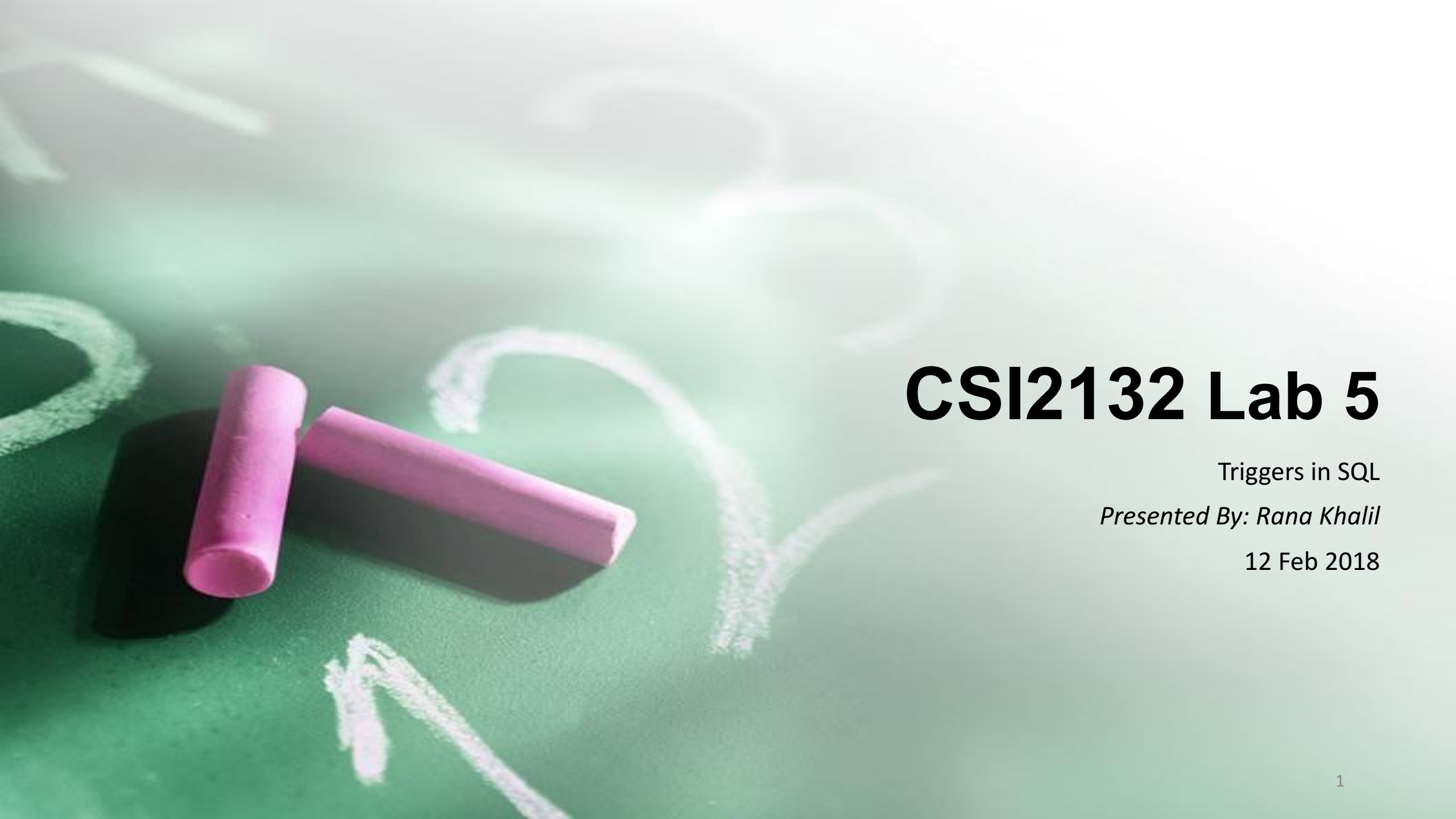


CSI2132 Lab 5

Triggers in SQL

Presented By: Rana Khalil

12 Feb 2018

Questions from Last Lab

- Casting Aggregate Functions

```
SELECT S.rating, AVG(S.age) AS avgage,
COUNT(*) numsailors
  ……
```

```
SELECT S.rating, CAST (AVG(S.age) AS
INTEGER) AS avgage, COUNT(*) numsailors
  ……..
```

	rating character varying (20)	avgage numeric	numsailors bigint
1	fair	18.0000000000000000	1
2	good	27.0000000000000000	2
3	poor	20.0000000000000000	2

	rating character varying (20)	avgage integer	numsailors bigint
1	fair	18	1
2	good	27	2
3	poor	20	2

