Intimate Computing

- Can digital technology express intimacy?
  - as something that relates to our innermost selves, something personal, closely felt
- Hug-T-Shirt (cute-circuit)
- Emotional Décor (NYU)
- Back-to-back chair

RU's crying surrogate
Course Administration

• Assignment #2 is out on the course homepage.
  – It is due 10/29/2007 (next Monday).
• Assignment #1 solution will be on the course homepage.
• Next week reading:
  – Chapter 8: Overview of Storage and Indexing

Long Reflection: DB design

• Step 1: Requirements Analysis
  – What data to store in the database?
• Step 2: Conceptual Database Design
  – Come up with the design: Entity-Relation (ER) model
  – Sketch the design with ER diagrams
• Step 3: Logical Database Design
  – Implement the design: relational data model
  – Map ER diagrams to relational tables
Recent Reflection: DB design

• Last lecture:
  – Query language: how to ask questions about the [relational] database?
  – Mathematical query language: Relational Algebra
• This lecture
  – A real query language: SQL (Structured Query Language)

Review: Relational Algebra

• A query is applied to table(s), and the result of a query is also a table.
• Find the names of sailors who have reserved boat 103

\[ \Pi_{\text{sname}}(\sigma_{\text{bid} = 103} \text{Reserves}) \bowtie \text{Sailors} \]
Example Table Definitions

\[ \text{Sailors}(\text{sid}: \text{integer}, \text{sname}: \text{string}, \text{rating}: \text{integer}, \text{age}: \text{real}) \]
\[ \text{Boats}(\text{bid}: \text{integer}, \text{bname}: \text{string}, \text{color}: \text{string}) \]
\[ \text{Reserves}(\text{sid}: \text{integer}, \text{bid}: \text{integer}, \text{day}: \text{date}) \]

Review: Relational Algebra

- Basic relational algebra operators:
  - Selection (\( \sigma \), pronounced sigma): Select a subset of rows from a table.
  - Projection (\( \pi \)): Delete unwanted columns from a table.
  - Cross-product (\( X \)): Combine two tables.
  - Set-difference (\( - \)): Tuples in table 1, but not in table 2.
  - Union (\( U \)): Tuples in tables 1 or 2.
Review: Relational Algebra (more)

- Additional relational algebra operators:
  - Intersection (∩): Tuples in tables 1 and 2.
  - Join (∞): conditional cross product
  - Division (⁄)
  - Renaming (ρ)
- Operations composed into complex query expression
- Query in English?

\[ \pi_{\text{sid}} (\sigma_{\text{age} > 20} \text{Sailors}) - \pi_{\text{sid}} ((\sigma_{\text{color} = 'red'} \text{Boats}) \bowtie \text{Reserves} \bowtie \text{Sailors}) \]

Relational Algebra to SQL

- Relational operators → SQL commands

Relational Algebra:
\[ \pi_{\text{sname}} (\sigma_{\text{bid} = 103} (\text{Sailors} \bowtie \text{Reserves})) \]

SQL:
\[ \text{SELECT S.sname} \]
\[ \text{FROM Sailors S, Reserves R} \]
\[ \text{WHERE S.sid=R.sid AND R.bid=103} \]

- Guess the mapping?
  - Notice the difference between SELECT (SQL) and \( \sigma \)
Lecture Outline

• Basic Query
  – `SELECT`
• Set Constructs
  – `UNION, INTERSECT, EXCEPT, IN, ANY, ALL, EXISTS`
• Nested Queries
• Aggregate Operators
  – `COUNT, SUM, AVG, MAX, MIN, GROUP BY, HAVING`

• Null Values
• Integrity Constraints
  – `CHECK, CREATE ASSERTION`
• Triggers
  – `CREATE TRIGGER, FOR EACH ROW`
Example Table Definitions

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

• Find names of sailors who’ve reserved boat #103

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

Basic SQL Query

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

• Relation-list: A list of relation names (possibly with range-variable after each name).
• Target-list: A list of attributes of relations in relation-list
• Qualification: conditions on attributes (<, >, =, and, or, not, etc.)
• DISTINCT: optional keyword for duplicate removal.
  – Default = no duplicate removal!
How to evaluate a query?

