Structured Query Language

Yanlei Diao

Slides Courtesy of R. Ramakrishnan, J. Gehrke, and G. Miklau

Structured Query Language (SQL)

- *Data Manipulation Language* (*DML*)
  - posing queries, operating on tuples
- *Data Definition Language* (*DDL*)
  - operating on tables/views
- Extension from Relational Algebra / Calculus
  - From a set to a multi-set (bag) based model
  - Extending first order expressive power with aggregation, and recursion

SQL Overview

- 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

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

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 using 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

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

- *relation-list*: cross-product (×)
- *qualification*: selection (σ)
- *target-list*: projection (π)
  - 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;
```

(sid)	sname	rating	age	(sid)	bid	day	
22	dustin	7	45.0	22	101	10/10/96	X
22	dustin	7	45.0	58	103	11/12/96	X
31	lubber	8	55.5	22	101	10/10/96	X X
31	lubber	8	55.5	58	103	11/12/96	X
58	rusty	10	35.0	22	101	10/10/96	X X
58	rusty	10	35.0	58	103	11/12/96	

☛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;
```

$$\pi_{sname}((\sigma_{bid=103}\text{Re}serves) \bowtie Sailors)$$

A Note on Range Variables

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

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

OR

```
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 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

- 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?

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

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

```
SELECT DISTINCT S.sid
FROM Reserves R, Boats B
WHERE  R.bid=B.bid
            AND B.color='red'
UNION
SELECT DISTINCT S.sid
FROM Reserves R, Boats B
WHERE  R.bid=B.bid
            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.

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

Need DISTINCT to be equivalent!

```
SELECT DISTINCT  S.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;
```

Find sid's of sailors who've reserved …

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

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

SQL Overview

- 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*.

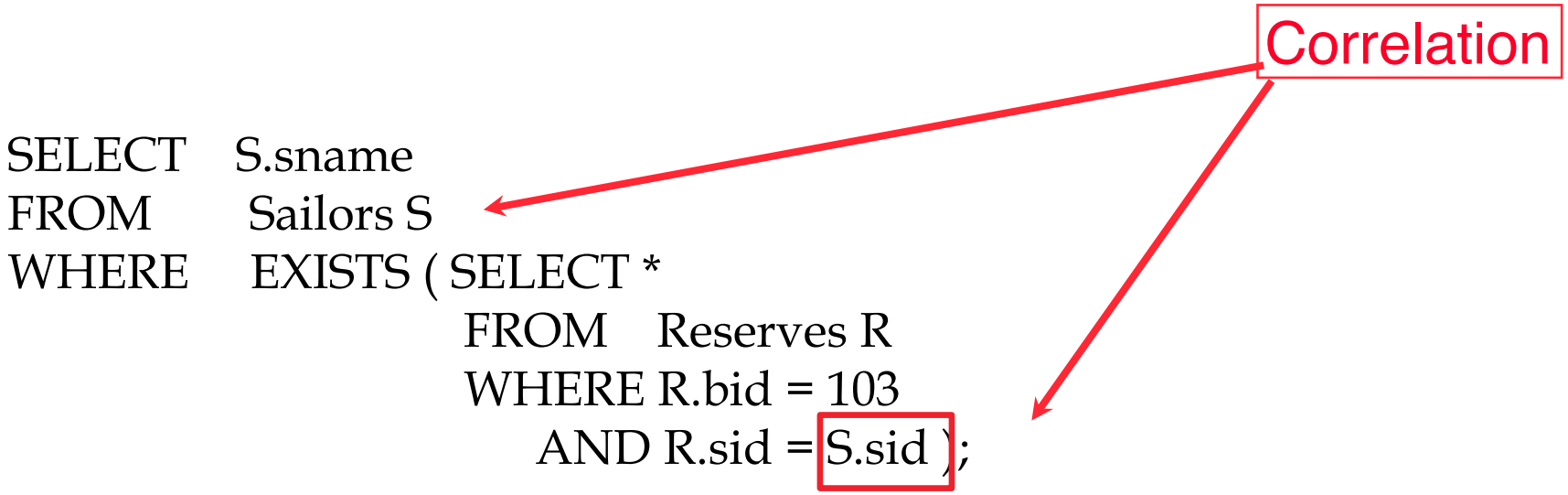
```
SELECT   S.sname
FROM     Sailors S
WHERE    S.sid IN ( SELECT R.sid
                    FROM Reserves R
                    WHERE R.bid = 103 );
```

