Universidad Nacional del Nordeste

Facultad de Ciencias Exactas y Naturales y Agrimensura

Lic. en Sistemas de Información

Base de datos 1

Grupo Nº 39

Integrantes:

Apellido y Nombre	**Número de DNI**
Orban, Tobias Naim	46.385.637
Firmapaz, Gabriel Andres	43.205.785
Gomez Hertler, Lisandro Leonel	46.461.232

Índice

Contenido:

Capítulo I — Introducción..... 6
1.1 Tema y contexto..... 6
1.2 Definición del problema..... 6
1.3 Objetivos..... 7
1.3.1 Objetivo general..... 7
1.3.2 Objetivos específicos..... 7
Capítulo II — Marco conceptual o referencial..... 9
2.1. Enfoque y delimitación del problema..... 9
2.1.1 Tema..... 9
2.1.2 Objeto de estudio..... 9
2.1.3 Unidad de análisis..... 10
2.1.4 Ámbito conceptual..... 10
2.2. Conceptos clave del marco conceptual..... 10
1. Plataforma social (social network)..... 11
2. Contenido generado por usuarios (UGC)..... 11
3. Experiencia de visualización..... 11
4. Calificación (rating)..... 11
5. Opinión / Reseña..... 11
6. Spoilers..... 11
7. Privacidad..... 12
8. Seguimiento y favoritos..... 12
9. Curaduría / Partidos destacados..... 12
10. Recordatorios..... 12
11. Reputación y confianza..... 12
12. Integridad e interoperabilidad de datos..... 12
13. Autenticación y Seguridad..... 13
2.3 Terminología y referencia de datos..... 13
2.3.1 Modelo lógico..... 13
2.3.2 Datos de ejemplo..... 13
2.3.3 Diccionario de datos..... 13
Capítulo III — Metodología..... 14
3.1 Descripción de cómo se realizó el Trabajo Práctico..... 14
3.1.1 Construcción del núcleo del modelo..... 14
3.1.2 Organización por ramas y flujo de trabajo..... 14
3.1.3 Desarrollo incremental por temas..... 15
3.1.4 Validación y cierre..... 15
3.2 Herramientas (instrumentos y procedimientos)..... 16
3.2.1 Herramientas técnicas..... 16
3.2.2 Procedimientos de trabajo..... 16

Capítulo IV — Resultados..... 18
4.1. Diseño del modelo de datos: Representación del dominio..... 18
4.1.1 Esquema relacional..... 18
4.1.2 Entidades principales del modelo..... 19
4.1.3 Highlights del diseño (objetivos del Capítulo I reflejados en el modelo)..... 19
4.2. Implementación física del modelo..... 20
4.2.1 Ejemplo de creación de tabla (dbo.partidos)..... 20
4.3 Desarrollo del Tema 1 — Procedimientos Almacenados y Funciones (SP & UDF)... 21
4.3.1 Marco Teórico Aplicado..... 21
1. Procedimientos almacenados (SP)..... 21
2. Funciones definidas por el usuario (UDF)..... 22
3. Consideraciones de rendimiento..... 22
4. Inserciones (INSERT) — síntesis teórico-práctica..... 23
4.3.2 Procedimientos almacenados implementados..... 24
1. dbo.sp_Insertar_Opinion..... 24
2. dbo.sp_Modificar_Opinion..... 24
3. dbo.sp_Eliminar_Opinion..... 25
4.3.3 Funciones definidas por el usuario..... 25
1. dbo.fn_ObtenerNombreUsuario(@usuario_id)..... 25
2. dbo.fn_CalcularPuntajePromedioPartido(@partido_id)..... 26
3. dbo.fn_FormatearResultadoPartido(@partido_id)..... 26
4.3.4 Inserción de datos: comparación entre inserción directa y vía SP..... 27
a) Resultados obtenidos (según la salida de Messages)..... 27
Insert Directo..... 28
Insert via SP..... 28
b) Análisis de los resultados..... 29
Inserción directa..... 29
Inserción vía SP..... 29
c) Interpretación técnica..... 29
4.4 Desarrollo del Tema 2 — Optimización de Consultas a Través de Índices..... 30
4.4.1 Objetivo del experimento..... 30
4.4.2 Carga masiva de datos (01-carga_inicial.sql)..... 30
4.4.3 Consultas de prueba sin índices (02-busqueda_sin_indice.sql)..... 31
1. Consulta de rango por fecha (conteo anual)..... 31
2. Consulta con JOINs y filtros combinados (fecha + estado)..... 31
3. Consulta de agregación mensual..... 32
4.4.4 Creación de índices y repetición de consultas..... 32
a) Índice simple por fecha (03-crear_indice_repetir_consultas.sql)..... 32
b) Índice compuesto con columnas incluidas (04-indice_compuesto.sql)..... 32
4.4.5 Resultados de rendimiento..... 33
4.5 Desarrollo del Tema 3: Manejo de Transacciones y Transacciones Anidadas..... 34
4.5.1 Objetivo del experimento..... 34
4.5.2 Procedimientos transaccionales implementados..... 34
a) sp_Registrar_Usuario_Completo..... 34

b) sp_Calificar_y_Opinar....34
c) sp_Seguir_Equipo_Con_Recordatorios....35
○ el recordatorio erróneo no quedó persistido....35
d) sp_Registrar_Actividad_Usuario_Completa....35
4.5.3 Ejemplos de transacciones anidadas y savepoints....35
4.5.4 Pruebas de éxito y de error....36
4.5.5 Observaciones de rendimiento....37
4.6 Desarrollo del Tema 4: Vistas y Vistas Indexadas....38
4.6.1 Marco teórico aplicado....39
4.6.2 Contexto del modelo: tabla dbo.partidos....39
4.6.3 Vista simple para operaciones CRUD (vw_partidos_basicos)....40
4.6.4 Vista indexada para estadísticas (vw_partidos_por_liga_y_anio)....40
4.6.5 Pruebas de rendimiento con y sin vista indexada....41
Bloque A – consulta directa sobre dbo.partidos....42
Bloque B – consulta sobre la vista indexada....42
Comparación de planes....43
4.6.6 Script de rollback del Tema 4....44
Capítulo V — Conclusiones....45
5.1. Conclusiones sobre Procedimientos Almacenados (SP) y Funciones (UDF)....45
5.2 Conclusiones sobre Optimización de Consultas a Través de Índices....46
5.3 Conclusiones sobre Transacciones y Transacciones Anidadas....47
5.4 Conclusiones sobre Vistas y Vistas Indexadas....48
Capítulo VI — Bibliografía....49
6.1 Referencias citadas....49
6.2 Recursos complementarios....49

Índice de figuras:

Figura 1: Diagrama Entidad Relación....18
Figura 2: Ejemplo creación tabla partidos....20
Figura 3: Ejemplo de uso dbo.sp_Insertar_Opinion....24
Figura 4: Ejemplo de uso dbo.sp_Modificar_Opinion....25
Figura 5: Ejemplo de uso dbo.sp_Eliminar_Opinion....25
Figura 6: Ejemplo de uso dbo.fn_ObtenerNombreUsuario....26
Figura 7: Ejemplo de uso dbo.fn_CalcularPuntajePromedioPartido....26
Figura 8: Ejemplo de uso dbo.fn_FormatearResultadoPartido....27
Figura 9: Captura de pantalla del insert directo....28
Figura 10: Captura de pantalla del insert via SP....29
Figura 11: Consulta de rango por fecha....31
Figura 12: Búsqueda Específica con JOINs....31
Figura 13: Consulta de agregación mensual....32
Figura 14: Creación del Índice....32
Figura 15: Creación del Índice compuesto....33

Figura 16: Creación dbo.vw_partidos_basicos.....40
Figura 17: Pruebas con la vista dbo.vw_partidos_basicos.....40
Figura 18: Creacion de dbo.vw_partidos_por_liga_y_anio.....41
Figura 19: Índice Clustered Unico sobre la vista..... 41
Figura 20:Consulta directa sobre dbo.partidos.....42
Figura 21: Captura de pantalla sobre el Plan Bloque A.....42
Figura 22: Consulta sobre la vista indexada.....42
Figura 23: Captura de pantalla sobre el Plan Bloque B.....43
Figura 24: Captura de pantalla sobre comparacion de Plan A vs Plan B44

Capítulo I — Introducción

1.1 Tema y contexto

El presente Trabajo Práctico Integrador aborda el **diseño, implementación y validación de un modelo de base de datos relacional** para *Tribuneros*, una red social orientada al registro, puntaje y comentario de partidos de fútbol por parte de la comunidad.

El tema se centra en el **estudio y resolución del problema de modelar de manera consistente, normalizada y verificable** las entidades, relaciones y reglas de negocio que intervienen en la interacción de usuarios con eventos futbolísticos.

El título del trabajo refleja este propósito:
“Tribuneros — Diseño, implementación y verificación del esquema relacional de una red social futbolera”, donde el foco principal es investigar cómo construir un modelo de datos que sostenga las funcionalidades esenciales de una plataforma social deportiva.

