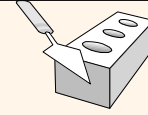
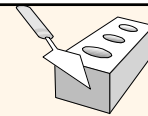


Null Values



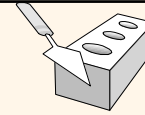
- ❖ Field values in a tuple are sometimes *unknown* (e.g., a rating has not been assigned) or *inapplicable* (e.g., no spouse's name).
 - SQL provides a special value *null* for such situations.
- ❖ The presence of *null* complicates many issues. E.g.:
 - Special operators needed to check if value is/is not *null*.
 - Is $rating > 8$ true or false when *rating* is equal to *null*? What about **AND**, **OR** and **NOT** connectives?
 - We need a 3-valued logic (true, false and *unknown*).
 - Meaning of constructs must be defined carefully. (e.g., WHERE clause eliminates rows that don't evaluate to true.)

Null Values (Cont.)



- Arithmetic operators: +, -, *, and / → null if one of their arguments is null.
- COUNT(*) handles null values just like other values; that is, **they get counted**.
- New operators (in particular, *outer joins*) possible/needed → left outer join, right outer join, and full outer join
- In a left outer join, Sailor rows without a matching Reserves row appear exactly once in the result, with the result columns inherited from Reserves assigned *null* values.

Integrity Constraints (Review)



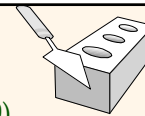
- ❖ An IC describes conditions that every *legal instance* of a relation must satisfy.
 - Inserts/deletes/updates that violate IC's are disallowed.
 - Can be used to ensure application semantics (e.g., *sid* is a key), or prevent inconsistencies (e.g., *sname* has to be a string, *age* must be < 20)
- ❖ Types of IC's: Domain constraints, primary key constraints, foreign key constraints, general constraints.
 - **Domain constraints**: Field values must be of right type. Always enforced.

General Constraints

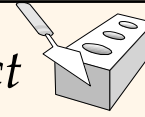
- ❖ Useful when more general ICs than keys are involved.
- ❖ Can use queries to express constraint.
- ❖ Constraints can be named.

```
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)))
```

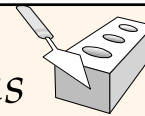


Domain Constraints and Distinct Types



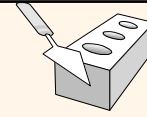
- CREATE DOMAIN ratingval INTEGER DEFAULT 1
CHECK (VALUE >= 1 AND VALUE <= 10)
- rating ratingval
- CREATE TYPE SailorId AS INTEGER
- ➔ To avoid the comparison of a SailorId value with a BoatId value

Constraints Over Multiple Relations



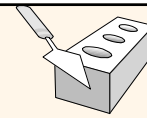
- CREATE TABLE Sailors
(sid INTEGER,
sname CHAR(10),
rating INTEGER,
age REAL,
PRIMARY KEY (sid),
CHECK
((SELECT COUNT (S.sid) FROM Sailors S)
+ (SELECT COUNT (B.bid) FROM Boats B) < 100)
- ❖ **Wrong!** If Sailors is empty, this constraint is defined to always hold. The number of Boats tuples can be anything!
- ❖ 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*

Triggers



- ❖ Trigger: procedure that starts automatically if specified changes occur to the DBMS
- ❖ Three parts:
 - Event (a change to the DB that activates the trigger)
 - Condition (tests whether the trigger should run)
 - Action (what happens if the trigger runs)
- ❖ A trigger can be thought of as a daemon that monitors a database.

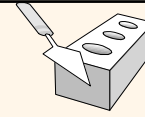
Triggers: Example (SQL:1999)



```
CREATE TRIGGER init_count BEFORE INSERT ON Students
DECLARE
    count INTEGER;
BEGIN
    count := 0;
END

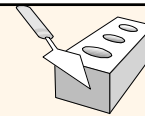
CREATE TRIGGER incr_count AFTER INSERT ON Students
WHEN (new.age < 18)
FOR EACH ROW
BEGIN
    count := count + 1;
END
```

Triggers: Example (SQL:1999)



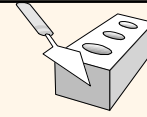
```
CREATE TRIGGER youngSailorUpdate
  AFTER INSERT ON Sailors
  REFERENCING NEW TABLE AS NewSailors
  FOR EACH STATEMENT
  INSERT
    INTO YoungSailors(sid, name, age, rating)
    SELECT sid, name, age, rating
    FROM NewSailors N
    WHERE N.age <= 18
```

Summary



- ❖ SQL was an important factor in the early acceptance of the relational model; more natural than earlier, procedural query languages.
- ❖ Relationally complete; in fact, significantly more expressive power than relational algebra.
- ❖ Even queries that can be expressed in RA can often be expressed more naturally in SQL.
- ❖ Many alternative ways to write a query; optimizer should look for most efficient evaluation plan.
 - In practice, users need to be aware of how queries are optimized and evaluated for best results.

Summary (Contd.)



- ❖ NULL for unknown field values brings many complications
- ❖ SQL allows specification of rich integrity constraints
- ❖ Triggers respond to changes in the database