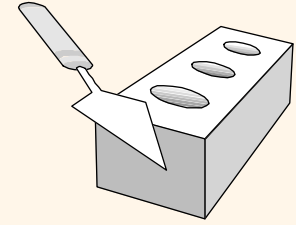


Introduction SQL: Part two

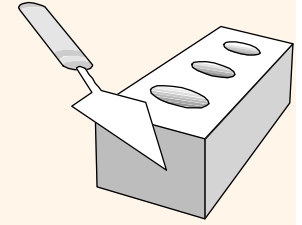
Book: Chapter 5

March 9, 2010



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.)
 - New operators (in particular, *outer joins*) possible/needed.



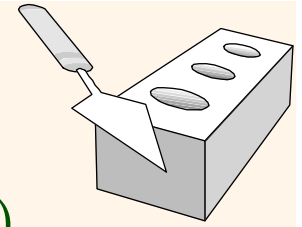
Integrity Constraints (Review)

- ❖ 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 < 100)
- ❖ Types of IC's: Domain constraints, primary key constraints, foreign key constraints, general constraints.
 - *Domain constraints*: Field values must be of the right type. Always enforced.

General Constraints

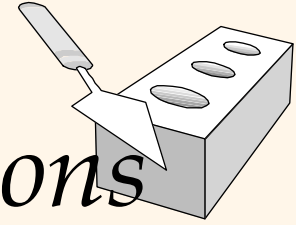
- ❖ Useful when more general ICs than keys are involved.
- ❖ Can use queries to express constraint.
- ❖ Constraints can be named.

```
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))));
```

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)
```

*Number of boats
plus number of
sailors is < 100*

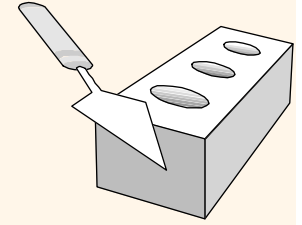
- ❖ Awkward and wrong!
- ❖ If Sailors is empty, the number of Boats tuples can be anything! **Why?**
- ❖ ASSERTION is the right solution; not associated with either table

```
CREATE ASSERTION smallClub
```

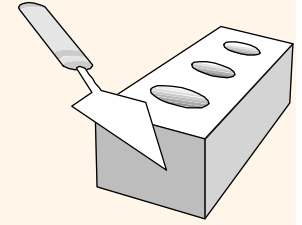
```
CHECK
```

```
((SELECT COUNT (S.sid) FROM Sailors S)  
+(SELECT COUNT (B.bid) FROM Boats B) < 100 )
```

Triggers



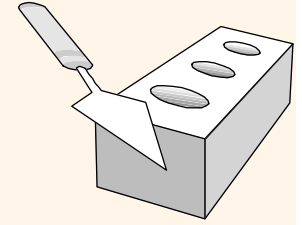
- ❖ Trigger: procedure that starts automatically if specified changes occur to the DBMS
- ❖ Three parts:
 - Event (activates the trigger)
 - Condition (tests whether the triggers should run)
 - Action (what happens if the trigger runs)



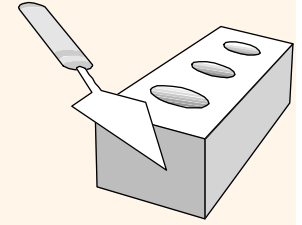
Triggers: Example (SQL:1999)

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

JOINS and NULL VALUES



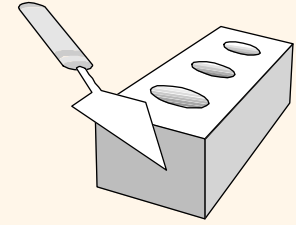
1. INNER JOINS
2. OUTER JOINS
3. NOT IN vs NOT EXISTS



JOINS revisited

Compare:

1. `SELECT *`
`FROM boats B, reserves R`
`WHERE B.bid = R.bid AND B.color = 'Gold';`
2. `SELECT *`
`FROM boats B INNER JOIN reserves R`
`ON B.bid = R.bid`
`WHERE B.color = 'Gold';`

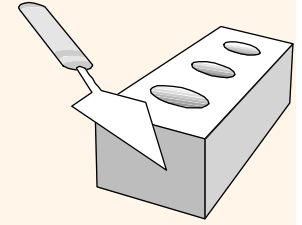


Need for outer joins

Query: for each boat give the bid, bname and the number of reservations.

Try this one:

```
SELECT B.bid, B.bname, COUNT(*) AS total  
FROM Boats B INNER JOIN Reserves R  
ON B.bid = R.bid  
GROUP BY B.bid;
```



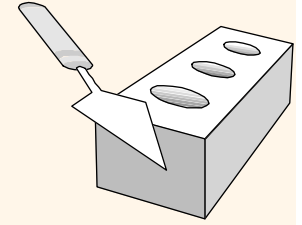
Boats and Reserves: example

<u>bid</u>	<u>bname</u>	<u>color</u>
1	P. Pan	green
2	M. Mouse	black
3	D. Duck	white
4	Snowy	white

<u>bid</u>	<u>sid</u>	<u>date</u>
1	12	1-1-2010
1	13	2-1-2010
4	12	9-3-2010
1	34	9-3-2010

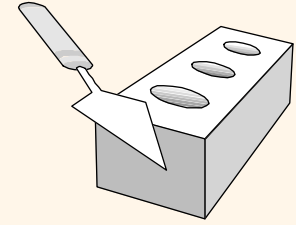
What kind of result do you expect???

The actual result:

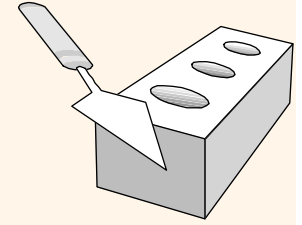


B.bid	B.bname	total
1	P. Pan	3
4	Snowy	1

But you wanted this:



bid	bname	total
1	P. Pan	3
2	M. Mouse	0
3	D. Duck	0
4	Snowy	1



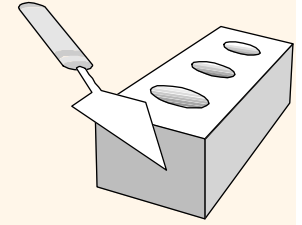
Using a left outer join:

Query: for each boat give the bid, bname and the number of reservations.

Try again:

```
SELECT B.bid, B.bname, COUNT(*) AS total  
FROM Boats B LEFT OUTER JOIN Reserves R  
ON B.bid = R.bid  
GROUP BY B.bid;
```

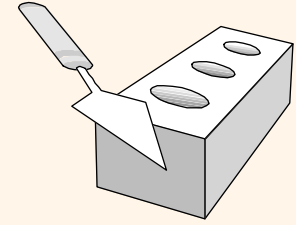
What kind of result do you expect???



The actual result:

bid	bname	total
1	P. Pan	3
2	M. Mouse	1
3	D. Duck	1
4	Snowy	1

Is this what you wanted???



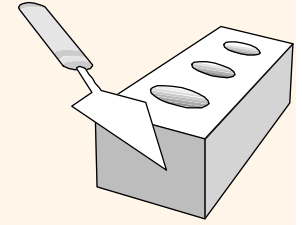
The right solution:

Query: *for each boat give the bid, bname and the number of reservations.*

One more time:

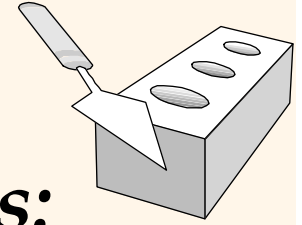
```
SELECT B.bid, B.bname, COUNT(R.bid) AS total  
FROM Boats B LEFT OUTER JOIN Reserves R  
ON B.bid = R.bid  
GROUP BY B.bid;
```

And the right result:



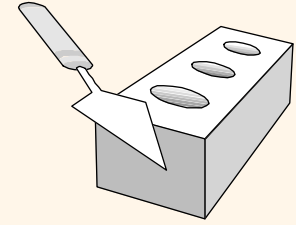
bid	bname	total
1	P. Pan	3
2	M. Mouse	0
3	D. Duck	0
4	Snowy	1

Another example of boats and reserves:



<u>bid</u>	bname	color
1	P. Pan	green
2	M. Mouse	black
3	D. Duck	white
4	Snowy	white

<u>rid</u>	bid	sid	date
r1	1	12	1-1-2010
r2	1	13	2-1-2010
r3	4	12	9-3-2010
r4	NULL	34	20-3-2010

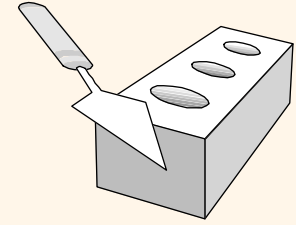


NOT IN vs NOT EXISTS

Query: give the bid's and bname's of the boats that **never** have been reserved.