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

- Conceptual query evaluation using relational operators:
  1) Compute the cross-product of relation-list.
  2) Discard resulting tuples if they fail qualifications.
  3) Delete attributes that are not in target-list. (called column-list)
  4) If DISTINCT is specified, eliminate duplicate rows.
- Only conceptual because of inefficiency computation
  - An optimizer can find better strategy

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

Example of Conceptual Evaluation (1)

(1) Compute the cross-product of relation-list.

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Reserves</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>bid</td>
</tr>
<tr>
<td>22</td>
<td>101</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
</tr>
<tr>
<td>58</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>103</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Reserves</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>name</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
</tr>
</tbody>
</table>
Example of Conceptual Evaluation (2)

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

(2) Discard tuples if they fail qualifications.

<table>
<thead>
<tr>
<th>S.sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>R.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 of Conceptual Evaluation (3)

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

(3) Delete attribute columns that not in target-list.

<table>
<thead>
<tr>
<th>(S.id)</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
<th>(R.id)</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>
A Note on Range Variables

• Really needed range variables only if the same relation appears twice in the FROM clause.

SQL Code:

```sql
SELECT S.sname
FROM   Sailors AS 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

SELECT sname
FROM   Sailors S, Reserves R1, Reserves R2
WHERE  S.sid = R1.sid
AND     S.sid = R2.sid
AND     R1.bid <> R2.bid
```

Find the sids of sailors who’ve reserved at least one boat

```sql
SELECT S.sid
FROM   Sailors S, Reserves R
WHERE  S.sid = R.sid
```

<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>
### DISTINCT

- Find the names and ages of all sailors
  
  ```
  SELECT S.sname, S.age 
  FROM Sailors S 
  ```

- Add `DISTINCT` to this query?

- Replace `S.sname` by `S.sid` in the `SELECT` clause?

- Add `DISTINCT` to the above?

<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>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>
</tbody>
</table>

### Find sailors whose names begin and end with B and contain at least three characters.

```sql
SELECT S.age, 
    age1=S.age-5, 
    2*S.age AS age2 
FROM Sailors S 
WHERE S.sname LIKE 'B_%B';
```
Find sid’s of sailors who’ve reserved a red or a green boats.

\[
\text{SELECT } S.\text{sid} \\
\text{FROM } S, B, R \\
\text{WHERE } S.\text{sid}=R.\text{sid} \text{ AND } R.\text{bid}=B.\text{bid} \text{ AND } B.\text{color}=\text{'red'} 
\]

\[
\text{SELECT } S.\text{sid} \\
\text{FROM } S, B, R \\
\text{WHERE } S.\text{sid}=R.\text{sid} \text{ AND } R.\text{bid}=B.\text{bid} \text{ AND } B.\text{color}=\text{'green'} 
\]

- **UNION**: work on two union-compatible sets of tuples

\[
\begin{align*}
\text{SELECT } & S.\text{sid} \\
\text{FROM } & S, B, R \\
\text{WHERE } & S.\text{sid}=R.\text{sid} \text{ AND } R.\text{bid}=B.\text{bid} \text{ AND } B.\text{color}=\text{'red'} \\
\text{UNION} & \\
\text{SELECT } & S.\text{sid} \\
\text{FROM } & S, B, R \\
\text{WHERE } & S.\text{sid}=R.\text{sid} \text{ AND } R.\text{bid}=B.\text{bid} \text{ AND } B.\text{color}=\text{'green'}
\end{align*}
\]

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

\[
\text{SELECT } S.\text{sid} \\
\text{FROM } S, B, R \\
\text{WHERE } S.\text{sid}=R.\text{sid} \text{ AND } R.\text{bid}=B.\text{bid} \text{ AND } B.\text{color}=\text{'red'} \\
\text{INTERSECT} \\
\text{SELECT } S.\text{sid} \\
\text{FROM } S, B, R \\
\text{WHERE } S.\text{sid}=R.\text{sid} \text{ AND } R.\text{bid}=B.\text{bid} \text{ AND } B.\text{color}=\text{'green'}
\]

