Foreign Key Constraints

- Foreign key: a rule that a value appearing in one relation must appear in the key component of another relation.
  - aka values for certain attributes must "make sense."
  - Potter example: Every professor who is listed as teaching a course in the Courses relation must have an entry in the Profs relation.
- How do we express such constraints in relational algebra?
- Consider the relations Courses(crn, year, name, proflast, …) and Profs(last, first).
- We want to require that every non-NULL value of proflast in Courses must be a valid professor last name in Profs.

- RA $\pi_{\text{ProfLast}}(\text{Courses}) \subseteq \pi_{\text{last}}(\text{Profs})$

Foreign Key Constraints in SQL

- We want to require that every non-NULL value of proflast in Courses must be a valid professor last name in Profs.
- In Courses, declare proflast to be a foreign key.
- ```
  CREATE TABLE Courses (
     proflast VARCHAR(8) REFERENCES Profs(last),...);
  ```
- ```
  CREATE TABLE Courses (
     proflast VARCHAR(8), ...,
     FOREIGN KEY proflast REFERENCES Profs(last));
  ```

Requirements for FOREIGN KEYs

- If a relation R declares that some of its attributes refer to foreign keys in another relation S, then these attributes must be declared UNIQUE or PRIMARY KEY in S.

- Values of the foreign key in R must appear in the referenced attributes of some tuple in S.

Enforcing Referential Integrity

- Three policies for maintaining referential integrity.
- Default policy: reject violating modifications.
- Cascade policy: mimic changes to the referenced attributes at the foreign key.
- Set-NULL policy: set appropriate attributes to NULL.

Default Policy for Enforcing Referential Integrity

- Reject violating modifications. There are four situations where this can happen.

- ***Insert*** a new Song tuple with an unreferenced Artist.
- ***Update*** the artist attribute in a Song tuple to an unreferenced Artist.
- ***Update*** the name attribute in an Artist tuple who has at least one Song tuple.
- ***Delete*** a tuple in Artists who has at least one Song tuple.

Cascade Policy for Enforcing Referential Integrity

- Only applies to deletions or updates to tuples in the referenced relation (e.g., Artists).

- If we delete a tuple in Artists, delete all tuples in Songs that refer to that tuple.
- If we change the name of an artists in Artists, update all Songs with the altered artist name as well.

Set-NULL Policy for Enforcing Referential Integrity

- Only applies to deletions or updates to tuples in the referenced relation (e.g., Artists).
- If we delete a tuple in Artists, set the artist attribute of all tuples in Songs that refer to the deleted tuple to NULL.
- If we change the name of an artist in Artists, set all of the artist attributes in any Song tuple that references the artist to NULL.

Specifying Referential Integrity Policies in SQL

- SQL allows the database designer to specify the policy for deletes and updates independently.
- Optionally follow the declaration of the foreign key with ON DELETE and/or ON UPDATE followed by the policy: SET NULL or CASCADE.
- Constraints can be circular, e.g., if there is a one-one mapping between two relations.
- SQL allows us to defer the checking of constraints (see Chapter 7.1.3).
- Good suggestions if you don't want default rejection: ON DELETE SET NULL, ON UPDATE CASCADE.

Specifying Referential Integrity Policies in SQL

- Remember – ON DELETE/ON UPDATE only applies in two situations:
  - Deleting a row from the ***REFERENCED*** table.
  - Updating a row from the ***REFERENCED*** table.
- Deletes and updates from the FOREIGN KEY table cannot be propagated; they are still automatically rejected.
- Example: updating a song in the songs table to change its artist.

Constraining Attributes and Tuples

- SQL also allows us to specify constraints on attributes in a relation and on tuples in a relation.
  - Disallow courses with a maximum enrollment greater than 100.
  - A chairperson of a department must teach at most one course every semester.
- How do we express such constraints in SQL?
- How can we change our minds about constraints?
- A simple constraint: NOT NULL
  - Declare an attribute to be NOT NULL after its type in a CREATE TABLE statement.
  - Effect is to disallow tuples in which this attribute is NULL.

Attribute-Based CHECK Constraints

- `CREATE TABLE name (`
  `attrib type CHECK (constraint), …)`
- constraint is anything that can be in a WHERE clause.
  - But usually it's a simple limit on values.
- CHECK statement may use a subquery to mention other attributes of the same relation or other relations.*
- An attribute-based CHECK constraint is checked only when any tuple gets a new value for this attribute.
  - INSERTs/UPDATEs
  - Not DELETEs! (e.g., do not use CHECK to simulate referential integrity with a foreign key)