```
SELECT B.bid, B.bname
FROM boats B
WHERE B.bid NOT IN (SELECT R.bid
                    FROM reserves R);
```

What result do you expect???

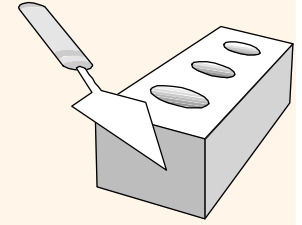


The actual result:

B.bid	B.bname
-------	---------

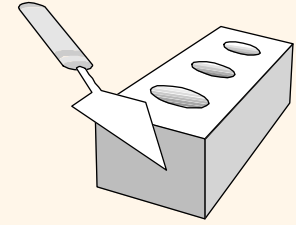
Is this what you wanted???

It's all in the NULL'S



2 = NULL ??? TRUE? FALSE? UNKNOWN!!!

3 = NULL ??? TRUE? FALSE? UNKNOWN!!!



One more time:

Query: give the bid's and bname's of the boats that **never** have been reserved.

```
SELECT B.bid, B.bname
```

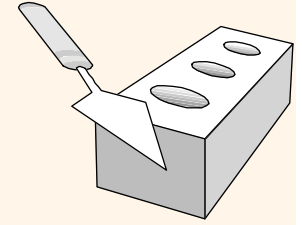
```
FROM boats B
```

```
WHERE B.bid NOT IN (SELECT R.bid
```

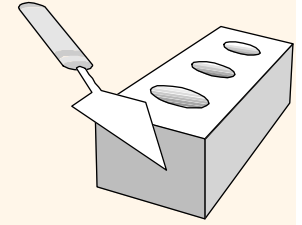
```
FROM reserves R
```

```
WHERE R.bid IS NOT NULL);
```

The right result



B.bid	B.bname
2	M. Mouse
3	D. Duck



Now try with NOT EXISTS:

Query: give the bid's and bname's of the boats that **never** have been reserved.

```
SELECT B.bid, B.bname
```

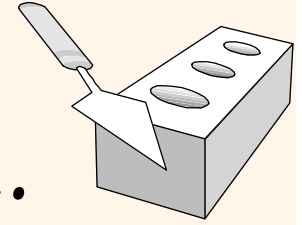
```
FROM boats B
```

```
WHERE NOT EXISTS (SELECT *
```

```
FROM reserves R
```

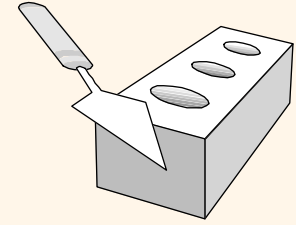
```
WHERE R.bid = B.bid);
```

And again we get the right result:



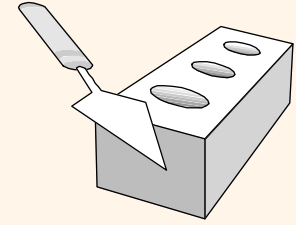
B.bid	B.bname
2	M. Mouse
3	D. Duck

.....WHERE 2 = NULL is FALSE
.....WHERE 3 = NULL is FALSE



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 (Continued)

- ❖ NULL for unknown field values brings many complications: deal with it!!!
- ❖ SQL allows specification of rich integrity constraints
- ❖ Triggers respond to changes in the database
- ❖ Be careful with NOT IN. It may be needed to test for NOT NULL.
- ❖ RTFM: some DMBS's don't adopt standard SQL.
- ❖ Use INNER JOIN instead of ,
- ❖ Use OUTER JOIN in case you want something like '... each...'.