Structured Query Language

Yanlei Diao
Structured Query Language (SQL)

- **Data Definition Language (DDL)**
  - operating on tables/views

- **Data Manipulation Language (DML)**
  - posing queries, operating on tuples

- **Extension from Relational Algebra / Calculus**
  1. From a set to a multi-set (bag) based model: duplicates are valid in the multi-set model
  2. Extending first order expressive power with aggregation and recursion
SQL Overview

- Table definition
- Query capabilities
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Aggregation & Grouping
  - Ordering
  - Null values
- Database updates
- Tables and views
- Integrity constraints
## Example Instances

**Sailor**

<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>

**Reserves**

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

**Boats**

<table>
<thead>
<tr>
<th>bid</th>
<th>sname</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>red</td>
</tr>
<tr>
<td>103</td>
<td>green</td>
</tr>
</tbody>
</table>
Creating Tables

CREATE TABLE Sailors
   ( sid INTEGER,
     sname CHAR(50) NOT NULL,
     rating INTEGER,
     age REAL,
     PRIMARY KEY (sid));

CREATE TABLE Boats
   ( bid INTEGER,
     bname CHAR(20),
     color CHAR(20),
     PRIMARY KEY (bid),
     UNIQUE (bname));

CREATE TABLE Reserves
   ( sid INTEGER,
     bid INTEGER,
     day DATE,
     PRIMARY KEY (sid,bid,day),
     FOREIGN KEY (sid) REFERENCES Sailors (sid)
     ON DELETE NO ACTION ON UPDATE CASCADE
     FOREIGN KEY (bid) REFERENCES Boats (bid)
     ON DELETE SET DEFAULT ON UPDATE CASCADE);
SQL Overview

- Table definition
- **Query capabilities**
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Aggregation and grouping
  - Ordering
  - Null values
- Database updates
- Tables and views
- Integrity constraints
# Example Instances

<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>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>yuppy</td>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>44</td>
<td>guppy</td>
<td>5</td>
<td>35.0</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>
## Basic SQL Query

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

- **relation-list**: a list of input relation names, possibly each with a range-variable.
- **qualification**: predicates combined with AND, OR and NOT
  - **predicate**: attr `op` const or attr1 `op` attr2, `op` is `<`, `>`, `>=`, `<=`, `=`, `<>
- **target-list**: a list of attributes to display in output
  - **DISTINCT** indicates no duplicates in the answer. Default is that duplicates are *not* eliminated!
## Conceptual Evaluation Strategy

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

- **relation-list**: cross-product (\( \times \))
- **qualification**: selection (\( \sigma \))
  - includes join predicates and restrictions on individual tuples
- **target-list**: projection (\( \pi \))
  - duplicate elimination if DISTINCT

- This is possibly the least efficient way to execute the query! Leave the issue to Query Optimization...
Example of Conceptual Evaluation

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid=R.sid AND R.bid=103;
```
Relational Algebra for the Query

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

\[ \pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie \text{Sailors})) \]
A Note on Range Variables

- Really needed only if the same relation appears **twice** in the `FROM` clause.

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

OR

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

*It is good style, however, to use range variables always!*
Find sailors who ’ve reserved some (at least one) boat

SELECT  S.sid
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid;

Would adding DISTINCT to this query change the answer set?

What if we replace S.sid by S.sname in the SELECT clause and then add DISTINCT?
String Pattern Matching

SELECT S.age
FROM   Sailors S
WHERE  S.sname LIKE 'A_%M';

- Find the ages of sailors whose names begin with ‘A’, end with ‘M’, and contain at least three characters.

- LIKE is used for string matching.
  - ‘_’ stands for any one character.
  - ‘%’ stands for 0 or more arbitrary characters.
Arithmetic Expressions

SELECT S.age, age1 = S.age-5, 2*S.age AS age2
FROM Sailors S
WHERE S.sname LIKE ‘A%M’;

- For sailors whose names begin with ‘A’ and end with ‘M’, return triples (of ages of sailors and two fields defined by expressions)
- Arithmetic expressions create derived attributes in SELECT.
  - AS and = are two ways to name fields in the result.
- They can also appear in the predicates in WHERE.
SQL Overview

- Table definition
- Query capabilities
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Aggregation and grouping
  - Ordering
  - Null values
- Database updates
- Tables and views
- Integrity constraints
Find sid’s of sailors who’ve reserved a red or a green boat

- If we replace OR by AND in this query, what do we get?

- UNION: computes the union of any two union-compatible sets of tuples (which are themselves the result of SQL queries).

```sql
SELECT DISTINCT R.sid
FROM Reserves R, Boats B
WHERE R.bid=B.bid
AND (B.color=‘red’ OR B.color=‘green’);
```

Why do we need DISTINCT in the query?

