Documentación: Iteración 2 – Sistemas Transaccionales

## RFC1 - MOSTRAR EL DINERO RECOLECTADO POR SERVICIOS EN CADA HABITACIÓN EN EL ÚLTIMO AÑO CORRIDO

1. **Creación Índice:**
   Con 800,000 registros en la base de datos, la creación de un índice podría mejorar significativamente el rendimiento de la consulta, especialmente si se ejecuta de manera recurrente por parte de usuarios administrativos como un gerente.

2. **Justificación Índice desde la selectividad:**
   La selectividad se refiere a qué tan bien un índice puede filtrar los registros relevantes. Si un índice puede reducir significativamente el conjunto de datos a revisar, será más selectivo y útil. En consultas que involucran sumas acumulativas y agrupaciones por habitación, y considerando que las habitaciones podrían ser muchas, pero no tanto como los registros de servicios y reservas, un índice en las tablas de reservas y servicios basado en las fechas y en los ID asociados podría ser altamente selectivo y por tanto muy útil para acelerar la consulta.

3. **Tipo de índice:**
   Basándonos en las dos consideraciones anteriores, sería prudente crear un índice compuesto. Para esta consulta específica, podríamos beneficiarnos de un índice compuesto que incluya las columnas utilizadas en las claves foráneas y las fechas que son utilizadas para filtrar registros dentro del rango del último año.

   Índice:
   CREATE INDEX idx_habitacion_alojamiento ON habitaciones (idalojamiento);
   CREATE INDEX idx_reserva_cuenta ON reservas (idcuenta);
   CREATE INDEX idx_servicio_reserva_fechas ON servicios (idreserva, horarioinicial, horariofinal);

4. **Consulta SQL:**

```
SELECT h.numhabitacion, SUM(s.costo) AS totalRecolectado
FROM habitaciones h
JOIN alojamientos a ON h.idalojamiento = a.idalojamiento
JOIN cuentas c ON a.idalojamiento = c.idalojamiento
JOIN reservas r ON c.idcuenta = r.idcuenta
JOIN servicios s ON r.idreserva = s.idreserva
WHERE s.horarioinicial >= CURRENT_DATE - INTERVAL '1' YEAR
AND s.horariofinal <= CURRENT_DATE
AND s.existe = 'SI'
GROUP BY h.numhabitacion;
```

## RFC2 - MOSTRAR LOS 20 SERVICIOS MÁS POPULARES

1. **Creación Índice:** Si es necesaria la creación de un índice para optimizar la consulta.

2. **Justificación Índice desde la selectividad:** La columna ‘idservicio’ tiene una alta selectividad ya que cada servicio tendría un identificador único. Un índice aquí permitiría búsquedas rápidas y eficientes, lo cual es crucial en operaciones de JOIN, GROUP BY y ORDER BY, todas presentes en la consulta.

3. **Tipo de índice:**
   Un índice B+ Tree en la columna idservicio en todas las tablas implicadas sería el más adecuado para mejorar el rendimiento de las operaciones de JOIN y GROUP BY. Este índice sería un índice primario en la tabla servicios y un índice secundario en las tablas relacionadas.

   Índice: CREATE INDEX idx_servicio_id ON servicios(idservicio);
   CREATE INDEX idx_bares_idservicio ON bares(idservicio);
   CREATE INDEX idx_conferencias_idservicio ON conferencias(idservicio);
   CREATE INDEX idx_gimnasios_idservicio ON gimnasios(idservicio);
   CREATE INDEX idx_piscinas_idservicio ON piscinas(idservicio);
   CREATE INDEX idx_restaurantes_idservicio ON restaurantes(idservicio);
   CREATE INDEX idx_reuniones_idservicio ON reuniones(idservicio);
   CREATE INDEX idx_spas_idservicio ON spas(idservicio);
   CREATE INDEX idx_tiendas_idservicio ON tiendas(idservicio);
   CREATE INDEX idx_utensilios_idservicio ON utensilios(idservicio);
   CREATE INDEX idx_wifi_idservicio ON wifi(idservicio);
   CREATE INDEX idx_lavados_idservicio ON lavados(idservicio);

