Caso 1: MBA Online

(12 p.) Sobre el caso expuesto se pide realizar lo siguiente:

Pregunta 1 (2 p.).
Crear un procedimiento almacenado o función que retorne los nombres de los asesores con la mayor cantidad de alumnos asignados.

```
CREATE PROCEDURE obtener_asesores_con_mas_alumnos
AS
BEGIN
    -- Obtener la cantidad máxima de alumnos asignados a un asesor
    DECLARE @max_alumnos INT;
    SELECT @max_alumnos = MAX(total_alumnos)
    FROM (
        SELECT advisers_id, COUNT(*) AS total_alumnos
        FROM students
        GROUP BY advisers_id
    ) AS t;

    -- Obtener los nombres de los asesores con la mayor cantidad de alumnos
asignados
    SELECT CONCAT(advisers.first_name, ' ', advisers.last_name) AS nombre_asesor
    FROM advisers
    INNER JOIN (
        SELECT advisers_id, COUNT(*) AS total_alumnos
        FROM students
        GROUP BY advisers_id
        HAVING COUNT(*) = @max_alumnos
    ) AS subquery ON advisers.id = subquery.advisers_id;
END;

EXEC obtener_asesores_con_mas_alumnos;
```

Pregunta 2 (2 p.).
Crear un procedimiento almacenado o función que retorne los nombres y versiones de las maestrías con la mayor cantidad de alumnos.

```
CREATE PROCEDURE obtener_maestrias_con_mas_alumnos
AS
BEGIN
    -- Variable para almacenar la cantidad máxima de alumnos
    DECLARE @max_alumnos INT;

    -- Obtener la cantidad máxima de alumnos asignados a una maestría
    SELECT @max_alumnos = MAX(total_alumnos)
    FROM (
        SELECT masters_id, COUNT(*) AS total_alumnos
        FROM students
        GROUP BY masters_id
    ) AS t;

    -- Obtener los nombres y versiones de las maestrías con la mayor cantidad de
alumnos asignados
    SELECT m.name, m.version
    FROM masters m
    JOIN (
        SELECT masters_id, COUNT(*) AS total_alumnos
        FROM students
```

```
        GROUP BY masters_id
        HAVING COUNT(*) = @max_alumnos
    ) AS subquery ON m.id = subquery.masters_id;

END;

EXEC obtener_maestrias_con_mas_alumnos;
```

Pregunta 3 (2 p.).
Crear un procedimiento almacenado o función que retorne los nombres completos de los estudiantes que no forman parte de ningún grupo de estudio.

```
CREATE PROCEDURE obtener_estudiantes_sin_grupo
AS
BEGIN
    -- Obtener los nombres completos de los estudiantes que no tienen grupo
asignado
    SELECT CONCAT(students.first_name, ' ', students.last_name) AS
nombre_completo
    FROM students
    LEFT JOIN (
        SELECT students_by_group.students_id
        FROM students_by_group
        INNER JOIN study_groups ON students_by_group.study_groups =
study_groups.id
    ) AS grupos_estudio ON students.id = grupos_estudio.students_id
    WHERE grupos_estudio.students_id IS NULL;
END;

EXEC obtener_estudiantes_sin_grupo;
```

Pregunta 4 (2 p.)
Crear un procedimiento almacenado o función que retorne los nombres de los cursos con la mayor cantidad de conferencias.

```
CREATE PROCEDURE obtener_cursos_con_mas_conferencias
AS
BEGIN
    -- Obtener la cantidad máxima de conferencias
    DECLARE @max_conferencias INT;
    SELECT @max_conferencias = MAX(total_conferencias)
    FROM (
        SELECT course_id, COUNT(*) AS total_conferencias
        FROM conferences
        GROUP BY course_id
    ) AS t;

    -- Obtener los nombres de los cursos con la mayor cantidad de conferencias
    SELECT courses.name
    FROM courses
    INNER JOIN (
        SELECT course_id, COUNT(*) AS total_conferencias
        FROM conferences
```

```
        GROUP BY course_id
        HAVING COUNT(*) = @max_conferencias
    ) AS subquery ON courses.id = subquery.course_id;
END;

EXEC obtener_cursos_con_mas_conferencias;
```

