Guía de Índices, Vistas Indexadas y Evidencias — Proyecto Academia2022

# 1. Objetivo del entregable

Este documento resume las vistas indexadas, índices propuestos y evidencias de rendimiento del sistema académico Academia2022. Se busca demostrar cómo las optimizaciones mediante vistas materializadas e índices reducen los costos de lectura y tiempo de ejecución.

# 2. Contexto general

El proyecto utiliza el esquema 'Academico' y 'App' de la base de datos Academia2022. Las consultas clave incluyen métricas de carga académica por alumno y ocupación por periodo.

# 3. Vistas Indexadas

## 3.1 App.vw\_CargaPorAlumno

Propósito: Materializar el número de cursos inscritos por alumno y periodo, reduciendo el costo de las agregaciones frecuentes sobre grandes volúmenes de datos.

CREATE VIEW App.vw\_CargaPorAlumno WITH SCHEMABINDING AS  
SELECT m.AlumnoID, m.MatriculaPeriodo, COUNT\_BIG(\*) AS TotalCursos  
FROM Academico.Matriculas AS m  
GROUP BY m.AlumnoID, m.MatriculaPeriodo;  
CREATE UNIQUE CLUSTERED INDEX UCI\_vw\_CargaPorAlumno ON App.vw\_CargaPorAlumno(AlumnoID, MatriculaPeriodo);

Por qué: acelera KPIs y rankings al evitar cálculos repetitivos.

## 3.2 App.vw\_OcupacionPorPeriodo

Propósito: Materializar el total de matrículas (ocupación) por periodo académico, agilizando consultas globales.

CREATE VIEW App.vw\_OcupacionPorPeriodo WITH SCHEMABINDING AS  
SELECT m.MatriculaPeriodo, COUNT\_BIG(\*) AS TotalMatriculas  
FROM Academico.Matriculas AS m  
GROUP BY m.MatriculaPeriodo;  
CREATE UNIQUE CLUSTERED INDEX UCI\_vw\_OcupacionPorPeriodo ON App.vw\_OcupacionPorPeriodo(MatriculaPeriodo);

Por qué: mejora consultas de dashboards y reportes institucionales.

# 4. Consultas Analíticas y KPIs

Se incluyen consultas de rendimiento con funciones de ventana, PIVOT y subconsultas correlacionadas.

## 4.1 Funciones de Ventana

Objetivo: Ranking de alumnos por carga académica y comparación con el periodo anterior usando DENSE\_RANK() y LAG().

## 4.2 PIVOT

Objetivo: Mostrar la cantidad de alumnos por curso y periodo, generando columnas para 24S1, 24S2 y 25S1.

## 4.3 Subconsulta Correlacionada

Objetivo: Obtener la matrícula más reciente (último periodo) por alumno mediante MAX(MatriculaPeriodo).

# 5. Propuesta de Índices

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Índice | Tabla | Tipo | Columnas | INCLUDE | Propósito |
| IX\_Matriculas\_Alumno\_Periodo | Academico.Matriculas | Nonclustered | (AlumnoID, MatriculaPeriodo DESC) | (CursoID) | Optimiza búsqueda del último periodo por alumno |
| IX\_Matriculas\_Curso\_Periodo | Academico.Matriculas | Nonclustered | (CursoID, MatriculaPeriodo) | (AlumnoID) | Optimiza KPIs y reportes por curso/periodo |
| IX\_Alumnos\_Activos | Academico.Alumnos | Filtrado | (CarreraID) | (NombreCompleto) | Filtra alumnos activos, mejora joins |
| UCI\_vw\_CargaPorAlumno | App.vw\_CargaPorAlumno | Clustered único | (AlumnoID, MatriculaPeriodo) | - | Materializa vista indexada |
| UCI\_vw\_OcupacionPorPeriodo | App.vw\_OcupacionPorPeriodo | Clustered único | (MatriculaPeriodo) | - | Materializa vista indexada global |

# 6. Evidencias de Performance

Para validar mejoras se usaron STATISTICS IO/TIME y SHOWPLAN\_XML en SQL Server. Se compararon consultas antes y después de implementar índices.

