CS2102

Database Systems

L08: Triggers

Requirement

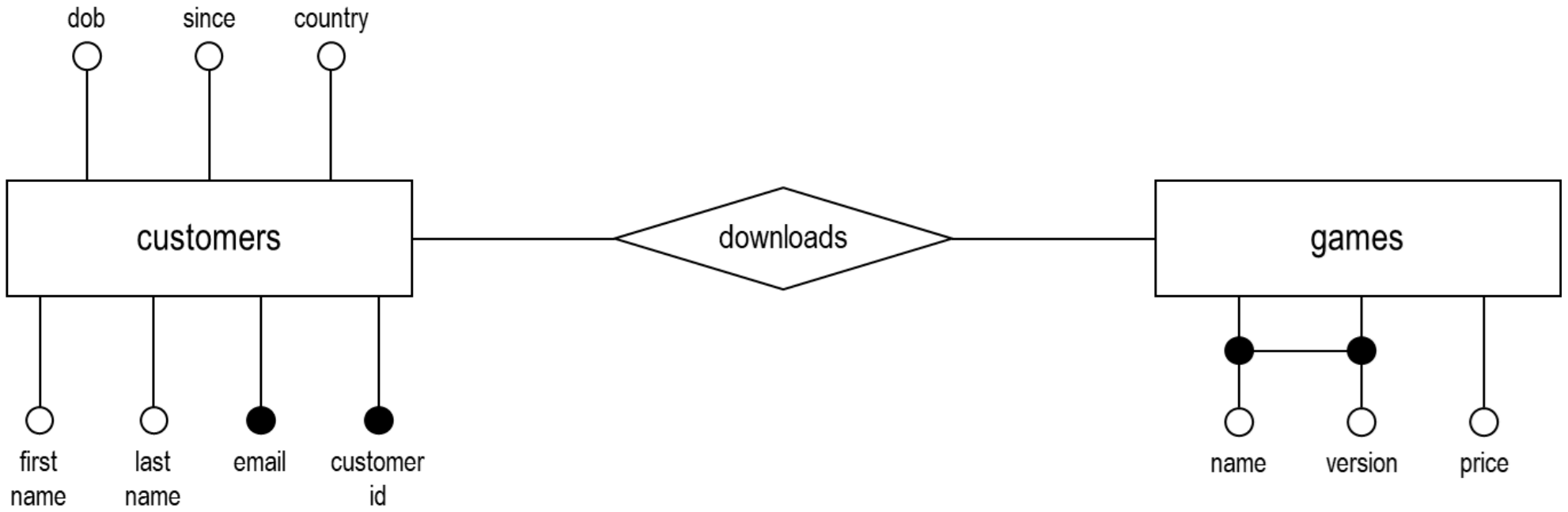
Game Store Requirement

Our company, **Apasaja Pte Ltd**, has been commissioned to develop an application to manage the data of an online app store. We want to store several items of information about our customers such as their **first name**, **last name**, **date of birth**, **e-mail**, **date** and **country of registration** to our online sales service and the **customer identifier** that they have chosen.

We also want to manage the list of our products, **games**, their **name**, their **version**, and their **price**. The price is fixed for each version of each game. Finally, our customers buy and **download** games. We record which version of which game each customer has downloaded. It is not essential to keep the download date for this application.

Design

Entity-Relationship Diagram



Constraint

Age Restrictions

Our company decides to enforce a suitability scheme to limit access to certain games based on their content to certain audiences based on their age.

For example, a customer who is not yet 21 cannot download the game 'Domainer'.

Stored Function

```
CREATE OR REPLACE FUNCTION is_r21()
RETURNS BOOLEAN AS $$
BEGIN
  -- code
END; $$ LANGUAGE plpgsql;
```

Constraint

Age Restrictions

Our company decides to enforce a suitability scheme to limit access to certain games based on their content to certain audiences based on their age.

For example, a customer who is not yet 21 cannot download the game 'Domainer'.