- (A Except B) returns tuples in A but not in B.
- What is the query in English if we replace INTERSECT by EXCEPT?
  - Find sids of all sailors who have reserved a red boat but not a green boat.
SET Construct: **UNION ALL**

- **UNION, INTERSECT, and** **EXCEPT** delete duplicate by default.
- To retain duplicates, use **UNION ALL, INTERSECT ALL, or** **EXCEPT ALL.**

```
Sid  Sname
71   Zorba
74   Horatio
74   Horatio
95   Bob
```

```
Sid  Sname
22   Dustin
```

```
Sid  Sname
71   Zorba
74   Horatio
```

```
Sid  Sname
74   Horatio
```

**Nested Queries**

- WHERE clause can contain an SQL subquery.
  - (Actually, so can FROM and HAVING clauses.)
- 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)
  ```
  ```
  Subquery: finds sids who have reserved bid 103
  ```
- \((x \text{ IN } B)\) returns true when \(x\) is in set \(B\).
  - To find sailors who’ve not reserved #103, use **NOT IN.**
- Nested loops Evaluation
  - For each Sailors tuple, check the qualification by computing the subquery.
  - Does the subquery result change for each Sailor row?
    - When would subquery result change for each Sailor row?
Nested Queries with Correlation

\[
\begin{align*}
\text{SELECT} & \quad S.sname \\
\text{FROM} & \quad \text{Sailors } S \\
\text{WHERE} & \quad \text{EXISTS} \left( \text{SELECT} \ast \\
& \quad \text{FROM} \quad \text{Reserves } R \\
& \quad \text{WHERE} \quad R.bid=103 \text{ AND } S.sid=R.sid \right)
\end{align*}
\]

- `EXISTS` is another set operator, like `IN`.
  - `(EXISTS S)` returns true when `S` is not empty.

- What is the above query in English?
  - Find sailors who have reserved boat #103

- In case of correlation, subquery must be re-computed for each Sailors tuple.

---

Nested Queries with `UNIQUE`

\[
\begin{align*}
\text{Sailors}(\text{sid}: \text{integer}, \text{sname}: \text{string}, \text{rating}: \text{integer}, \text{age}: \text{real}) \\
\text{Boats}(\text{bid}: \text{integer}, \text{bname}: \text{string}, \text{color}: \text{string}) \\
\text{Reserves}(\text{sid}: \text{integer}, \text{bid}: \text{integer}, \text{day}: \text{date})
\end{align*}
\]

- `(UNIQUE S)` returns true if `S` has no duplicate tuples or `S` is empty.

\[
\begin{align*}
\text{SELECT} & \quad S.sname \\
\text{FROM} & \quad \text{Sailors } S \\
\text{WHERE} & \quad \text{UNIQUE} \left( \text{SELECT} \ast \\
& \quad \text{FROM} \quad \text{Reserves } R \\
& \quad \text{WHERE} \quad R.bid=103 \text{ AND } S.sid=R.sid \right)
\end{align*}
\]

- What is the above query in English?
  - Finds sailors with at most one reservation for boat #103.

- Replace `R.bid` with `*`?
  - Finds sailors with at most one reservation for boat #103 in a given `day`.
More on Set-Comparison Operators

• Have seen IN, EXISTS and UNIQUE. Can also use NOT IN, NOT EXISTS, and NOT UNIQUE.
• Also available: \( \text{op ANY, op ALL} \), where \( \text{op} \) can be \( >, <, =, \neq, \leq, \geq \)
  \( (a > \text{ANY} B) \) returns true when \( a \) is greater than any one element in set \( B \).
  \( (a > \text{ALL} B) \) returns true when \( a \) is greater than all elements in set \( B \).
  \[
  \text{SELECT * FROM Sailors S WHERE S.rating > ANY (SELECT S2.rating FROM Sailors S2 WHERE S2.sname='Horatio')}
  \]
• What is the above query in English?
  – Find sailors whose rating is greater than that of some sailor called Horatio.
• What is the above query in English if \( > \) ANY is replaced by \( > \) ALL?
  – Find sailors whose rating is greater than all sailors called Horatio.

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

\[
\begin{align*}
\text{SELECT S.sid} \\
\text{FROM Sailors S, Boats B, Reserves R} \\
\text{WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='red'} \\
\text{INTERSECT} \\
\text{SELECT S.sid} \\
\text{FROM Sailors S, Boats B, Reserves R} \\
\text{WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='green'}
\end{align*}
\]
• Rewrite INTERSECT with IN (plus a subquery)
  – \( (x \text{IN } B) \) returns true when \( x \) is in set \( B \).
  – Strategy?
Rewriting \textit{INTERSECT} Using \textit{IN}

\begin{verbatim}
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')

Find sids who've reserved a green boat

• Find \textit{sid}'s of Sailors who've reserved red but not green boats \textit{(EXCEPT)}
  – Replace \textit{IN} with \textit{NOT IN}.
\end{verbatim}

Division in SQL

• Find sailors who've reserved all boats.

• Strategy?
  – Find all boats that have been reserved by a sailor
  – Compare with all boats
  – Do the sailor's reserved boats include all boats?
    \begin{itemize}
    \item Yes \rightarrow include this sailor
    \item No \rightarrow exclude this sailor
    \end{itemize}

\begin{verbatim}
SELECT  S.sname
FROM    Sailors S
WHERE   NOT EXISTS ((SELECT B.bid
                      FROM  Boats B)
                     EXCEPT
                     (SELECT R.bid
                      FROM  Reserves R
                      WHERE R.sid=S.sid))
\end{verbatim}

\begin{itemize}
\item (A \textit{EXCEPT} B) returns tuples in A but not in B.
\end{itemize}
Division in SQL

- Can you do it the hard way, without EXCEPT & with NOT EXISTS?

- Strategy:
  - For each sailor, check that there is no boat that has not been reserved by this sailor.

