# NULL values, SQL Constraints & Triggers

Kathleen Durant PhD

CS 3200

Lesson 7

# Lecture Outline

- Constraints
- NULL Values
- Trigger Description
- My SQL trigger example
- MY SQL Installation

# NULL and Missing Data

# Missing data values in relations

- Allowing fields to have no value allows us to model the real world as well as mathematics
- We need to be able to semantically represent the NAN or a value that does not have a value (yet)  or a value that does exist we just do not know it
- Missing data is prevalent in many fields of study – the majority of data is missing
  - Goal: get a good representation of what is not there given the minority that is provided
  - Many statistical and data mining techniques defined to deal with missing data

# NULLS in SQL

- NULL is a placeholder for missing or unknown value of an attribute.
  - It is not itself a value, therefore it has no data type.
- Codd proposed to distinguish two kinds of NULLs:
- A-marks: data is applicable but not known (for example, someone’s age)
- I-marks: data is Inapplicable (telephone number for someone who does not have a telephone, or spouse’s name for someone who is not married)

# NULL and its impact on SQL

- SQL allows field values not to have a value
  - Sometimes the field's value will not be known until later or it is inapplicable
  - Example: Later: (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.
- Presence of NULL complicates many issues:
  - Special operators needed to check if value is (not) NULL.
  - Is rating>8 true or false for rating=NULL? What about AND, OR and NOT connectives?
- We need a 3-valued logic (true, false and unknown).
- Semantics of 3-valued logic must be defined consistently.
  - WHERE clause eliminates rows that do not evaluate to true.

# Problems with NULLs

- Defining selection operation: if we check tuples for some property like Mark > 40 and for some tuple Mark is NULL, do we include it?
- Defining intersection or difference of two relations: are two tuples <John, NULL> and <John,NULL> the same or not?
- Additional problems for SQL: do we treat NULLs as duplicates?
- Do we include them in count, sum, average and if yes, how? How do arithmetic operations behave when an argument is NULL?

# Solutions to NULL: three values in Logic

- Use three-valued logic instead of classical two-valued logic to evaluate conditions.
- When there are no NULLs around, conditions evaluate to true or false, but if a null is involved, a condition will evaluate to the third value (‘undefined’, or ‘unknown’).
- This is the idea behind testing conditions in WHERE clause of SQL SELECT: **only tuples where the condition evaluates to true are returned.**

# 3-VALUED LOGIC

| X | Y | X AND Y | X OR Y | NOT X |
|---|---|---|---|---|
| TRUE | TRUE | TRUE | TRUE | FALSE |
| TRUE | UNKNOWN | UNKNOWN | TRUE | FALSE |
| TRUE | FALSE | FALSE | TRUE | FALSE |
| UNKNOWN | TRUE | UNKNOWN | TRUE | UNKNOWN |
| UNKNOWN | UNKNOWN | UNKNOWN | UNKOWN | UNKNOWN |
| UNKNOWN | FALSE | FALSE | UNKNOWN | UNKNOWN |
| FALSE | TRUE | FALSE | TRUE | TRUE |
| FALSE | UKNOWN | FALSE | UNKNOWN | TRUE |
| FALSE | FALSE | FALSE | FALSE | TRUE |

FALSE = 0, TRUE = 1, UNKNOWN =1/2 NOT(X) = 1-X, AND(X,Y) =MIN(X,Y), OR(X,Y) = MAX(X,Y)