CHECK?

```
ALTER TABLE downloads
ADD CONSTRAINT is_r21
    CHECK (is_r21());
```

Check!

PostgreSQL checks the new constraints against existing data.

Test Cases

Fail?

The customer with identifier `Jonathan2000` is underaged.

```
INSERT INTO downloads VALUES ('Jonathan2000', 'Domainer', '1.0');
```

Success?

The customer with identifier `Jonathan2000` is underaged.

```
INSERT INTO downloads VALUES ('Jonathan2000', 'Aerified', '1.0');
```

Inconsistent State

No `INSERT/UPDATE` but allow `DELETE`.

Limitation

Stable Functions

The `CHECK` constraints require that functions be **immutable** or **stable**. In other words, they must **always** return the same result for the same inputs/queries.

Our function `is_r21()` depends on data from **multiple rows and tables**. So, it is neither immutable nor stable.

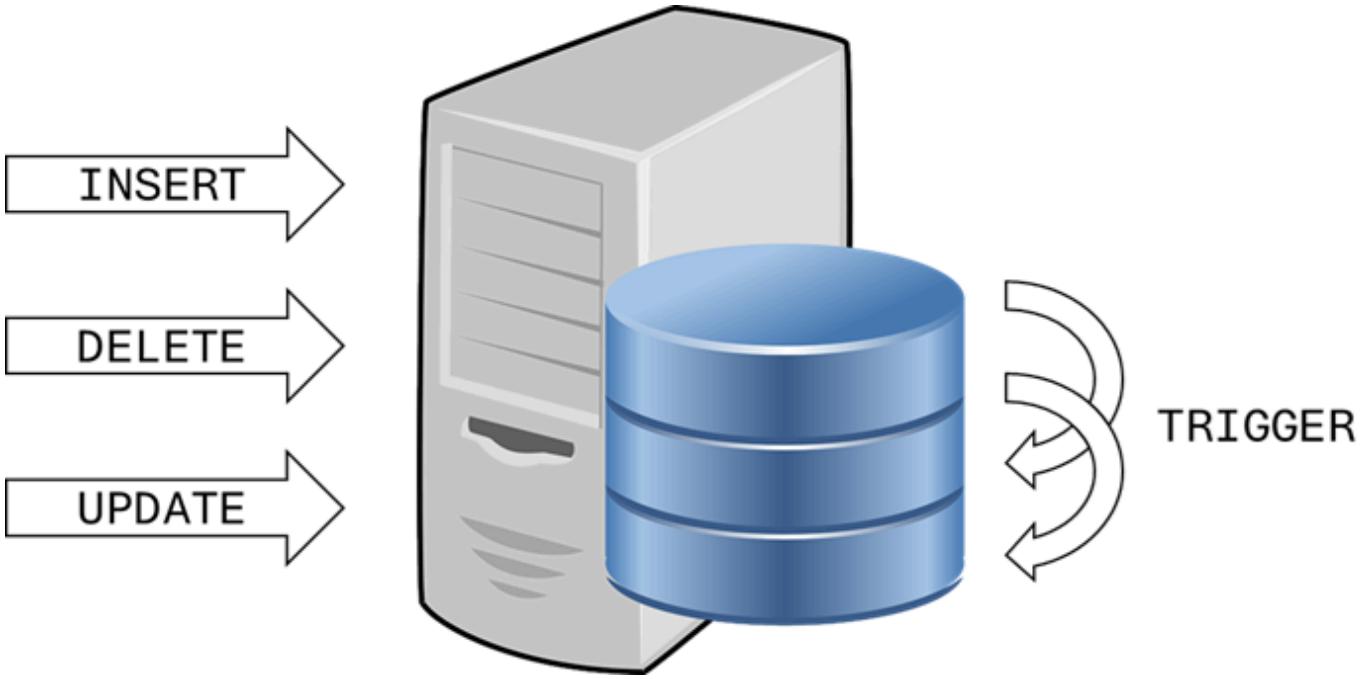
`CHECK` constraints also **cannot be deferred**. We want `is_r21()` to be checked after insertion and not before insertion.