Ejemplo:

SET STATISTICS IO ON; SET STATISTICS TIME ON;  
SELECT TOP 5 \* FROM App.vw\_CargaPorAlumno;  
SET STATISTICS IO OFF; SET STATISTICS TIME OFF;

La reducción de lecturas lógicas y tiempo total confirma que las vistas indexadas y los índices materializados reducen significativamente el costo de ejecución.

# 7.ENTREGABLE EQUIPO C: SQL DE REPORTES Y PERFORMANCE, SCRIPT DOCUMENTADO (QUÉ HACE, CÓMO FUNCIONA, POR QUÉ)

**¿QUÉ HACE?**

Asegura que todos los comandos siguientes se ejecuten en la base de datos 'Academia2022'.

**¿CÓMO FUNCIONA?**

Es un comando de T-SQL que cambia el contexto de la sesión actual a la base de datos especificada.

**¿POR QUÉ?**

Es una buena práctica para evitar ejecutar accidentalmente scripts en la base de datos incorrecta (como 'master').

USE Academia2022;

GO

## 7.1 Hito 3: Vistas Indexadas para KPIs , (Top cursos, Carga por alumno, Ocupación por periodo)

**¿QUÉ HACE?**

Este bloque crea (o verifica que ya existe) la Vista Indexada para el KPI "Top Cursos", tal como se definió en el script original.

**¿CÓMO FUNCIONA?**

* `IF OBJECT\_ID... IS NULL`: Comprueba si la vista 'App.vw\_MatriculasPorCurso'. NO existe. Si no existe (es la primera ejecución), entra al bloque.
* `CREATE VIEW ... WITH SCHEMABINDING`: Crea la vista "amarrándola" (SCHEMABINDING) a la estructura de la tabla base 'Academico.Matriculas'. Esto es un REQUISITO OBLIGATORIO para poder indexar una vista.
* `COUNT\_BIG(\*)`: Agrupa por CursoID y cuenta las matrículas. Usar COUNT\_BIG(\*) es otro REQUISITO OBLIGATORIO para vistas indexadas.
* `CREATE UNIQUE CLUSTERED INDEX ...`: Este es el comando clave que \*materializa\* la vista. Físicamente guarda el resultado agregado (el conteo por curso) en disco, como si fuera una tabla.

**¿POR QUÉ?**

Esta es la optimización de rendimiento MÁXIMA para reportes de agregados. En lugar de contar millones de matrículas cada vez que alguien abre el dashboard de "Top Cursos", SQL Server simplemente lee este resultado pre-calculado y guardado. La consulta al KPI es casi instantánea.

PRINT 'Verificando KPI 1: App.vw\_MatriculasPorCurso...';

GO

IF OBJECT\_ID('App.vw\_MatriculasPorCurso', 'V') IS NULL

BEGIN

PRINT 'Creando App.vw\_MatriculasPorCurso...';

CREATE VIEW App.vw\_MatriculasPorCurso

WITH SCHEMABINDING -- Obligatorio para indexar

AS

SELECT

m.CursoID,

COUNT\_BIG(\*) AS Total -- Obligatorio usar COUNT\_BIG

FROM

Academico.Matriculas AS m

GROUP BY

m.CursoID;

**-- Este índice es el que "materializa" la vista**

CREATE UNIQUE CLUSTERED INDEX IX\_vw\_MatriculasPorCurso

ON App.vw\_MatriculasPorCurso(CursoID);

END

ELSE

BEGIN

PRINT 'Vista App.vw\_MatriculasPorCurso ya existe.';

END

GO

**¿QUÉ HACE?**

Crea la Vista Indexada para el KPI "Carga por Alumno".

