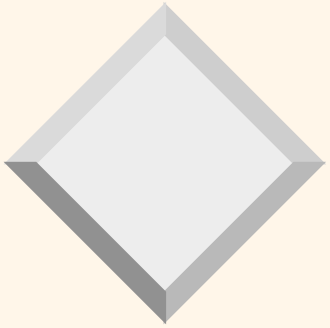


# *SQL: DDL, ICs, Updates and Views*

Module 3, Lecture 5

# *SQL is More Than Just a Query Language*

- *Data-definition language (DDL):*
  - Create / destroy / alter *relations* and *views*.
  - Define *integrity constraints* (IC's).
- *Update language:*
  - Insert / delete / modify (update) tuples.
  - Interact closely with ICs.
- *Access Control:*
  - Can grant / revoke the right to access and manipulate tables (relations / views).

# *Creating Relations*

```
CREATE TABLE Boats
        (bid: INTEGER, bname: CHAR(10), color: CHAR(10))
```

- Creates the Boats relation that we know and love. Three fields, names and types as shown.

```
CREATE TABLE Reserves
        (sname: CHAR(10), bid: INTEGER, day: DATE)
```

- A small change: Reserves uses *sname* instead of *sid.*
- *No ICs have been specified.* (We'll discuss this later.)

# *Destroying and Altering Relations*

DROP TABLE Boats

- Destroys the relation Boats. The schema information *and* the tuples are deleted.

ALTER TABLE Boats
ADD COLUMN boatkind: CHAR(10)

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

# *Creating Indexes*

CREATE INDEX NameColorInd ON Boats (*bname*, *color*)

- Creates a B+-tree index on Boats, with (*bname*, *color*) as the search key.
  - Question: What is order at bottom of tree?
- *This statement is NOT included in the SQL/92 standard!*
  - Syntax usually differs slightly between systems.
  - e.g., CREATE INDEX NameColorInd ON Boats
    WITH STRUCTURE = BTREE, KEY = (bname,color)
- To drop an index (Sybase):

  DROP INDEX Boats.NameColorInd

# *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.
  - *Domain constraints*: Field values must be of right type. Always enforced.

# *Primary and Candidate Keys (Review)*

- *Key* for a relation: Minimal set of fields such that in any legal instance, two distinct tuples do not agree upon the key field values.
  - Possibly many *candidate keys* (specified using UNIQUE), one of which is chosen as the *primary key*.
  - Primary key fields cannot contain *null* values.

```
CREATE TABLE Reserves
  ( sname CHAR(10)
    bid INTEGER,
    day DATE,
    PRIMARY KEY (sname, bid, day) )
```

```
CREATE TABLE Reserves
  ( sname CHAR(10) NOT NULL,
    bid INTEGER,
    day DATE,
    PRIMARY KEY (bid, day)
    UNIQUE (sname) )
```

# *Foreign Keys (Review)*

- *Foreign key:* Set of fields in one relation R that is used to `refer' to tuples in another relation S.
  - Fields should be a key (ideally, primary) of S.
  - In tuples of R, field values must match values in some S tuple, or be *null*.

```
CREATE TABLE Boats
  ( bid INTEGER,
    bname CHAR(10)
    color CHAR(10),
    PRIMARY KEY (bid) )
```

```
CREATE TABLE Reserves
  ( sname CHAR(10) NOT NULL,
    bid INTEGER,
    day DATE,
    PRIMARY KEY (bid, day)
    UNIQUE (sname)
    FOREIGN KEY (bid)
    REFERENCES Boats )
```

# 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!
- 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 )
```

# *Inserting New Records*

- Single record insertion:

```
INSERT INTO  Sailors (sid, sname, rating, age)
VALUES  (12, 'Emmanuel', 5, 21.0)
```

- Multiple record insertion:

```
INSERT INTO  Sailors (sid, sname, rating, age)
SELECT  S.sid, S.name, null, S.age
FROM  Students S
WHERE  S.age >= 18
```

☛ *An INSERT command that causes an IC violation is rejected.*

# *Deleting Records*

- Can delete all tuples that satisfy condition in a WHERE clause:

```
DELETE
FROM  Sailors S
WHERE  S.rating  IS NULL
```

- Example deletes all unrated sailors; WHERE clause can contain nested queries etc., in general.
- *What should be done when a deletion causes a violation of a foreign key constraint?*

# *Modifying Records*

- UPDATE command used to modify fields of existing tuples.
- WHERE clause is applied first and determines fields to be modified. SET clause determines new values.
- If field being modified is also used to determine new value, value on rhs is *old* value.

```
UPDATE  Sailors S
SET  S.rating=S.rating-1
WHERE  S.age < 15
```

| sid | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 62 | rusty | 8 | 25.0 |
| 58 | rusty | 10 | 35.0 |