4. **Consulta SQL:**

```
SELECT s.idservicio,
    s.horarioinicial,
    s.horariofinal,
    s.costo,
    b.estilo AS bar_estilo,
    c.capacidad AS conferencia_capacidad,
    c.fecha AS conferencia_fecha,
    g.capacidad AS gimnasio_capacidad,
    p.capacidad AS piscina_capacidad,
    r.estilo AS restaurante_estilo,
    re.capacidad AS reunion_capacidad,
    spa.capacidad AS spa_capacidad,
    t.tipo AS tienda_tipo,
    u.estado AS utensilio_estado,
    w.anchobanda AS wifi_anchobanda,
    la.prendas AS lavado_prendas,
    COUNT(res.idreserva) AS cantidad_reservas
FROM servicios s
LEFT JOIN bares b ON s.idservicio = b.idservicio
```

```
LEFT JOIN conferencias c ON s.idservicio = c.idservicio
LEFT JOIN gimnasios g ON s.idservicio = g.idservicio
LEFT JOIN piscinas p ON s.idservicio = p.idservicio
LEFT JOIN restaurantes r ON s.idservicio = r.idservicio
LEFT JOIN reuniones re ON s.idservicio = re.idservicio
LEFT JOIN spas spa ON s.idservicio = spa.idservicio
LEFT JOIN tiendas t ON s.idservicio = t.idservicio
LEFT JOIN utensilios u ON s.idservicio = u.idservicio
LEFT JOIN wifi w ON s.idservicio = w.idservicio
LEFT JOIN lavados la ON s.idservicio = la.idservicio
JOIN reservas res ON s.idreserva = res.idreserva
GROUP BY s.idservicio,
    s.horarioinicial,
    s.horariofinal,
    s.costo,
    b.estilo,
    c.capacidad,
    c.fecha,
    g.capacidad,
    p.capacidad,
    r.estilo,
    re.capacidad,
    spa.capacidad,
    t.tipo,
    u.estado,
    w.anchobanda,
    la.prendas
ORDER BY cantidad_reservas DESC
FETCH FIRST 20 ROWS ONLY;
```

## RFC3 - MOSTRAR EL ÍNDICE DE OCUPACIÓN DE CADA UNA DE LAS HABITACIONES DEL HOTEL

1. **Creación Índice:** Sí es necesario la creación de un índice con el fin de optimizar la consulta debido a la magnitud de la base de datos.

2. **Justificación Índice desde la selectividad:** La combinación de checkin y idalojamiento en un índice compuesto mejora la selectividad ya que es menos probable que la combinación de ambas sea la misma en múltiples registros. Este índice compuesto permitiría rápidamente los registros de alojamiento relevantes para un rango de fechas específico, antes de realizar cálculos de agregación.

3. **Tipo de índice:** El índice idx_alojamiento_checkin_idalojamiento es un índice compuesto B+ que facilita la búsqueda eficiente por rango de fechas y la asociación rápida con la tabla habitaciones. El índice B+ se prefiere sobre un índice hash

porque los índices hash son ideales para búsquedas de igualdad, pero no para búsquedas de rangos, que son necesarias cuando se filtra por fechas.

Indice: CREATE INDEX idx_alojamiento_checkin_idalojamiento ON alojamientos(checkin, idalojamiento);

4. **Consulta SQL:**

```
SELECT
   h.numhabitacion,
   SUM(a.checkout - a.checkin) AS dias_ocupados,
   (SUM(a.checkout - a.checkin) / 365) * 100 AS indice_ocupacion
FROM
   habitaciones h
JOIN
   alojamientos a ON h.idalojamiento = a.idalojamiento
WHERE
   a.checkin BETWEEN ADD_MONTHS(SYSDATE, -12) AND SYSDATE
GROUP BY
   h.numhabitacion;
```

## RFC4 - MOSTRAR LOS SERVICIOS QUE CUMPLEN CON CIERTA CARACTERÍSTICA (Categoría: Ventas en tiendas)

1. **Creación Índice:** No es necesaria la creación de un índice puesto que la clave primaria tiene un índice B+ automático generado por Oracle.

