Base de Datos 1

Práctica 2

1) Crear usuarios para las bases de datos, usando los nombres reparacion para la versión normalizada y reparacion_dn para la desnormalizada. Habiendo creado estos usuarios, evitar el uso de root para el resto del trabajo práctico.

```
CREATE USER 'reparacion'@'%' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON reparacion.* TO 'reparacion'@'%';

CREATE USER 'reparacion_dn'@'%' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON reparacion_dn.* TO 'reparacion_dn'@'%';
```

1.1) Adicionalmente, en ambas bases:

- cree un usuario sólo con permisos para realizar consultas de selección, es decir que no puedan realizar cambios en la base. Use los nombres `reparacion_select` y `reparacion_dn_select`.

```
CREATE USER 'reparacion_select'@'%' IDENTIFIED BY 'password';
GRANT SELECT ON reparacion.* TO 'reparacion_select'@'%';

CREATE USER 'reparacion_dn_select'@'%' IDENTIFIED BY 'password';
GRANT SELECT ON reparacion_dn.* TO 'reparacion_dn_select'@'%';
```

- cree un usuario que pueda realizar consultas de selección, actualización y eliminación a nivel de filas, pero que no puedan modificar el esquema. Use los nombres `reparacion_update` y `reparacion_dn_update`.

```
CREATE USER 'reparacion_update'@'%' IDENTIFIED BY 'password';
GRANT SELECT, DELETE, UPDATE ON reparacion.* TO 'reparacion_update'@'%';

CREATE USER 'reparacion_dn_update'@'%' IDENTIFIED BY 'password';
GRANT SELECT, DELETE, UPDATE ON reparacion_dn.* TO 'reparacion_dn_update'@'%';
```

- cree un usuario que tenga los permisos de los anteriores, pero que además pueda modificar el esquema de la base de datos. Use los nombres `reparacion_schema` y `reparacion_dn_schema`.

```
CREATE USER 'reparacion_schema'@'%' IDENTIFIED BY 'password';
GRANT SELECT, DELETE, UPDATE, CREATE, DROP, INSERT, ALTER ON reparacion.* TO
'reparacion_schema'@'%';

CREATE USER 'reparacion_dn_schema'@'%' IDENTIFIED BY 'password';
GRANT SELECT, DELETE, UPDATE, CREATE, DROP, INSERT, ALTER ON reparacion_dn.* TO
'reparacion_dn_schema'@'%';
```

2) Listar dni, nombre y apellido de todos los clientes ordenados por dni en forma ascendente. Realice la consulta en ambas bases. ¿Qué diferencia nota en cuanto a performance? ¿Arrojan los mismos resultados? ¿Qué puede concluir en base a las diferencias halladas?

```
USE reparacion;
SELECT dniCliente, nombreApellidoCliente FROM cliente ORDER BY dniCliente ASC;
--> 20000 rows in set (0.01 sec)

USE reparacion_dn;
SELECT dniCliente, nombreApellidoCliente FROM reparacion ORDER BY dniCliente ASC;
--> 162252 rows in set (0.07 sec)

SELECT DISTINCT dniCliente, nombreApellidoCliente FROM reparacion ORDER BY dniCliente
ASC;
--> 20000 rows in set (1.63 sec)
```

Se puede observar una gran diferencia en el tiempo de ejecución de las consultas y se aprecia una diferencia en la cantidad de resultados. Esto es por la presencia de dependencias multivaluadas en el esquema no normalizado, las cuales generan duplicación de información.

3) Hallar aquellos clientes que para todas sus reparaciones siempre hayan usado su tarjeta de crédito primaria (nunca la tarjeta secundaria). Realice la consulta en ambas bases.

```
USE reparacion;
SELECT dniCliente, nombreApellidoCliente
FROM cliente c WHERE NOT EXISTS (
    SELECT * FROM reparacion r
    WHERE c.dniCliente = r.dniCliente AND c.tarjetaSecundaria = r.tarjetaReparacion
);
--> 11976 rows in set (0.05 sec)

USE reparacion_dn;
SELECT DISTINCT dniCliente, nombreApellidoCliente
FROM reparacion r1 WHERE NOT EXISTS (
    SELECT * FROM reparacion r2
    WHERE r1.dniCliente = r2.dniCliente AND r1.tarjetaSecundaria = r2.tarjetaReparacion
);
--> 11976 rows in set (0.74 sec)
```