```
SELECT  S.sname
FROM    Sailors S
WHERE   NOT EXISTS (SELECT  B.bid
                    FROM    Boats B
                    WHERE   NOT EXISTS (SELECT  R.bid
                                        FROM    Reserves R
                                        WHERE   R.bid = B.bid AND R.sid = S.sid))
```

<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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bid</th>
<th>bname</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>xyz</td>
<td>red</td>
</tr>
<tr>
<td>103</td>
<td>abc</td>
<td>green</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>31</td>
<td>101</td>
<td>11/12/96</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>12/12/96</td>
</tr>
</tbody>
</table>

Aggregate Operators

- `COUNT (*)`
- `COUNT ([DISTINCT] A)`
  - `A` is a column
- `SUM ([DISTINCT] A)`
- `AVG ([DISTINCT] A)`
- `MAX (A)`
- `MIN (A)`
- Count the number of sailors

```
SELECT COUNT (*)
FROM Sailors S
```
Find the average age of sailors with rating = 10

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

Count the number of different sailor names

\[ \text{SELECT COUNT (DISTINCT S.sname)} \]
\[ \text{FROM Sailors S} \]
Find the age of the oldest sailor

\[ \text{Sailors}(\text{sid}: \text{integer}, \text{sname}: \text{string}, \text{rating}: \text{integer}, \text{age}: \text{real}) \]

\[
\text{SELECT MAX(S.AGE)} \\
\text{FROM Sailors S}
\]

Find name and age of the oldest sailor(s)

\[
\text{SELECT S.sname, MAX(S.age)} \\
\text{FROM Sailors S}
\]

- This is illegal, but why?
  - Cannot combine a column with a value (unless we use \text{GROUP BY})

\[
\text{SELECT S.sname, S.age} \\
\text{FROM Sailors S} \\
\text{WHERE S.age = (SELECT MAX(S2.age) FROM Sailors S2)}
\]

- Okay, but not supported in every system
  - Convert a table (of a single aggregate value) into a single value for comparison
GROUP BY and HAVING

• So far, aggregate operators are applied to all (qualifying) tuples.
  – Can we apply them to each of several groups of tuples?
• Example: find the age of the youngest sailor for each rating level.
  – In general, we don’t know how many rating levels exist, and what the rating values for these levels are!
  – Suppose we know that rating values go from 1 to 10; we can write 10 queries that look like this:

\[
\text{SELECT \ MIN (S.age) } \\
\text{FROM \ Sailors S} \\
\text{WHERE \ S.rating = i}
\]

For \( i = 1, 2, \ldots, 10 \):

\[
\text{SELECT \ MIN (S.age) } \\
\text{FROM \ Sailors S} \\
\text{WHERE \ S.rating = i}
\]

\[
\text{For \( i = 1, 2, \ldots, 10 \):}
\]

Find the age of the youngest sailor for each rating level

\[
\text{SELECT S.rating, } \text{MIN (S.age) as age} \\
\text{FROM Sailors S} \\
\text{GROUP BY \ S.rating}
\]

(1) The sailors tuples are put into “same rating” groups.
(2) Compute the Minimum age for each rating group.

<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>85</td>
<td>Art</td>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>32</td>
<td>Andy</td>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>3</td>
<td>63.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rating</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>8</td>
<td>25.5</td>
</tr>
</tbody>
</table>
Find the age of the youngest sailor for each rating level that has at least 2 members.

SELECT S.rating, MIN(S.age) as age
FROM Sailors S
GROUP BY S.rating
HAVING COUNT(*) > 1

1. The sailors tuples are put into “same rating” groups.
2. Eliminate groups that have < 2 members.
3. Compute the Minimum age for each rating group.

<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>85</td>
<td>Art</td>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>32</td>
<td>Andy</td>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>3</td>
<td>63.5</td>
</tr>
</tbody>
</table>

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., AVG(S.age)).
- The attribute list (e.g., S.rating) in target-list must be in grouping-list.
- The attributes in group-qualification must be in grouping-list.
Say if Attribute list is not in grouping list