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

```
SELECT  S.sname
FROM    Sailors S, Reserves R
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.



Set Comparison Operators in WHERE

- Set comparison, optionally with a proceeding NOT:
  - EXISTS *R* -- true if *R* is non-empty
  - *attr* IN *R* -- true if *R* contains *attr*
  - UNIQUE *R* -- true if no duplicates in *R*
- Arithmetic operator *op* {<,<=,=,< >, >=,>} and ALL/ANY:
  - *attr op* ALL *R*-- all elements of R satisfy condition
  - *attr op* ANY *R* -- some element of R satisfies condition

'attr IN *R'* equivalent to *'attr* = ANY *R'*

'attr NOT IN *R'* equivalent to *'attr* < > ALL *R'*

Finding Extreme Values

- Find the sailors with the *highest* rating

```
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.

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

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

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

Find sailors who've reserved all boats.

(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));
```

(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));
```

SQL Overview

- 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;
```

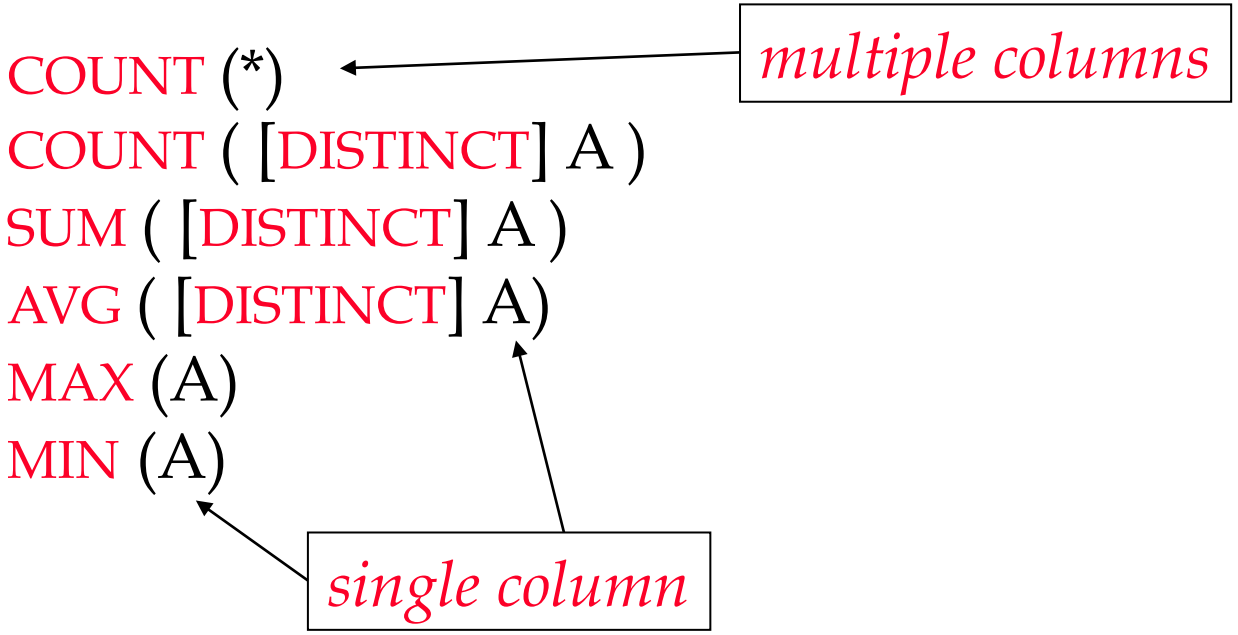
```
SELECT  AVG (S.age)
FROM  Sailors S
WHERE  S.rating=10;
```

```
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.

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 ∈[1, 10], write 10 queries like:

For i = 1, 2, ... , 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

```
SELECT  S.rating,  MIN (S.age)
                   AS minage
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1;
```

Answer relation:

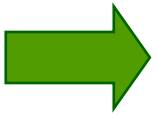
rating	minage
3	25.5
7	35.0
8	25.5

Sailors instance:

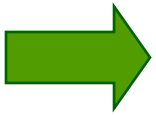
sid	sname	rating	age
22	dustin	7	45.0
29	brutus	1	33.0
31	lubber	8	55.5
32	andy	8	25.5
58	rusty	10	35.0
64	horatio	7	35.0
~~71~~	~~zorba~~	~~10~~	~~16.0~~
74	horatio	9	35.0
85	art	3	25.5
95	bob	3	63.5
96	frodo	3	25.5

Find age of the youngest sailor with age ≥ 18, for each rating with at least 2 such sailors.