# SQL: NULLs in conditions

- Select SID from Sailor where rating > 5
- Execution: rating > 5 evaluates to ‘unknown’ on the last tuple

| SID | Sname | Rating | Age |
|---|---|---|---|
| 28 | Yuppy | 9 | 35.0 |
| 31 | Lubber | 3 | 55.5 |
| 44 | Guppy | 5 | 35.0 |
| 58 | Rusty | NULL | 35.0 |

| SID | Sname | Rating | Age |
|---|---|---|---|
| 28 | Yuppy | 9 | 35.0 |

# SQL: NULLs in conditions

- Select SID from Sailor where rating > 5 **OR** Name = ‘Rusty’
- Execution: rating > 5 evaluates to true on the last tuple

| SID | Sname | Rating | Age |
|---|---|---|---|
| 28 | Yuppy | 9 | 35.0 |
| 31 | Lubber | 3 | 55.5 |
| 44 | Guppy | 5 | 35.0 |
| 58 | Rusty | NULL | 35.0 |

| SID | Sname | Rating | Age |
|---|---|---|---|
| 28 | Yuppy | 9 | 35.0 |
| 58 | Rusty | NULL | 35.0 |

# SQL: NULLs in Arithmetic

- Select SID, Rating * 10 as NewRating from Sailor
- Arithmetic operations applied to NULL result in NULLs

| SID | Sname | Rating | Age |
|---|---|---|---|
| 28 | Yuppy | 9 | 35.0 |
| 31 | Lubber | 3 | 55.5 |
| 44 | Guppy | 5 | 35.0 |
| 58 | Rusty | NULL | 35.0 |

| SID | NewRating |
|---|---|
| 28 | 90 |
| 31 | 30 |
| 44 | 50 |
| 58 | NULL |

# SQL with NULLS: Aggregates

- Select avg(Rating) as AVG, COUNT(Rating) as NUM, COUNT(*) as ALLNUM, SUM(Rating) as SUM from Sailors

  - AVG = 5.67
  - NUM = 3
  - ALLNUM = 4
  - SUM = 17

| SID | Sname | Rating | Age |
|---|---|---|---|
| 28 | Yuppy | 9 | 35.0 |
| 31 | Lubber | 3 | 55.5 |
| 44 | Guppy | 5 | 35.0 |
| 58 | Rusty | NULL | 35.0 |

# Outer Joins

- When we take the join of two relations we match up tuples which share values
  - Some tuples have no match are ‘lost’
  - These are called dangles
- Outer joins include dangles in the result set and use NULLs to fill in the blanks
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
  - FULL OUTER JOIN

# Alternative Solution: Default Values to Express Loss of Data

- Default values are an alternative to the use of NULLs
  - If a value is not known a particular placeholder value –the default is used
  - Actual values within the domain type so no need for 3 value-logic
  - Default values can provide more meaning than NULLs
    - None
    - Unknown
    - Not supplied
    - Not applicable

# Default Value Example

- Default values are
  - ???? For Name
  - -1 for Rating and Age
- Hopefully no one has a name of ???? and rating and age cannot really be = -1 so can identify your default values
- What about
- Update Sailors

  set age = age +1?

| SID | Sname | Rating | Age |
|---|---|---|---|
| 28 | Yuppy | 9 | 35.0 |
| 31 | Lubber | 3 | 55.5 |
| 44 | ???? | 5 | -1 |
| 58 | Rusty | -1 | 35.0 |

# Problems with default values

- They are real values in the domain of the variable
  - They can be updated like any other field value
  - You need to use a value that will not appear in any other circumstances
  - They may not be interpreted correctly
  - You need compatibility in the domains
    - You can’t have a string such as ‘unknown’ stored in an integer field
- You may want to just use NULL

# NULL support in SQL

- SQL allows you to INSERT NULLs
  - Example: UPDATE Sailors set rating = NULL where Name = ‘Mark’
- Separate function to test for NULL
  - Example: SELECT Name from Sailor where rating IS NOT NULL
  - Example: SELECT Name from Sailor
    where rating IS NULL

# NULL or Default Value

- Which method to use?
- Default values should not be used when they might be confused with ‘real’ values
- NULLs can (and often are) used where the other approaches seem inappropriate

# Integrity Constraints

- 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. This is always enforced.

