

# Constraints, Triggers and Views

Lecture By  
Binu Jasim  
19-Sep-2016



# Constraints

- Also known as Integrity Constraints
- Limits the allowed database states
- Eg:- can't have NULL values in the name field
- $0 \leq \text{CGPA} \leq 10$



# Why Constraints

- Catch data entry errors and update errors
- Consistency check - Referential integrity constraints
- Query optimization and efficient indexing



# Non NULL Constraints

```
create table Student (rollNo int,  
name text, CGPA real not null);
```

- `text`, `int` and `real` are part of the schema. They are not considered as constraints
- `not null` is a constraint



```
insert into Student values(123, 'Alice',  
null);
```

```
update Student set CGPA = null where  
rollNo = 123;
```

Error: NOT NULL constraint failed:



# Key Constraint

```
create table Faculty (id primary key,  
name);
```

Error: UNIQUE constraint failed

(if tried to insert two rows with the same id)

Note: NULL values in the key field are allowed (some systems (like sqlite) even allow repeated NULL!)

```
create table Tathva_Reg (rollNo, college,  
primary key (rollNo, college));
```



# Unique Constraints

```
create table Faculty (id primary key,  
name unique);
```

```
create table Tathva_Reg (rollNo, college,  
unique (rollNo, college));
```

Error: UNIQUE constraint failed

(if tried to insert two rows with the same id)

Note: NULL values in the Unique field are allowed  
(even repeated NULL!)



# Attribute Based Constraints

```
create table Student (rollNo primary key,  
name,  
CGPA check (CGPA <= 10 and CGPA >= 0) );
```

```
insert into Student values(1, 'Bob', 20);  
Error: CHECK constraint failed
```



# Tuple Based Constraints

- Constraints on more than one attributes together in a tuple
- Eg:- Only CS department can offer DBMS course

Course

rollNo	cName	dept	marks
123	DBMS	CSE	48
123	OS	CSE	36
399	DBMS	ECE	25



Only CSE dept can offer the DBMS course

```
create table Course (rollNo, cName,  
dept, marks,  
check (cName <> 'DBMS'  
or dept = "CSE" ) );
```



- Note that MySQL doesn't support 'check' constraints. PostgreSQL and SQLite support
- But it is valid under SQL specification
- We can implement other constraints using check constraint
- `check(CGPA is not null)`



# Assertions

- CHECK constraints if violated only prevents the current insertion or update or delete command
- ASSERTIONS can involve or any number of other rows in the same table, or even any number of other tables - more global in scope
- CREATE ASSERTION - DDL statement



Employee salary shouldn't be greater than the salary of their manager

```
create assertion salary_constraint  
check(not exists  
  (select * from EMP E, EMP M  
    where E.salary > M.salary  
    and E.mngr_ssn = M.ssn));
```



All students in Course table should be in the Student table

```
create assertion ReferentialIntegrity  
  check (not exists (select * from Course  
    where rollNo not in  
      (select rollNo from Student)));
```



# Assertions

- Assertions are global constraints. DBMS will throw an error whenever it is violated
- we can delete assertions using drop assertion
- Significantly more difficult and slower than simple checks
- Most DBMS (MySQL, Postgre or sqlite) don't support Assertions
- MariaDB (created by MySQL creators) supports



# Deferred Constraint Checking

- During a single database update, a constraint could be violated.
- But after a transaction the constraints should hold
- eg:- the column 'percent' should add to 100% over all rows
- PostgreSQL, Oracle etc. support deferred constraints



# Referential Integrity Constraints

customer_id	first_name	last_name	email	phone
1	John	Bonham	bonham@example	020394984
2	Dave	Grohl	grohl@example.	920930938
3	Robert	Smith	smith@example.	9873219847
4	Frank	Black	black@example.	892372039872



order_id	order_date	customer
1	2012-02-19 01:14:30	2
2	2012-03-11 16:14:59	2
3	2012-03-01 18:15:15	1
4	2012-03-11 01:00:26	4
5	2012-03-11 01:27:51	3