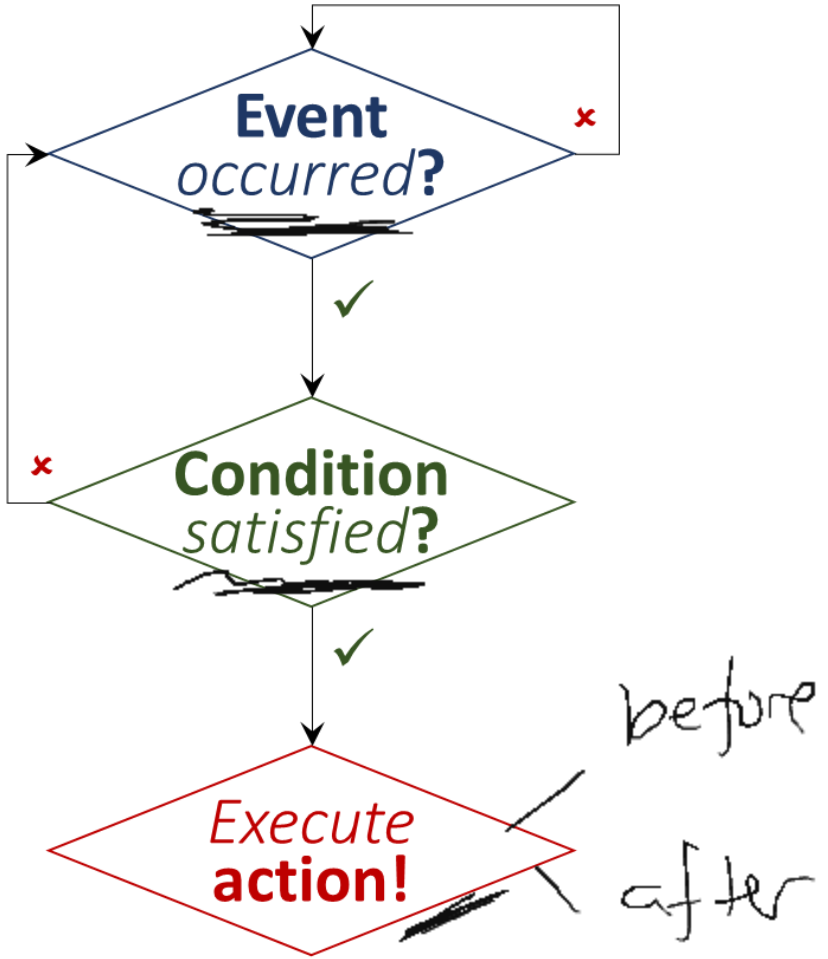
```
ALTER TABLE downloads
DROP CONSTRAINT is_r21;
```

Basic

Event-Condition-Action

A **trigger** is a procedure or function executed when a database **event** occurs on a table.