2. **Justificación Índice desde la selectividad:** La justificación de la no creación del índice se basa en que ya no se necesita crear el índice porque la llave primaria por default ya es un índice que crea Oracle aunque la selectividad es bastante buena.

3. **Tipo de índice:** B+ generado automáticamente por Oracle.

4. **Consulta SQL:**

```
SELECT s.*, t.tipo
FROM servicios s
INNER JOIN tiendas t ON s.idservicio = t.idservicio;
```

## RFC5 - MOSTRAR EL CONSUMO EN HOTELANDES POR UN USUARIO DADO, EN UN RANGO DE FECHAS INDICADO

1. **Creación Índice:** Si es necesario la creación de un índice para este requerimiento funcional.

2. **Justificación Índice desde la selectividad:** Aunque las fechas en sí mismas podrían no ser únicas para cada entrada, la combinación de un rango de fechas con un iduser probablemente será única o tendrá una selectividad alta, porque es poco probable que un mismo usuario se aloje en el hotel varias veces en el mismo rango de fechas.

3. **Tipo de índice:** Dado que las fechas de alojamiento (checkin y checkout) se usarán para buscar rangos, un índice B+ sería apropiado. Estos serían índices secundarios, ya que no son la clave primaria de la tabla.

   Índices: CREATE INDEX idx_alojamientos_checkin ON alojamientos(checkin);
   CREATE INDEX idx_alojamientos_checkout ON alojamientos(checkout);
   CREATE INDEX idx_alojamientos_user_date ON alojamientos(iduser, checkin, checkout);

4. **Consulta SQL:**

```
SELECT
  u.nombreuser,
  h.numhabitacion,
  h.precionoche,
  a.checkin,
  a.checkout,
  s.costo AS costo_servicio,
  p.nombre AS nombre_producto,
  p.precio AS precio_producto
FROM
  usuarios u
  JOIN alojamientos a ON u.idalojamiento = a.idalojamiento
  JOIN habitaciones h ON a.idalojamiento = h.idalojamiento
  LEFT JOIN reservas r ON u.iduser = r.idreserva
  LEFT JOIN servicios s ON r.idreserva = s.idreserva
  LEFT JOIN productos p ON s.idservicio = p.idproducto
WHERE
  u.iduser = 1
  AND a.checkin BETWEEN TO_DATE('2023-01-01', 'YYYY-MM-DD') AND
TO_DATE('2023-01-10', 'YYYY-MM-DD')
  AND a.checkout BETWEEN TO_DATE('2023-01-01', 'YYYY-MM-DD') AND
TO_DATE('2023-01-10', 'YYYY-MM-DD')
  AND a.activa = 'SI';
```

## RFC6 - ANALIZAR LA OPERACIÓN DE HOTELANDES

1. **Creación Índice:** Sí se necesita de la creación de un índice

2. **Justificación Índice desde la selectividad:** Si los cálculos de ingresos implican sumar valores frecuentemente desde la tabla cuentas, un índice compuesto que incluya netocuenta podría ser beneficioso para acelerar estas operaciones de agregación.

3. **Tipo de índice:** Este es un índice secundario, ya que no es un índice de clave primaria. Este índice secundario ayudará en consultas que buscan registros específicos por idalojamiento y que además realizan operaciones con netocuenta, como sumas o promedios para calcular ingresos totales.

```
CREATE INDEX idx_cuentas_compuesto ON cuentas (idalojamiento, netocuenta);
```

