Mis evaluables corregidas:

1.Consultas:

```
-- 1. Show the attacks found by employees working with hacker whose alias starts with the
--    letter C, the alias of the hacker and a field with the full name of the employee with
--    their alias in this format:  name surname, "alias". Sort the result alphabetically by
--    that field, type_code and at_type.

SELECT
   CONCAT(employee.name_emp, ' ', employee.surname_emp, ', "', employee.alias, '"') AS
NombreCompleto,
   hacker.alias AS AliasHacker,
   attack.index_at,
   attack.type_code
FROM
   employee
JOIN
   hacker ON employee.alias = hacker.contact
JOIN
   attack ON employee.alias = attack.found_by
WHERE
   hacker.alias LIKE 'C%'
ORDER BY
   NombreCompleto,
   attack.type_code;

-- 2. Show the name of the target and the average severity of all effects caused by attacks
--    on that target, only if the average is greater than 60%. Sort the result from highest
--    to lowest average severity.

2 consulta A -- GROUP BY
SELECT name_ta, AVG(severity) AS severity_avg
FROM effect E, target T
WHERE E.ip_target=T.ip_target AND E.service=T.service
GROUP BY E.ip_target, E.service, name_ta
HAVING AVG(severity)>0.6
ORDER BY 2 DESC;
2 consulta B -- Derived table with GROUP BY
SELECT name_ta, severity_avg
FROM target T, (SELECT E.ip_target, E.service,
             AVG(severity) AS severity_avg
             FROM effect E
             GROUP BY E.ip_target, E.service) S
WHERE T.ip_target=S.ip_target AND T.service=S.service
AND severity_avg>0.6
ORDER BY 2 DESC;
```

```
-- 3. Display the aliases of employees who have never worked with another employee,
--    sorted by the last name of the employee. Use the following methods:


3 consulta A CON  -- NOT IN with DISTINCT and UNION
SELECT
   alias,
   surname_emp
FROM
   employee
WHERE
   alias NOT IN (SELECT DISTINCT emp1 FROM collaborate UNION SELECT DISTINCT
emp2 FROM collaborate)
ORDER BY
   surname_emp;

3 consulta B --  NOT EXISTS
SELECT
   alias,
   surname_emp
FROM
   employee e
WHERE
   NOT EXISTS (SELECT 1 FROM collaborate c WHERE c.emp1 = e.alias OR c.emp2 = e.alias)
ORDER BY
   surname_emp;

-- 4. Show the attacks, their timestamps, and the first and last name of their
--    finders (if any) of **all** attacks of the same type as the most recent
--    attack, except this one. Sort the results by surname and from most recent
--    to oldest.
Consulta 4 --MAX
SELECT
   attack.type_code
   attack.index_at,
   attack.timestamp_at,
   employee.name_emp,
   employee.surname_emp
FROM
   attack
 LEFT JOIN
   employee ON attack.found_by = employee.alias
WHERE
   attack.type_code = (SELECT timestamp_at FROM attack
                WHERE timestamp_at = (SELECT MAX(timestamp_at) FROM attack))
   AND attack.timestamp_at <> (SELECT timestamp_at FROM attack
                WHERE timestamp_at = (SELECT MAX(timestamp_at) FROM attack))
ORDER BY
   employee.surname_emp,
   attack.timestamp_at DESC;
```

```
-- 5. List how many attacks each target has received and sort the targets
--    from least to most attacked.
CONSULTA 5 --Mucho mas simple que la mia

SELECT ip_target, service, COUNT(DISTINCT type_at,index_at) AS n_attacks
FROM effect E
GROUP BY ip_target, service
ORDER BY 3;
-- Now list the average number of attacks suffered by targets according to
-- their service.