4) Crear una vista llamada `sucursalesPorCliente` que muestre los dni de los clientes y los códigos de sucursales de la ciudad donde vive el cliente. Cree la vista en ambas bases.

```
USE reparacion;
CREATE VIEW sucursalesPorCliente AS
    SELECT c.dniCliente, s.codSucursal
    FROM cliente c INNER JOIN sucursal s ON (c.ciudadCliente = s.ciudadSucursal);
--> Query OK, 0 rows affected (0.01 sec)

USE reparacion_dn;
CREATE VIEW sucursalesPorCliente
    AS SELECT c.dniCliente, s.codSucursal
    FROM reparacion r WHERE (r.ciudadCliente = r.ciudadSucursal)
--> Query OK, 0 rows affected (0.01 sec)
```

5) En la base normalizada, hallar los clientes que dejaron vehículos a reparar en todas las sucursales de la ciudad en la que viven

Restricción: resolver este ejercicio sin usar la cláusula “NOT EXIST”.

Nota: limite su consulta a los primeros 100 resultados, caso contrario el tiempo que tome puede ser excesivo.

```
USE reparacion;
```

a) Realice la consulta sin utilizar la vista creada en el ej 4.

```
SELECT c.dniCliente
FROM cliente c
WHERE c.dniCliente NOT IN (
    SELECT c.dniCliente
    FROM sucursal s
    WHERE s.ciudadSucursal = c.ciudadCliente
        AND s.codSucursal <> All (
            SELECT r.codSucursal
            FROM reparacion r
            WHERE r.ciudadReparacionCliente = c.ciudadCliente
                AND c.dniCliente = r.dniCliente
    )
)
LIMIT 100;
--> 100 rows in set (0.00 sec)
-- 13007 rows in set (0.12 sec)
```

b) Realice la consulta utilizando la vista creada en el ej 4.

```
SELECT c.dniCliente
FROM cliente c
WHERE c.dniCliente NOT IN  (
    SELECT s.dniCliente
    FROM sucursalesPorCliente s
    WHERE s.dniCliente = c.dniCliente
        AND s.codSucursal <> ALL (
            SELECT r.codSucursal
            FROM reparacion r
            WHERE r.ciudadReparacionCliente = c.ciudadCliente
                AND r.dniCliente = s.dniCliente
    )
)
LIMIT 100;
--> 100 rows in set (0.01 sec)
-- 13007 rows in set (0.74 sec)
```

Se encontro un gran boost de performance de orden 10 en comparación a las consultas usando count.

6) Hallar los clientes que en alguna de sus reparaciones hayan dejado como dato de contacto el mismo domicilio y ciudad que figura en su DNI. Realice la consulta en ambas bases.

```
USE reparacion;

Select DISTINCT c.dniCliente FROM cliente c INNER JOIN reparacion r ON (r.dniCliente =
c.dniCliente) WHERE (r.direccionReparacionCliente = c.domicilioCliente AND
r.ciudadReparacionCliente = c.ciudadCliente;
--> 18307 rows in set (0.05 sec)

Select DISTINCT c.dniCliente FROM cliente c INNER JOIN reparacion r ON (r.dniCliente =
c.dniCliente AND r.direccionReparacionCliente = c.domicilioCliente AND
r.ciudadReparacionCliente = c.ciudadCliente;
--> 18307 rows in set (0.04 sec)

SELECT dniCliente
FROM cliente
WHERE EXISTS
(SELECT *
        FROM reparacion
    WHERE cliente.domicilioCliente = reparacion.direccionReparacionCLiente
       AND cliente.ciudadCliente = reparacion.ciudadReparacionCliente
       AND cliente.dniCliente = reparacion.dniCliente);
--> 18307 rows in set (0.05 sec)

USE reparacion_dn;
Select DISTINCT dniCliente FROM reparacion WHERE direccionReparacionCliente =
domicilioCliente AND ciudadReparacionCliente = ciudadCliente;
--> 18307 rows in set (0.09 sec)
```

