SELECT sname
FROM Sailors
```

```
<table>
<thead>
<tr>
<th>sname</th>
</tr>
</thead>
<tbody>
<tr>
<td>brutus</td>
</tr>
<tr>
<td>art</td>
</tr>
<tr>
<td>bob</td>
</tr>
<tr>
<td>frodo</td>
</tr>
<tr>
<td>dustin</td>
</tr>
<tr>
<td>horatio</td>
</tr>
<tr>
<td>lubber</td>
</tr>
<tr>
<td>andy</td>
</tr>
</tbody>
</table>
```

Default behavior does not eliminate duplicates.
Ordering the Results

```
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.

\[
\text{SELECT} \quad \text{[DISTINCT]} \quad \text{target-list}
\text{FROM} \quad \text{relation-list}
\text{WHERE} \quad \text{qualifications}
\]

RA equiv:
\[
\sigma \quad \Pi
\]
Example of Conceptual Evaluation

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid=R.sid AND R.bid=103
```
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*
Find the colors of boats reserved by ‘Lubber’

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

Sailors

<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</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>

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

<table>
<thead>
<tr>
<th>name</th>
<th>x</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

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

Which address?
Range Variables in SQL

Purchase (buyer, seller, store, product)

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'
```
Please write in SQL

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
```

<table>
<thead>
<tr>
<th>depart</th>
<th>arrive</th>
</tr>
</thead>
<tbody>
<tr>
<td>NYC</td>
<td>Reno</td>
</tr>
<tr>
<td>NYC</td>
<td>Oakland</td>
</tr>
<tr>
<td>Boston</td>
<td>Tampa</td>
</tr>
<tr>
<td>Oakland</td>
<td>Boston</td>
</tr>
<tr>
<td>Tampa</td>
<td>NYC</td>
</tr>
</tbody>
</table>
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
UNION example, alternative:

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.

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' \textbf{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

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

SQL supports several aggregation operations:

- `COUNT (*)`
- `COUNT ( [DISTINCT] A)`
- `SUM ( [DISTINCT] A)`
- `AVG ( [DISTINCT] A)`
- `MAX (A)`
- `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

\[
\text{SELECT } \text{Sum(price} \times \text{quantity}) \text{ FROM Purchase}
\]

Example 1’: find total sales of bagels

\[
\text{SELECT } \text{Sum(price} \times \text{quantity}) \text{ FROM Purchase WHERE product = ‘bagel’}
\]
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
## Grouping

### 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>

### 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>

SQL:
```
SELECT S.rating, MIN(S.age)
FROM  Sailors S
GROUP BY S.rating
```
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.
Conceptual Evaluation

- The cross-product of \textit{relation-list} is computed, tuples that fail \textit{qualification} are discarded, `unnecessary’ fields are deleted, and the remaining tuples are partitioned into groups by the value of attributes in \textit{grouping-list}.
- The \textit{group-qualification} is then applied to eliminate some groups. Expressions in \textit{group-qualification} must have a \textit{single value per group}!
- One answer tuple is generated per qualifying group.
Find age of the youngest sailor with age $\geq 18$, 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
```

**Answer relation:**

<table>
<thead>
<tr>
<th>rating</th>
<th>minage</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>8</td>
<td>25.5</td>
</tr>
</tbody>
</table>

**Sailors instance:**

<table>
<thead>
<tr>
<th>sid</th>
<th>surname</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>29</td>
<td>brutus</td>
<td>1</td>
<td>33.0</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>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>64</td>
<td>horatio</td>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>71</td>
<td>zorba</td>
<td>10</td>
<td>16.0</td>
</tr>
<tr>
<td>74</td>
<td>horatio</td>
<td>9</td>
<td>35.0</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>
</tbody>
</table>