**¿CÓMO FUNCIONA?**

* `IF OBJECT\_ID... DROP VIEW`: Primero borra la vista si ya existe. Esto permite que el script se pueda ejecutar múltiples veces sin errores (lo hace "idempotente").
* `CREATE VIEW ... WITH SCHEMABINDING`: Crea la vista 'App.vw\_CargaPorAlumno'.
* `GROUP BY m.AlumnoID, m.MatriculaPeriodo`: Pre-calcula el conteo de cursos por cada alumno en cada periodo.
* `CREATE UNIQUE CLUSTERED INDEX ...`: Materializa este resultado agrupado.

**¿POR QUÉ?**

Cumple el requisito del KPI "Carga por alumno". Optimiza reportes que necesiten saber cuántos cursos lleva un alumno en un periodo específico. Previene que la base de datos tenga que calcular esto una y otra vez.

PRINT 'Creando KPI 2: App.vw\_CargaPorAlumno...';

GO

IF OBJECT\_ID('App.vw\_CargaPorAlumno', 'V') IS NOT NULL

DROP VIEW App.vw\_CargaPorAlumno;

GO

CREATE VIEW App.vw\_CargaPorAlumno

WITH SCHEMABINDING -- Requerido para indexar

AS

SELECT

m.AlumnoID,

m.MatriculaPeriodo,

COUNT\_BIG(\*) AS TotalCursos -- Requerido para indexar (COUNT\_BIG)

FROM

Academico.Matriculas AS m

GROUP BY

m.AlumnoID, m.MatriculaPeriodo;

GO

**--Creación del índice que materializa la vista**

IF NOT EXISTS (SELECT \* FROM sys.indexes WHERE name = 'UCI\_vw\_CargaPorAlumno')

CREATE UNIQUE CLUSTERED INDEX UCI\_vw\_CargaPorAlumno

ON App.vw\_CargaPorAlumno(AlumnoID, MatriculaPeriodo);

GO

**¿QUÉ HACE?**

Crea la Vista Indexada para el KPI "Ocupación por Periodo".

**¿CÓMO FUNCIONA?**

Idéntico a las vistas anteriores, pero esta agrupa todas las matrículas únicamente por 'MatriculaPeriodo' para obtener el total de inscripciones en cada periodo académico.

**¿POR QUÉ?**

Es perfecto para un dashboard principal que muestre un gráfico de barras con la ocupación histórica de la academia. La consulta será instantánea gracias a esta vista.

PRINT 'Creando KPI 3: App.vw\_OcupacionPorPeriodo...';

GO

IF OBJECT\_ID('App.vw\_OcupacionPorPeriodo', 'V') IS NOT NULL

DROP VIEW App.vw\_OcupacionPorPeriodo;

GO

CREATE VIEW App.vw\_OcupacionPorPeriodo

WITH SCHEMABINDING

AS

SELECT

m.MatriculaPeriodo,

COUNT\_BIG(\*) AS TotalMatriculas

FROM

Academico.Matriculas AS m

GROUP BY

m.MatriculaPeriodo;

GO

IF NOT EXISTS (SELECT \* FROM sys.indexes WHERE name = 'UCI\_vw\_OcupacionPorPeriodo')

CREATE UNIQUE CLUSTERED INDEX UCI\_vw\_OcupacionPorPeriodo

ON App.vw\_OcupacionPorPeriodo(MatriculaPeriodo);

GO

PRINT '--- Vistas Indexadas de KPIs creadas con éxito. ---';

GO

Alcance: Consultas analíticas (Ventanas, PIVOT, Subconsultas) Estas consultas son las que USAREMOS para el análisis de performance.

PRINT '--- EJECUTANDO CONSULTAS ANALÍTICAS ---';

GO

**¿QUÉ HACE?**

Genera un reporte analítico que crea un ranking de alumnos por la cantidad de cursos que llevan en un periodo, Y ADEMÁS, muestra cuántos cursos llevaron en el periodo inmediatamente anterior para comparar su evolución.