7) Para aquellas reparaciones que tengan registrados mas de 3 repuestos, listar el DNI del cliente, el código de sucursal, la fecha de reparación y la cantidad de repuestos utilizados. Realice la consulta en ambas bases.

```
USE reparacion;

SELECT r.dniCliente, r.codSucursal, r.fechaInicioReparacion, COUNT(DISTINCT
re.repuestoReparacion)
FROM reparacion r INNER JOIN repuestoreparacion re ON (
    r.fechaInicioReparacion = re.fechaInicioReparacion
    AND r.dniCliente = re.dniCliente
)
GROUP BY r.dniCliente, r.fechaInicioReparacion
HAVING COUNT(DISTINCT re.repuestoReparacion) > 3;
--> 3964 rows in set (0.09 sec)

USE reparacion_dn;
SELECT dniCliente, codSucursal, fechaInicioReparacion, COUNT(DISTINCT
repuestoReparacion)
FROM reparacion
GROUP BY dniCliente, fechaInicioReparacion
HAVING COUNT(DISTINCT repuestoReparacion) > 3;
--> 3964 rows in set (0.16 sec)
```

En la base normalizada realice los siguientes ejercicios:

```
USE reparacion;
```

8) Agregar la siguiente tabla:

```
REPARACIONESPORCLIENTE:
   idRC: *int(11)* **PK** *AI*
   dniCliente: *int(11)*
   cantidadReparaciones: *int(11)*
   fechaultimaactualizacion: *datetime*
   usuario: *char(16)*
```

```
CREATE TABLE REPARACIONESPORCLIENTE (
    idRC INT NOT NULL AUTO_INCREMENT,
    dniCliente INT NOT NULL,
    cantidadReparaciones INT NOT NULL,
    fechaultimaactualizacion DATE NOT NULL,
    usuario CHAR(16) NOT NULL,
    PRIMARY KEY (idRC)
);
--> Query OK, 0 rows affected (0.01 sec)
```

9) Stored procedures

a) Crear un stored procedure que realice los siguientes pasos dentro de una transacción:

- Realizar una consulta que para cada cliente (dniCliente), calcule la cantidad de reparaciones que tiene registradas. Registrar la fecha en la que se realiza la consulta y el usuario con el que la realizó.
- Guardar el resultado de la consulta en un cursor.
- Iterar el cursor e insertar los valores correspondientes en la tabla REPARACIONESPORCLIENTE.

```
CREATE PROCEDURE logReparaciones()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE cant INT;
    DECLARE dni INT;
    DECLARE cur CURSOR FOR
        SELECT count(*) as cant, dniCliente
        FROM reparacion GROUP BY dniCliente;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    START TRANSACTION;
    OPEN cur;
    loop_1: LOOP
        FETCH cur INTO cant, dni;
        IF done THEN
            LEAVE loop_1;
        END IF;
        INSERT INTO REPARACIONESPORCLIENTE
            (dniCliente, cantidadReparaciones, fechaultimaactualizacion, usuario)
            VALUES (dni, cant, NOW(), CURRENT_USER);
    END LOOP;
    CLOSE cur;
    COMMIT;
END;
--> Query OK, 0 rows affected (0.00 sec)
```

b) Ejecute el stored procedure.

```
CALL logReparaciones();
--> Query OK, 0 rows affected (0.41 sec)
```

10) Crear un trigger de modo que al insertar un dato en la tabla `REPARACION`, se actualice la cantidad de reparaciones del cliente, la fecha de actualización y el usuario responsable de la misma (actualiza la tabla `REPARACIONESPORCLIENTE`).

```
CREATE TRIGGER after_reparacion_insert
    AFTER INSERT ON reparacion
    FOR EACH ROW
BEGIN
        IF (NEW.dniCliente IN (SELECT dniCliente FROM reparacionesporcliente))
        THEN
            UPDATE REPARACIONESPORCLIENTE
            SET cantidadReparaciones = cantidadReparaciones + 1,
                fechaultimaactualizacion = NOW(), usuario = CURRENT_USER()
            WHERE NEW.dniCliente = REPARACIONESPORCLIENTE.dniCliente;
        ELSE
            INSERT INTO reparacionesporcliente
                (dniCliente, cantidadReparaciones, fechaultimaactualizacion, usuario)
                VALUES (NEW.dniCliente, 1, NOW(), CURRENT_USER());
        END IF;
END;
```