Basic

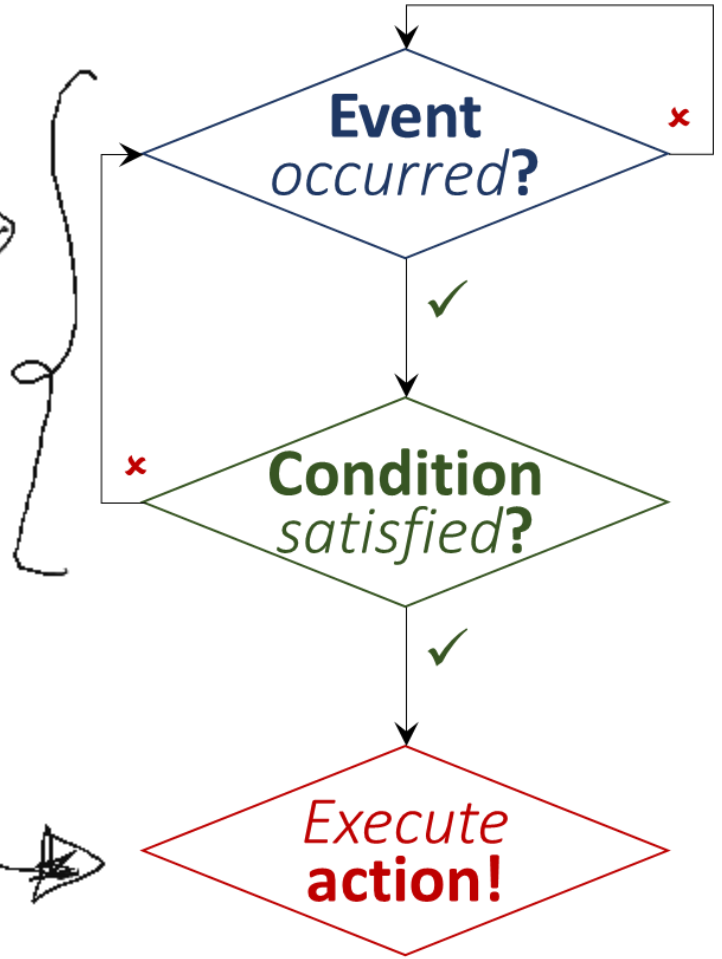
Event-Condition-Action

A **trigger** is a procedure or function executed when a database **event** occurs on a table.

Component

Trigger: Checking if event occurred.
- Binds the **trigger function** to operations on a **table** or a **view**.

Trigger Function: Action executed.
- The function invoked when a **statement** *(e.g., insert, update, or delete)* is executed.



Business Rule Trigger

Trigger Function

```
CREATE OR REPLACE FUNCTION fr21()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (
    SELECT c.customerid
    FROM customers c NATURAL JOIN downloads d
    WHERE d.name = 'Domainer'
      AND EXTRACT(year FROM AGE(c.dob)) < 21
  ) THEN
    RAISE EXCEPTION 'Underaged!'; -- STOP!
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger

```
CREATE CONSTRAINT TRIGGER tr21
AFTER INSERT OR UPDATE ON downloads
DEFERRABLE INITIALLY DEFERRED
FOR EACH ROW
EXECUTE PROCEDURE fr21();
```

Business Rule Trigger

Trigger Function

```
CREATE OR REPLACE FUNCTION fr21()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (
    SELECT c.customerid
    FROM customers c NATURAL JOIN downloads d
    WHERE d.name = 'Domainer'
      AND EXTRACT(year FROM AGE(c.dob)) < 21
  ) THEN
    RAISE EXCEPTION 'Underaged!'; -- STOP!
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger

```
CREATE CONSTRAINT TRIGGER tr21
AFTER INSERT OR UPDATE ON downloads
DEFERRABLE INITIALLY DEFERRED
FOR EACH ROW
EXECUTE PROCEDURE fr21();
```

Trigger Function

A *trigger function* must return a TRIGGER.

Business Rule Trigger

Trigger Function

```
CREATE OR REPLACE FUNCTION fr21()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (
    SELECT c.customerid
    FROM customers c NATURAL JOIN downloads d
    WHERE d.name = 'Domainer'
      AND EXTRACT(year FROM AGE(c.dob)) < 21
  ) THEN
    RAISE EXCEPTION 'Underaged!'; -- STOP!
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger

```
CREATE CONSTRAINT TRIGGER tr21
AFTER INSERT OR UPDATE ON downloads
DEFERRABLE INITIALLY DEFERRED
FOR EACH ROW
EXECUTE PROCEDURE fr21();
```

Trigger Event

The *trigger* checks for `INSERT` statement as well as `UPDATE` statement for the table `downloads`.

Business Rule Trigger

Trigger Function

```
CREATE OR REPLACE FUNCTION fr21()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (
    SELECT c.customerid
    FROM customers c NATURAL JOIN downloads d
    WHERE d.name = 'Domainer'
      AND EXTRACT(year FROM AGE(c.dob)) < 21
  ) THEN
    RAISE EXCEPTION 'Underaged!'; -- STOP!
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger

```
CREATE CONSTRAINT TRIGGER tr21
AFTER INSERT OR UPDATE ON downloads
DEFERRABLE INITIALLY DEFERRED
FOR EACH ROW
EXECUTE PROCEDURE fr21();
```

Trigger Timing and Granularity

The *trigger* checks **after** the statement is performed. It also checks **for changes to each row**.