- If a variable is set to integer however the output is a decimal does it get rounded up or down?

```
Select 9.5 as original, cast(9.5 as int);
```

```
Select 9.4 as original, cast(9.4 as int);
```

	original numeric	int4 integer
1	9.5	10

	original numeric	int4 integer
1	9.4	9

- Not Exists Explanation – covered that in the DGD

Outline

- **(30 min)** Revision and students have the chance to complete any missing part of lab 4 (if students need it). We will not use backup file this time.
- **(10 min)** Quiz on sailor DB
- **(40 min)** Lab 05 material, assertion and triggers. The first 5 pages of the lab will be self-reading.
  - Specifying Constraints in the Database Schema
  - Constraint Checking
  - Declaring Assertions in SQL (will not cover)
  - Declaring Triggers in SQL

Example

First, make sure you have the following tuples in each relation of the Sailors schema.
If not, create them accordingly

Sailors			
sid	sname	age	rating
1	Salvador	26	good
2	Rafael	28	good
3	John	10	good
4	Bruce	18	fair
5	James	17	fair
6	Smith	18	poor
7	Peter	22	poor

Boats	
bid	color
1	red
2	green
3	blue

Reserves	
sid	bid
1	1
1	2
2	1
7	1
7	2
7	3
4	1
3	3

Create a new schema named: “sailors”. Download and execute all the queries within: “Create Sailors DB” file

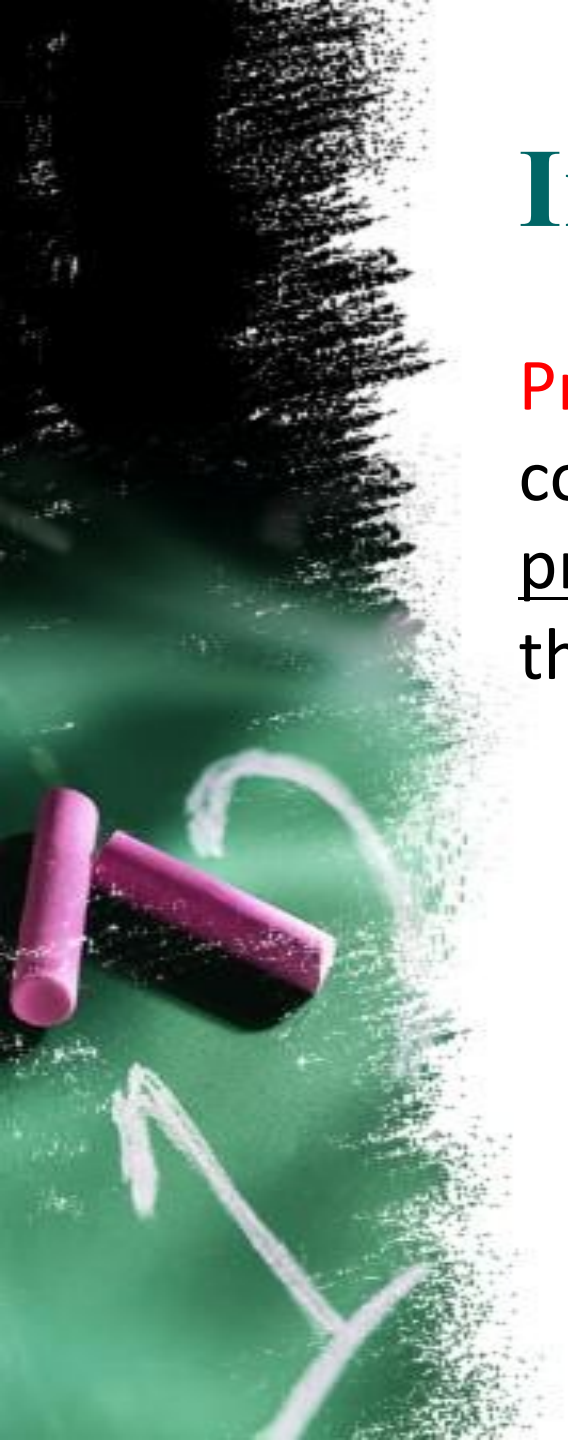
QUIZ TIME!