- From S.b  $\rightarrow$  R.a
- field a has to be primary key field of R
- field b is called foreign key



# Actions upon delete and update of R.a

- Restrict - Throw error - default
- set Null and cascade
- cascade



```
create table Course(rollNo  
    references Student(rollNo)  
        on delete set null),  
    cName, dept, marks);
```



```
create table T(A, B, C,  
    primary key(A,B),  
    foreign key (B,C) references T(A,B)  
    on delete cascade);
```

```
insert into T values (1,1,1);  
insert into T values (2,1,1);  
insert into T values (3,2,1);  
insert into T values (4,3,2);
```

Q. What happens if the first row is deleted?



# Triggers

- a trigger is a piece of SQL to execute either before or after an update, insert, or delete in a database.
- More dynamic compared to constraints
- eg: All new customers opening an account must have a balance of \$100; however, once the account is opened their balance can fall below that amount - constraints are not appropriate here



# Triggers

- Event - Condition - Action rules
- When an event occurs, check condition and if true take an action
- Application: (DRY) To move monitoring logic from the application to the DBMS
- More expressive constraints can be implemented using triggers
- Implementations vary across DBMS



All new students with CGPA $\geq$ 8 should automatically enrol for Seminar

```
create trigger R1
after insert on Student
for each row
when New.CGPA >= 8
begin
    insert into Course
    values (New.rollNo, 'Seminar', 'CSE',
null);
end;
```



# Referential Integrity Constraint using triggers

```
create trigger R2
after delete on Student
for each row
begin
    delete from Course
    where rollNo = Old.rollNo;
end;
```



## Key Constraint using triggers

```
create trigger R3
before insert on Student
for each row
when exists (select * from Student
where rollNo = New.rollNo)
begin
    select raise(ignore);
end;
```