4. **Consulta SQL:**

```
WITH Ocupacion AS (
   SELECT
      a.checkin,
      COUNT(*) AS habitaciones_ocupadas
   FROM alojamientos a
   JOIN habitaciones h ON a.idalojamiento = h.idalojamiento
   WHERE a.activa = 'SI'
   GROUP BY a.checkin
),
Ingresos AS (
   SELECT
      a.checkin,
      SUM(c.netocuenta) AS ingresos_totales
   FROM alojamientos a
   JOIN cuentas c ON a.idalojamiento = c.idalojamiento
   WHERE a.activa = 'SI'
   GROUP BY a.checkin
)
SELECT
   (SELECT checkin FROM Ocupacion ORDER BY habitaciones_ocupadas DESC
FETCH FIRST ROW ONLY) AS dia_mayor_ocupacion,
   (SELECT checkin FROM Ingresos ORDER BY ingresos_totales DESC FETCH
FIRST ROW ONLY) AS dia_mayores_ingresos,
   (SELECT checkin FROM Ocupacion ORDER BY habitaciones_ocupadas ASC
FETCH FIRST ROW ONLY) AS dia_menor_demanda
FROM dual;
```

## RFC7 - ENCONTRAR LOS BUENOS CLIENTES

1. **Creación Índice:** Sí es necesaria la creación de índices

2. **Justificación Índice desde la selectividad:** En este caso, ya que la distribución de los gastos de los clientes (netocuenta) y las fechas de estancia (checkin y checkout) varían significativamente, un índice será útil para acelerar las consultas que utilizan estas columnas como condiciones de filtro. Por ejemplo, un índice sobre la columna netocuenta en la tabla cuentas mejoraría el rendimiento de la consulta que busca clientes que han gastado más de $15'000.000.oo. Además, un índice en las columnas de fecha (checkin y checkout) en la tabla alojamientos ayudaría a filtrar rápidamente los registros dentro del último año. (Ya creado anteriormente)

3. **Tipo de índice:** Un índice secundario no único de tipo B+, que facilitaría la búsqueda rápida de cuentas por montos de gastos, particularmente para identificar aquellos clientes que han gastado una cantidad significativa.

   Índice: CREATE INDEX idx_cuentas_netocuenta ON cuentas(netocuenta);

4. **Consulta SQL:**

```
SELECT
  u.nombreuser,
  u.tipodocuser,
  u.numdocuser,
  u.correouser,
  SUM(NVL(a.checkout - a.checkin, 0)) AS total_dias_estadia,
  SUM(NVL(c.netocuenta, 0)) AS total_consumos
FROM
  usuarios u
JOIN alojamientos a ON u.idalojamiento = a.idalojamiento
JOIN cuentas c ON a.idalojamiento = c.idalojamiento
WHERE
  -- Filtramos para el último año de operaciones
  (a.checkin BETWEEN ADD_MONTHS(SYSDATE, -12) AND SYSDATE
   OR a.checkout BETWEEN ADD_MONTHS(SYSDATE, -12) AND SYSDATE)
GROUP BY
  u.nombreuser,
  u.tipodocuser,
  u.numdocuser,
  u.correouser
HAVING
  -- El cliente ha estado al menos 14 días (dos semanas) o ha consumido más de 15
millones
  SUM(NVL(a.checkout - a.checkin, 0)) >= 14
  OR SUM(NVL(c.netocuenta, 0)) > 15000000;
```

## RFC8 - ENCONTRAR LOS SERVICIOS QUE NO TIENEN MUCHA DEMANDA

1. **Creación Índice:** Sí se requiere de la creación de un índice

2. **Justificación Índice desde la selectividad:** Un índice podría ser especialmente útil sobre la columna horareserva en la tabla reservas, dado que la consulta filtrará y agrupará basándose en fechas que caen dentro de un rango de tiempo específico (el último año).

3. **Tipo de índice:**

   idx_horareserva: Este es un índice secundario de tipo B+ Tree en la columna horareserva. Se utiliza para mejorar la velocidad de las consultas que filtran reservas dentro de un rango de fechas, en este caso, el último año.
   idx_idservicio_reservas: Este es otro índice secundario de tipo B+ Tree, pero en la columna idservicio de la tabla reservas. Mejora la eficiencia de las operaciones de join entre las tablas reservas y servicios.