11) Crear un stored procedure que sirva para agregar una reparación, junto con una revisión de un empleado (`REVISIONREPARACION`) y un repuesto (`REPUESTOREPARACION`) relacionados dentro de una sola transacción. El stored procedure debe recibir los siguientes parámetros: `dniCliente`, `codSucursal`, `fechaReparacion`, `cantDiasReparacion`, `telefonoReparacion`, `empleadoReparacion`, `repuestoReparacion`.

```
CREATE PROCEDURE addRep(IN dni INT(11), IN sucursal INT, IN fecha DATETIME,
    IN dias INT, IN tel VARCHAR(45), IN empleado VARCHAR(30), IN repuesto VARCHAR(30))
BEGIN
    DECLARE domicilio VARCHAR(255);
    DECLARE ciudad VARCHAR(255);
    DECLARE tarjeta VARCHAR(255);

    SELECT domicilioCliente, ciudadCliente, tarjetaPrimaria
        INTO domicilio, ciudad, tarjeta
    FROM cliente WHERE cliente.dniCliente = dni;

    START TRANSACTION;
        INSERT INTO reparacion
            (codSucursal, dniCliente, fechaInicioReparacion, cantDiasReparacion,
            telefonoReparacionCliente,direccionReparacionCliente,
            ciudadReparacionCliente, tarjetaReparacion)
        VALUES (sucursal, dni, fechaReparacion, dias, tel, domicilio, ciudad, tarjeta);

        INSERT INTO revisionreparacion
            (dniCliente, fechaInicioReparacion, empleadoReparacion)
            VALUES (dni, fechaReparacion, empleado);

        INSERT INTO repuestoreparacion
            (dniCliente, fechaInicioReparacion, repuestoReparacion)
            VALUES (dni, fechaReparacion, repuesto);
    COMMIT;
END;
```

12) Ejecutar el stored procedure del punto 11 con los siguientes datos:

- dniCliente: 1009443
- codSucursal: 100
- fechaReparacion: 2013-12-14 12:20:31
- empleadoReparacion: 'Maidana'
- repuestoReparacion: 'bomba de combustible'
- cantDiasReparacion: 4
- telefonoReparacion: 4243-4255

```
CALL addRep(1009443, 100, "2013-12-14 12:20:31", 4, "4243-4255", "Maidana", "bomba de
combustible");
```

13) Realizar las inserciones provistas en el archivo inserciones.sql. Validar mediante una consulta que la tabla REPARACIONESPORCLIENTE se esté actualizando correctamente.

```
mysql -u reparacion -ppassword reparacion < ./inserciones.sql
```

14) Considerando la siguiente consulta

```
EXPLAIN select count(r.dniCliente) from reparacion r, cliente c, sucursal s,
revisionreparacion rv where r.dnicliente=c.dnicliente
and r.codsucursal=s.codsucursal
and r.dnicliente=rv.dnicliente
and r.fechainicioreparacion=rv.fechainicioreparacion
and empleadoreparacion = 'Maidana'
and s.m2<200
and s.ciudadsucursal='La Plata';
```

Analice su plan de ejecución mediante el uso de la sentencia EXPLAIN.

```
| id | select_type | table | partitions | type   | possible_keys | key     | key_len |
ref                                                                 | rows  | filtered |
Extra                                          |
+----+-------------+-------+------------+--------+---------------+---------+---------+--
-------------------------------------------------------------------+-------+----------+----
-----------------------------------------------+
|  1 | SIMPLE      | s     | NULL       | ALL    | PRIMARY       | NULL    | NULL    |
NULL                                                                |    18 |     5.56 |
Using where                                    |
|  1 | SIMPLE      | r     | NULL       | ALL    | PRIMARY       | NULL    | NULL    |
NULL                                                                | 39174 |    10.00 |
Using where; Using join buffer (Block Nested Loop) |
|  1 | SIMPLE      | c     | NULL       | eq_ref | PRIMARY       | PRIMARY | 4       |
reparacion.r.dniCliente                                             |     1 |   100.00 |
Using index                                    |
|  1 | SIMPLE      | rv    | NULL       | eq_ref | PRIMARY       | PRIMARY | 101     |
reparacion.r.dniCliente,reparacion.r.fechaInicioReparacion,const |     1 |   100.00 |
Using index                                    |
+----+-------------+-------+------------+--------+---------------+---------+---------+--
-------------------------------------------------------------------+-------+----------+----
-----------------------------------------------+
4 rows in set, 1 warning (0.00 sec)
```

a) ¿Qué atributos del plan de ejecución encuentra relevantes para evaluar la performance de la consulta?