SELECT E.service, AVG(n_attacks) as avg_attacks
FROM effect E, (SELECT service, COUNT(DISTINCT type_at,index_at) AS n_attacks
              FROM effect E
              GROUP BY ip_target, service) A
WHERE E.service=A.service
GROUP BY service;

/* Create a procedure that displays the attacks that have been found by a given employee
on the same day that they worked in collaboration with another employee. The procedure must
receive an employee *alias* and display the attack identifier, the *timestamp_at* of the attack
and the *alias* of the employee who worked with them that day. */

 CREATE OR REPLACE VIEW attacks_found AS
SELECT E.alias AS employee, H.alias AS hacker, COUNT(found_by) AS at_found
FROM hacker H, employee E, attack
WHERE H.contact=E.alias AND E.alias=found_by
AND H.dni IS NULL
GROUP BY E.alias,H.alias
ORDER BY COUNT(found_by) DESC;

-- With VIEW
-- Subquery with >= ALL

SELECT employee, at_found
FROM attacks_found
WHERE at_found >= ALL (SELECT at_found
                    FROM attacks_found);

-- Without VIEW

-- Subquery with >= ALL (HAVING)
SELECT E.alias, COUNT(found_by) AS at_found
FROM hacker H, employee E, attack A
WHERE E.alias=found_by AND H.contact=E.alias
AND H.dni IS NULL
GROUP BY E.alias
HAVING COUNT(found_by) >= ALL (SELECT COUNT(found_by)
                           FROM hacker H, employee E, attack A
                           WHERE E.alias=found_by AND H.contact=E.alias
                           AND H.dni IS NULL
                           GROUP BY E.alias);
```

Funciones y procedures corregidas:

```
/* SCRIPT 1
   Create a function that returns the number of times a target has been attacked.
   The function must receive the target identifier and return the number of attacks,
   or raise an error if the target does not exist. */

DELIMITER $$
DROP FUNCTION IF EXISTS  get_attack_count$$
CREATE FUNCTION get_attack_count(input_ip_target VARCHAR(15), service_target
VARCHAR(20))
RETURNS INT
NOT DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE attack_count INT;
    SELECT COUNT((DISTINCT type_at, index_at)) FROM effect WHERE ip_target =
input_ip_target AND service = service_target INTO attack_count;

  IF (SELECT COUNT(*) FROM target WHERE ip_target=input_ip_target AND
service=service_target) = 0
    THEN
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT =
       '\nUnexpected parameter\n*****\n====> Target does not exists.\n*****\n';
  END IF;

    RETURN attack_count;
END$$


Select get_attack_count('104.26.10.78', 'PaaS');

/* SCRIPT 2
   Create a function that receives a timestamp and returns only the date part of it.
   If the timestamp is not valid the function must raise an error. */
"2 FUNCTION TIEMPO"
 DELIMITER $$
DROP FUNCTION IF EXISTS get_date_part //
CREATE FUNCTION get_date_part(p_timestamp_at TIMESTAMP)
RETURNS DATE
DETERMINISTIC
BEGIN
  IF p_timestamp_at IS NULL
  THEN
     SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT =
     '\nUnexpected parameter\n*****\n====> Timestamp is NULL.\n*****\n';
  END IF;
  RETURN DATE(p_timestamp_at);
END$$
DELIMITER ;
```

```
SELECT get_date_part('2021-06-01 12:30:45') AS date_part; -- 2021-06-01
SELECT get_date_part(NULL) AS date_part; -- User Error
SELECT get_date_part('Hi') AS date_part; -- Default error
```

```
/* Create a procedure that displays the attacks that have been found by a given employee
on the same day that they worked in collaboration with another employee. The procedure must
receive an employee *alias* and display the attack identifier, the *timestamp_at* of the attack
and the *alias* of the employee who worked with them that day. */
```

```
"ATAQUE COLABORADOR CON UNION"
DELIMITER $$
DROP PROCEDURE IF EXISTS Ataque_Colaborador$$
CREATE PROCEDURE Ataque_Colaborador(IN employee_alias VARCHAR(20))
BEGIN
IF (SELECT alias FROM employee WHERE alias=pi_alias) IS NULL
  THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT =
    '\nUnexpected parameter\n*****\n====> Employee does not exists.\n*****\n';
  END IF;
   (
      SELECT
         a.index_at AS Identificador,
         a.type_code AS Codigo
         a.timestamp_at AS Fecha_Ataque,
         c.emp2 AS colaborador
      FROM
         attack a
      JOIN
         collaborate c ON DATE(a.timestamp_at) = c.day_col
      WHERE
         a.found_by = employee_alias AND c.emp1 = employee_alias
   )
   UNION
   (
      SELECT
         a.index_at AS Identificador,
         a.type_code AS Codigo
         a.timestamp_at AS Fecha_Ataque,
         c.emp1 AS colaborador
      FROM
         attack a
      JOIN
         collaborate c ON DATE(a.timestamp_at) = c.day_col
      WHERE
         a.found_by = employee_alias AND c.emp2 = employee_alias
   );
END$$
DELIMITER ;
```

```
Call Ataque_Colaborador('FireWall');
INSERT INTO collaborate VALUES
```

```
('CyberGata','FireWall','2023-02-01');
INSERT INTO attack VALUES
('XSS',10,'Cybergata','2023-02-01 16:14:18');


/* SCRIPT 3
  Create a procedure that receives a *type_code* and displays a report about the attacks of that
type.