**¿CÓMO FUNCIONA?**

* `WITH CargaPorPeriodo AS (...)`: Se usa un "CTE" (Common Table Expression)para crear una tabla virtual temporal llamada 'CargaPorPeriodo'. Esta tabla contiene el conteo de cursos por alumno y periodo.
* `DENSE\_RANK() OVER(...)`: Esta es la FUNCIÓN DE VENTANA de ranking. `PARTITION BY c.MatriculaPeriodo`: Le dice a `DENSE\_RANK()` que reinicie el conteo (el ranking) por cada nuevo periodo. `ORDER BY c.CursosInscritos DESC`: Define cómo se calcula el ranking (el que tiene más cursos inscritos obtiene el ranking #1).
* `LAG(...) OVER(...)`: Esta es la FUNCIÓN DE VENTANA de serie temporal. `PARTITION BY c.AlumnoID`: Le dice a `LAG()` que solo mire filas del mismo alumno. `ORDER BY c.MatriculaPeriodo`: Ordena los periodos del alumno de más antiguo a más nuevo.
* `LAG(c.CursosInscritos, 1, 0)`: "Espía" la fila anterior (`1`) en la ventana ordenada y devuelve el valor de 'CursosInscritos' de esa fila. Si no hay fila anterior (es su primer periodo), devuelve un cero (`0`).

**¿POR QUÉ?**

Cumple el requisito de "Consultas con ventanas". Este tipo de consulta es fundamental para Inteligencia de Negocios (BI), ya que permite crear rankings y análisis de series temporales (crecimiento) sin usar subconsultas o JOINS complejos.

PRINT 'Consulta 1: Funciones de Ventana (Ranking y LAG)';

GO

WITH CargaPorPeriodo AS (

-- 1. Agregamos la carga base

SELECT

AlumnoID,

MatriculaPeriodo,

COUNT(\*) AS CursosInscritos

FROM Academico.Matriculas

GROUP BY AlumnoID, MatriculaPeriodo

)

**2. Aplicamos ventanas y unimos con Alumnos**

SELECT

a.NombreCompleto,

c.MatriculaPeriodo,

c.CursosInscritos,

**KPI: Ranking de alumnos por periodo (más inscritos primero)**

DENSE\_RANK() OVER(PARTITION BY c.MatriculaPeriodo

ORDER BY c.CursosInscritos DESC) AS RankingCarga,

**KPI: Comparativa de carga vs periodo anterior**

LAG(c.CursosInscritos, 1, 0) OVER(PARTITION BY c.AlumnoID

ORDER BY c.MatriculaPeriodo) AS CargaPeriodoAnterior

FROM CargaPorPeriodo c

JOIN Academico.Alumnos a ON c.AlumnoID = a.AlumnoID -- Join costoso

ORDER BY

c.MatriculaPeriodo, RankingCarga;

GO

**¿QUÉ HACE?**

Rota la tabla de matrículas. Toma los valores de la fila 'MatriculaPeriodo' (ej: '24S1', '24S2') y los convierte en COLUMNAS en el resultado final, mostrando un conteo de alumnos.

**CÓMO FUNCIONA:**

* `WITH BasePivot AS (...)`: Un CTE que pre-calcula el total de alumnos por CursoID y Periodo.
* `PIVOT (...)`: Este es el operador clave. `SUM(TotalAlumnos)`: Es la función de agregación que irá \*dentro\* de las nuevas celdas de la tabla pivotada. `FOR MatriculaPeriodo IN ([24S1], [24S2], [25S1])`: Esta es la instrucción de rotación. Le dice a SQL que la columna 'MatriculaPeriodo' es la que debe "pivotar", y que queremos crear \*nuevas columnas\* llamadas [24S1], [24S2] y [25S1].
* `ISNULL(p.[24S1], 0)`: Se usa para mostrar un '0' en lugar de 'NULL' si un curso no tuvo matrículas en ese periodo.

**POR QUÉ:**

Cumple el requisito de "PIVOT". Transforma un formato de datos normalizado (bueno para almacenar en la BD) en un formato de reporte tipo "hoja de cálculo" (bueno para leer por humanos).