1.2 Definición del problema

En el ecosistema actual, las personas aficionadas al fútbol utilizan múltiples herramientas aisladas para gestionar su actividad futbolera: aplicaciones de resultados, redes sociales generalistas, anotaciones personales y sitios con estadísticas incompletas. Esta fragmentación genera varios problemas:

- Dificultad para **centralizar el historial personal** de partidos vistos.
- Ausencia de un espacio unificado para **emitir opiniones, puntuar encuentros y compartir reseñas**.
- Falta de **trazabilidad real** entre partido–usuario–interacción (visualización, calificación, comentario).
- Imposibilidad de obtener **estadísticas personalizadas**, recomendaciones o contenido destacado curado por la comunidad.
- Debilidad en los sistemas actuales para manejar la **privacidad**, el control de spoilers y las preferencias de los usuarios.

Desde una perspectiva académica, este problema se traduce en una pregunta central:

> **¿Cómo diseñar un modelo de datos relacional que permita capturar de manera íntegra, coherente y escalable todas las interacciones que los usuarios realizan dentro de una red social basada en partidos de fútbol?**

De esta pregunta derivan interrogantes específicos:

- ¿Cuáles son las entidades, relaciones y restricciones necesarias para representar el dominio del fútbol y la actividad social asociada?
- ¿Cómo garantizar la integridad referencial, la consistencia temporal y la normalización del esquema?
- ¿Qué mecanismos SQL (procedimientos, funciones, transacciones, índices, vistas) son adecuados para implementar la lógica requerida?
- ¿Cómo validar que la estructura implementada satisface los requisitos del problema?

El Trabajo Práctico se desarrolla como una investigación aplicada, donde los alumnos deben explorar, diseñar, implementar y verificar soluciones técnicas que respondan a estos interrogantes.

1.3 Objetivos

1.3.1 Objetivo general

Diseñar, documentar e implementar el modelo de datos relacional de *Tribuneros* en SQL Server, resolviendo el problema de representar de forma íntegra y verificable las interacciones futboleras de los usuarios dentro de una plataforma social.

1.3.2 Objetivos específicos

Derivados del problema planteado, se busca:

1. **Identificar** las entidades clave del dominio: usuarios, perfiles, equipos, ligas, partidos e interacciones sociales.
2. **Construir** un esquema relacional normalizado con claves primarias, foráneas y restricciones de integridad adecuadas.
3. **Implementar** la base mediante los scripts de creación y cargar datos representativos usando los archivos:
    - creacion.sql
    - carga_inicial.sql
4. **Validar** la consistencia estructural y de datos utilizando los scripts:

- verificacion.sql
- conteo.sql

5. **Aplicar y documentar** los contenidos de los temas de investigación:
    - Procedimientos y funciones (Tema 1)
    - Índices y optimización (Tema 2)
    - Transacciones y manejo de concurrencia (Tema 3)
    - Vistas y vistas indexadas (Tema 4)
6. **Justificar técnicamente** cada decisión de diseño y relacionarla con los conceptos teóricos de Bases de Datos I.
7. **Elaborar documentación académica** que permita evaluar el diseño, su implementación y su correspondencia con las necesidades del sistema.

Capítulo II — Marco conceptual o referencial

El presente capítulo expone los conceptos, fundamentos y categorías teóricas que permiten contextualizar el problema abordado por este trabajo. Su propósito es **situar a Tribuneros dentro de un marco conceptual sólido**, que facilite comprender la lógica del dominio, las decisiones de diseño del modelo de datos y las relaciones entre los elementos que lo componen.

Al tratarse de un sistema sociotécnico —una red social centrada en experiencias futboleras— es necesario establecer definiciones operativas, límites conceptuales y vínculos entre nociones tecnológicas, sociales y de gestión de información.

2.1. Enfoque y delimitación del problema

El trabajo se inscribe en la intersección entre **gestión de información**, **plataformas sociales** y **consumo deportivo**, específicamente en el contexto del fútbol profesional. A efectos analíticos, se establecen los siguientes elementos del marco conceptual:

2.1.1 Tema

El registro social y estructurado de experiencias de visionado de fútbol y la producción de calificaciones y reseñas con control de spoilers y privacidad configurable.

2.1.2 Objeto de estudio

Las **interacciones verificables entre usuarios, partidos y contenido generado**, incluyendo:

- visualizaciones (qué se vio, cuándo, por cuánto tiempo)
- calificaciones
- opiniones con o sin spoilers
- favoritos y seguimientos
- recordatorios

- curaduría editorial/algorítmica de partidos destacados

2.1.3 Unidad de análisis

Las tres entidades centrales involucradas en el proceso completo “ver → calificar → opinar”:

1. El **usuario** (identidad, preferencias, seguridad).
2. El **partido** (evento deportivo con atributos propios).
3. La **publicación** (opinión y/o calificación como resultado social pos-partido).

2.1.4 Ámbito conceptual

El dominio de ligas y equipos profesionales, extensible a selecciones, copas y torneos internacionales. Se tiene especial foco en:

- el **consumo post partido**,
- la **conversación segura**,
- el **acceso controlado al contenido** cuando incluye spoilers, y
- la **trazabilidad** de la interacción usuario–partido.

El marco abarca tanto componentes teóricos (plataformas sociales, UGC, reputación, integridad de datos) como elementos técnicos (hash de contraseñas, claves foráneas, normalización, restricciones).

2.2. Conceptos clave del marco conceptual

Se presentan las categorías fundamentales que estructuran el problema y orientan la construcción del modelo relacional. Cada concepto incluye **una definición teórica breve**, una **definición operativa en el contexto del sistema** y el **rol** que desempeña dentro del modelo.

1. **Plataforma social (social network)**

- **Definición teórica:** sistema sociotécnico que habilita interacción, producción de contenido y vínculos entre usuarios.
- **Definición operativa**: en Tribuneros se materializa mediante las tablas *usuarios*, *perfiles*, *opiniones*, *favoritos*, *seguidos*, *visualizaciones*.
- **RoI**: provee el andamiaje para el efecto red, la personalización y el control de acceso.

2. **Contenido generado por usuarios (UGC)**

- **Definición teórica:** información producida directamente por las personas, no por la plataforma.
- **Definición operativa:** opiniones textuales y calificaciones numéricas asociadas a partidos.
- **RoI:** constituye el insumo que nutre la conversación, la relevancia y la curaduría.

3. **Experiencia de visualización**

- **Definición operativa:** acto de ver un partido con registro estructurado *(minutos vistos, medio, timestamp*).
- **RoI:** requisito habilitante para emitir una calificación u opinión; base para métricas de engagement.

4. **Calificación (rating)**

- **Definición operativa:** valoración numérica discreta almacenada en *calificaciones.puntaje.*
- **RoI:** permite rankings, agregaciones grupales y recomendaciones básicas.

5. **Opinión / Reseña**

- **Definición operativa:** texto con control de visibilidad (pública/privada) y bandera de spoiler.
- **RoI:** soporte cualitativo de discusión, memoria y narrativa del partido.

6. **Spoilers**

- **Definición teórica:** revelación anticipada de información crítica que afecta la experiencia del espectador.

- **Definición operativa:** atributo *tiene_spoilers* en opiniones.
- **Rol:** regula el feed y la visibilidad según reglas de protección de experiencia.

7. Privacidad

- **Definición operativa:** atributo *publica* en opiniones y configuraciones del perfil.
- **Rol:** determina el alcance del contenido y garantiza el cumplimiento de preferencias del usuario.

8. Seguimiento y favoritos

- **Definición operativa:** seguimiento de equipos (*seguimiento_equipos*) y marcación de partidos (*favoritos*).
- **Rol:** nutren el feed, recomendaciones y notificaciones.

9. Curaduría / Partidos destacados

- **Definición operativa:** marca editorial o algorítmica registrada en *partidos_destacados*.
- **Rol:** optimiza descubrimiento y concentra la conversación en encuentros relevantes.

10. Recordatorios

- **Definición operativa:** programación de alertas para visualizar partidos (*recordatorios*).
- **Rol:** propicia completar el ciclo completo (ver → calificar → opinar).

11. Reputación y confianza

- **Definición teórica:** nociones derivadas del comportamiento histórico en plataformas sociales.
- **Definición operativa:** se infiere a partir de actividad (opiniones, consistencia, visibilidad pública).
- **Rol:** orientar ordenamiento del feed y distribución de contenido.

12. Integridad e interoperabilidad de datos

- **Definición teórica:** propiedad de los sistemas que asegura coherencia interna y confiabilidad de la información.

- **Definición operativa:** claves foráneas, tipos válidos, estados permitidos, relaciones entre ligas–equipos–partidos.
- **Rol:** garantiza consistencia en consultas internas y reportes.

13. **Autenticación y Seguridad**

- **Definición teórica:** conjunto de mecanismos que protegen identidad y acceso a datos.
- **Definición operativa:**
  contraseñas hash SHA2_512 en *usuarios.password_hash*, procedimientos *sp_usuario_set_password_simple* y *sp_usuario_login_simple.*
- **Rol:** proteger credenciales y preservar integridad del sistema.