# General Constraints

- Allows you to define a constraint beyond key or unique fields.
  - Can use queries to express constraint.
  - Constraints can be named uses the CHECK keyword

- 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 tables

- Create a contraint such that: *Number of boats plus number of sailors is < 100*
- CREATE ASSERTION smallClub CHECK ( (SELECT COUNT (S.sid) FROM Sailors S) +(SELECT COUNT (B.bid) FROM Boats B) < 100 )

# Triggers

- Similar to Integrity constraints

# Triggers

- Trigger: procedure that starts automatically if specified changes occur to the DBMS
- A trigger has three parts:
- Event
  - Change to the database that activates the trigger
- Condition
  - Query or test that is run when the trigger is activated
- Action
  - Procedure that is executed when the trigger is activated and its condition is true

# Trigger Options

- **Event** can be insert, delete, or update on DB table
- **Condition:**
  - Condition can be a true/false statement
    - All employee salaries are less than $100K
  - Condition can be a query
    - Interpreted as true if and only if answer set is not empty
- **Action** can perform DB queries and updates that depend on:
  - Answers to query in condition part
  - Old and new values of tuples modified by the statement that activated the trigger
  - Action can also contain data-definition commands, e.g., create new tables

# When to Fire the Trigger

- Triggers can be executed once per modified record or once per activating statement
  - Row-level trigger versus a Statement Level Trigger
  - Trigger looking at the set of records that are modified versus the actual individual values of the old and the new values
- Should trigger action be executed before or after the statement that activated the trigger?
  - Consider triggers on insertions
    - Trigger that initializes a variable for counting how many new tuples are inserted: execute **trigger before insertion**
    - Trigger that updates this count variable for each inserted tuple: **execute after each tuple is inserted** (might need to examine values of tuple to determine action)
    - Trigger can also be run **in place  of the action**

# Trigger Example

- 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
```

Trigger has access to **NEW** and **OLD** values

# Trouble with Triggers

- Action can trigger multiple triggers
  - Execution order is arbitrary
- Challenge: Trigger action can fire other triggers
  - Very difficult to reason about what exactly will happen
    - Trigger can fire “itself” again
  - Unintended effects possible
- Many religious wars on triggers evil vs. not evil
  - Analogous to the gun control debate in society
    - Triggers do not corrupt databases people who write triggers do
- Example: Triggers defined to monitor Stock prices
  - Once multiple triggers are activated can’t shut off
  - Sit back and watch the world’s economic system collapse
- Introducing Triggers leads you to deductive databases
  - Need rule analysis tools that allow you to deduce truths about the data

# MY SQL limits the use of triggers

- Triggers not introduced until 5.0
- Not activated for foreign key actions
- No triggers on the mysql system database
- Active triggers are not notified when the meta data of the table is changed while it is running
- No recursive triggers
- Triggers cannot modify/alter the table that is already being used
    - For example the table that triggered it

# MY SQL Trigger

```
CREATE TRIGGER <trigger-name> trigger_time trigger_event
 ON table_name
              FOR EACH ROW
                  BEGIN
                   END
```

- Syntax
  - Trigger_time is [BEFORE | AFTER]
  - Trigger_event [INSERT|UPDATE|DELETE]
  - Other key words – OLD AND NEW
  - Naming convention for a trigger
    trigger_time_tablename_trigger_event
  - Found in the directory associated with the database
    - File tablename.tdg – maps the trigger to the correspnoding table
    - Triggername.trn contains the trigger definition

# Reviewing your trigger

- Go to the trigger directory and read the file (.trg)

Program Data\MySQL\MySQL5.5\data\<db-name>*.trg

- Use the DBMS to locate the trigger for you

**Triggers in current schema**