Para esto, debemos ver la cantidad de filas escaneadas, y también si se han usado índices. Si no hubiera indices, se recorrerían todos los registros de la tabla para obtener el resultado.

En el campo REF vemos que una fila está en NULL, y las otras muestran atributos.

(REF: Las columnas del índice que se está usando, o una constante si esta es posible.)

Por ende, vemos que en la fila que está en NULL no se está usando ninguna columna, por ende ningún index.

b) Observe en particular el atributo type ¿cómo se están aplicando los JOIN entre las tablas involucradas?

Index: Escaneo completo de la tabla para cada combinación de filas de las tablas previas, revisando únicamente el índice.

Ref: Todas las filas con valores en el índice que coincidan serán leídos desde esta tabla por cada combinación de filas de las tablas previas. Similar a eq_ref, pero usado cuando usa sólo un prefijo más a la izquierda de la clave o si la clave no es UNIQUE o PRIMARY KEY. Si la clave que es usada coincide sólo con pocas filas, esta union es buena.

Eq_ref: Una fila de la tabla 1 será leída por cada combinación de filas de la tabla 2. Este tipo es usado cuando todas las partes de un índice se usan en la consulta y el índice es UNIQUE o PRIMARY KEY.

c) Según lo que observó en los puntos anteriores, ¿qué mejoras se pueden realizar para optimizar la consulta?

Primero que nada, vemos que la fila en la cual se observa mayor cantidad de rows, es la fila en la cual se usa 'index', y 'ref' es null.

Deberíamos usar indices.

d) Aplique las mejoras propuestas y vuelva a analizar el plan de ejecución. ¿Qué cambios observa?

Como se puede observar, elegí crear distintos indices, como también modificar la consulta. Respecto a esta consulta en particular, solo con agregar el índice "empleado", la cantidad de rows disminuyó mucho. Sin embargo, se añadieron otros índices debido a que ante un hipotético crecimiento de la base de datos en cuanto a datos que contiene, podrían beneficiarme en otras consultas.

A su vez, también modifiqué la consulta. Para empezar, notamos que la tabla "cliente" era innecesaria. El único atributo que se usaba era dniCliente, el cuál también está presente en reparación. Tener una tabla menos nos va a facilitar la consulta, en el sentido de que se deben cruzar menos tablas y por ende habrá menos operaciones.

Varias cuestiones a analizar en el WHERE en la consulta original se han movido al FROM. Esto es debido a que estoy intentando que en resultados intermedios de la consulta, me queden conjuntos más chicos (es decir, estoy filtrando). Por ende, las últimas operaciones se hacen sobre conjuntos mas chicos, y por ende, tenemos mejor performance.

```
CREATE INDEX sucursal_ciudad_m2
    ON sucursal (ciudadsucursal, m2);
CREATE INDEX empleadoreparacion_revisionreparacion
    ON revisionreparacion (empleadoreparacion);
CREATE INDEX reparacion_fechaInicioReparacion
    ON reparacion (fechaInicioReparacion);

EXPLAIN
    SELECT COUNT(r.dniCliente)
    FROM reparacion r INNER JOIN
        (SELECT codsucursal FROM sucursal WHERE m2<200 AND ciudadsucursal='La Plata') s
            ON(r.codsucursal = s.codsucursal) INNER JOIN
        revisionreparacion rv
            ON(r.dnicliente=rv.dnicliente AND
r.fechainicioreparacion=rv.fechainicioreparacion)
    WHERE empleadoreparacion = 'Maidana';
```

15) Análisis de permisos.

a) Para cada punto de la práctica incluido en el cuadro, ejecutarlo con cada uno de los usuarios creados en el punto 1 e indicar con cuáles fue posible realizar la operación.

Punto	r	rdn	r_select	rdn_select	r_update	rdn_update	r_schema	rdn_schem
2	○	○	○	○	□	□	□	□
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								

b) Determine para cada caso, cuál es el conjunto de permisos mínimo.

c) Desde su punto de vista y contemplando lo visto en la materia, explique cuál es la manera óptima de asignar permisos a los usuarios.