2.3 Terminología y referencia de datos

Para consolidar el marco conceptual y asegurar coherencia entre teoría e implementación, se definen los siguientes elementos técnicos de referencia:

2.3.1 Modelo lógico

Se detalla en el Capítulo III y se implementa físicamente mediante el script creacion.sql, que contiene la definición de tablas, claves primarias, claves foráneas y restricciones de integridad.

2.3.2 Datos de ejemplo

Los datos representativos utilizados en las pruebas y validaciones provienen del archivo carga_inicial.sql, que incluye ligas, equipos, usuarios, partidos y contenido social simulado.

2.3.3 Diccionario de datos

El diccionario completo —columnas, tipos, claves y descripciones funcionales— se encuentra documentado en docs/diccionario_datos.md y constituye un insumo esencial para interpretar el modelo.

Capítulo III — Metodología

El presente capítulo describe el proceso metodológico adoptado para el desarrollo del Trabajo Práctico Integrador *Tribuneros*, desde la planificación y la organización grupal hasta la implementación técnica y la elaboración de los informes correspondientes. Se detallan las acciones realizadas, las herramientas utilizadas y las estrategias de trabajo aplicadas para resolver los cuatro temas de investigación solicitados por la cátedra.

3.1 Descripción de cómo se realizó el Trabajo Práctico

El proyecto fue desarrollado por un equipo de tres integrantes bajo una modalidad de trabajo colaborativo. El proceso general se estructuró en dos grandes etapas:
(1) construcción del núcleo del modelo de datos y **(2) desarrollo incremental por temas de investigación**.

3.1.1 Construcción del núcleo del modelo

En primera instancia, el grupo definió y estructuró el esqueleto del proyecto mediante la creación de los siguientes scripts base:

- creacion.sql – definición completa del esquema tribuneros_bdi, claves primarias, foráneas y restricciones.
- carga_inicial.sql – inserción de datos representativos para pruebas.
- verificacion.sql – consultas de integridad, relaciones y consistencia del modelo.
- conteo.sql – métricas generales y validación cuantitativa.
- limpieza_datos.sql – restablecimiento del entorno para ensayos sucesivos.

Esta primera fase permitió establecer un entorno estable, reproducible y versionado para la ejecución del resto del trabajo práctico.

3.1.2 Organización por ramas y flujo de trabajo

El equipo utilizó **GitHub** como entorno colaborativo:

- Todo el desarrollo se integró en la **rama *develop***.

- Se mantuvo un flujo de trabajo basado en commits, revisiones y ordenamiento de carpetas.

- Los informes teóricos y scripts se organizaron en directorios separados:

    - **docs/tema-informe.md** – Documento académico de cada tema (procedimientos, índices, transacciones, vistas).

    - **scripts/tema/** – Scripts ordenados, limpios y numerados para ejecutar pruebas, validar hipótesis y replicar resultados por tema.

3.1.3 Desarrollo incremental por temas

A diferencia de un desarrollo paralelo, adoptamos un enfoque **individual por tema**, donde:

- **Tema 1**: desarrollado por un integrante.

- **Tema 2**: desarrollado por un segundo integrante.

- **Tema 3**: desarrollado por un tercer integrante.

- **Tema 4**: desarrollado **en conjunto por los tres**

Cada integrante siguió siempre el mismo orden metodológico:

1. **Investigación teórica** (Microsoft Docs, material de la cátedra, fuentes externas, IA).

2. **Implementación técnica** en SQL Server (scripts del tema).

3. **Pruebas prácticas**, validación, correcciones.

4. **Elaboración del informe** una vez que los scripts funcionaban correctamente.

5. **Integración en el repositorio**, manteniendo prolijidad y consistencia.

No se realizó documentación simultánea:
primero se investigaba, luego se implementaba y recién al final se documentaba.

3.1.4 Validación y cierre

Una vez finalizados los cuatro temas:

- Se realizaron pruebas de consistencia mediante los scripts de verificación.

- Se estandarizaron nombres, rutas, numeración y estructura general del repositorio.

- Finalmente, se consolidaron los informes en un único documento y se realizó la exportación a PDF para su presentación formal.

3.2 Herramientas (instrumentos y procedimientos)

Para la recolección, análisis, implementación y validación del modelo de datos se utilizaron las siguientes herramientas y métodos:

3.2.1 Herramientas técnicas

- **SQL Server / SQL Server Management Studio (SSMS):**
  Motor principal de base de datos y entorno de ejecución de todos los scripts. Permitió validar restricciones, índices, transacciones y vistas.

- **Visual Studio Code (VSCode):**
  Editor utilizado para redactar los scripts, organizar carpetas y escribir documentación en formato Markdown.

- **GitHub (repositorio colaborativo):**
  - Control de versiones (branches, commits y merges).
  - Integración de informes, scripts y documentación técnica.
  - Coordinación grupal mediante issues e historial de trabajo.

- **Internet y documentación oficial (Microsoft Learn, SQL Docs):**
  Base de apoyo conceptual para comprender índices, procedimientos, transacciones, vistas indexadas y buenas prácticas SQL.

- **Inteligencia Artificial (IA):**
  Se empleó como asistente de investigación y como apoyo en la generación de ejemplos, refactorización de consultas y verificación de conceptos teóricos aplicados al proyecto.

3.2.2 Procedimientos de trabajo

- **Investigación individual distribuida:**
  Cada integrante investigó uno de los temas 1, 2 o 3; el tema 4 fue trabajado por los tres.

- **Desarrollo secuencial por integrante:**
  Investigación → Scripts → Pruebas → Informe del tema.

- **Validación en equipo:**
  Se realizaron revisiones grupales antes de integrar cualquier tema a la rama ***develop***.

- **Revisión final conjunta:**
  Consolidación de informes, ordenamiento del repositorio y limpieza del proyecto.

Capítulo IV — Resultados

El presente capítulo expone los resultados obtenidos a partir del diseño, implementación y experimentación realizados sobre la base de datos **Tribuneros**, en relación directa con los objetivos planteados en el Capítulo I.

Se presentan evidencias objetivas —diagramas, tablas, scripts ejecutados, capturas de planes de ejecución y resultados de pruebas— que permiten evaluar el funcionamiento del modelo y la aplicación práctica de los cuatro temas de investigación: procedimientos y funciones, índices, transacciones y vistas.

En este capítulo **no se formulan conclusiones**, sino que se documentan los hallazgos y resultados generados durante el desarrollo.

4.1. Diseño del modelo de datos: Representación del dominio

Como resultado de las actividades de análisis y modelado (Capítulos I y II), se elaboró el **Esquema Relacional** que sustenta toda la implementación. Este esquema fue derivado del diagrama ER y se presenta como evidencia del diseño conceptual y lógico.

4.1.1 Esquema relacional

Figura 1: Diagrama Entidad Relación

4.1.2 Entidades principales del modelo

- **Usuarios:** usuarios, perfiles
- **Catálogos:** ligas, equipos
- **Partidos:** partidos
- **Interacciones sociales:** visualizaciones, calificaciones, opiniones, favoritos
- **Relaciones sociales:** seguimiento_equipos, seguimiento_ligas, seguimiento_usuarios
- **Curaduría:** partidos_destacados, recordatorios

4.1.3 Highlights del diseño (objetivos del Capítulo I reflejados en el modelo)

- **Integridad:**
    - PK/FK explícitas
    - Restricciones UNIQUE en (usuario_id, partido_id) para evitar duplicidades
- **Validación automática:**
    - CHECK en partidos.estado, visualizaciones.medio, recordatorios.estado
- **Reglas de negocio:**
    - ON DELETE CASCADE para usuarios
    - RESTRICT y SET NULL en catálogos para conservar historial
- **Rendimiento:**
    - Índices sobre partidos.fecha_utc, liga_id y columnas de usuario en tablas de interacción

El diseño resultante satisface los objetivos específicos de **estructurar un esquema normalizado**, garantizar **consistencia referencial**, y permitir **consultas eficientes**.

4.2. Implementación física del modelo

La implementación del modelo se realizó mediante los scripts del repositorio:

- **creacion.sql** — Definición completa del esquema.
- **carga_inicial.sql** — Datos representativos del dominio.
- **verificacion.sql** — Consultas de chequeo de integridad.
- **conteo.sql** — Métricas iniciales de auditoría.