Integrity Constraints: a Recap

To ensure the validity of the data held at the database schema, the relational model defines different types of integrity constraints:

- Primary key constraints
- Foreign key constraints
- Referential integrity constraints
- Domain constraints
- General constraints

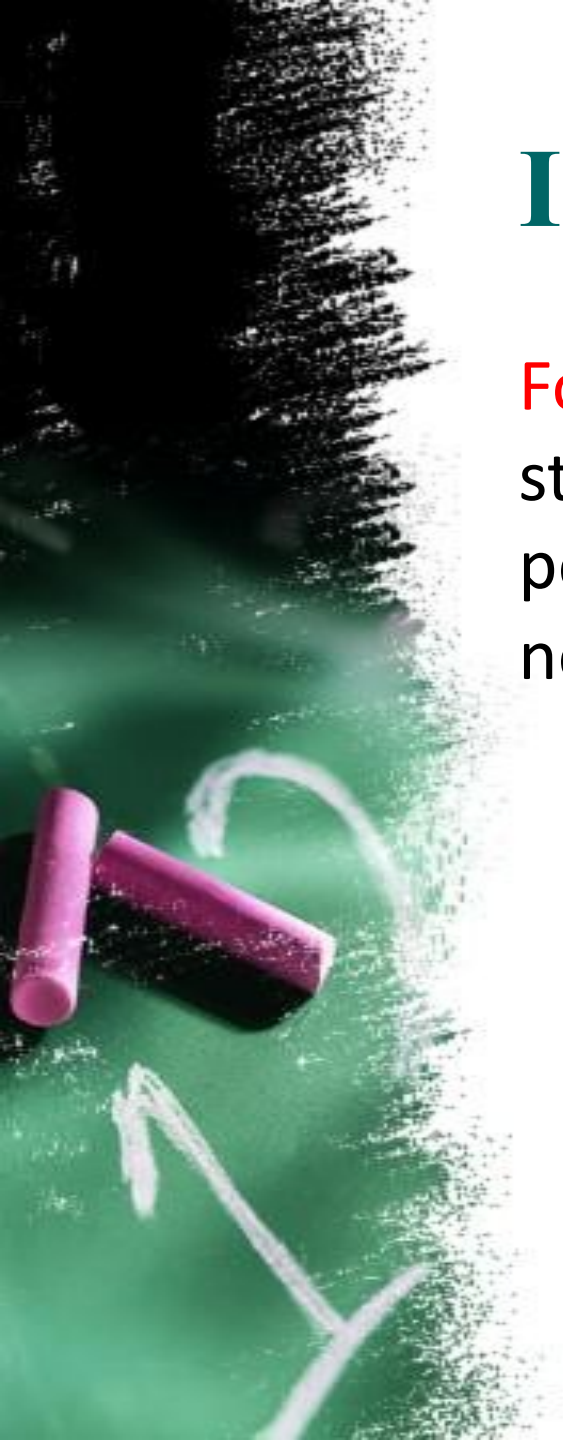
Operations that violate any integrity constraint at the tuple level are disallowed.



Integrity Constraints: a Recap

Primary key constraints: By default, DBMS checks that the combination of values for those attributes declared as primary key remains unique in the relation and that none of them are null.