PRINT 'Consulta 2: PIVOT (Matrículas por Curso/Periodo)';

GO

**Definimos los periodos que queremos como columnas**

DECLARE @ColumnasPivot NVARCHAR(MAX) = N'[24S1], [24S2], [25S1]';

(Se podrían hacer dinámicas, pero para el reporte se definen estáticas)

**1. Obtenemos la base de matriculas por curso y periodo**

WITH BasePivot AS (

SELECT CursoID, MatriculaPeriodo, COUNT(\*) AS TotalAlumnos

FROM Academico.Matriculas

GROUP BY CursoID, MatriculaPeriodo

)

**2. Aplicamos PIVOT**

SELECT

c.CursoNombre,

ISNULL(p.[24S1], 0) AS [Periodo 24S1],

ISNULL(p.[24S2], 0) AS [Periodo 24S2],

ISNULL(p.[25S1], 0) AS [Periodo 25S1]

FROM (

SELECT CursoID, MatriculaPeriodo, TotalAlumnos FROM BasePivot

) AS FuenteDatos

PIVOT (

SUM(TotalAlumnos) -- Función de Agregado

FOR MatriculaPeriodo IN ([24S1], [24S2], [25S1]) -- Columnas a crear

) AS p

JOIN Academico.Cursos c ON p.CursoID = c.CursoID -- Join para nombre

ORDER BY

c.CursoNombre;

GO

**QUÉ HACE:**

Obtiene la matrícula \*más reciente\* (el último periodo registrado) de cada alumno, junto con el nombre del curso y el nombre del alumno.

**CÓMO FUNCIONA:**

1. La consulta principal (`SELECT ... FROM Academico.Matriculas m ...`) recorre la tabla `Matriculas` fila por fila (alias 'm').

2. `WHERE m.MatriculaPeriodo = (...)`: En la cláusula WHERE, compara el periodo de la fila actual (`m.MatriculaPeriodo`) con el resultado de una subconsulta.

3. Esta subconsulta es "CORRELACIONADA" porque depende de la consultaprincipal. Nota la línea: `WHERE m2.AlumnoID = m.AlumnoID`.

4. Por cada fila 'm' que la consulta principal procesa, la subconsulta se re-ejecuta para encontrar el periodo MÁXIMO (MAX) \*solo para ese alumno específico\* (`m.AlumnoID`).

5. Si el periodo de la fila 'm' es igual al periodo máximo de ese alumno, la fila se incluye en el resultado.

**POR QUÉ:**

Cumple el requisito de "subconsultas correlacionadas". Es una técnica clásica (aunque a veces costosa en rendimiento) para resolver problemas de "el más grande por grupo" o "el último por grupo". Es fundamental entenderla.

PRINT 'Consulta 3: Subconsulta Correlacionada (Última Matrícula)';

GO

SELECT

a.NombreCompleto,

c.CursoNombre,

m.MatriculaPeriodo

FROM

Academico.Matriculas m

JOIN

Academico.Alumnos a ON m.AlumnoID = a.AlumnoID

JOIN

Academico.Cursos c ON m.CursoID = c.CursoID

WHERE

m.MatriculaPeriodo = (

-- Subconsulta Correlacionada:

-- Por cada fila 'm', esta subconsulta se re-ejecuta

-- para encontrar el periodo máximo SOLO para ese alumno.

SELECT MAX(m2.MatriculaPeriodo)

FROM Academico.Matriculas m2

WHERE m2.AlumnoID = m.AlumnoID

);

GO

PRINT '--- Fin del script de reportes. ---';

# 8. Conclusiones

• Las vistas indexadas mejoran consultas de agregación y KPIs.  
• Los índices propuestos reducen I/O y optimizan los planes de ejecución.  
• Las funciones analíticas aprovechan las vistas materializadas para rendimiento óptimo.  
• Las evidencias IO/TIME demuestran mejoras cuantificables.