   Indice: CREATE INDEX idx_horareserva ON reservas (horareserva) INDEXTYPE IS btree;
   CREATE INDEX idx_idservicio_reservas ON reservas (idservicio) INDEXTYPE IS btree;

4. **Consulta SQL:**

```
SELECT s.idservicio,
    COUNT(r.idreserva) AS cantidad_solicitudes,
    TO_CHAR(r.horareserva, 'IW') AS semana_iso,
    TO_CHAR(r.horareserva, 'YYYY') AS ano
FROM servicios s
JOIN reservas r ON s.idservicio = r.idreserva
WHERE r.horareserva BETWEEN ADD_MONTHS(SYSDATE, -12) AND
SYSDATE
GROUP BY s.idservicio, TO_CHAR(r.horareserva, 'IW'), TO_CHAR(r.horareserva,
'YYYY')
HAVING COUNT(r.idreserva) < 3
ORDER BY ano, semana_iso;
```

## DISEÑO Y CARGA MASIVA DE DATOS

Se realizaron scripts en formato SQL para cargar masivamente los datos y verificando que cumpla la lógica del negocio y del esquema:
**Alojamientos:**

```
BEGIN
  FOR i IN 1..10000 LOOP -- Cambia 10000 por la cantidad de filas que quieres generar.
    INSERT INTO alojamientos (
      activa,
```

```
    checkin,
    checkout,
    acompanantes,
    idplan,
    idalojamiento
  ) VALUES (
    CASE MOD(i, 2) WHEN 0 THEN 'SI' ELSE 'NO' END, -- Alterna entre 'SI' y 'NO'.
    TO_DATE('2005-01-01', 'YYYY-MM-DD') + DBMS_RANDOM.VALUE(0, 365 *
(EXTRACT(YEAR FROM SYSDATE) - 2005)), -- Genera una fecha aleatoria de checkin
desde el año 2005 hasta la fecha actual.
    TO_DATE('2005-01-01', 'YYYY-MM-DD') + DBMS_RANDOM.VALUE(365 *
(EXTRACT(YEAR FROM SYSDATE) - 2004), 365 * (EXTRACT(YEAR FROM
SYSDATE) - 2003)), -- Genera una fecha de checkout posterior al checkin.
    DBMS_RANDOM.VALUE(0, 8), -- Genera un número aleatorio de acompañantes
entre 0 y 8.
    MOD(i, 200) + 1, -- Asigna un idplan de manera cíclica entre 1 y 200.
    i -- Asigna un idalojamiento incremental.
  );
 END LOOP;
 COMMIT;
END;
/
```

**Usuarios:**

```
DECLARE
 v_nombreuser    VARCHAR2(64);
 v_tipodocuser   VARCHAR2(32);
 v_numdocuser    NUMBER;
 v_correouser    VARCHAR2(64);
 v_iduser        NUMBER;
 v_idalojamiento NUMBER;
BEGIN
 FOR i IN 1..10000 LOOP -- Cambia 10000 por la cantidad de usuarios que desees
insertar.
  -- Genera un nombre de usuario aleatorio. Puede que necesites una función para esto o
usar un conjunto fijo de nombres.
  v_nombreuser := 'Usuario ' || TO_CHAR(i);

  -- Asigna un tipo de documento de usuario aleatorio.
  v_tipodocuser := CASE MOD(i, 5)
                   WHEN 0 THEN 'CC'
                   WHEN 1 THEN 'CE'
                   WHEN 2 THEN 'NIT'
                   WHEN 3 THEN 'PASAPORTE'
                   WHEN 4 THEN 'TI'
                 END;
```

```
    -- Genera un número de documento de usuario aleatorio.
    v_numdocuser := TRUNC(DBMS_RANDOM.VALUE(10000000, 99999999));

    -- Genera un correo electrónico de usuario aleatorio. Puede que necesites una función
para esto.
    v_correouser := 'usuario' || TO_CHAR(i) || '@email.com';

    -- Establece el ID de usuario y alojamiento.
    v_iduser := i;
    v_idalojamiento := i;

    INSERT INTO usuarios (
      nombreuser,
      tipodocuser,
      numdocuser,
      correouser,
      iduser,
      idalojamiento
    ) VALUES (
      v_nombreuser,
      v_tipodocuser,
      v_numdocuser,
      v_correouser,
      v_iduser,
      v_idalojamiento
    );

    -- Realiza un commit cada 1000 registros para evitar bloqueos y uso excesivo de
memoria.
    IF MOD(i, 1000) = 0 THEN
      COMMIT;
    END IF;
  END LOOP;
  COMMIT;
END;
/
```