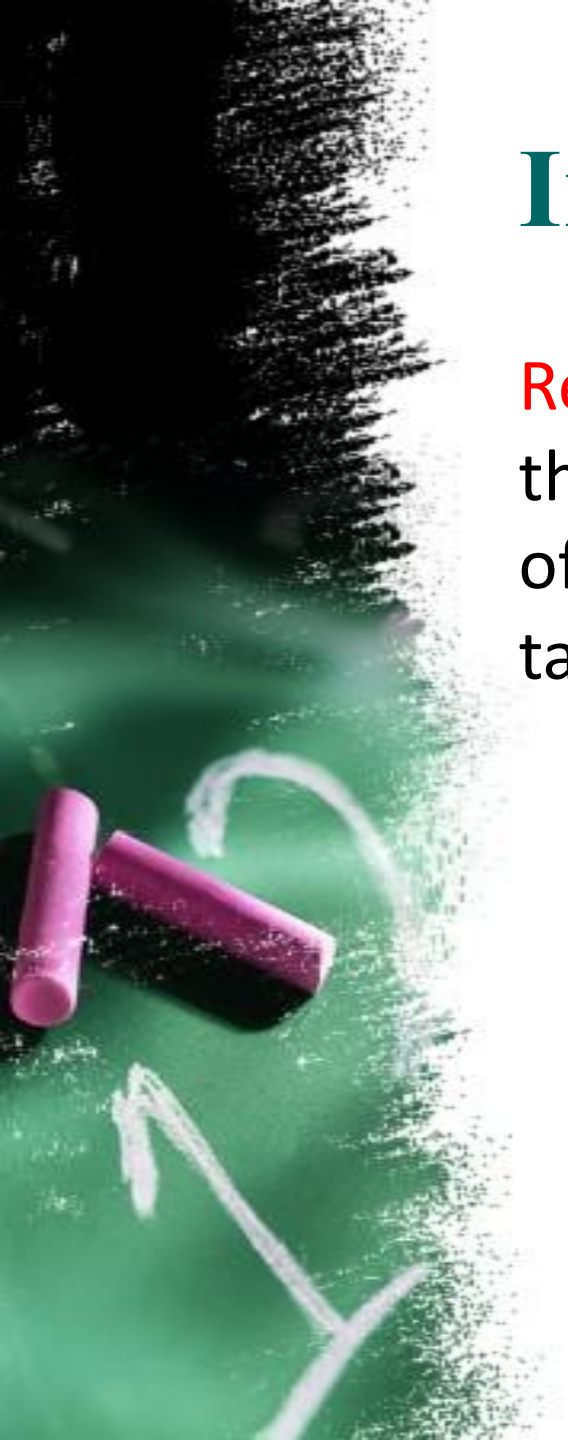
```
CREATE TABLE artist
(
  aname character varying(20) NOT NULL,
  birthplace character varying(20),
  style character varying(20),
  dateofbirth date,
  country character varying(20),
  CONSTRAINT "pk of artist" PRIMARY KEY (aname)
)
```



Integrity Constraints: a Recap

Foreign key constraints: Control what attribute values can be stored in the relation holding the foreign key field. It can point to a primary key attribute in another relation or to a non-PK attribute having the UNIQUE constraint.

```
CREATE TABLE artwork
(
  title character varying(20) NOT NULL,
  "year" integer,
  "type" character varying(20),
  price numeric(8,2),
  aname character varying(20),
  CONSTRAINT artwork_pkey PRIMARY KEY (title),
  CONSTRAINT artwork_aname_fkey FOREIGN KEY (aname)
      REFERENCES artist (aname) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
)
```



Integrity Constraints: a Recap

Referential integrity constraints: Specifies what happens to the tuples in the foreign relation when a deletion or update of a primary key attribute value is about to occur in the main table.

```
CREATE TABLE artwork
(
  title character varying(20) NOT NULL,
  "year" integer,
  "type" character varying(20),
  price numeric(8,2),
  aname character varying(20),
  CONSTRAINT artwork_pkey PRIMARY KEY (title),
  CONSTRAINT artwork_aname_fkey FOREIGN KEY (aname)
      REFERENCES artist (aname) MATCH SIMPLE
      ON UPDATE CASCADE ON DELETE CASCADE
)
```

Integrity Constraints: a Recap

Domain constraints: Restricts the set of values an attribute can take to lie within a particular domain.
A CHECK clause is added to the attribute definition.

```
CREATE TABLE customer
(
  custid integer NOT NULL,
  "name" character varying(20),
  address character varying(20),
  amount numeric(8,2),
  rating integer,
  CONSTRAINT customer_pkey PRIMARY KEY (custid),
  CONSTRAINT customer_rating_check CHECK (rating >= 1 AND rating <= 10)
)
```

Triggers in SQL

- It is important to monitor the database schema and take appropriate action upon the occurrence of an event.
- For example, if an artwork is appraised in over five million dollars, we notify the manager of the museum where it is located so as to enforce tighter anti-theft security measures.
- Or when an artist is removed from the database, we execute a stored procedure that forwards his/her customer list to a foreign server for cross validation.
- In such cases, we rely on triggers to notify about the event and take further action.