4.2.1 Ejemplo de creación de tabla (dbo.partidos)

```
/* ==================== 3) Partidos ==================== */
CREATE TABLE dbo.partidos (
  id              INT           NOT NULL,
  id_externo      VARCHAR(80)   NULL,
  liga_id         INT           NULL,
  temporada       SMALLINT      NULL,                    -- año
  ronda           NVARCHAR(40)  NULL,
  fecha_utc       DATETIME2(0)  NOT NULL,                -- precisión a minuto
  estado          TINYINT       NOT NULL,                -- 0=prog,1=vivo,2=fin,3=posp,4=canc
  estadio         NVARCHAR(120) NULL,
  equipo_local    INT           NOT NULL,
  equipo_visitante INT          NOT NULL,
  goles_local     TINYINT       NULL,
  goles_visitante TINYINT       NULL,
  creado_en       DATETIME2(3)  NOT NULL DEFAULT SYSUTCDATETIME(),
  CONSTRAINT PK_partidos PRIMARY KEY (id),
  CONSTRAINT CK_partidos_estado CHECK (estado IN (0,1,2,3,4)),
  CONSTRAINT CK_partidos_equipos_distintos CHECK (equipo_local <> equipo_visitante),
  CONSTRAINT FK_partidos_liga       FOREIGN KEY (liga_id)          REFERENCES dbo.ligas(id) ON DELETE SET NULL,
  CONSTRAINT FK_partidos_local      FOREIGN KEY (equipo_local)     REFERENCES dbo.equipos(id),
  CONSTRAINT FK_partidos_visitante  FOREIGN KEY (equipo_visitante) REFERENCES dbo.equipos(id)
);

CREATE INDEX IX_partidos_fecha ON dbo.partidos(fecha_utc);
CREATE INDEX IX_partidos_liga  ON dbo.partidos(liga_id);
GO
```

Figura 2: Ejemplo creación tabla partidos

Se presenta este ejemplo como evidencia del diseño físico aplicado a tablas clave del sistema.

4.3 Desarrollo del Tema 1 — Procedimientos Almacenados y Funciones (SP & UDF)

En esta sección se presentan los resultados obtenidos durante la implementación y prueba de los **procedimientos almacenados (SP)** y **funciones definidas por el usuario (UDF)** desarrollados para el proyecto integrador.
El objetivo operativo fue incorporar mecanismos de encapsulación de lógica SQL en la base de datos, evaluar su funcionamiento y comparar su comportamiento frente a operaciones directas de manipulación de datos.

A continuación, se describen los componentes implementados, las ejecuciones realizadas y los resultados obtenidos.

4.3.1 Marco Teórico Aplicado

1. **Procedimientos almacenados (SP)**

Los **procedimientos almacenados** son rutinas precompiladas que se ejecutan en el motor SQL. Al invocarse por primera vez generan un **plan de ejecución cacheado**, el cual puede reutilizarse o recompilarse según sea necesario. Su uso permite:

- Encapsular lógica de negocio dentro de la base de datos.
- Reutilizar código y centralizar validaciones.
- Controlar permisos vía GRANT EXECUTE o EXECUTE AS.
- Implementar lógica transaccional (BEGIN TRAN, COMMIT, ROLLBACK) con manejo de errores estructurado.

Aplicación en el proyecto:
Se desarrollaron SP destinados a **insertar, actualizar y eliminar opiniones**, incluyendo validaciones de integridad como:

- Unicidad por combinación (partido_id, usuario_id)
- Verificación de claves foráneas
- Control de ownership basado en usuario_id
- Ejecución bajo transacción para asegurar consistencia

---

2. Funciones definidas por el usuario (UDF)

Las **UDF** permiten encapsular lógica de consulta y devolver:

- Valores escalares (scalar UDF), o
- Tablas (table-valued functions)

En T-SQL, las funciones **no generan efectos secundarios**: no pueden modificar datos ni invocar SP.

Se distinguen principalmente dos tipos:

- **iTVF (inline table-valued functions):** mayor rendimiento y mejores estimaciones de cardinalidad.
- **mTVF (multi-statement):** más flexibles, pero generan peor estimación y afectan el plan.

Aplicación en el proyecto:
Se utilizaron UDF para:

- Obtener nombre_usuario a partir del usuario_id
- Calcular puntajes promedio
- Formatear información para reportes y vistas

---

3. Consideraciones de rendimiento

Tanto SP como UDF interactúan con el optimizador, el **caché de planes** y la estimación de datos. Algunas consideraciones:

- En SP puede ocurrir *parameter sniffing*. Se mitiga con:
    - OPTION (RECOMPILE)
    - Captura de parámetros en variables locales
    - OPTIMIZE FOR en casos especiales
- Desde SQL Server 2019, algunas UDF escalares pueden **inlinarse**, reduciendo su costo.

- Para grandes volúmenes, **priorizar operaciones set-based** sobre ciclos fila a fila.

---

4. Inserciones (INSERT) — síntesis teórico-práctica

Las operaciones de inserción influyen en integridad, concurrencia y el plan de ejecución. Puntos clave:

1. **Set-based vs por fila:**
    - Preferir INSERT ... SELECT y **TVP (Table-Valued Parameters)** para envíos por lotes.
    - Evitar invocar el mismo SP por cada fila, salvo en escenarios OLTP.
    - En caso de UPSERT, MERGE sólo si está bien delimitado.
2. **SP vs INSERT directo:**
    - Los SP agregan verificaciones, transacciones y control de permisos.
    - El overhead es mínimo en lotes pequeños.
    - Para cargas masivas, usar SP set-based o TVP.
3. **Concurrencia e integridad:**
    - Las FKs y UNIQUE introducen bloqueos.
    - Es importante planificar índices de soporte.
    - Para mucha lectura concurrente, evaluar SNAPSHOT o READ COMMITTED SNAPSHOT.
4. **Medición y optimización:**
    - Analizar con SET STATISTICS IO/TIME
    - Revisar el plan (seek > scan, evitar key lookups)
    - Usar SET NOCOUNT ON
    - Retornar la PK generada vía SCOPE_IDENTITY()

4.3.2 Procedimientos almacenados implementados

Se desarrollaron tres procedimientos almacenados principales (según el archivo *01_procedimientos.sql*), destinados a gestionar la inserción y modificación de registros de la base tribuneros_bdi.

1. **dbo.sp_Insertar_Opinion**

Propósito:
Inserta una nueva opinión sobre un partido, validando puntaje, existencia de usuario/partido y evitando datos inconsistentes.

Parámetros: @usuario_id, @partido_id, @puntaje, @comentario, @opinion_id (OUTPUT — devuelve el nuevo ID generado)

Validaciones incluidas:

- Inserta correctamente registros válidos.
- Rechaza puntajes fuera del rango permitido.
- Rechaza registros con referencias inexistentes.
- Genera un nuevo opinion_id incremental.

Ejemplo de uso:

```
DECLARE @nueva_opinion_id INT;
EXEC dbo.sp_Insertar_Opinion
    @partido_id = 1,
    @usuario_id = 2, -- ID de 'ana.ferro'
    @titulo = 'Visto desde la neutralidad',
    @cuerpo = 'Un partido que quedará en la historia del fútbol sudamericano.',
    @publica = 1,
    @tiene_spoilers = 0,
    @opinion_id = @nueva_opinion_id OUTPUT;

SELECT * FROM dbo.opiniones WHERE id = @nueva_opinion_id;
```

Figura 3: Ejemplo de uso dbo.sp_Insertar_Opinion

2. **dbo.sp_Modificar_Opinion**

Propósito: Actualiza el contenido de una opinión existente.

Parámetros: @opinion_id, @usuario_id (para seguridad), @titulo, @cuerpo, @publica, @tiene_spoilers.

Lógica de seguridad: Solo el usuario que creó la opinión puede modificarla.

Ejemplo de uso:

```
EXEC dbo.sp_Modificar_Opinion
    @opinion_id = 1, -- ID de la opinión creada por 'tobiager'
    @usuario_id = 1,  -- ID de 'tobiager'
    @titulo = 'La gloria eterna en Madrid (editado)',
    @cuerpo = 'Partidazo histórico. River campeón con autoridad. 3-1 y a casa. Inolvidable.',
    @publica = 1,
    @tiene_spoilers = 1;
```

Figura 4: Ejemplo de uso dbo.sp_Modificar_Opinion

3. dbo.sp_Eliminar_Opinion

Propósito:
Elimina una opinión registrada de la base de datos.

Parámetros: @opinion_id

Resultados obtenidos:

- Borra correctamente el registro indicado.
- No afecta opiniones de otros usuarios o partidos.
- Verificado mediante consultas de post-eliminación.

Ejemplo de uso:

```
-- Suponiendo que la opinión con ID 101 fue creada por el usuario con ID 1
EXEC dbo.sp_Borrar_Opinion
    @opinion_id = 101,
    @usuario_id = 1;
```

Figura 5: Ejemplo de uso dbo.sp_Eliminar_Opinion

4.3.3 Funciones definidas por el usuario

Las funciones fueron implementadas en *02_funciones.sql*, con el objetivo de encapsular operaciones de consulta y facilitar la reutilización desde el sistema.

1. dbo.fn_ObtenerNombreUsuario(@usuario_id)

Propósito:
Devuelve el *nombre_usuario* correspondiente al *usuario_id* recibido.
Permite simplificar consultas evitando combinaciones (JOIN) con la tabla perfiles o usuarios.