**Habitaciones:**

```
DECLARE
  v_numhabitacion NUMBER;
  v_disponible    VARCHAR2(2);
  v_precionoche   NUMBER;
  v_idhotel       NUMBER;
  v_idtipo        NUMBER;
  v_idalojamiento NUMBER;
BEGIN
  FOR i IN 1..10000 LOOP -- Cambia 10000 por la cantidad de filas que deseas insertar.
    v_numhabitacion := i;
```

```
    v_disponible    := CASE MOD(i, 2) WHEN 0 THEN 'SI' ELSE 'NO' END;
    v_precionoche   := TRUNC(DBMS_RANDOM.VALUE(50000, 1500000), -3); --
Redondea al múltiplo de 1000 más cercano dentro del rango.
    v_idhotel       := MOD(i-1, 20) + 1; -- Asigna un ID de hotel de manera cíclica entre 1 y
20.
    v_idtipo        := MOD(i-1, 20) + 1; -- Asigna un ID de tipo de manera cíclica entre 1 y
20.
    v_idalojamiento := i; -- Asume que idalojamiento se incrementa igual que idhabitacion.

    INSERT INTO habitaciones (
      numhabitacion,
      disponible,
      precionoche,
      idhotel,
      idtipo,
      idalojamiento,
      idhabitacion
    ) VALUES (
      v_numhabitacion,
      v_disponible,
      v_precionoche,
      v_idhotel,
      v_idtipo,
      v_idalojamiento,
      i -- idhabitacion se incrementa automáticamente con el bucle.
    );

    IF MOD(i, 1000) = 0 THEN
      COMMIT; -- Realiza un commit cada 1000 registros para evitar bloqueos y uso
excesivo de memoria.
    END IF;
  END LOOP;
  COMMIT; -- Hace commit de la última transacción si es que no alcanzó el múltiplo de
1000.
END;
/
```

**Cuentas:**

```
DECLARE
  v_netocuenta    NUMBER;
  v_idcuenta      NUMBER;
  v_idalojamiento NUMBER;
BEGIN
  FOR i IN 1..10000 LOOP -- Cambia 10000 por la cantidad de filas que deseas insertar.
    v_netocuenta    := TRUNC(DBMS_RANDOM.VALUE(100000, 20000000), -3); --
Redondea al múltiplo de 1000 más cercano dentro del rango.
    v_idcuenta      := i;
    v_idalojamiento := i; -- Asume que idalojamiento se incrementa igual que idcuenta.
```

```
    INSERT INTO cuentas (
      netocuenta,
      idcuenta,
      idalojamiento
    ) VALUES (
      v_netocuenta,
      v_idcuenta,
      v_idalojamiento
    );

    IF MOD(i, 1000) = 0 THEN
      COMMIT; -- Realiza un commit cada 1000 registros para evitar bloqueos y uso
excesivo de memoria.
    END IF;
  END LOOP;
  COMMIT; -- Hace commit de la última transacción si es que no alcanzó el múltiplo de
1000.
END;
/
```