Tuple-Based CHECK Constraints

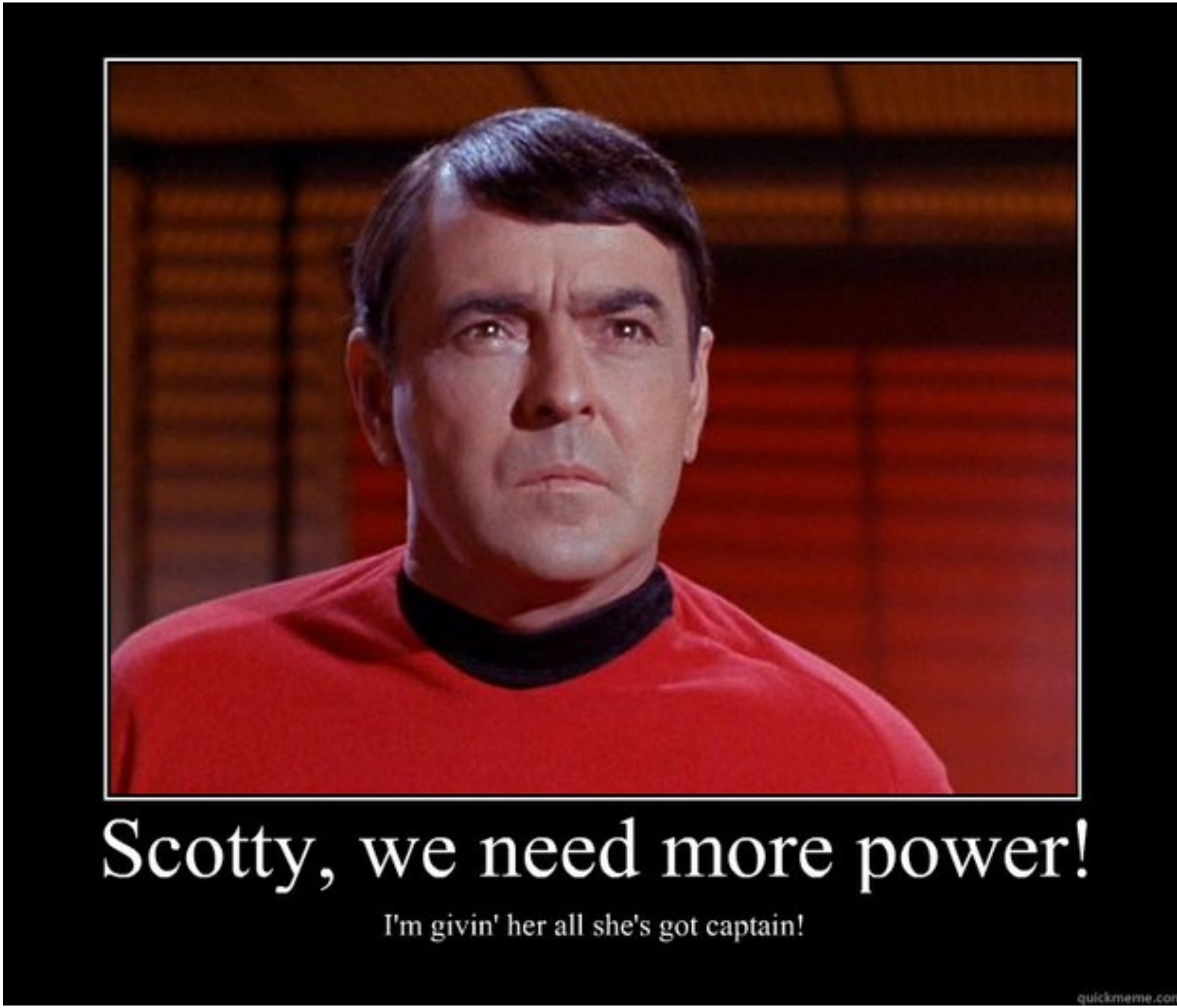
- Tuple-based CHECK constraints are checked whenever a tuple is inserted into or updated in a relation.
- Designer may add these constraints after the list of attributes in a CREATE TABLE statement.
- ```
  CREATE TABLE name (
    attrib1 type1, attrib2 type2, …
    CHECK (constraint));
  ```
  ```

CHECK Caveats

- CHECK constraints are only triggered when the table on which they are declared is INSERTed into or UPDATEd.
- If a CHECK constraint on table T1 references another table T2 in a constraint and that other table changes, it will not re-trigger the CHECK.
- Many DBMSs (SQLite, MySQL, PostgreSQL, Oracle) prohibit subqueries in CHECKs for this reason.

Modifying Constraints

- SQL allows constraints to be named, so that they can later be modified.
- See SQL documentation.


Scotty, we need more power!
I'm givin' her all she's got captain!
quickmeme.com


NEEDS MORE
JIGGAWATTS


Did you say
MORE POWER?

Assertions and Triggers

- Assertion – a schema-level condition that must be true at all times (a more powerful CHECK).
- Trigger – a series of actions associated with some database event.

Assertions

- These are database-schema elements, like relations
- Defined by:
  - CREATE ASSERTION <name>

    CHECK (<condition>);
- Condition may refer to any relation or attribute in the database schema.

Assertions: Example

- Can’t have more courses than students (‘Pigeonhole Principle’)

```
CREATE ASSERTION FewStudents CHECK (
 (SELECT COUNT(*) FROM Students) >=
 (SELECT COUNT(*) FROM Courses)
);
```

Assertions

- Bad news – no popular DBMSs support them.

Triggers: Motivation

- Assertions are powerful, but the DBMS often can't tell when they need to be checked.
- Attribute- and tuple-based checks are checked at known times, but not as powerful.
- ***Triggers*** let the user decide when to check for any condition.

Triggers

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

Maintain a unary relation New_Courses which has the list of brand new courses

Triggers

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

```
CREATE TRIGGER incr_count
AFTER INSERT ON Teach           // Event
REFERENCING NEW ROW AS new
FOR EACH ROW
WHEN (new.id NOT IN (SELECT ID FROM Courses))    // Condition
INSERT INTO New_Courses(id) VALUES(new.id);      // Action
```

OK, what could have been done?