Ejemplo de uso:

```sql
SELECT
  titulo,
  cuerpo,
  dbo.fn_ObtenerNombreUsuario(usuario_id) AS autor
FROM dbo.opiniones
WHERE partido_id = 1;
```

Figura 6: Ejemplo de uso dbo.fn_ObtenerNombreUsuario

2. dbo.fn_CalcularPuntajePromedioPartido(@partido_id)

Propósito:
Calcula y devuelve el puntaje promedio otorgado por los usuarios a un partido determinado.
Se utiliza para reportes y rankings dentro del sistema.

Ejemplo de uso:

```sql
SELECT
  p.id,
  eq_local.nombre AS local,
  eq_visit.nombre AS visitante,
  dbo.fn_CalcularPuntajePromedioPartido(p.id) AS puntaje_promedio
FROM dbo.partidos p
JOIN dbo.equipos eq_local ON p.equipo_local = eq_local.id
JOIN dbo.equipos eq_visit ON p.equipo_visitante = eq_visit.id;
```

Figura 7: Ejemplo de uso dbo.fn_CalcularPuntajePromedioPartido

3. dbo.fn_FormatearResultadoPartido(@partido_id)

Propósito:
Devuelve una cadena con el resultado final del partido en formato "goles_local - goles_visitante". Si el partido aún no tiene marcador cargado, devuelve "vs" para indicar que no se disputó o no se registró el resultado.

Ejemplo de uso:

```
SELECT
  p.id,
  dbo.fn_FormatearResultadoPartido(p.id) AS resultado
FROM dbo.partidos p;
```

Figura 8: Ejemplo de uso dbo.fn_FormatearResultadoPartido

4.3.4 Inserción de datos: comparación entre inserción directa y vía SP

Para evaluar el comportamiento de las operaciones de inserción, se realizaron dos lotes de carga, ambos compuestos por **3 inserciones** de opiniones:

- **03_datos_insert_directo.sql** → Inserción directa mediante INSERT INTO
- **04_datos_insert_via_sp.sql** → Inserción mediante tres llamadas a EXEC dbo.sp_insertar_opinion

En ambos casos se utilizó:

SET STATISTICS IO ON;

SET STATISTICS TIME ON;

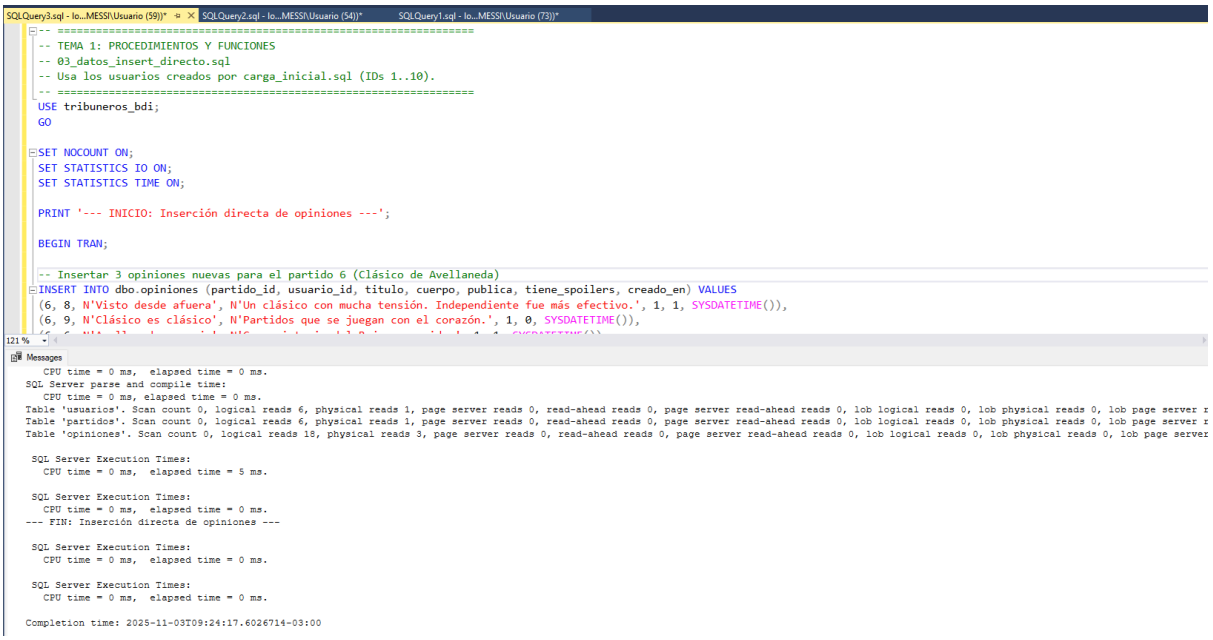
lo que permitió registrar el costo en lecturas lógicas y tiempos de CPU/elapsed reportados por SQL Server Management Studio.

---

a) Resultados obtenidos (según la salida de *Messages*)

Método	usuarios LReads	partidos LReads	opiniones LReads	Total LReads	CPU (ms)	Elapsed (ms)
INSERT directo (03_)	6	6	18	**30**	≈0	≈0
Vía SP (04_) — 3× EXEC	6	6	18	**30**	picos 0–18	picos 17–18

Insert Directo

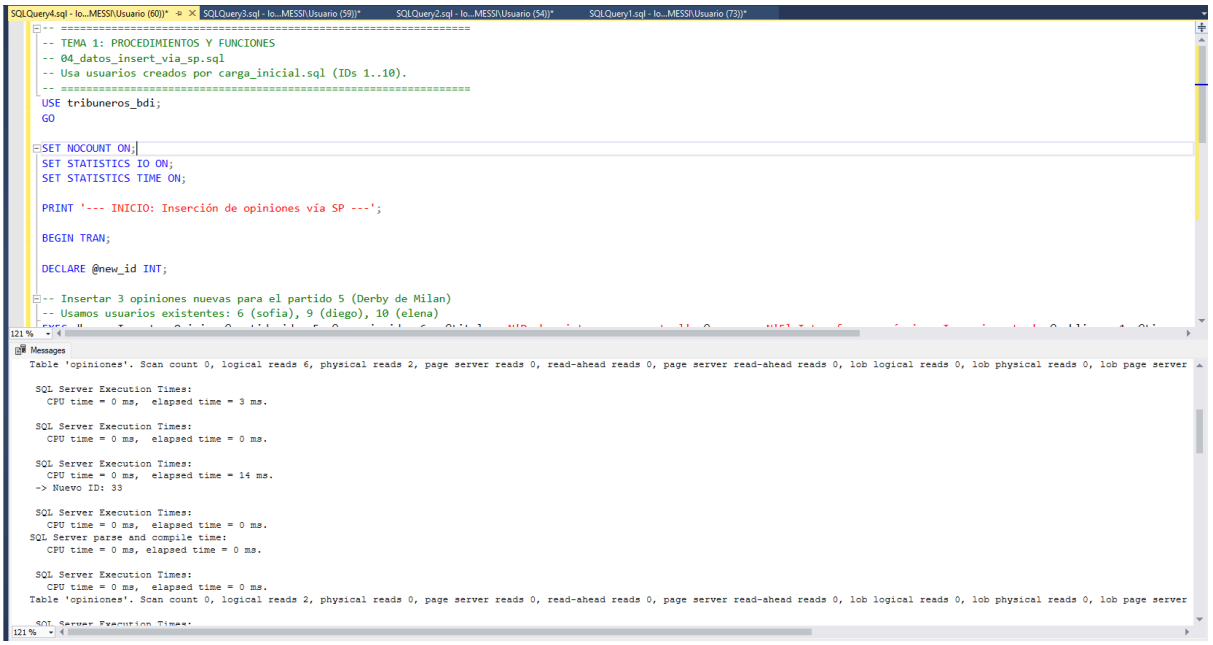


```
-- ================================================================
-- TEMA 1: PROCEDIMIENTOS Y FUNCIONES
-- 03_datos_insert_directo.sql
-- Usa los usuarios creados por carga_inicial.sql (IDs 1..10).
-- ================================================================
USE tribuneros_bdi;
GO

SET NOCOUNT ON;
SET STATISTICS IO ON;
SET STATISTICS TIME ON;

PRINT '--- INICIO: Inserción directa de opiniones ---';

BEGIN TRAN;

-- Insertar 3 opiniones nuevas para el partido 6 (Clásico de Avellaneda)
INSERT INTO dbo.opiniones (partido_id, usuario_id, titulo, cuerpo, publica, tiene_spoilers, creado_en) VALUES
(6, 8, N'Visto desde afuera', N'Un clásico con mucha tensión. Independiente fue más efectivo.', 1, 1, SYSDATETIME()),
(6, 9, N'Clásico es clásico', N'Partidos que se juegan con el corazón.', 1, 0, SYSDATETIME()),
```

```
Messages
   CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
Table 'usuarios'. Scan count 0, logical reads 6, physical reads 1, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server r
Table 'partidos'. Scan count 0, logical reads 6, physical reads 1, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server r
Table 'opiniones'. Scan count 0, logical reads 18, physical reads 3, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 5 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
--- FIN: Inserción directa de opiniones ---

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.

Completion time: 2025-11-03T09:24:17.6026714-03:00
```