Pregunta 5 (4 p.).
Crear un procedimiento almacenado o función que retorne la cantidad consolidad de actividades (exámenes, ensayos y presentaciones) para cada curso.

```
CREATE PROCEDURE obtener_cantidad_actividades_por_curso
AS
BEGIN
    -- Obtener la cantidad consolidada de actividades por curso
    SELECT c.name AS nombre_curso,
           COALESCE(SUM(e.total_actividades), 0) AS cantidad_actividades
    FROM courses c
    LEFT JOIN (
        SELECT course_id, COUNT(*) AS total_actividades
        FROM (
            SELECT course_id
            FROM exams
            UNION ALL
            SELECT course_id
            FROM essays
            UNION ALL
            SELECT course_id
            FROM keynotes
        ) AS actividades
        GROUP BY course_id
    ) AS e ON c.id = e.course_id
    GROUP BY c.name;
END;

EXEC obtener_cantidad_actividades_por_curso;
```

Caso 2: AudioSlave

Pregunta 6 (2 p.).
Establecer una regla de validación utilizando JSON Schema para la colección de documentos que represente una lista de reproducción creada por un usuario.

```
db.createCollection("listas_reproduccion", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["nombre", "usuario", "fecha_creacion",
"canciones"],
      properties: {
        nombre: {
          bsonType: "string",
          description: "El nombre de la lista de
reproducción"
        },
        usuario: {
          bsonType: "string",
          description: "El usuario que creó la lista de
reproducción"
        },
        fecha_creacion: {
          bsonType: "date",
          description: "La fecha de creación de la lista de
reproducción"
        },
        canciones: {
          bsonType: "array",
          description: "La lista de canciones en la lista
de reproducción",
          items: {
            bsonType: "object",
            required: ["nombre", "fecha_agregada"],
            properties: {
              nombre: {
                bsonType: "string",
                description: "El nombre de la canción"
              },
              fecha_agregada: {
                bsonType: "date",
                description: "La fecha en que se agregó la
canción a la lista"
              }
            }
          }
```

```
        }
      }
    }
  }
});
```

Pregunta 7 (2 p.).
Indicar los patrones de modelado de datos utilizados para el documento que representa una lista de reproducción creada por un usuario.

- Patrón de Incrustación (Embedding):

```
db.listas_reproduccion.insertOne({
  _id: ObjectId("5bd761dcae323e45a93ccfe8"),
  nombre: "Lista de reproducción 1",
  usuario: {
    gender: "M",
    age: 42,
    email: "cauho@witwuta.sv",
    satisfaction: 4
  },
  purchaseMethod: "Online",
  saleDate: ISODate("2015-03-23T21:06:49.506Z"),
  storeLocation: "Denver",
  canciones: [
    { nombre: "Canción 1", duracion: 4.6 },
    { nombre: "Canción 2", duracion: 4.2 }
  ]
});
```

- Patrón de Referencias (Referencing):

```
db.listas_reproduccion.insertOne({
  _id: ObjectId("5bd761dcae323e45a93ccfe8"),
  nombre: "Lista de reproducción 1",
  usuario: {
    gender: "M",
    age: 42,
    email: "cauho@witwuta.sv",
    satisfaction: 4
  },
  purchaseMethod: "Online",
  saleDate: ISODate("2015-03-23T21:06:49.506Z"),
  storeLocation: "Denver",
  canciones: [
    ObjectId("cancion1_id"),
    ObjectId("cancion2_id")
  ]
});
```

Caso 3: Ventas

Pregunta 8 (2 p.).
Escribir una consulta que permita mostrar la cantidad de ventas realizadas en cada ciudad. Considerar solo aquellas ventas en la cuales se haya utilizado un cupón de descuento.

```
db.ventas.aggregate([
  {
    $match: {
      couponUsed: true
    }
  },
  {
    $group: {
      _id: "$storeLocation",
      cantidadVentas: { $sum: 1 }
    }
  }
]);
```

Pregunta 9 (2 p.).
Escribir una consulta que permita mostrar la cantidad de ventas realizadas por cada método de compra. Considerar solo aquellas ventas en las cuales la satisfacción del cliente haya sido mayor o igual a 4.

```
db.ventas.aggregate([
  {
    $match: {
      "customer.satisfaction": { $gte: 4 }
    }
  },
  {
    $group: {
      _id: "$purchaseMethod",
      cantidadVentas: { $sum: 1 }
    }
  }
]);
```