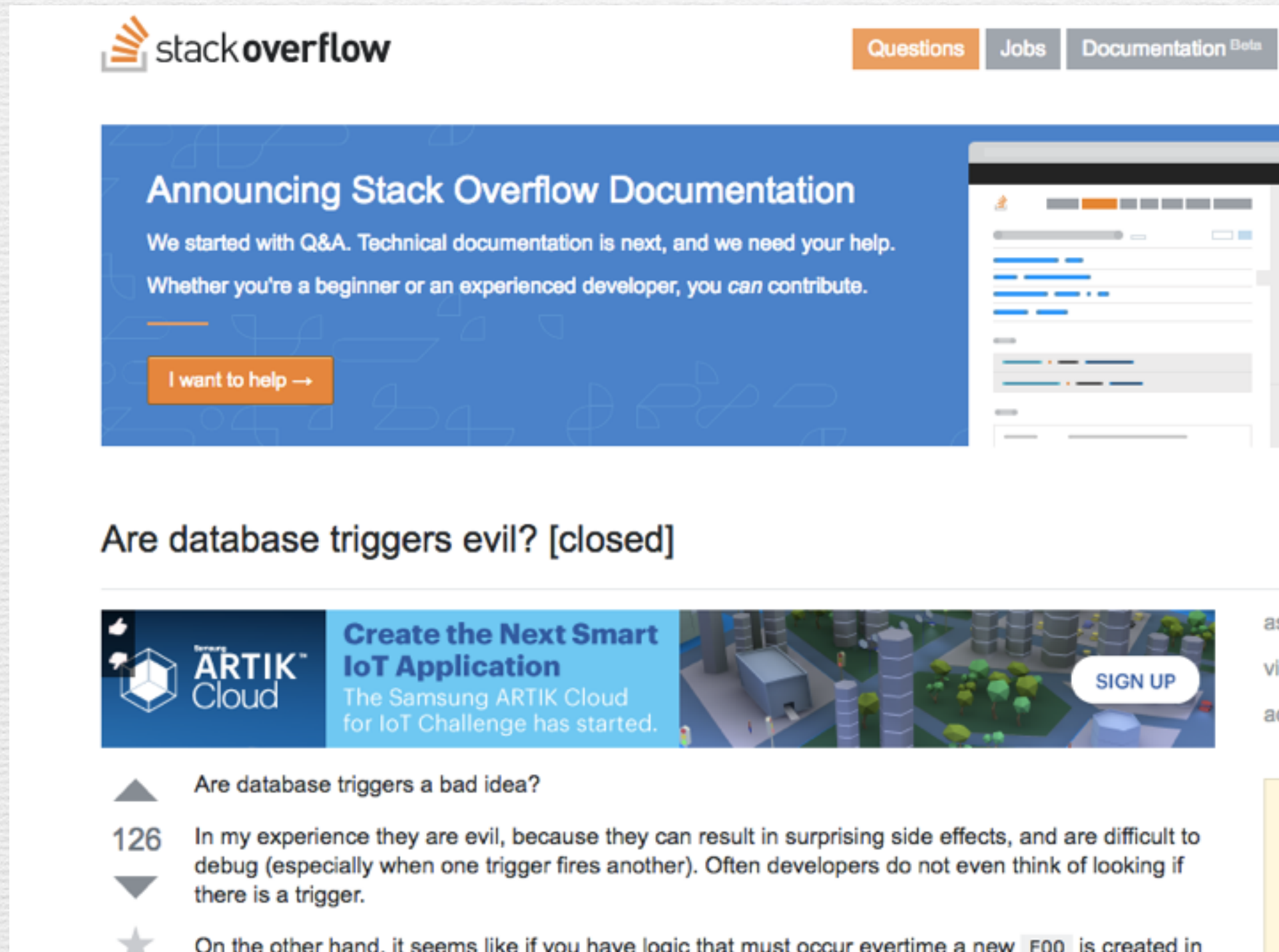


# Triggers

- Can be table level references as well - e.g: postgres
- SQLite and MySQL support only row level references - new and old rows
- Can be confusing - chaining of triggers should be avoided



# Exercise: Find out the relative merits and demerits of triggers



The screenshot shows the Stack Overflow homepage. At the top is the Stack Overflow logo and navigation links for Questions, Jobs, and Documentation Beta. A large blue banner announces the new documentation, stating that technical documentation is next and users are needed to help. Below the banner is a question titled "Are database triggers evil? [closed]". Under the question is a Samsung ARTIK Cloud advertisement. The question has 126 answers, with the first one stating that triggers are evil due to side effects and being difficult to debug. The second answer starts with "On the other hand, it seems like if you have logic that must occur everytime a new F00 is created in".

stackoverflow

Questions Jobs Documentation Beta

**Announcing Stack Overflow Documentation**

We started with Q&A. Technical documentation is next, and we need your help.

Whether you're a beginner or an experienced developer, you *can* contribute.

[I want to help →](#)

**Are database triggers evil? [closed]**

**Create the Next Smart IoT Application**  
The Samsung ARTIK Cloud for IoT Challenge has started.

[SIGN UP](#)

▲ Are database triggers a bad idea?