```sql
SELECT DISTINCT R.sid
FROM Reserves R, Boats B
WHERE R.bid=B.bid
UNION
SELECT DISTINCT R.sid
FROM Reserves R, Boats B
WHERE R.bid=B.bid
AND B.color=‘green’;
```
Set Operations on Multisets

- Consider a multiset set $A$, for each element, $x \in A$, define an indicator function:
  \[1_A(x) = \# \text{ occurrences of } x \text{ in } A\]

- Both regular sets and multisets use consistent definitions:
  \[1_{A \cup B}(x) = \max(1_A(x), 1_B(x))\]
  \[1_{A \cap B}(x) = \min(1_A(x), 1_B(x))\]
Find sid’s of sailors who’ve reserved a red and a green boat

- **INTERSECT**: computes the intersection of any two *union-compatible* sets of tuples.

```sql
SELECT DISTINCT R1.sid
FROM Reserves R1, Boats B1,
     Reserves R2, Boats B2
WHERE R1.bid=B1.bid
  AND R2.bid=B2.bid
  AND (B1.color='red'
       AND B2.color='green')
  AND R1.sid=R2.sid;
```

```sql
SELECT DISTINCT R.sid
FROM Reserves R,
     Boats B
WHERE R.bid=B.bid
  AND B.color='red'
INTERSECT
SELECT DISTINCT R.sid
FROM Reserves R,
     Boats B
WHERE R.bid=B.bid
  AND B.color='green';
```

Need **DISTINCT** to be equivalent!
Find sid’s of sailors who’ve reserved …

- Also available: **EXCEPT** (What does this query return?)

```
SELECT DISTINCT R.sid
FROM Reserves R, Boats B
WHERE  R.bid=B.bid
       AND B.color= 'red'
EXCEPT
SELECT DISTINCT R.sid
FROM Reserves R, Boats B
WHERE  R.bid=B.bid
       AND B.color= 'green';
```
SQL Overview

- Table definition
- Query capabilities
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
    - Nested queries (correlation)
  - Aggregation & Grouping
  - Ordering
  - Null values
- Database updates
- Tables and views
- Integrity constraints
Nested Queries

- A nested query has another query embedded within it.
- The embedded query is called the subquery.

- The subquery often appears in the WHERE clause:

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

- Subqueries are also possible in the FROM clause.
Conceptual Evaluation, extended

- For each row in the cross-product of the outer query, evaluate the WHERE condition by *re-computing the subquery*.

\[
\text{SELECT S.sname} \\
\text{FROM Sailors S} \\
\text{WHERE S.sid IN ( SELECT R.sid} \\
\quad \text{FROM Reserves R} \\
\quad \text{WHERE R.bid = 103 );}
\]

However, this query is equivalent to (can be *simplified* to):

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

- A subquery that depends on the table(s) mentioned in the outer query is a **correlated subquery**.
- In conceptual evaluation, must recompute subquery for each row of the outer query.

```sql
SELECT S.sname
FROM Sailors S
WHERE EXISTS (SELECT *
               FROM Reserves R
               WHERE R.bid = 103
               AND R.sid = S.sid);
```
Set Comparison Operators in WHERE

- Set comparison, optionally with a proceeding NOT:
  - **EXISTS** \( R \) -- true if \( R \) is non-empty
  - \( \text{attr IN} \ \ R \) -- true if \( R \) contains \( \text{attr} \)
  - **UNIQUE** \( R \) -- true if no duplicates in \( R \)

- Arithmetic operator \( \text{op} \ \{<,\leq,=,<>,\geq,>\} \) and **ALL/** ANY:
  - \( \text{attr op ALL} \ \ R \) -- all elements of \( R \) satisfy condition
  - \( \text{attr op ANY} \ \ R \) -- some element of \( R \) satisfies condition

\[\text{‘attr IN R’} \quad \text{equivalent to} \quad \text{‘attr = ANY R’}\]
\[\text{‘attr NOT IN R’} \quad \text{equivalent to} \quad \text{‘attr <> ALL R’}\]
Finding Extreme Values

- Find the sailors with the *highest* rating

```sql
SELECT S.sid
FROM   Sailors S
WHERE  S.rating >= ALL ( SELECT S2.rating
                           FROM Sailors S2   );
```
Please Write SQL

- Find sailors whose rating is higher than some sailor named Dustin.

```sql
SELECT S.sid
FROM Sailors S
WHERE S.rating > ANY (SELECT S2.rating
                      FROM Sailors S2
                      WHERE S2.name = 'Dustin');
```

- Find sailors whose rating is higher than all sailors named Dustin.