Business Rule Trigger

Trigger Function

```
CREATE OR REPLACE FUNCTION fr21()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (
    SELECT c.customerid
    FROM customers c NATURAL JOIN downloads d
    WHERE d.name = 'Domainer'
      AND EXTRACT(year FROM AGE(c.dob)) < 21
  ) THEN
    RAISE EXCEPTION 'Underaged!'; -- STOP!
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger

```
CREATE CONSTRAINT TRIGGER tr21
AFTER INSERT OR UPDATE ON downloads
DEFERRABLE INITIALLY DEFERRED
FOR EACH ROW
EXECUTE PROCEDURE fr21();
```

Trigger Activation

The *trigger* currently do not check for additional conditions. It invoked the trigger function named `fr21()`.

Business Rule Trigger

Trigger Function

```
CREATE OR REPLACE FUNCTION fr21()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (
    SELECT c.customerid
    FROM customers c NATURAL JOIN downloads d
    WHERE d.name = 'Domainer'
      AND EXTRACT(year FROM AGE(c.dob)) < 21
  ) THEN
    RAISE EXCEPTION 'Underaged!'; -- STOP!
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger

```
CREATE CONSTRAINT TRIGGER tr21
AFTER INSERT OR UPDATE ON downloads
DEFERRABLE INITIALLY DEFERRED
FOR EACH ROW
EXECUTE PROCEDURE fr21();
```

Deferrable Trigger

The ***trigger*** is **deferrable** *(i.e., checked at end of transaction)*. Only **AFTER** trigger can be deferred. You also need the keyword **CONSTRAINT**.

Business Rule Trigger

Trigger Function

```
CREATE OR REPLACE FUNCTION fr21()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (
    SELECT c.customerid
    FROM customers c NATURAL JOIN downloads d
    WHERE d.name = 'Domainer'
      AND EXTRACT(year FROM AGE(c.dob)) < 21
  ) THEN
    RAISE EXCEPTION 'Underaged!'; -- STOP!
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

checks for violations

Trigger

```
CREATE CONSTRAINT TRIGGER tr21
AFTER INSERT OR UPDATE ON downloads
DEFERRABLE INITIALLY DEFERRED
FOR EACH ROW
EXECUTE PROCEDURE fr21();
```

(In)-Consistency Check

The *trigger function* checks if there are any inconsistent rows. The check is done via SQL query.

Business Rule Trigger

Trigger Function

```
CREATE OR REPLACE FUNCTION fr21()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (
    SELECT c.customerid
    FROM customers c NATURAL JOIN downloads d
    WHERE d.name = 'Domainer'
      AND EXTRACT(year FROM AGE(c.dob)) < 21
  ) THEN
    RAISE EXCEPTION 'Underaged!'; -- STOP!
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger

```
CREATE CONSTRAINT TRIGGER tr21
AFTER INSERT OR UPDATE ON downloads
DEFERRABLE INITIALLY DEFERRED
FOR EACH ROW
EXECUTE PROCEDURE fr21();
```

Rollback

The *trigger function* stops execution by raising an EXCEPTION. This causes the effect to be rolled back.