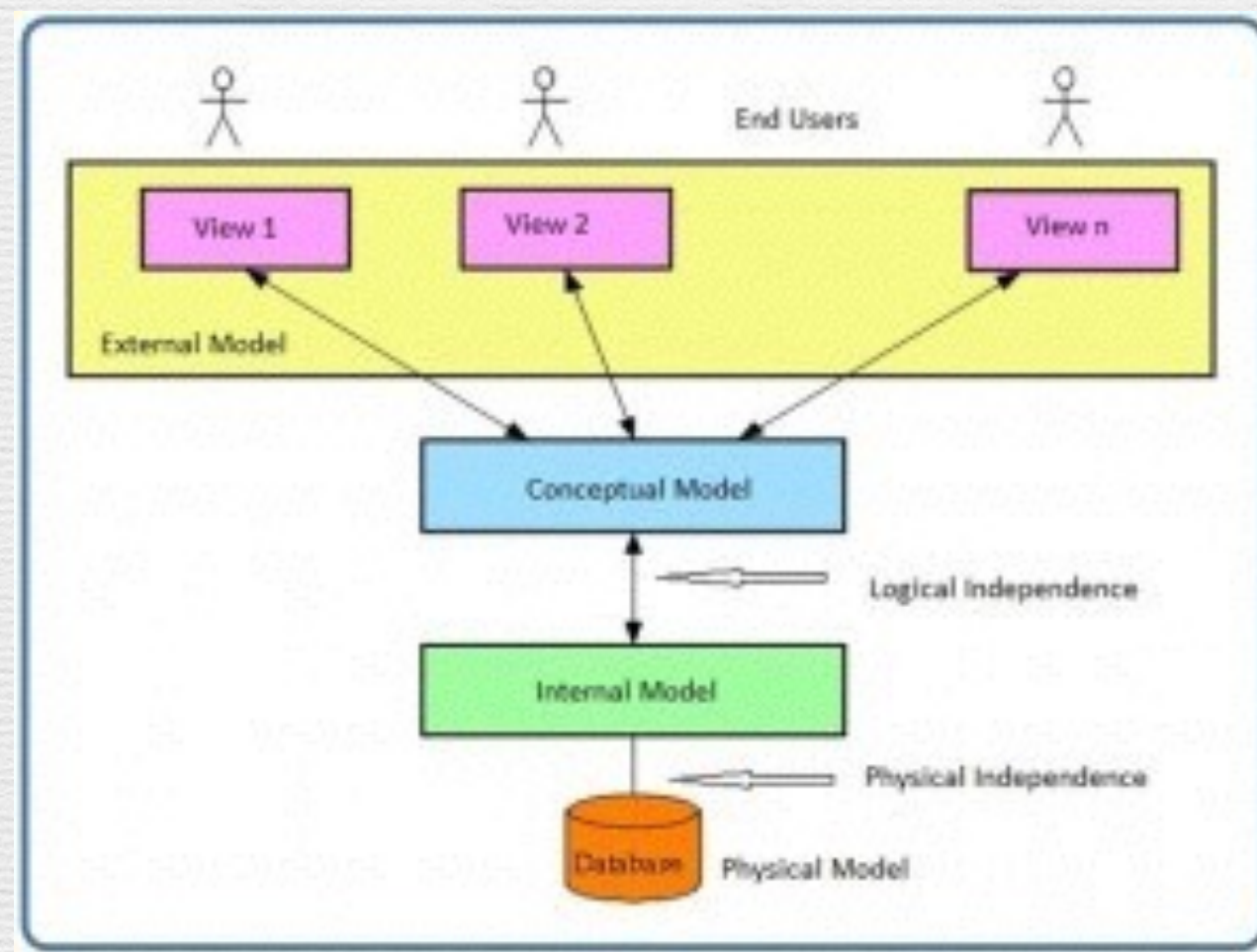
126 In my experience they are evil, because they can result in surprising side effects, and are difficult to debug (especially when one trigger fires another). Often developers do not even think of looking if there is a trigger.

★ On the other hand, it seems like if you have logic that must occur everytime a new F00 is created in



# Views

- Virtual Tables
- Physical - Conceptual - Logical Layers





```
create view CSenrollments as
select rollNo, cName
from Course
where dept = 'CSE';
```

- We can use the View CSenrollments like any other table



```
create view FCSstudents as
  select rollNo, name, cName
  from Student S, Course C
  where S.rollNo = C.rollNo
  and S.CGPA < 4.0
  and C.dept = 'CSE';
```



# Applications of Views

- Security - Give permission only on a view, not on the entire table (some fields (e.g.password) shouldn't be visible)
- Easiness to the application developers - eg; denormalised tables - Joined tables



# Views are not stored as Tables

- Views are not stored physically
- Only the code is stored
- So if a table is updates, it will be reflected in the view when we access it