```sql
SELECT S.sid
FROM Sailors S
WHERE S.rating > ALL (SELECT S2.rating
                      FROM Sailors S2
                      WHERE S2.name = 'Dustin');
```
Find names of sailors who’ve reserved all boats

\{X_{\text{name}} \mid \exists X_{\text{sid}}, X_{\text{rating}}, X_{\text{age}} \langle X_{\text{sid}}, X_{\text{name}}, X_{\text{rating}}, X_{\text{age}} \rangle \in \text{Sailors} \land \\
\forall \langle X_{\text{bid}}, X_{\text{color}} \rangle \in \text{Boats} \\
(\exists X_{\text{day}} \langle X_{\text{sid}}, X_{\text{bid}}, X_{\text{day}} \rangle \in \text{Reserves}) \} 

\forall x \in R \ F(x) \equiv \neg \exists x \in R \ \neg F(x)

\{X_{\text{name}} \mid \exists X_{\text{sid}}, X_{\text{rating}}, X_{\text{age}} \langle X_{\text{sid}}, X_{\text{name}}, X_{\text{rating}}, X_{\text{age}} \rangle \in \text{Sailors} \land \\

\neg \exists \langle X_{\text{bid}}, X_{\text{color}} \rangle \in \text{Boats} \\
(\neg \exists X_{\text{day}} \langle X_{\text{sid}}, X_{\text{bid}}, X_{\text{day}} \rangle \in \text{Reserves} ) \} 

How do we write it in SQL?
Find sailors who’ve reserved *all* boats.

(1)
```
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)
));
```

(2)
```
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)
));
```
SQL Overview

- Table definition
- Query capabilities
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Aggregation and grouping
  - Ordering
  - Null values
- Database updates
- Tables and views
- Integrity constraints
Example Aggregate Operators

SELECT COUNT(*) FROM Sailors S;

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

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

SELECT COUNT(DISTINCT S.rating) FROM Sailors S WHERE S.sname='Bob';

SELECT S.sname FROM Sailors S WHERE S.rating=(SELECT MAX(S2.rating) FROM Sailors S2);
Aggregate Operators

- Take a relation (single column or multiple columns), return a value.
- Significant extension of relational algebra.

- \( \text{COUNT (\{DISTINCT\} A)} \)
- \( \text{SUM (\{DISTINCT\} A)} \)
- \( \text{AVG (\{DISTINCT\} A)} \)
- \( \text{MAX (A)} \)
- \( \text{MIN (A)} \)

---

multiple columns

single column
Find name and age of the oldest sailor(s)

- The first query is illegal! (We’ll look into the reason a bit later, when we discuss GROUP BY.)

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

```
SELECT s.sname, s.age FROM Sailors S
WHERE s.age =
(SELECT MAX(s2.age) FROM Sailors S2);
```
Motivation for Grouping

- What if we want to apply aggregate operators to each group (subset) of tuples?

- Find the age of the youngest sailor for each rating level.
  - If we know that rating values $\in [1, 10]$, write 10 queries like:

  ```sql
  SELECT MIN (S.age)
  FROM Sailors S
  WHERE S.rating = i
  
  For $i = 1, 2, \ldots, 10$:

  SELECT MIN (S.age)
  FROM Sailors S
  WHERE S.rating = i
  
  - In general, we don’t know how many rating levels exist, and what the rating values for these levels are!
Queries with GROUP BY and HAVING

```
SELECT [DISTINCT] target-list
FROM relation-list
WHERE qualification
GROUP BY grouping-list
[HAVING group-qualification];
```

- A group is a set of tuples that have the same value for all attributes in grouping-list.
- Query returns a single answer tuple for each group!
- The target-list can only contain:
  (i) attributes in the grouping-list (e.g., S.rating), or
  (ii) aggregate operations on other attributes, e.g., MIN (S.age).
Conceptual Evaluation, extended

- The cross-product of relation-list is computed.
- Tuples that fail qualification are discarded.
- The remaining tuples are partitioned into groups by the value of attributes in grouping-list.
- The group-qualification, if present, eliminates some groups.
  - Group-qualification must have a single value per group!
- A single answer tuple is produced for each qualifying group.
Find age of the youngest sailor with age ≥ 18, for each rating with at least 2 such sailors

```sql
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>
Find age of the youngest sailor with age $\geq 18$, for each rating with at least 2 such sailors.

<table>
<thead>
<tr>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>10</td>
<td>16.0</td>
</tr>
<tr>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>3</td>
<td>63.5</td>
</tr>
<tr>
<td>3</td>
<td>25.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>3</td>
<td>63.5</td>
</tr>
<tr>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>7</td>
<td>45.0</td>
</tr>
<tr>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>9</td>
<td>35.0</td>
</tr>
<tr>
<td>10</td>
<td>35.0</td>
</tr>
</tbody>
</table>

<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>
Find those ratings for which the average age is the minimum over all ratings

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

- **Derived table**: result of an SQL query as input to the FROM clause of another query
  - Computed once before the other query is evaluated.
SQL Overview

- Table definition
- Query capabilities
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Aggregation & Grouping
    - Ordering
    - Null values
- Database updates
- Tables and views
- Integrity constraints
ORDER BY

- Return the name and age of sailors rated level 8 or above in increasing (decreasing) order of age.

```sql
SELECT  S.sname, S.age
FROM    Sailors S
WHERE   S.rating > 8
ORDER BY S.age [ASC | DESC];
```
TOP-K Queries

Return the name and age of the ten youngest sailors rated level 8 or above.

```
SELECT  S.sname, S.age
FROM    Sailors S
WHERE   S.rating >= 8
ORDER BY S.age ASC
LIMIT 10;
```