Figura 9: Captura de pantalla del insert directo

Insert via SP



```
-- ================================================================
-- TEMA 1: PROCEDIMIENTOS Y FUNCIONES
-- 04_datos_insert_via_sp.sql
-- Usa usuarios creados por carga_inicial.sql (IDs 1..10).
-- ================================================================
USE tribuneros_bdi;
GO

SET NOCOUNT ON;
SET STATISTICS IO ON;
SET STATISTICS TIME ON;

PRINT '--- INICIO: Inserción de opiniones vía SP ---';

BEGIN TRAN;

DECLARE @new_id INT;

-- Insertar 3 opiniones nuevas para el partido 5 (Derby de Milan)
-- Usamos usuarios existentes: 6 (sofia), 9 (diego), 10 (elena)
```

```
Messages
Table 'opiniones'. Scan count 0, logical reads 6, physical reads 2, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 3 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 14 ms.
 -> Nuevo ID: 33

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 0 ms.
Table 'opiniones'. Scan count 0, logical reads 2, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server
```

Figura 10: Captura de pantalla del insert via SP

b) Análisis de los resultados

Inserción directa

- Ejecución en un único lote INSERT … VALUES (…), (…), (…).
- El total de lecturas lógicas fue de **30**, distribuidas entre:
    - Tabla **usuarios**: 6 LReads
    - Tabla **partidos**: 6 LReads
    - Tabla **opiniones**: 18 LReads

Inserción vía SP

- Tres llamados consecutivos a sp_insertar_opinion.
- El total acumulado también fue de **30 lecturas lógicas**.
- Se observaron picos aislados de tiempo por llamada (entre 0 y 18 ms), esperables por la ejecución fila-a-fila.

---

c) Interpretación técnica

Costo de IO equivalente:
Ambos enfoques mostraron la misma cantidad total de lecturas lógicas (**30**), ya que:

- La inserción requiere validar claves foráneas → lecturas en usuarios y partidos.
- La tabla opiniones realiza:
    - Inserción del registro
    - Verificación de unicidad (UQ (partido_id, usuario_id))

Diferencia operativa:

- **INSERT directo:** inserta las 3 filas en un solo lote.
- **SP:** ejecuta la lógica + validaciones para cada fila por separado.

Para volúmenes pequeños, ambos métodos presentan un rendimiento equivalente.

4.4 Desarrollo del Tema 2 — Optimización de Consultas a Través de Índices

Esta sección presenta los resultados obtenidos al aplicar distintas estrategias de indexación sobre la tabla *partidos* de la base *tribuneros_bdi*, con el objetivo de evaluar el impacto en el rendimiento de consultas sobre al menos **1.000.000 de registros**.

4.4.1 Objetivo del experimento

- Medir el costo de ejecutar consultas de filtrado y agregación **sin índices**, con un **índice simple** por fecha, y con un **índice compuesto** que incluya columnas adicionales.

- Registrar planes de ejecución, costos estimados y comportamiento del motor de SQL Server.

Los scripts utilizados están organizados en script/tema2-indices.

4.4.2 Carga masiva de datos (01-carga_inicial.sql)

Se implementó un script para poblar masivamente las tablas de dominio:

- **Ligas:** 50 registros.

- **Equipos:** 500 registros.

- **Partidos:** 1.000.000 registros.

Características principales de la carga:

- Inserción controlada mediante bucles WHILE para mantener la secuencia de IDs y permitir **ejecuciones repetidas** del script.

- Asignación cíclica de ligas y países mediante expresiones CASE.

- Generación de campos derivados (por ejemplo, URL de escudos en formato https://escudos.tribuneros.com/{id}.png).

- Uso de **batches** y eliminación temporal de índices para acelerar la inserción.

- Garantía de integridad referencial: se evitan equipos inexistentes y se respetan las claves foráneas.

Resultado: se obtiene una tabla ***partidos*** con volumen suficiente para evaluar el comportamiento de los índices en escenarios de carga realista.

4.4.3 Consultas de prueba sin índices (02-busqueda_sin_indice.sql)

Con la tabla partidos sin índices adicionales sobre fecha_utc, se ejecutaron tres tipos de consultas:

1. **Consulta de rango por fecha (conteo anual)**

```sql
SELECT COUNT(*) AS total_partidos_2023
FROM dbo.partidos
WHERE fecha_utc >= '2023-01-01'
  AND fecha_utc < '2024-01-01';
```

Figura 11: Consulta de rango por fecha

2. **Consulta con JOINs y filtros combinados (fecha + estado)**

```sql
SELECT p.id, p.fecha_utc, …
FROM partidos p
JOIN equipos el ON p.equipo_local = el.id
JOIN equipos ev ON p.equipo_visitante = ev.id
WHERE p.fecha_utc >= '2024-01-01'
  AND p.fecha_utc < '2024-04-01'
  AND p.estado = 2;
```

Figura 12: Búsqueda Específica con JOINs

   - Recupera datos descriptivos de equipos y resultados.
   - Utiliza filtros por fecha y estado.

3. Consulta de agregación mensual

```
SELECT YEAR(fecha_utc), MONTH(fecha_utc), COUNT(*), SUM(...)
FROM partidos
WHERE fecha_utc >= '2023-01-01'
  AND fecha_utc < '2024-01-01'
GROUP BY YEAR(fecha_utc), MONTH(fecha_utc)
ORDER BY anio, mes;
```

Figura 13: Consulta de agregación mensual

a. Agrupa por año y mes.

b. Recorre todas las filas del período solicitado.

En todos los casos, los planes de ejecución mostraron **lecturas masivas** sobre la tabla base y costos elevados.

4.4.4 Creación de índices y repetición de consultas

a) Índice simple por fecha (03-crear_indice_repetir_consultas.sql)

Se creó un índice no agrupado sobre la columna fecha_utc:

```
CREATE INDEX IX_partidos_fecha ON dbo.partidos(fecha_utc);
```

Figura 14: Creación del Índice

Luego se repitieron las mismas consultas de la sección 4.4.3, registrando:

- cambio de Table Scan a **Index Seek/Scan** sobre IX_partidos_fecha,
- disminución del costo estimado y de las lecturas lógicas.

b) Índice compuesto con columnas incluidas (04-indice_compuesto.sql)

Posteriormente se reemplazó el índice anterior por uno compuesto:

```
CREATE INDEX IX_partidos_fecha_compuesto
ON dbo.partidos(fecha_utc, estado)
INCLUDE (liga_id, equipo_local, equipo_visitante, goles_local, goles_visitante, estadio);
```

Figura 15: Creación del Índice compuesto

Las consultas de prueba se ejecutaron nuevamente, observándose que:

- el índice cubre completamente las columnas consultadas
- se evita el acceso adicional a la tabla base (**cobertura total de la consulta**)
- los planes de ejecución muestran operadores de acceso más selectivos

---

4.4.5 Resultados de rendimiento

A partir de los planes de ejecución estimados se obtuvieron los siguientes valores relativos de costo para una de las consultas de rango por fecha:

Escenario	Costo estimado relativo
Sin índice sobre fecha_utc	1,5307
Índice simple IX_partidos_fecha	0,466
Índice compuesto IX_partidos_fecha_compuesto	0,23385

Hechos observados:

- El plan sin índice obliga a leer la tabla completa.
- Con el índice simple se reduce significativamente el costo y el número de páginas leídas.
- Con el índice compuesto más columnas incluidas se obtiene el **menor costo estimado**, aprovechando mejor la selectividad por fecha y estado evitando accesos adicionales a la tabla.

Estos resultados cuantitativos muestran el efecto de la indexación progresiva sobre el rendimiento de las consultas definidas en los objetivos del Tema 2.

4.5 Desarrollo del Tema 3: Manejo de Transacciones y Transacciones Anidadas

Esta sección presenta los resultados obtenidos al diseñar e implementar procedimientos almacenados con transacciones explícitas, transacciones anidadas y puntos de guardado (SAVE TRANSACTION) en la base tribuneros_bdi. Los scripts utilizados se encuentran en script/tema3-transacciones.

4.5.1 Objetivo del experimento

- Implementar procedimientos almacenados con manejo robusto de transacciones.
- Demostrar el uso de transacciones anidadas y puntos de guardado.
- Observar el comportamiento de COMMIT y ROLLBACK en distintos escenarios de éxito y error.
- Registrar patrones de uso de TRY/CATCH, SET XACT_ABORT ON y @@TRANCOUNT.

4.5.2 Procedimientos transaccionales implementados

Se desarrollaron cuatro procedimientos almacenados que cubren escenarios frecuentes del dominio Tribuneros:

a) sp_Registrar_Usuario_Completo

- Crea un usuario y su perfil dentro de una única transacción.
- Incluye dos operaciones INSERT (en usuarios y perfiles).
- Utiliza SET XACT_ABORT ON y bloque TRY/CATCH.
- En las pruebas, cuando una de las inserciones falló, se observaron **rollback completos**, sin filas huérfanas.

b) sp_Calificar_y_Opinar

- Registra una calificación y una opinión sobre un partido en una misma transacción.