Business Rule Trigger

Trigger Function

```
CREATE OR REPLACE FUNCTION fr21()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (
    SELECT c.customerid
    FROM customers c NATURAL JOIN downloads d
    WHERE d.name = 'Domainer'
      AND EXTRACT(year FROM AGE(c.dob)) < 21
  ) THEN
    RAISE EXCEPTION 'Underaged!'; -- STOP!
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger

```
CREATE CONSTRAINT TRIGGER tr21
AFTER INSERT OR UPDATE ON downloads
DEFERRABLE INITIALLY DEFERRED
FOR EACH ROW
EXECUTE PROCEDURE fr21();
```

Return

This has no effect on AFTER trigger. It has effect on BEFORE trigger. We can even modify the rows to be operated on.

Business Rule Trigger

Trigger Function

```
CREATE OR REPLACE FUNCTION fr21()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (
    SELECT c.customerid
    FROM customers c NATURAL JOIN downloads d
    WHERE d.name = 'Domainer'
      AND EXTRACT(year FROM AGE(c.dob)) < 21
  ) THEN
    RAISE EXCEPTION 'Underaged!'; -- STOP!
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Trigger

```
CREATE CONSTRAINT TRIGGER tr21
AFTER INSERT OR UPDATE ON downloads
DEFERRABLE INITIALLY DEFERRED
FOR EACH ROW
EXECUTE PROCEDURE fr21();
```

OLD and NEW

`NEW` (or `OLD`) keyword refers to the what the row looks like **after** (or **before**) the operation. They are not always available.

Typical Scenario

Lawful Download

Should pass.

```
INSERT INTO downloads
VALUES ('Deborah84', 'Domainer', '1.0');

INSERT INTO downloads
VALUES ('Deborah84', 'Aerified', '1.0');

INSERT INTO downloads
VALUES ('Jonathan2000', 'Aerified', '1.0');
```

Unlawful Download

Should fail.

```
INSERT INTO downloads
VALUES ('Jonathan2000', 'Domainer', '1.0');

UPDATE downloads
SET name = 'Domainer'
WHERE customerid = 'Jonathan2000'
  AND name = 'Aerified';
```

Worst-Case Scenario

Lawful Updates

Should pass.

```
UPDATE customers
SET dob = '1990-01-01'
WHERE customerid = 'Jonathan2000';
```

Unlawful Download

Should fail.

```
UPDATE customers
SET dob = '2024-01-01'
WHERE customerid = 'Deborah84';
```

Other Cases?

- Can you think of **other cases**?
- Do you need to use a **different trigger function**?
- How do we know we have **covered all cases**?
- Is our trigger function **efficient**? *(Not an issue in our course.)*

Before vs After

Transition Variable

`NEW` contains the modified row **after** the triggering event.
`OLD` contains the modified row **before** the triggering event.

	NEW	OLD
`INSERT`	yes	no
`UPDATE`	yes	yes
`DELETE`	no	yes

Note

Note that not all make sense all the time.

However, if available, they are available on before and after the triggering event.

Break

Back by 12:46

Log Changes

Logging Operations

We want to know the operations performed on downloads. We can do this by creating trigger on `INSERT`, `UPDATE`, and `DELETE` statements and insert a log on another table.

```
CREATE OR REPLACE FUNCTION log_ops()
RETURNS TRIGGER AS $$
BEGIN
  INSERT INTO downloads_log VALUES
    (NEW.customerid, NEW.name,
     TG_OP, CURRENT_DATE);
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER op_log
AFTER INSERT OR DELETE OR UPDATE
  ON downloads
FOR EACH ROW
  EXECUTE FUNCTION log_ops();
```

TG_OP

`TG_OP` stores the operation.

Before Operation

Checking Age Before Download

Consider the business rule for Domainer from before. Can we check and prevent insertion before the operation?

```
CREATE OR REPLACE FUNCTION fr21()
RETURNS TRIGGER AS $$
BEGIN
  IF (NOT is_r21()) THEN

    RAISE EXCEPTION 'Underaged!'; -- STOP!
  END IF;
  RETURN NEW;
END; $$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER tr21
BEFORE INSERT
  ON downloads
FOR EACH ROW
  EXECUTE FUNCTION fr21();
```

Issue?