Components of a Trigger

- Event(s): An INSERT, UPDATE or DELETE operation on a particular tuple.
- Condition: Determines whether the action should be executed. If no condition is specified, the trigger is executed once the event takes place.
- Action: Usually a sequence of SQL statements, but could be also a database transaction or running an external program.

Triggers in PostgreSQL

```
CREATE TRIGGER name { BEFORE | AFTER } { event [ OR ... ] }
    ON table [ FOR [ EACH ] { ROW | STATEMENT } ]
    EXECUTE PROCEDURE funcname ( arguments )
```

- **name**: The name to give the new trigger. This must be distinct from the name of any other trigger for the same table.
- **BEFORE/AFTER**: Determines whether the function is called before or after the event.
- **event**: One of INSERT, UPDATE, or DELETE; this specifies the event that will fire the trigger. Multiple events can be specified using OR.
- **table**: The name of the table the trigger is for.
- **FOR EACH ROW/STATEMENT**: This specifies whether the trigger procedure should be fired once for every row affected by the trigger event, or just once per SQL statement. If neither is specified, FOR EACH STATEMENT is the default.
- **funcname**: A user-supplied function that is declared as taking no arguments and returning type trigger, which is executed when the trigger fires.
- **arguments**: An optional comma-separated list of arguments to be provided to the function when the trigger is executed.

Triggers in PostgreSQL

```
CREATE TRIGGER name {BEFORE | AFTER}
{ event [OR… ] } [OF attribute] ON table
[ FOR [EACH] {ROW | STATEMENT} ]
[ WHEN (condition) ]
EXECUTE PROCEDURE funcname(arguments)
```

Trigger Examples

Execute the check_sailor_rating_age( ) function whenever a row of the SAILORS table is about to be updated.

```
CREATE TRIGGER check_sailor
      BEFORE UPDATE ON sailors
      FOR EACH ROW
      EXECUTE PROCEDURE check_sailor_rating_age();
```

Trigger Examples

Same as before, but only if the sailor's rating is to be updated.

```
CREATE TRIGGER check_sailor
      BEFORE UPDATE OF rating ON sailors
      FOR EACH ROW
      EXECUTE PROCEDURE check_sailor_rating_age();
```

Trigger Examples

Same as before, but only if the sailor's age will in fact change its value. Notice the WHEN clause.

```
CREATE TRIGGER check_sailor
      BEFORE UPDATE ON sailors
      FOR EACH ROW
      WHEN (OLD.age IS DISTINCT FROM NEW.age)
      EXECUTE PROCEDURE check_sailor_rating_age();
```

Trigger Examples

Call a function to log any sailors' updates, but only if something changed:

```
CREATE TRIGGER log_sailor_update
   AFTER UPDATE ON sailors
   FOR EACH ROW
   WHEN (OLD.* IS DISTINCT FROM NEW.*)
   EXECUTE PROCEDURE log_sailor_update();
```

Trigger Procedures (functions)

- Must be defined before the CREATE TRIGGER statement can execute.
- Must be declared as a function taking no arguments and returning type trigger.
- Can be written in C (low-level) or PL-PGSQL (high-level)
- We will use PL-PGSQL (Procedural Language for PostgreSQL) from pgAdmin

Writing Trigger Procedures with PL-PGSQL

- Open the Query Editor and type the following:

```
CREATE FUNCTION check_sailor_name_age()
   RETURNS trigger AS
$BODY$
BEGIN

-- Check sailor's name
IF NEW.sname IS NULL THEN
        RAISE EXCEPTION 'The sailor must have a name';
END IF;

-- Check sailor's age
IF NEW.age > 50 THEN
        RAISE EXCEPTION 'The sailor must be 50 or below';
END IF;

RETURN NEW;

END

$BODY$ LANGUAGE plpgsql;
```

Writing Trigger Procedures with PL-PGSQL

- Open the Query Editor and type the following:

```
CREATE TRIGGER check_sailor
BEFORE UPDATE ON sailors
FOR EACH ROW
EXECUTE PROCEDURE check_sailor_name_age()
```

Writing Trigger Procedures with PL-PGSQL

Some variables that are often needed:

NEW = Holds the contents of the row to insert or update.