- Valida existencia de partido y usuario, y que el partido esté finalizado.
- Devuelve los IDs generados mediante parámetros OUTPUT.
- Ante errores de validación o duplicidad, los registros de calificación y opinión no quedan almacenados.

c) sp_Seguir_Equipo_Con_Recordatorios

- Inserta un seguimiento a un equipo y crea recordatorios para sus próximos partidos.
- Implementa una transacción externa y puntos de guardado (SAVE TRANSACTION) para cada recordatorio.
- En el flujo de prueba, cuando un recordatorio se intentó crear con datos inválidos, se ejecutó ROLLBACK TO el savepoint correspondiente:
    - el seguimiento y los recordatorios válidos se mantuvieron,
    - **el recordatorio erróneo no quedó persistido.**

d) sp_Registrar_Actividad_Usuario_Completa

- Registra la actividad completa de un usuario sobre un partido dentro de una única transacción.
- Incluye tres operaciones encadenadas:
  • INSERT en **visualizaciones** (medio y minutos vistos).
  • INSERT en **calificaciones** (puntaje de 1 a 5).
  • UPDATE en **perfiles** (actualización del timestamp de última actividad).
- Valida previamente la existencia del partido, del usuario, el dominio del medio y el rango permitido del puntaje.
- Devuelve los IDs generados (visualización y calificación) mediante parámetros **OUTPUT**.
- La transacción solo se confirma si las tres operaciones se ejecutan correctamente; si alguna falla, se revierte todo el bloque manteniendo la atomicidad.
- En las pruebas, la detección de puntajes inválidos o medios inexistentes produjo **rollback completo**, sin crear visualizaciones ni calificaciones parciales, y sin actualizar el perfil del usuario.

4.5.3 Ejemplos de transacciones anidadas y savepoints

El archivo 02_transacciones_anidadas.sql contiene ejemplos didácticos, entre ellos:

- **SAVE TRANSACTION básico:**
  Se observó que, tras crear varios savepoints y provocar un error en una de las operaciones, ROLLBACK TO revertía únicamente la parte afectada, manteniendo los cambios previos dentro de la misma transacción.

- **Transacciones multinivel y @@TRANCOUNT:**
  La ejecución de varios BEGIN TRANSACTION anidados mostró que @@TRANCOUNT se incrementa por cada llamada, que cada COMMIT solo lo decrementa en 1 y que un ROLLBACK lo restablece a 0, anulando todos los cambios pendientes.

- **Procesamiento batch con savepoints:**
  En un cursor sobre un conjunto de registros, cada iteración creó un savepoint. Los registros con error se revirtieron de manera individual con ROLLBACK TO, mientras que el resto se confirmó con COMMIT al final del lote.

4.5.4 Pruebas de éxito y de error

Las pruebas se organizaron en dos grupos principales, utilizando los scripts 03_pruebas_transacciones.sql y 04_pruebas_rollback.sql.

a) Casos de éxito

Prueba	Descripción	Resultado observado
1	Registrar usuario completo	Usuario y perfil creados en forma atómica
2	Calificar y opinar	Calificación y opinión creadas en la misma Tx
3	Seguir equipo con recordatorios	Seguimiento + 3 recordatorios insertados
4	Transferir favoritos	5 favoritos copiados correctamente
5	Medición de IO y tiempo	Métricas capturadas con STATISTICS IO/TIME

En todas ellas, el valor de @@TRANCOUNT retornó a 0 tras el COMMIT final.

b) Casos de error / rollback

Prueba	Escenario	Comportamiento esperado	Resultado observado
1	Correo duplicado	Rollback completo	Usuario no fue creado
2	Partido no finalizado	Rollback completo	Calificación no fue registrada
3	Calificación duplicada	Rechazo de segunda calificación	Solo una calificación persistió
4	Equipo inválido con savepoint	Rollback parcial	Usuario y equipo válido permanecieron
5	Error crítico (división por cero)	Rollback completo	Ningún dato de la transacción quedó

c) Escenarios particulares verificados

- **Rollback parcial:** escenario “crear usuario + seguir equipo válido + seguir equipo inválido”
    - Tras ROLLBACK TO el segundo savepoint y COMMIT final, las verificaciones mostraron:
        - usuario creado,
        - seguimiento al equipo válido presente,
        - seguimiento al equipo inválido ausente.
- **Rollback completo con error crítico:**
    - Se ejecutó una transacción con inserciones en usuarios, perfiles y seguimientos, seguida de un error deliberado (1/0) con SET XACT_ABORT ON.
    - Las consultas de verificación confirmaron que ninguno de los registros se había persistido.

4.5.5 Observaciones de rendimiento

Las mediciones de tiempo y de lecturas lógicas se realizaron con SET STATISTICS IO ON y SET STATISTICS TIME ON, comparando operaciones con y sin transacciones explícitas.

a) Tiempos promedios medidos

Operación	Sin transacción explícita	Con BEGIN/COMMIT	Diferencia aproximada
INSERT simple	~1 ms	~1 ms	despreciable
3 INSERT relacionados	~3 ms	~3 ms	despreciable
INSERT + validaciones complejas	~2 ms	~2–3 ms	< 1 ms

b) Lecturas lógicas por procedimiento (ejemplos)

- sp_Registrar_Usuario_Completo
    - usuarios: 2 lecturas lógicas
    - perfiles: 2 lecturas lógicas
    - **Total:** 4 lecturas lógicas
- sp_Calificar_y_Opinar
    - partidos: 1 lectura lógica (validación)
    - usuarios: 1 lectura lógica (validación)
    - calificaciones: 3 lecturas (INSERT + índice)
    - opiniones: 3 lecturas (INSERT + índice)
    - **Total:** 8 lecturas lógicas

En todas las pruebas, el impacto temporal de introducir transacciones explícitas se mantuvo bajo, mientras que el control sobre commits y rollbacks se observó claramente en estados intermedios y finales de las tablas.

4.6 Desarrollo del Tema 4: Vistas y Vistas Indexadas

Esta sección presenta los resultados obtenidos al implementar vistas y una vista indexada sobre la tabla dbo.partidos de la base tribuneros_bdi. Los scripts correspondientes se encuentran en script/tema4-vistas.

4.6.1 Marco teórico aplicado

En SQL Server, una **vista** es un objeto que almacena una consulta predefinida y devuelve su resultado como si fuera una tabla virtual. En el proyecto se utilizó este mecanismo para:

- encapsular lógica de selección,
- simplificar el acceso a subconjuntos de columnas,
- desacoplar a la aplicación de cambios estructurales en las tablas base.

Una **vista indexada** es una vista cuyo resultado se materializa físicamente mediante un índice. El motor puede leer directamente el conjunto de datos pre-agregado, lo cual resulta útil en consultas de lectura intensiva. A diferencia de las vistas normales, las operaciones DML sobre las tablas base deben mantener actualizado el índice asociado.

El diseño y las pruebas se basaron en la documentación oficial de Microsoft y en material de referencia técnico (Microsoft Docs, SQLShack y Red-Gate/Simple-Talk) sobre creación y uso de vistas indexadas.

4.6.2 Contexto del modelo: tabla dbo.partidos

La tabla dbo.partidos almacena los encuentros de fútbol registrados por el sistema. Entre sus columnas, las más relevantes para este tema son:

- id (INT, PK): identificador del partido.
- liga_id (INT, FK): liga o torneo.
- equipo_local, equipo_visitante (INT, FK): equipos participantes.
- fecha_utc (DATETIME2(0)): fecha y hora del partido.
- temporada (SMALLINT): temporada o año.
- estado (TINYINT): estado del partido (pendiente, jugado, etc.).
- goles_local, goles_visitante (SMALLINT): resultado.

A partir de esta tabla se definieron vistas para:

- operaciones CRUD básicas sobre partidos,
- cálculos de estadísticas por liga y año.

4.6.3 Vista simple para operaciones CRUD (vw_partidos_basicos)

Se implementó una vista simple que expone solo las columnas necesarias para tareas básicas:

```
CREATE VIEW dbo.vw_partidos_basicos AS
SELECT id, liga_id, fecha_utc, equipo_local, equipo_visitante, estado, temporada
FROM dbo.partidos;
```

Figura 16: Creación dbo.vw_partidos_basicos

Motivación

- Exponer solo columnas necesarias para operaciones CRUD.
- Mantenerla actualizable (sin JOIN ni GROUP BY).

Operaciones probadas

Con la vista creada se realizaron las siguientes pruebas:

```
-- INSERT de pruebas
INSERT INTO dbo.vw_partidos_basicos (id, liga_id, fecha_utc, equipo_local, equipo_visitante, estado, temporada)
VALUES (9000, 1, SYSDATETIME(), 1, 2, 0, 2025);

-- UPDATE (cambiar estado)
UPDATE dbo.vw_partidos_basicos SET estado = 1 WHERE id = 9000;

-- DELETE de registros de prueba
DELETE FROM dbo.vw_partidos_basicos WHERE id BETWEEN 9000 AND 9004;
```

Figura 17: Pruebas con la vista dbo.vw_partidos_basicos

En todas las ejecuciones se verificó que:

- las inserciones, actualizaciones y eliminaciones realizadas a través de la vista se reflejaban en la tabla partidos,
- el rango de IDs 9000–9009 permitió repetir las pruebas sin afectar datos productivos.