```
SELECT S.sname, S.rating,
       AVG (S.age) as age
FROM Sailors S
GROUP BY S.rating
HAVING COUNT(*) > 1
```

<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>85</td>
<td>Art</td>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>32</td>
<td>Andy</td>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>3</td>
<td>63.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sname</th>
<th>Rating</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Art</td>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>Bob</td>
<td>3</td>
<td>63.5</td>
</tr>
<tr>
<td>Dustin</td>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>Lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>Andy</td>
<td>8</td>
<td>25.5</td>
</tr>
</tbody>
</table>

Say if Group qualification is not in grouping list

```
SELECT S.rating, AVG (S.age) as age
FROM Sailors S
GROUP BY S.rating
HAVING S.sname ≠ 'Dustin'
```

<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>85</td>
<td>Art</td>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>32</td>
<td>Andy</td>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>95</td>
<td>Bob</td>
<td>3</td>
<td>63.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rating</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>40.5</td>
</tr>
</tbody>
</table>
Conceptual Evaluation

• Without \textit{GROUP BY} and \textit{HAVING}:
  – Compute cross-product of \textit{relation-list}
  – Remove tuples that fail \textit{qualification}
  – Delete unnecessary columns

• With \textit{GROUP BY} and \textit{HAVING}, continue with
  – Partition remaining tuples into groups by the value of attributes in \textit{grouping-list} (specified in \textit{GROUP-BY} clause)
  – Remove groups that fail \textit{group-qualification} (specified in \textit{HAVING} clause).
  – Compute one answer tuple per qualifying group.

\textbf{For each red boat, find the number of reservations for this boat}

\begin{verbatim}
SELECT B.bid, COUNT(*) AS num_reservations
FROM Boats B, Reserves R
WHERE R.bid=B.bid AND B.color='red'
GROUP BY B.bid

SELECT B.bid, COUNT(*) AS num_reservations
FROM Boats B, Reserves R
WHERE R.bid=B.bid
GROUP BY B.bid
HAVING B.color='red'
\end{verbatim}

• Illegal, why?
  – \textit{B.color} does not appear in \textit{group-list}
Find the 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 COUNT(S) > 1

• What is wrong?
  – COUNT(S) is counting tuples after the qualification (S.age > 18).
  – Eliminate groups with multiple sailors but only one sailor with age > 18.

• How to fix it?
  – Use subquery in the HAVING clause.

SELECT S.rating, MIN(S.age) FROM Sailors S WHERE S.age > 18 GROUP BY S.rating HAVING 1 < ANY (SELECT COUNT(*) FROM Sailors S2 WHERE S.rating=S2.rating)

Find rating(s) for (which the average age is the minimum) over all rating groups

SELECT S.rating FROM Sailors S WHERE S.age = (SELECT MIN(AVG(S2.age)) FROM Sailors S2 GROUP BY S2.rating)

• What’s wrong?
  – Aggregate operations cannot be nested

• How to fix it?

SELECT Temp.rating FROM (SELECT Temp.rating, AVG (S.age) AS avgage FROM Sailors S GROUP BY S.rating) AS Temp WHERE Temp.avgage = (SELECT MIN(Temp.avgage) FROM Temp)
Table Constraints

- Specify constraints over a single table
  - Useful when more general ICs than keys are involved.

- Constraints can be named.

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

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)))
```