OLD = Holds the contents of the original row.

TG_NARGS = Number of input arguments passed on to the trigger procedure.

TG_ARGV[ ] = Text array containing the arguments, accessed as $1, $2, etc.

Writing Trigger Procedures with PL-PGSQL

Testing the Trigger Procedure

Go to the Edit Data window of the Sailors table.

Try to update Peter's age to 51. What do you get?

Now update it to 23. Was the operation successful?

Implementing Triggers

When does the event below happen?

There must never be more than two sailors traveling in the green boat because it is fragile and could sink.

- When a new tuple in the RESERVES relation is inserted or an existing one is updated.
- Then we create triggers accordingly.
- But first, we define the trigger procedure.

A Glimpse Into PL-PGSQL

Structure of the Main Code Block

```
CREATE FUNCTION identifier (arguments) RETURNS type AS '
  DECLARE
    declaration;
    [...]
  BEGIN
    statement;
    [...]
    RETURN { variable_name | value }
  END;' LANGUAGE 'plpgsql';
```

variables and constants here

Examples

```
id INTEGER;
title VARCHAR(10);
price FLOAT;
six CONSTANT INTEGER := 6;
ten INTEGER NOT NULL := 10;
```

A Glimpse Into PL-PGSQL

Assigning Query Results to Variables

```
SELECT INTO target_variable [, ...] target_column [, ...] select_clauses;

- Using the SELECT INTO statement

  DECLARE
    customer_fname varchar(100);
    customer_lname varchar(100);
  BEGIN
    SELECT INTO customer_fname, customer_lname
                first_name, last_name
           FROM customers;
    IF NOT FOUND THEN
      return -1;
    END IF;
    return 1;
  END;
```

variables

attributes

The Fragile Green Boat Procedure

Type it in the Query Tool and run it.

```
CREATE OR REPLACE FUNCTION check_green_boat()
  RETURNS trigger AS
$BODY$
DECLARE
        howMany INTEGER;
BEGIN
        IF NEW.bid = 2 THEN
                SELECT COUNT(*) INTO howMany FROM reserves WHERE bid = 2;
                IF howMany >= 2 THEN
                        RAISE EXCEPTION 'The green boat cannot be booked by more than two sailors.
                                         There are % thus far.', howMany;
                END IF;
        END IF;

        RETURN NEW;
END

$BODY$
  LANGUAGE plpgsql VOLATILE;
```

The Fragile Green Boat Procedure

Then declare the corresponding trigger.

```
CREATE TRIGGER green_boat
  BEFORE INSERT OR UPDATE
  ON reserves
  FOR EACH ROW
  EXECUTE PROCEDURE check_green_boat();
```

Now try it. Go to the Edit Data window for the RESERVES table and try to break the green boat rule by inserting a new record or updating an existing one in such a way that its capacity is overloaded.

ERROR: The green boat cannot be booked by more than two sailors. There are 2 thus far. CONTEXT: PL/pgSQL function sailors.check_green_boat() line 8 at RAISE .

Your turn: The Blue Boat Rule

Write the trigger and its corresponding PL-PGSQL procedure to enforce the blue boat rule.

The blue boat can only be booked by an inexperienced sailor if he travels along with an expert sailor.

- Inexperienced sailor = poor rating
- Expert sailor = good rating

Your turn: Artist DB

It is impossible to admire Caravaggio and not Josefa.

It happens when

- A Caravaggio fan is inserted -> add Josefa tuple
- A Caravaggio fan is deleted -> delete Josefa tuple
- A Josefa fan is inserted -> disallow insertion if no Caravaggio tuple for that fan is found
- A Josefa fan is updated -> disallow update if a Caravaggio tuple for that fan is found.
- A Josefa tuple is deleted -> disallow deletion if a Caravaggio tuple for that fan is found.

References

- Triggers in PostgreSQL

https://www.postgresql.org/docs/10/static/triggers.html

- Quick Intro to PL-PGSQL

https://www.codeproject.com/Articles/33734/How-to-write-PL-pgSQL-functions-for-PostgreSQL

- Basic PL-PGSQL statements

https://www.postgresql.org/docs/10/static/plpgsql-statements.html

- Creating trigger procedures with PL-PGSQL

https://www.postgresql.org/docs/10/static/plpgsql-trigger.html