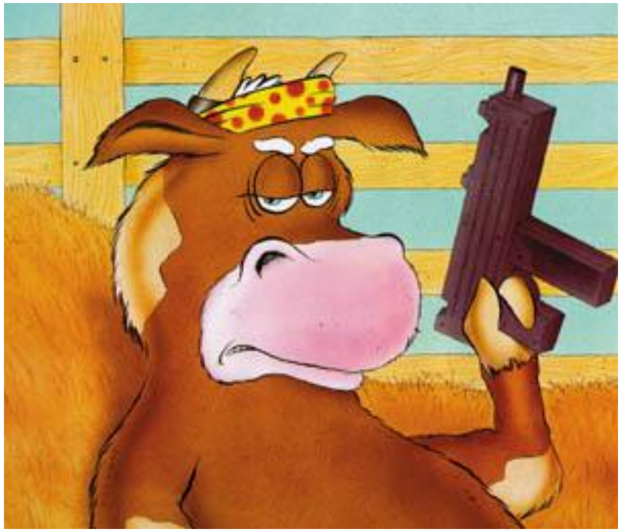
# *SQL: Constraints & Triggers*

Chapter 5.6-5.10



# Controlling Output Order

- ❖ SQL's "ORDER BY" clause is used to sort tuples in either ascending or descending order.
- ❖ ORDER BY specifies attributes used in the sort

```
SELECT *
FROM Sailors
WHERE age > 18
ORDER BY rating
```

| sid | sname | rating | age |
|---|---|---|---|
| 29 | Brutus | 1 | 33.0 |
| 85 | | | |
| 95 | | | |
| 22 | | | |
| 64 | | | |
| 31 | | | |
| 32 | | | |
| 74 | | | |
| 58 | | | |

```
SELECT *
FROM Sailors
WHERE age > 18
ORDER BY rating DESC
```

| sid | sname | rating | age |
|---|---|---|---|
| 58 | Rusty | 10 | 35.0 |
| 74 | | | |
| 31 | | | |
| 32 | | | |
| 22 | | | |
| 64 | | | |
| 85 | | | |
| 95 | | | |
| 29 | | | |

```
SELECT *
FROM Sailors
WHERE age > 18
ORDER BY rating DESC, sname ASC
```

| sid | sname | rating | age |
|---|---|---|---|
| 58 | Rusty | 10 | 35.0 |
| 74 | Horatio | 9 | 35.0 |
| 32 | Andy | 8 | 25.5 |
| 31 | Lubber | 8 | 55.5 |
| 22 | Dustin | 7 | 45.0 |
| 64 | Horatio | 7 | 35.0 |
| 85 | Art | 3 | 25.5 |
| 95 | Bob | 3 | 63.5 |
| 29 | Brutus | 1 | 33.0 |

# Controlling output size

- The "LIMIT" clause is used to limit the number of tuples returned by a "SELECT" statement
- Useful for seeing a small number of examples, or "top-X" in combination with "ORDER BY"

```
SELECT *
FROM Sailors
LIMIT 5
```

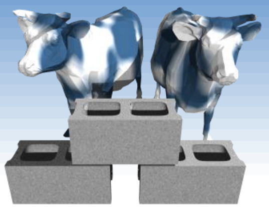
| sid | sname | rating | age |
|---|---|---|---|
| 22 | Dustin | 7 | 45.0 |
| 29 | Brutus | 1 | 33.0 |
| 31 | Lubber | 8 | 55.5 |
| 32 | Andy | 8 | 25.5 |
| 58 | Rusty | 10 | 35.0 |

```
SELECT *
FROM Sailors
ORDER BY rating DESC
LIMIT 5
```

| sid | sname | rating | age |
|---|---|---|---|
| 58 | Rusty | 10 | 35.0 |
| 74 | Horatio | 9 | 35.0 |
| 31 | Lubber | 8 | 55.5 |
| 32 | Andy | 8 | 25.5 |
| 22 | Dustin | 7 | 45.0 |

# *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.)
- Joins can also generate *null* entries



# Types of JOINS

- Recall our "Baby" sailor database

Sailors:

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

Reserves:

| sid | bid | day |
|---|---|---|
| 22 | 101 | 1996-10-10 |
| 31 | 103 | 1996-11-12 |

- An "implied" join

```
SELECT S.sname, R.day
FROM Sailors S, Reserves R
WHERE  S.sid=R.sid
```

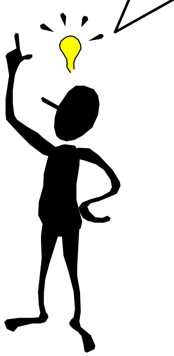
| sname | day |
|---|---|
| dustin | 1996-10-10 |
| rusty | 1996-11-12 |

- An "explicit" join (inner join)

```
SELECT S.sname, R.day
FROM Sailors S JOIN Reserves R ON S.sid=R.sid
```

```
SELECT S.sname, R.day
FROM Sailors S INNER JOIN Reserves R ON S.sid=R.sid
```

```
SELECT S.sname, R.day
FROM Sailors S NATURAL JOIN Reserves R
```



"inner" implies only tuples that share the join condition appear in the result set

| sname | day |
|---|---|
| dustin | 1996-10-10 |
| rusty | 1996-11-12 |

# *Left and Right JOINS*

Sailors:

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

Reserves:

| sid | bid | day |
|---|---|---|
| 22 | 101 | 1996-10-10 |
| 31 | 103 | 1996-11-12 |

Boats:

| bid | bname | color |
|---|---|---|
| 101 | Interlake | blue |
| 102 | Interlake | red |
| 103 | Clipper | green |

- A “Left” JOIN returns a tuple for every row of the first, “left”, relation, even if it requires adding “Null” values

```
SELECT S.sname, R.day
FROM Sailors S LEFT JOIN Reserves R ON S.sid=R.sid
```

→

```
SELECT S.sname, R.day
FROM Sailors S NATURAL LEFT JOIN Reserves R
```

→

| sname | day |
|---|---|
| dustin | 1996-10-10 |
| lubber | Null |
| rusty | 1996-11-12 |

- Likewise a “Right” join returns a tuple for every row in the second, “right”, relation

```
SELECT R.day, B.bname
FROM Reserves R NATURAL RIGHT JOIN Boats B
```

| day | bname |
|---|---|
| 1996-10-10 | Interlake |
| Null | Interlake |
| 1996-11-12 | Clipper |

# *FULL OUTER Joins*

Sailors:

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

Reserves:

| sid | bid | day |
|---|---|---|
| 22 | 101 | 1996-10-10 |
| 31 | 103 | 1996-11-12 |

Boats:

| bid | bname | color |
|---|---|---|
| 101 | Interlake | blue |
| 102 | Interlake | red |
| 103 | Clipper | green |

- The FULL OUTER JOIN keyword returns *all* rows from *all* tables with the specified attributes joined or *null* if there is no match

```
SELECT S.sname, R.day, B.bname
FROM (Sailors S NATURAL LEFT JOIN Reserves R)
          FULL OUTER JOIN Boats B ON R.bid=B.bid
```

| sname | day | bname |
|---|---|---|
| dustin | 1996-10-10 | Interlake |
| lubber | Null | Null |
| Null | Null | Interlake |
| rusty | 1996-11-12 | Clipper |

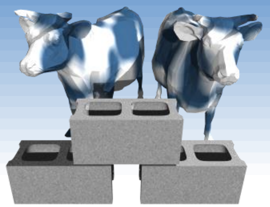
# *Integrity Constraints (Review)*

- 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 nonempty string, *age* must be < 200)
- *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 Constraint CHECKs*

- CHECK clause
- Useful when more general ICs than keys are involved.
- Example: All ratings must be between 1 and 10

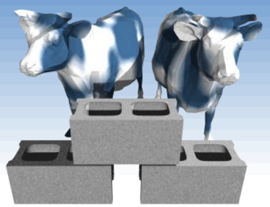
```
CREATE TABLE  Sailors(
          sid       INTEGER,
          sname     TEXT,
          rating    INTEGER,
          age       REAL,
          PRIMARY KEY  (sid),
          CHECK  (rating >= 1
                    AND rating <= 10)
```



# More complicated CHECKs

- Constraints can be named.
- Checks can contain nested subqueries
- Example: Disallow reservations of boats named "Interlake" by sailors with ratings less than 7
- "bid" and "sid" refer to values from the associated INSERT or UPDATE

```
CREATE TABLE  Reserves(
          sid    INTEGER,
          bid   INTEGER,
          day  DATE,
          PRIMARY KEY  (bid,day),
          CONSTRAINT  NoInterlakeIfLessThan7
          CHECK  ('Interlake' <> ( SELECT  B.bname
                                    FROM     Boats B
                                    WHERE  B.bid=bid)
                      OR 7  <= (SELECT S.rating
                                FROM Sailor S
                                WHERE S.sid=sid))
```



# Constraints Over Multiple Relations

- Awkward and wrong!
- If Sailors is empty, the number of Boats tuples can be anything!

```
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*

- 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

- Triggers have three parts:
  - *Event* (that activates the trigger)
  - *Condition* (tests whether the triggers should run)
  - *Action* (what happens if the trigger runs)

# Triggers: Example

- Suppose there was a rule that "*no one with a rating less than 5 can reserve a green boat*". The following trigger would enforce this rule, and generate a failure message:

```
CREATE TRIGGER RatingRuleForGreen
  BEFORE INSERT ON Reserves                                   Event
  BEGIN
     SELECT RAISE(FAIL, 'Sailor is not qualified')     Action
     WHERE EXISTS (SELECT * FROM Sailors, Boats     Condition
                         WHERE sid = new.sid AND rating < 5
                         AND bid = new.bid AND color = 'green');
  END;
```

- Note the special variable "new" for accessing parameters of the invoking INSERT query

# *Triggers: Another Example*

- Changes in one table can cause side-effects in other tables via triggers
- Example “Event Logging”
- We know dates of reservations, but not when they were made. This can be remedied using a trigger as follows:

```
CREATE TRIGGER insertLog
AFTER INSERT ON Reserves
BEGIN
    INSERT INTO ReservesLog (sid, bid, resDate, madeDate)
    VALUES (new.sid, new.bid, new.date, DATE(‘NOW’));
END;
```

# *Summary*

- NULLs provide a means for representing "unspecified" attribute values
- NULLs can be generated by special JOINs
- Wide range of JOIN operations-- Some retain the cardinality of specified relations
- SQL allows specification of rich integrity constraints
- Triggers respond to changes in the database