  For each attack, the report must display a table with the *index_at*, *timestamp_at*, *found_by*
  (1 row) and a second table with the *eff_description*, *severity* and target identifier for each
  effect of the attack. */

DELIMITER $$
DROP PROCEDURE IF EXISTS Reporte_de_Ataque_Loop$$
CREATE PROCEDURE Reporte_de_Ataque_Loop(IN codigo_tipo VARCHAR(5))
BEGIN
   DECLARE v_current_id INT DEFAULT 1;
   DECLARE v_max_id INT;

IF (SELECT code_type FROM at_type WHERE code_type=codigo_tipo) IS NULL
  THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT =
    '\nUnexpected parameter\n*****\n====> Attack type does not exists.\n*****\n';
  END IF;

   SELECT MAX(index_at) INTO v_max_id FROM attack WHERE type_code = codigo_tipo;
   WHILE v_current_id <= v_max_id DO
      SELECT
         a.index_at AS Identificador,
         a.timestamp_at AS Fecha_Ataque,
         a.found_by AS Descubridor
      FROM
         attack AS a
      WHERE
         a.type_code = codigo_tipo AND a.index_at = v_current_id;

      SELECT
         e.eff_description AS Descripcion_Efecto,
         e.severity AS Severidad
      FROM
         effect AS e
      WHERE
        type_at = codigo_tipo AND index_at = v_current_id;
      SET v_current_id = v_current_id + 1;
   END WHILE;
END$$
DELIMITER ;

CALL Reporte_de_Ataque_Loop('DDos');
```

```
/* SCRIPT 4
   Create a procedure to insert a new attack. The procedure must receive a *type_code* and insert the new
   attack with the current timestamp and the *index_at* field being the next number for that type of attack.
   The founder of the attack must be the user that is executing the procedure, without host part. If the user
   is not an employee, then *found_by* will be empty. Make the necessary arrangements to check the procedure
   works correctly.

   If the *type_code* is not valid, the procedure must raise an error.
   The procedure must "return" 1 if the attack has been inserted by an employee and 0 if it was not.
*/

/*SOLUCION DE PAU*/
DELIMITER //

/* We will use get_user() function developed in the unit */
DROP FUNCTION IF EXISTS get_user//
CREATE FUNCTION get_user ()
RETURNS CHAR(20)
DETERMINISTIC NO SQL
BEGIN
  RETURN SUBSTRING_INDEX(USER(), '@', 1);
END//
DROP PROCEDURE IF EXISTS insert_attack //
CREATE PROCEDURE insert_attack(pi_type VARCHAR(5), OUT po_ret BOOLEAN)
BEGIN
  DECLARE v_index INT DEFAULT 0;
  DECLARE v_host VARCHAR(20);
  START TRANSACTION;
  IF (SELECT code_type FROM at_type WHERE code_type=pi_type) IS NULL
  THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT =
    '\nUnexpected parameter\n*****\n====> Attack type does not exists.\n*****\n';
  END IF;
  /* This way v_host will contain the user if it is an employee, otherwise it will be null */
  SELECT alias INTO v_host FROM employee WHERE alias=get_user();
  /* We get the number of attacks for the type, we assume there are no gaps in the index */
  SELECT COUNT(index_at) INTO v_index FROM attack WHERE type_code=pi_type;

 INSERT INTO attack VALUES (pi_type, v_index+1, v_host, NOW());
  COMMIT;
  IF v_host IS NULL
  THEN
    SET po_ret = 0;
  ELSE
    SET po_ret = 1;
  END IF;
END //
```

```
CALL insert_attack('SQLI', @ret)// -- Error
SELECT @ret// -- NULL
CALL insert_attack('RCE', @ret)// -- index_at 1 and found_by is NULL
SELECT * FROM attack WHERE type_code='RCE' ORDER BY index_at//
SELECT @ret// -- 0 Your user is not an employee
-- Create user Neo and give permissions to execute any procedure in database
CREATE USER 'Neo'@'localhost' IDENTIFIED BY 'Neo'//
GRANT EXECUTE ON db_csd.* TO 'Neo'@'localhost'//
-- After switching to user Neo, insert an attack
 -- index_at 7 and found_by is Neo
SELECT @ret; -- 1 Neo is an employee
/* To check the attack table must switch back to your user or give Neo
   SELECT permission on at least attack table */
SELECT * FROM attack WHERE type_code='SQLi' ORDER BY index_at;
```