**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("Sbd761dcae323e45a93ccfe8"),  
 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("Sbd761dcae323e45a93ccfe8"),  
 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 }  
 }  
 }  
]);