**Reservas:**

```
DECLARE
  v_horareserva DATE;
  v_idreserva   NUMBER;
  v_idcuenta    NUMBER;
BEGIN
  FOR i IN 1..10000 LOOP -- Cambia 10000 por la cantidad de filas que deseas insertar.
    -- Genera una fecha y hora aleatoria, evitando las 24 horas.
    v_horareserva := TO_DATE('2023-01-01', 'YYYY-MM-DD') + (i - 1) +
DBMS_RANDOM.VALUE(0, 23) / 24;
    v_idreserva   := i;
    v_idcuenta    := i; -- Asume que idcuenta se incrementa igual que idreserva.

    INSERT INTO reservas (
      horareserva,
      idreserva,
      idcuenta
    ) VALUES (
      v_horareserva,
      v_idreserva,
      v_idcuenta
    );

    IF MOD(i, 1000) = 0 THEN
      COMMIT; -- Realiza un commit cada 1000 registros para evitar bloqueos y uso
excesivo de memoria.
    END IF;
```

```
  END LOOP;
  COMMIT; -- Hace commit de la última transacción si es que no alcanzó el múltiplo de 1000.
END;
/
```

**Servicios:**

```
DECLARE
  v_horarioinicial DATE;
  v_horariofinal   DATE;
  v_costo          NUMBER;
  v_cargado        VARCHAR2(2);
  v_existe         VARCHAR2(2);
  v_idservicio     NUMBER;
  v_idreserva      NUMBER;
BEGIN
  FOR i IN 1..10000 LOOP -- Cambia 10000 por la cantidad de servicios que desees insertar.
    v_horarioinicial := TO_DATE('2023-01-01 07:00', 'YYYY-MM-DD HH24:MI') + (i-1); -- Asume que el servicio comienza a las 7 AM y añade un día por cada iteración.
    v_horariofinal   := v_horarioinicial + INTERVAL '16' HOUR; -- Asume que el servicio termina 16 horas después.
    v_costo          := TRUNC(DBMS_RANDOM.VALUE(100000, 3000000), -3); -- Genera un costo aleatorio entre 100000 y 3000000, redondeado al múltiplo de 1000 más cercano.
    v_cargado        := CASE MOD(i, 2) WHEN 0 THEN 'SI' ELSE 'NO' END; -- Alterna entre 'SI' y 'NO'.
    v_existe         := 'SI'; -- Asume que todos los servicios existen.
    v_idservicio     := i; -- Asume que idservicio se incrementa automáticamente con el bucle.
    v_idreserva      := i; -- Asume que idreserva se incrementa automáticamente con el bucle.

    INSERT INTO servicios (
      horarioinicial,
      horariofinal,
      costo,
      cargado,
      existe,
      idservicio,
      idreserva
    ) VALUES (
      v_horarioinicial,
      v_horariofinal,
      v_costo,
      v_cargado,
      v_existe,
      v_idservicio,
      v_idreserva
    );
```

```
    IF MOD(i, 1000) = 0 THEN
      COMMIT; -- Realiza un commit cada 1000 registros para evitar bloqueos y uso
excesivo de memoria.
    END IF;
  END LOOP;
  COMMIT; -- Hace commit de la última transacción si es que no alcanzó el múltiplo de
1000.
END;
```

**Bares:**

```
DECLARE
  TYPE estilos_typ IS VARRAY(5) OF VARCHAR2(50);
  estilos estilos_typ := estilos_typ('Frances', 'Italiano', 'Mediterraneo', 'Mexicano',
'Oriental');
BEGIN
  FOR i IN 1..10000 LOOP -- Cambia 10000 por la cantidad de bares que deseas insertar.
    INSERT INTO bares (
      estilo,
      idservicio,
      idbar
    ) VALUES (
      estilos(MOD(i, 5) + 1), -- Esto ciclará a través del arreglo de estilos.
      i, -- idservicio incrementa de 1 en 1.
      i  -- idbar coincide con idservicio.
    );

    IF MOD(i, 1000) = 0 THEN
      COMMIT; -- Realiza un commit cada 1000 registros para evitar bloqueos y uso
excesivo de memoria.
    END IF;
  END LOOP;
  COMMIT; -- Hace commit de la última transacción si es que no alcanzó el múltiplo de
1000.
END;
/
```