SQL: Queries and Constraints

Yanlei Diao
UMass Amherst
Feb 12 & 14, 2007
DML versus DDL

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

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

- **Query capabilities**
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Aggregation & Grouping
  - Null values

- **Database updates**

- **Views**
**Example Instances**

<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>name</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>

**S2**

<table>
<thead>
<tr>
<th>sid</th>
<th>name</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 relation names (possibly with a range-variable after each relation name).
- **qualification**: Predicates combined using AND, OR and NOT.
  - Predicate: Attr op const or Attr1 op Attr2, where op is one of <, >, >=, <=, =, <>
- **target-list**: A list of attributes of relations in relation-list
  - DISTINCT indicates no duplicates in the answer. Default is that duplicates are **not** eliminated!
- **Conceptual evaluation strategy**: cross-product, selection, projection…
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>
<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>

What is the relational algebra for this query?
Relational Algebra for the Query

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

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

- Really needed only if the same relation appears twice in the FROM clause. The previous query can also be written as:

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

OR

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

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

![SQL Query Image]

- Would adding DISTINCT to this query make a difference?
- What is the effect of replacing S.sid by S.sname in the SELECT clause and adding DISTINCT to this variant of the query?
String Pattern Matching

SELECT S.age
FROM Sailors S
WHERE S.sname LIKE 'B_%B';

- Find the ages of sailors whose names begin and end with ‘B’ 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

\[
\text{SELECT } S.\text{age}, \text{age1}=S.\text{age}-5, 2*S.\text{age} \text{ AS age2}
\text{FROM } \text{Sailors S}
\text{WHERE } S.\text{sname} \text{ LIKE ‘B}_\%\text{B’};
\]

- Find triples (of ages of sailors and two fields defined by expressions) for sailors whose names begin and end with ‘B’ and contain at least three characters.
- AS and = are two ways to name fields in result.
- Arithmetic expressions can also appear in the predicates in WHERE.
SQL Overview

- Query capabilities
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Aggregation & Grouping
  - Null values

- Database updates

- Views
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).

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

SELECT S.sid
FROM Sailors S, Boats B, Reserves R
AND B.color='red'
UNION
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
AND B.color='green';
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.
  - Included in SQL-92, but some systems don’t support it.

```
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
    AND B.color='red'
INTERSECT
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
    AND B.color='green';
```

Need **DISTINCT** to be equivalent!

```
SELECT S.sid
FROM Sailors S, Boats B1, Reserves R1,
    Boats B2, Reserves R2
WHERE S.sid=R1.sid AND R1.bid=B1.bid
    AND S.sid=R2.sid AND R2.bid=B2.bid
    AND (B1.color='red' AND B2.color='green');
```
Find sid’s of sailors who’ve reserved …

- Also available: \textbf{EXCEPT} (What does this query return?)

\begin{verbatim}
SELECT  S.sid
FROM    Sailors S, Boats B, Reserves R
        AND B.color='red'
\textbf{EXCEPT}
SELECT  S.sid
FROM    Sailors S, Boats B, Reserves R
        AND B.color='green';
\end{verbatim}
SQL Overview

- Query capabilities
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Aggregation & Grouping
  - Null values
- Database updates
- Views
Nested queries

- A nested query is a query with another query embedded within it.
- The embedded query is called the subquery.
- The subquery usually appears in the WHERE clause:

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

(Subqueries also possible in FROM or HAVING clause.)
Conceptual evaluation, extended

- For each row in cross product of outer query, evaluate the WHERE clause conditions, (re)computing the subquery.

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

equivalent to:

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

- If the inner subquery depends on tables mentioned in the outer query then it is a correlated subquery.
- In terms of conceptual evaluation, we must recompute subquery for each row of outer query.

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

- Optional NOT may precede these:
  - EXISTS R -- true if R is non-empty
  - attr IN R -- true if R contains attr
  - UNIQUE R -- true if no duplicates in R

- For arithmetic operator op \{<,\leq,=,\geq,\rangle\}
  - attr op ALL R -- all elements of R satisfy condition
  - attr op ANY R -- some element of R satisfies condition

IN equivalent to \(=\) ANY
NOT IN equivalent to \(<\geq\rangle\) ALL
Example

- 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 Harry.