Assertions: Constraints Over Multiple Tables

- Awkward and wrong!
  - If Sailors is empty, the number of Boats tuples can be anything!

- ASSERTION is the right solution; not associated with either table.

```sql
CREATE TABLE Sailors
    ( sid INTEGER,
      sname CHAR(10),
      rating INTEGER,
      age REAL,
      PRIMARY KEY (sid),
      CHECK ( (SELECT COUNT (S.sid) FROM Sailors S)
              + (SELECT COUNT (B.bid) FROM Boats B) < 100 )
    )

CREATE ASSERTION smallClub
    CHECK ( (SELECT COUNT (S.sid) FROM Sailors S)
            + (SELECT COUNT (B.bid) FROM Boats B) < 100 )
```
Triggers

- Trigger: procedure that starts automatically if specified changes occur to the DBMS
- A trigger has three parts:
  - Event (activates the trigger)
  - Condition (tests whether the triggers should run)
  - Action (what happens if the trigger runs)

```
CREATE TRIGGER incr_count AFTER INSERT ON Students // Event
WHEN (new.age < 18) // Condition
FOR EACH ROW
  BEGIN // ACTION: a procedure in Oracle’s PL/SQL syntax
    count := count + 1
  END
```

Starwar Exercises

```
char(name, race, homeworld, affiliation)
planets(name, type, affiliation)
timetable(cname, pname, movie, arrival, departure)
```

Which planet does Princess Leia go to in movie3?

```
SELECT distinct pname
FROM timetable
WHERE cname = 'Princess Leia' and movie=3
```
Starwar Exercises

char(name, race, homeworld, affiliation)
planets(name, type, affiliation)
timetable(cname, pname, movie, arrival, departure)

• How many people stay on Dagobah in movie 3?

SELECT count(*)
FROM timetable, characters
WHERE movie=3 and pname = 'Dagobah' and
timetable.cname=characters.name and
characters.race='Human'

Starwar Exercises

char(name, race, homeworld, affiliation)
planets(name, type, affiliation)
timetable(cname, pname, movie, arrival, departure)

• Who has been to his/her homeworld in movie 2?

SELECT distinct c.name
FROM characters c, timetable t
WHERE c.name=t.cname and t.pname=c.homeworld
and movie=2
Starwar Exercises

char(name, race, homeworld, affiliation)
planets(name, type, affiliation)
timetable(cname, pname, movie, arrival, departure)

• Find all characters that have been on all planets of rebels.

SELECT name
FROM characters c
WHERE not exists (SELECT p.name FROM planets p
WHERE affiliation='rebels' and p.name NOT IN
(SELECT pname from timetable t where
  t.cname=c.name and t.pname=p.name))

Starwar Exercises

char(name, race, homeworld, affiliation)
planets(name, type, affiliation)
timetable(cname, pname, movie, arrival, departure)

• Find distinct names of the planets visited by those of race "droid".

SELECT distinct t.pname
FROM char c, timetable t
WHERE c.name=t.cname and c.race='droid'
Starwar Exercises

char(name, race, homeworld, affiliation)
planets(name, type, affiliation)
timetable(cname, pname, movie, arrival, departure)

• For each character and for each neutral planet, how much time total did the character spend on the planet?

SELECT c.name, p.name, SUM(t.departure-t.arrival+1) as amount
FROM characters c, timetable t, planets p
WHERE t.cname=c.name and t.pname=p.name and p.affiliation='neutral'
GROUP BY c.name, p.name