```
SHOW TRIGGERS;
```

**ALL Triggers in DBMS using the System Catalog**

```
SELECT * FROM Information_Schema.Triggers
WHERE Trigger_schema = 'database_name' AND
   Trigger_name = 'trigger_name';
select trigger_schema, trigger_name, action_statement
from information_schema.triggers;
```

# Changing your trigger

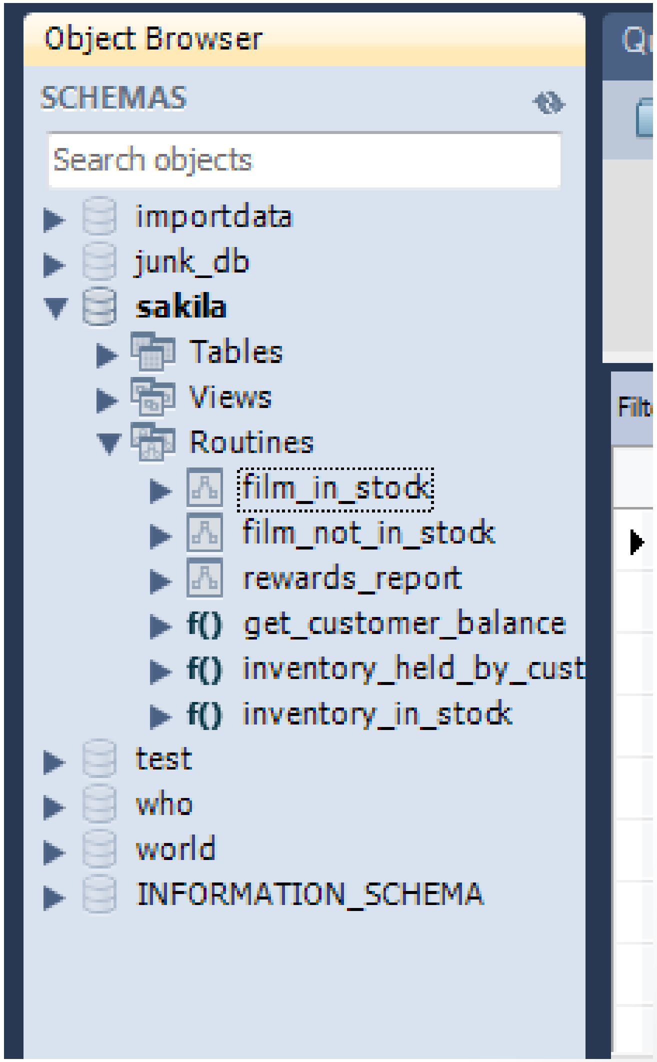
- There is no edit of a trigger
- CREATE TRIGGER …
- DROP TRIGGER <TRIGGERNAME>;
- CREATE TRIGGER …

# Information_schema tables in MySQL

- Description of schema for all databases within your MySQL instance
  - Every vendor implements the catalog differently
- Actually implemented as views as opposed to tables – so no files or directory structure associated with them
- Can query on many schema objects : information_schema.tables, information_schema.triggers, information_schema.routines….
- https://dev.mysql.com/doc/refman/5.5/en/information-schema.html
  - 20 or so different views of catalog data

# Procedures and Functions in My SQL

- Where are they stored?
- Stored with your database
- Find them in the database directory



Object Browser

SCHEMAS

Search objects

- importdata
- junk_db
- sakila
  - Tables
  - Views
  - Routines
    - film_in_stock
    - film_not_in_stock
    - rewards_report
    - f() get_customer_balance
    - f() inventory_held_by_cust
    - f() inventory_in_stock
- test
- who
- world
- INFORMATION_SCHEMA

# MY SQL Notes

- One directory for the catalog, which is itself a database.
- Consists of tables specifying privileges – who can access what
  - database-level privileges
  - table-level privileges  etc.
- One directory for each user database.
- Each table is represented by three files:
  - one for the per-table metadata
  - one for the data
  - one for any indices on the table

# Summary

- NULL for unknown field values brings many complications to a DBMS
  - However, unknown values are part of the real world
- SQL allows specification of rich integrity constraints
  - Define constraints across tables
- Triggers respond to changes in the database
  - Strength: Very Powerful
  - Weakness: Very Powerful