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

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

```sql
SELECT  S.sid
FROM     Sailors S
WHERE  S.rating > ALL (SELECT S2.rating
                        FROM Sailors S2
                        S2.name = 'Harry')
```
Simulating INTERSECT

- Suppose we have tables R(a,b) and S(a,b)
- The following computes $R \cap S$:

```sql
SELECT DISTINCT *
FROM R
WHERE (R.a, R.b) IN (SELECT * FROM S);
```

This can be expressed without nesting:

- Given R(a,b), S(a,b), what is $R \bowtie S$?

```
SELECT DISTINCT R.a, R.b
FROM R, S
WHERE R.a = S.a AND R.b = S.b;
```

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

using INTERSECT

```
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'
```

without INTERSECT (1)

```
SELECT sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
    AND S.sid IN
    (SELECT S2.sid
     FROM Sailors S2, Reserves R2, Boats B2
     WHERE S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = 'green')
```

“Find all sailors who have reserved a red boat and, further, have sids that are included in the set of sids of sailors who have reserved a green boat.”
Find the names of sailors who reserved a red and a green boat.

without INTERSECT (2)

```sql
SELECT S1.sname
FROM   (SELECT S.sid, S.name
        FROM Sailors S, Reserves R, Boats B
        WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'green') AS S1,
       (SELECT S.sid, Sname
        FROM Sailors S, Reserves R, Boats B
        WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red') AS S2
WHERE S1.sid = S2.sid AND S1.sname = S2.sname
```
Simulating EXCEPT (set difference)

- What does this query compute?

```
SELECT B.bid
FROM Boats B
WHERE B.bid NOT IN (SELECT R.bid
                     FROM Reserves R
                     WHERE R.sid = 100);
```

**Find boats not reserved by sailor with sid = 100.**

- Inner on R: boats reserved by sailor with sid=100
- All boats − inner is what we want.
Find sailors who’ve reserved all boats.

Sailors $S$ such that ...

there is no boat $B$ without ...

a Reserves tuple showing $S$ reserved $B$

(1) \[
\text{SELECT S.sname} \\
\text{FROM Sailors S} \\
\text{WHERE NOT EXISTS (} \\
\text{SELECT B.bid} \\
\text{FROM Boats B} \\
\text{WHERE NOT EXISTS (} \\
\text{SELECT R.bid} \\
\text{FROM Reserves R} \\
\text{WHERE R.sid=S.sid));}
\]

(2) \[
\text{SELECT S.sname} \\
\text{FROM Sailors S} \\
\text{WHERE NOT EXISTS} \\
((\text{SELECT B.bid} \\
\text{FROM Boats B}) \\
\text{EXCEPT} \\
(\text{SELECT R.bid} \\
\text{FROM Reserves R} \\
\text{WHERE R.sid=S.sid}));
\]
SQL Overview

- Query capabilities
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Aggregation & Grouping
  - Null values
- Database updates
- Views
Aggregate Operators

- Significant extension of relational algebra.

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

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

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

- 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

- So far, we’ve applied aggregate operators to all (qualifying) tuples. Sometimes, we want to apply them to each of several groups of tuples.

- Find the age of the youngest sailor for each rating level.
  - If we know that rating values go from 1 to 10; we can write 10 queries that look like this (!):
    
    \[
    \text{SELECT } \text{MIN (S.age)} \\
    \text{FROM Sailors S} \\
    \text{WHERE S.rating} = i
    \]
    
    For \( i = 1, 2, \ldots, 10 \):
  - 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

- A *group* is a set of tuples that have the same value for all attributes in *grouping-list*.
- Each answer tuple for a *group*!
- The *target-list* contains:
  (i) *attribute names*: each attribute must be in the *grouping-list*
  (ii) terms with *aggregate operations*, e.g., MIN (S.age).
Conceptual Evaluation

- The cross-product of \textit{relation-list} is computed.
- Tuples that fail \textit{qualification} are discarded.
- 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;
```

**Sailors instance:**

<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>
<tr>
<td>96</td>
<td>frodo</td>
<td>3</td>
<td>25.5</td>
</tr>
</tbody>
</table>

**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>
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>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

- **Query capabilities**
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Aggregation & Grouping
    - Null values

- **Database updates**

- **Views**
**NULLS in SQL**

- Whenever we don’t have a value, we can put a NULL.
- Can mean many things:
  - Value does not exist
  - Value exists but is unknown
  - Value not applicable
  - Etc.
- The schema specifies for each attribute whether it can be null (nullable attribute)
- How does SQL cope with tables that have NULLs?
Null Values

- If $x = \text{NULL}$ then $4*(3-x)/7$ is still NULL
- If $x = \text{NULL}$ then $x = \text{“Joe”}$ is UNKNOWN
- In SQL there are three boolean values:
  - FALSE = 0
  - UNKNOWN = 0.5
  - TRUE = 1
Null Values

- $C_1$ AND $C_2 = \min(C_1, C_2)$
- $C_1$ OR $C_2 = \max(C_1, C_2)$
- NOT $C_1 = 1 - C_1$

**SELECT** *
FROM Person
WHERE (age < 25) AND (height > 6 OR weight > 190)

**E.g.**
- age=20
- heigth=NULL
- weight=200

- Rule in SQL: include only tuples that yield TRUE
Null Values

- Unexpected behavior:

```
SELECT * 
FROM Person 
WHERE age < 25 OR age >= 25
```

- Some Persons are not included!
Null Values

- Can test for NULL explicitly:
  - x IS NULL
  - x IS NOT NULL

```sql
SELECT *
FROM Person
WHERE age < 25 OR age >= 25 OR age IS NULL
```

- Now it includes all Persons
SQL Overview

- Query capabilities
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Aggregation & Grouping
  - Null values

- Database updates

- Views
Modifying the Database

- Three kinds of modifications
  - Insertion - creates new tuple(s)
  - Deletion - remove existing tuple(s)
  - Updates - modify existing tuple(s)

- Sometimes they are all called “updates”
Insertions

General form:

```
INSERT INTO R(A1, ..., An) VALUES (v1, ..., vn);
```

Example: Insert a new sailor to the database:

```
INSERT INTO Sailors(sid, sname, rating, age)
VALUES (3212, 'Fred', 9, 44);
```

Missing attribute is NULL.
May drop attribute names if give values of all attributes in order.
The query replaces the VALUES keyword. Here we insert many tuples into Sailor.
Deletions

Example:

```
DELETE FROM Sailors WHERE S.sname = 'Harry';
```

Factoid about SQL: there is no way to delete only a single occurrence of a tuple that appears twice in a relation.
Updates

Examples:

UPDATE Employees
SET salary = salary * 1.1

UPDATE Sailors S
SET S.rating = s.rating + 1
WHERE S.sid IN
(SELECT sid
  FROM Reserves R
  WHERE R.date = 'Oct, 25');
SQL Overview

- **Query capabilities**
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Aggregation & Grouping
  - Null values

- **Database updates**

- **Views**
Views

- A **view** is just a relation, but we store a **definition**, rather than a set of tuples.

```sql
CREATE VIEW YoungActiveStudents (name, grade)
AS SELECT S.name, E.grade
FROM Students S, Enrolled E
WHERE S.sid = E.sid and S.age<21;
```

- Views can be dropped using **DROP VIEW** command.
  - **DROP TABLE** if there’s a view on the table?
    - **DROP TABLE** command has options to let the user specify this.
Views and Security

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  - Given YoungStudents, but not Students or Enrolled, we can find students s who are enrolled, but not the cid’s of the courses they are enrolled in.
SQL Overview

- Query capabilities
  - SELECT-FROM-WHERE blocks
  - Set operations (union, intersect, except)
  - Nested queries (correlation)
  - Aggregation & Grouping
  - Null values

- Database updates

- Views

- Integrity constraints
Integrity Constraints (Review)

- An IC describes conditions that every legal instance of a relation must satisfy.
  - Inserts/deletes/updates that violate IC’s are disallowed.
  - Can be used to ensure application semantics (e.g., sid is a key), or prevent inconsistencies (e.g., sname has to be a string, age must be < 200)

- Types of IC’s:
  - Domain constraints
  - Primary key constraints
  - Foreign key constraints
  - General constraints.
General Constraints

- Useful for ICs more general than keys.
- Two forms: table constraints and assertions.

```
CREATE TABLE Sailors
    ( sid INTEGER,
      sname CHAR(10),
      rating INTEGER,
      age REAL,
      PRIMARY KEY (sid),
      CHECK ( rating >= 1
        AND rating <= 10));
```
General Constraints (contd.)

- Can use queries to express constraints. Can also name constraints.

CREATE TABLE Reserves
    ( sname CHAR(10),
    bid INTEGER,
    day DATE,
    PRIMARY KEY (bid,day),
    FOREIGN KEY (bid) REFERENCES Boats
    ON DELETE CASCADE,
    CONSTRAINT noInterlakeRes
    CHECK (`Interlake` <>
    (SELECT B.bname
    FROM Boats B
    WHERE B.bid=bid)));
Constraints Over Multiple Relations

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 )
);

- Wrong! Table constraints are required to hold only if the associated table is nonempty.
  - What if Sailors is empty?

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

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

Number of boats plus number of sailors is < 100