The data is still valid before insertion!

Before Operation

Checking Age Before Download

Consider the business rule for **Domainer** from before. Can we check and prevent insertion before the operation?

```
CREATE OR REPLACE FUNCTION fr21()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.name = 'Domainer' AND
     get_age(NEW.customerid) < 21 THEN
    RAISE EXCEPTION 'Underaged!'; -- STOP!
  END IF;
  RETURN NEW;
END; $$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER tr21
BEFORE INSERT
  ON downloads
FOR EACH ROW
  EXECUTE FUNCTION fr21();
```

Get Age?

We leave `get_age` as an exercise.

Quiz

Question

Can we use `fr21()` in the following trigger and still prevent insertion? Select all options that apply.

	Choice	Comment	
A	`AFTER INSERT` trigger on `downloads`		?
B	`AFTER UPDATE` trigger on `downloads`		?
C	`BEFORE UPDATE` trigger on `customers`		?
D	`BEFORE UPDATE` trigger on `downloads`		?

Instead Of

Instead Of Trigger

We can use `INSTEAD OF` trigger to perform operation on a `VIEW`. This allows us to insert/delete/update a non-updatable `VIEW`.

```
CREATE TRIGGER update_copy_view
INSTEAD OF UPDATE ON copy_view     -- from tutorial 01
FOR EACH ROW EXECUTE FUNCTION update_copy_view_function();
```

Before Operation

Checking Age Before Download

Consider the business rule for Domainer from before. Can we check and prevent insertion before the operation? Can we NOT raise exception?

```
CREATE OR REPLACE FUNCTION fr21()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.name = 'Domainer' AND
     get_age(NEW.customerid) < 21 THEN
    RETURN NULL;  -- STOP!
  END IF;
  RETURN NEW;
END; $$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER tr21
BEFORE INSERT
  ON downloads
FOR EACH ROW
  EXECUTE FUNCTION fr21();
```

Question

What about non-NULL result?

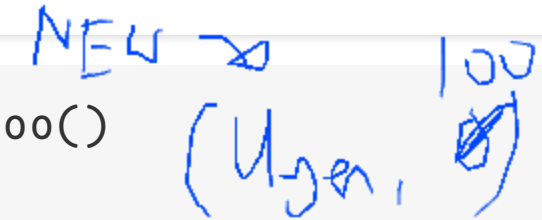
Effect of Return Value

	NULL Tuple	Non-NULL Tuple `t`
`BEFORE INSERT`	No tuple inserted	Tuple `t` will be inserted
`BEFORE UPDATE`	No tuple updated	Tuple `t` will be the updated tuple
`BEFORE DELETE`	No tuple deleted	Deletion proceed as normal
`AFTER ...`	**No effect**	

Quiz

Question

Consider the following trigger and trigger function. What is the effect of the trigger?




```
CREATE OR REPLACE FUNCTION foo()
RETURNS TRIGGER AS $$
BEGIN
  IF (NEW.Name = 'Uyen') THEN
    NEW.Mark := 100;
  END IF;   RETURN NEW;
END;  $$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER bar
BEFORE INSERT ON Scores
FOR EACH ROW
EXECUTE FUNCTION foo();
```

Name	Mark

	Choice	Comment	
A	Uyen will always get 100		?
B	No tuple inserted		?
C	No effect *(i.e., normal insertion)*		?

Quiz

Question

Consider the following trigger and trigger function. What is the effect of the trigger?

```
CREATE OR REPLACE FUNCTION foo()
RETURNS TRIGGER AS $$
BEGIN
  IF (NEW.Name = 'Uyen') THEN
    NEW.Mark := 100;
  END IF;   RETURN NULL;
END;  $$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER bar
BEFORE INSERT ON Scores
FOR EACH ROW
EXECUTE FUNCTION foo();
```

Name	Mark

	Choice	Comment	
A	Uyen will always get 100		?
B	No tuple inserted		?
C	No effect *(i.e., normal insertion)*		?