```
UPDATE  Sailors S
SET  S.rating=S.rating-1
WHERE  S.rating >= 8
```

| sid | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 7 | 55.5 |
| 62 | rusty | 7 | 25.0 |
| 58 | rusty | 9 | 35.0 |

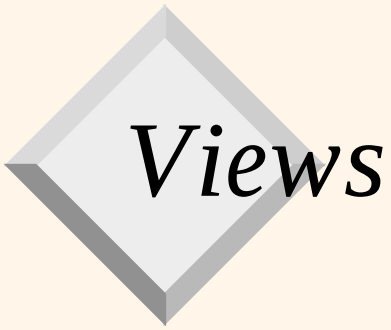
# *Enforcing Referential Integrity*

- Consider Boats and Reserves; *bid* in Reserves is a foreign key that references Boats.
- What should be done if a Reserves tuple with a non-existent boat id is inserted? (*Reject it!*)
- What should be done if a Boats tuple is deleted?
  - Also delete all Reserves tuples that refer to it.
  - Disallow deletion of a Boats tuple that is referred to.
  - Set bid of Reserves tuples that refer to it to a *default bid*.
  - Set bid of Reserves tuples that refer to it to *null*.
- Same choices if primary key of Boats tuple is updated.

# *Referential Integrity in SQL/92*

- SQL/92 supports all 4 options on deletes and updates.
  - Default is NO ACTION (*delete/update is rejected*)
  - CASCADE (also delete all tuples that refer to deleted tuple)
  - SET NULL / SET DEFAULT (sets foreign key value of referencing tuple)

```
CREATE TABLE Reserves
  ( sname CHAR(10) NOT NULL,
    bid INTEGER DEFAULT 1000,
    day DATE,
    PRIMARY KEY (bid, day)
    UNIQUE (sname)
    FOREIGN KEY (bid)
        REFERENCES Boats
        ON DELETE CASCADE
        ON UPDATE SET DEFAULT )
```



# *Views*

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

```
CREATE VIEW ActiveSailors (name, age, day)
        AS SELECT S.sname, S.age, R.day
        FROM Sailors S, Reserves R
        WHERE S.name=R.sname AND S.rating>6
```

- Views can be dropped using the DROP VIEW command.
  - How to handle DROP TABLE if there's a view on the table?
    - DROP TABLE command has options to let the user specify this.

# *Queries on Views*

- Evaluated using a technique known as *query modification.*
  - Reference to view is replaced by its definition.

```
SELECT A.name, MAX ( A.day )
FROM Active Sailors A
GROUP BY A.name
```

- Note how *sname* has been renamed to *name* to match the view definition.

```
SELECT name, MAX ( A.Day )
FROM
  ( SELECT S.sname AS name, S.age, R.day
    FROM Sailors S, Reserves R
    WHERE S.sname=R.sname
          AND S.rating>6 ) AS A
GROUP BY A.name
```

# *Updates on Views*

- Views just like base relations on queries.
- Not true for updates!
  - View update → updating the underlying relations.
  - Sometimes ambiguous or even impossible!
  - E.g.: delete (just) the highlighted tuple from instance A of view ActiveSailors.

***R***

| sname | bid | day |
|---|---|---|
| dustin | 101 | 10/10/96 |
| rusty | 104 | 12/15/96 |
| rusty | 103 | 11/12/96 |

***S***

| sid | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 62 | rusty | 8 | 25.0 |
| 58 | rusty | 10 | 35.0 |

A

| name | age | day |
|---|---|---|
| dustin | 45.0 | 10/10/96 |
| rusty | 25.0 | 12/15/96 |
| rusty | 25.0 | 11/12/96 |
| rusty | 35.0 | 12/15/96 |
| rusty | 35.0 | 11/12/96 |

# *Updatable Views*

- SQL/92 only allows updates to views on single tables with no aggregates.

```
CREATE VIEW YoungSailors (sid, age,rating)
        AS SELECT S.sid, S.age, S.rating
        FROM Sailors S
        WHERE S.age < 18
```

- Each view tuple generated from exactly one tuple in underlying relation; so any update/delete command on the view can be easily *translated* onto the relation.
- Should insertion of (94, 22.0, 7) be allowed?
  - —Adding WITH CHECK OPTION to view definition would disallow this (otherwise, it is allowed).

# *Views and Security*

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  - Given ActiveSailors, but not Sailors or Reserves, we can find sailors who have a reservation, but not the *bid*'s of boats that have been reserved.
- The GRANT/REVOKE commands can be used to control access to relations and views.
- Together with the ability to define views, this provides a very powerful access control mechanism.

# *GRANT and REVOKE of Privileges*

- GRANT INSERT, SELECT ON Sailors TO Horatio
  - Horatio can query Sailors or insert tuples into it.
- GRANT DELETE ON Sailors TO Yuppy
  WITH GRANT OPTION
  - Yuppy can delete tuples, and also authorize others to do so.
- GRANT UPDATE (*rating*) ON Sailors TO Dustin
  - Dustin can update (only) the *rating* field of Sailors tuples.
- GRANT SELECT ON ActiveSailors TO Guppy, Yuppy
  - This does NOT allow the 'uppies to query Sailors directly!
- REVOKE: When a privilege is revoked from X, it is also revoked from all users who got it *solely* from X.

# *Security to the Level of a Field!*

- Can create a view that only returns one field of one tuple. (How?)
- Then grant access to that view accordingly.
- Allows for *arbitrary* granularity of control
  - A bit clumsy to specify.
  - Can be hidden under a good UI.

# *Summary of SQL's DDL*

- DDL supports creation of relations, views and indexes. Tables can also be altered (by adding or dropping fields and ICs).
- Views can be queried just like ordinary relations, but only limited forms of updates are allowed.
- The GRANT / REVOKE commands for controlling privileges (ability to read or modify a relation), in conjunction with views, provide a powerful security and access control mechanism.

# *Summary (Contd.)*

- Many kinds of integrity constraints are supported in SQL/92.
  - Domain constraints, primary and candidate key specification, foreign keys, and general constraints over one or more relations.
  - Foreign key constraints, in particular, interact closely with insert / delete / modify commands, and users have several choices wrt this interaction.