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
~~10~~	~~16.0~~
9	35.0
3	25.5
3	63.5
3	25.5



rating	age
1	33.0
3	25.5
3	63.5
3	25.5
7	45.0
7	35.0
8	55.5
8	25.5
9	35.0
10	35.0



rating	minage
3	25.5
7	35.0
8	25.5

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
  - 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*.

```
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;
```

SQL Overview

- 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

NULL Values in SQL

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

Null Values

- If x= NULL, then 4*(3-x)/7 is still NULL
- If x= NULL, then x="Joe" is UNKNOWN
- In SQL there are three boolean values:
  - FALSE = 0
  - UNKNOWN = 0.5
  - TRUE = 1

Coping with Unknown Values

- C1 AND C2 = min(C1, C2)
- C1 OR C2 = max(C1, C2)
- NOT C1 = 1 – C1

```
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

Anomaly Associated with Null's

- Unexpected behavior:

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

E.g.
John's age is NULL

- Some person is not included!

Null Values

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

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

- Now it includes all people.

SQL Overview

- 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

Modifying the Database

Three kinds of modifications:

- Insert - create new tuple(s)
- Delete - remove existing tuple(s)
- Update - 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);
```

Can omit attributes; a missing attribute is NULL.
May drop attribute names if give values of all attributes in order.

Insertions

Example: Insert *multiple* tuples to Sailors:

```
INSERT   INTO   Sailors(sid, sname)

    SELECT  B.id,  B.name
    FROM      Boaters B
    WHERE   Boaters.rank = 'captain';
```

The query replaces the VALUES keyword.

Deletions

Example: delete all tuples that satisfy a condition

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

Fact 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
  - Ordering
  - Null values
- Database updates
- Tables and views
- Integrity constraints

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
            ON DELETE NO ACTION ON UPDATE CASCADE
    FOREIGN KEY (bid) REFERENCES Boats
            ON DELETE SET DEFAULT ON UPDATE CASCADE);
```

Destroying and Altering Tables

```
DROP TABLE Sailors;
```

- Destroys the Sailors relation, including schema and data.

```
ALTER TABLE Sailors
        ADD COLUMN credit_card:CHAR(40);
```

- The schema is altered by adding a new field; every tuple in the current instance is extended with a *null* value in the new field.

Views

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

```
CREATE VIEW RedBoatLovers (sid, name, bid)
    AS SELECT  S.sid, S.sname, B.bid
       FROM    Sailors S, Reserves R, Boats B
       WHERE   S.sid = R.sid and R.bid = B.bid
               and  B.color= 'red' ;
```

- Views can be dropped using DROP VIEW command.
  - DROP TABLE if there's a view on the table?

Uses of Views

- Views can be used to present necessary information (or a summary), while *hiding details in underlying relation(s)*.
- Security/Privacy
  - E.g., hiding sailors' credit card from the boat repair dept.
- Logical data independence
  - User application defined on a view is unchanged when underlying table changes
- Computational benefits
  - Result of a complex query is frequently used; materialize it.
  - Online Analytical Processing (OLAP)

SQL Overview

- 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

Integrity Constraints (Review)

- Types of *integrity constraints* in SQL:
  - *Attribute constraints*: domain, NOT NULL
  - *Key constraints*: PRIMARY KEY, UNIQUE
  - *Foreign key constraints*: FOREIGN KEY
  - *General constraints*: CHECK, ASSERTION
- Inserts/deletes/updates that violate IC's are disallowed.

General Constraints

- Two forms: CHECK (single table constraint) and ASSERTION (multiple-table constraint).

```
CREATE TABLE   Sailors
        (   sid  INTEGER,
            sname  CHAR(50),
            rating  INTEGER,
            age  REAL,
            PRIMARY KEY  (sid),
            CHECK  ( rating >= 1
                        AND rating <= 10));
```

Constraints over Multiple Relations

Number of boats plus number of sailors is < 100:

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

- ASSERTION is a constraint over both tables; checked whenever one of the table is modified.

Questions