Quiz

Question

Consider the following trigger and trigger function. What is the effect of the trigger?

```
CREATE OR REPLACE FUNCTION foo()
RETURNS TRIGGER AS $$
BEGIN
  IF (NEW.Name = 'Uyen') THEN
    NEW.Mark := 100;
  END IF;   RETURN OLD;
END;  $$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER bar
BEFORE INSERT ON Scores
FOR EACH ROW
EXECUTE FUNCTION foo();
```

Name	Mark

	Choice	Comment	
A	Uyen will always get 100		?
B	No tuple inserted		?
C	No effect *(i.e., normal insertion)*		?

IF vs WHEN

Motivation

Consider the following check. This is done when the trigger function is invoked.

```
CREATE OR REPLACE FUNCTION foo()
RETURNS TRIGGER AS $$
BEGIN
  IF (NEW.Name = 'Uyen') THEN
    NEW.Mark := 100;
  END IF;   RETURN NEW;
END;  $$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER bar
BEFORE INSERT ON Scores
FOR EACH ROW
EXECUTE FUNCTION foo();
```

IF vs WHEN

Motivation

We can move the check to the trigger. In this case, trigger function is not invoked.

```
CREATE OR REPLACE FUNCTION foo()
RETURNS TRIGGER AS $$
BEGIN
  NEW.mark := 100;
END;  $$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER bar
BEFORE INSERT ON Scores
FOR EACH ROW
WHEN (NEW.name = 'Uyen')
EXECUTE FUNCTION foo();
```

WHEN Condition

Requirement

In general, the condition in `WHEN()` can be more complicated.

Subject to the following requirements.

- `NO SELECT` in `WHEN()`.
- `NO OLD` in `WHEN()` for `INSERT`.
- `NO NEW` in `WHEN()` for `DELETE`.
- `NO WHEN()` for `INSTEAD OF`.

Row vs Statement

Trigger Granularity

We can execute the trigger function for ***every tuple*** operated using **row-level trigger**.

We can execute the trigger function ***once per statement*** using **statement-level trigger**.

Note

A statement *(e.g., `INSERT`/`UPDATE`/`DELETE`)* may affect multiple rows.

Motivating Example

Consider the `downloads_log` table from before. What if we want to prevent deletion from log.

Statement Level

Motivating Example

Consider the `downloads_log` table from before. What if we want to prevent deletion from log.

```
CREATE OR REPLACE FUNCTION no_delete()
RETURNS TRIGGER AS $$
BEGIN
  RAISE EXCEPTION 'No delete from log...';
END; $$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER warn_delete
BEFORE DELETE ON downloads_log
FOR EACH STATEMENT
EXECUTE FUNCTION no_delete();
```

Note

Since we're preventing the operation, once is enough.

Granularity and Timing

Timing	Row-Level	Statement-Level
`AFTER`	Tables Only	Tables and Views
`BEFORE`	Tables Only	Tables and Views
`INSTEAD OF`	Views Only	not allowed

NEW/OLD are not available

Activation Order

Multiple Triggers

There can be multiple triggers defined for the same event on the same table.

The general order of activation is as follows.
1. `BEFORE` statement-level trigger.
2. `BEFORE` row-level trigger.
3. `AFTER` row-level trigger.
4. `AFTER` statement-level trigger.

Within Category

Within each category, the trigger is activated in alphabetical order.

Quiz

Question

Consider the following trigger functions and its triggers. Which trigger function is executed first?

Trigger Function	Trigger
`aardvark`	`AFTER` row-level
`check`	`BEFORE` row-level
`precheck`	`BEFORE` statement-level
`aaron_checker`	`BEFORE` row-level

	Choice	Comment	
A	`aardvark`		?
B	`check`		?
C	`precheck`		?
D	`aaron_check`		?

```
postgres=# exit
```

```
Press any key to continue . . .
```