4.6.4 Vista indexada para estadísticas (vw_partidos_por_liga_y_anio)

Con el objetivo de optimizar consultas de resumen del tipo “cantidad de partidos por liga y año”, se diseñó una vista con agregación y se materializó con un índice clustered.

Definición de la vista

```
CREATE VIEW dbo.vw_partidos_por_liga_y_anio
WITH SCHEMABINDING
AS
SELECT
    p.liga_id,
    YEAR(p.fecha_utc) AS anio,
    COUNT_BIG(*) AS cantidad_partidos
FROM dbo.partidos AS p
GROUP BY p.liga_id, YEAR(p.fecha_utc);
```

Figura 18: Creacion de dbo.vw_partidos_por_liga_y_anio

Índice clustered único sobre la vista

```
CREATE UNIQUE CLUSTERED INDEX IX_vw_partidos_por_liga_y_anio
    ON dbo.vw_partidos_por_liga_y_anio (liga_id, anio);
```

Figura 19: Índice Clustered Unico sobre la vista

Durante la creación se cumplieron los requisitos de SQL Server para vistas indexadas:

- uso de WITH SCHEMABINDING y referencias a dbo.partidos con nombre de dos partes,
- utilización de COUNT_BIG como agregación principal,
- activación de las opciones SET obligatorias (como ANSI_NULLS ON, QUOTED_IDENTIFIER ON, ARITHABORT ON, entre otras) antes de la creación

---

4.6.5 Pruebas de rendimiento con y sin vista indexada

Las pruebas se realizaron con SET STATISTICS IO ON y SET STATISTICS TIME ON, ejecutando dos bloques de consultas.

Bloque A – consulta directa sobre dbo.partidos

```
SELECT p.liga_id, YEAR(p.fecha_utc) AS anio, COUNT(*) AS cantidad_partidos
FROM dbo.partidos AS p
GROUP BY p.liga_id, YEAR(p.fecha_utc)
ORDER BY p.liga_id, anio;
```

Figura 20:Consulta directa sobre dbo.partidos

Resultados observados en el plan de ejecución:

- operador principal Clustered Index Scan (Clustered) sobre dbo.partidos,
- presencia de operadores Sort y Stream Aggregate para calcular los totales,
- mayor costo estimado y lectura de un volumen elevado de páginas de datos.

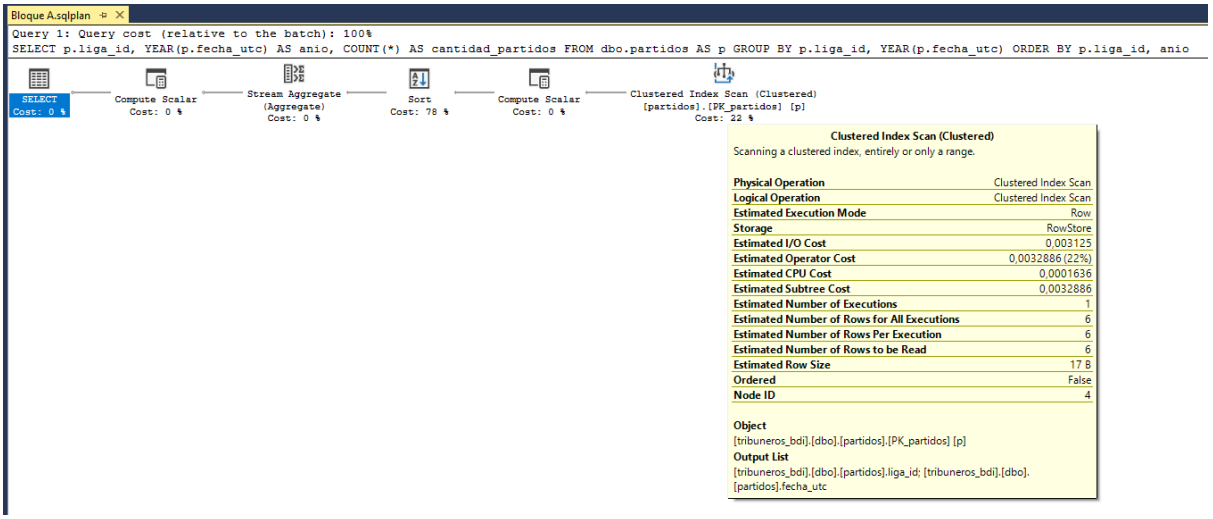


Figura 21: Captura de pantalla sobre el Plan Bloque A

Bloque B – consulta sobre la vista indexada

Tras crear la vista y su índice, se ejecutó la consulta equivalente:

```
SELECT v.liga_id, v.anio, v.cantidad_partidos
FROM dbo.vw_partidos_por_liga_y_anio AS v
ORDER BY v.liga_id, v.anio;
```

Figura 22: Consulta sobre la vista indexada

En este caso, el plan de ejecución mostró:

- un Clustered Index Scan (ViewClustered) sobre IX_vw_partidos_por_liga_y_anio,
- ausencia de operadores de agregación adicionales, ya que los datos se leen materializados,
- menor costo estimado y reducción de lecturas lógicas en comparación con el bloque A.

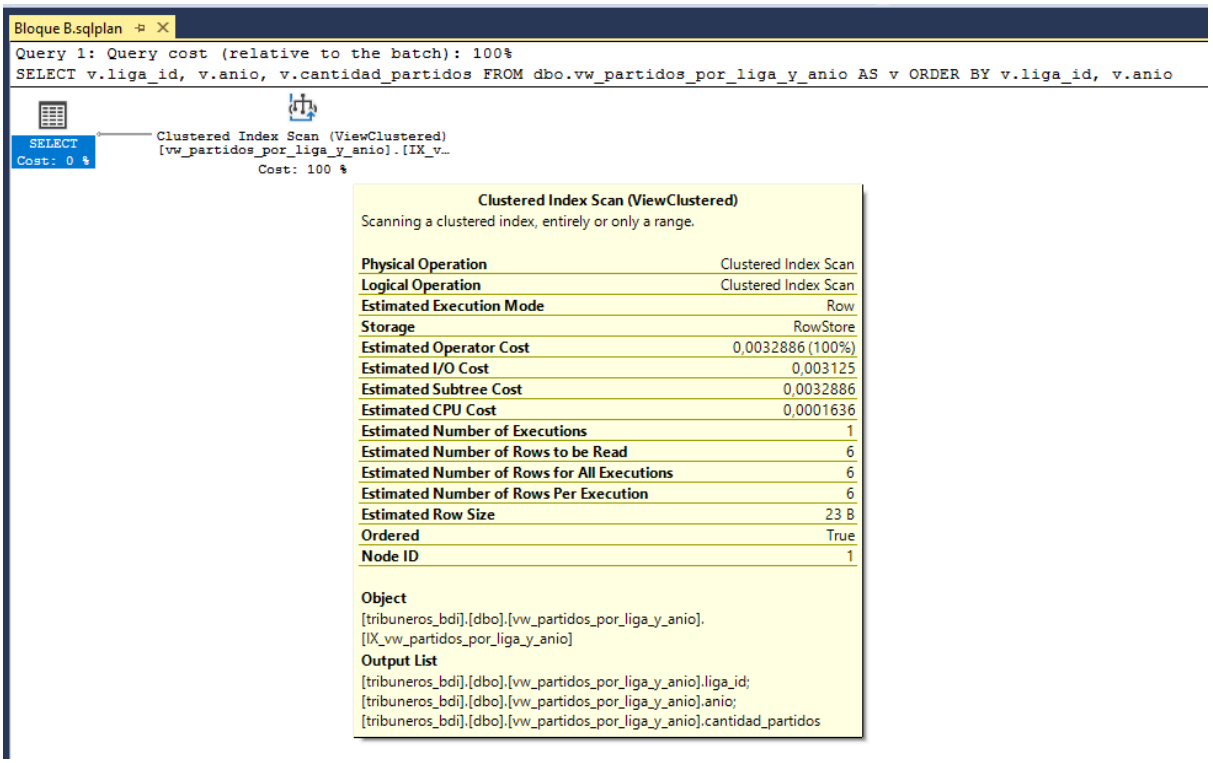


Figura 23: Captura de pantalla sobre el Plan Bloque B

Comparación de planes

Al comparar ambos planes (Showplan Comparison) se registraron las siguientes diferencias:

Característica	Bloque A (sin vista indexada)	Bloque B (con vista indexada)
Fuente de datos	dbo.partidos	vw_partidos_por_liga_y_anio

Operador principal	Clustered Index Scan	Clustered Index Scan (View)
Operadores adicionales	Sort + Stream Aggregate	ninguno
Costo estimado relativo	más alto	más bajo
Lecturas lógicas	tabla base completa	menor (uso de índice materializado)

Además, al volver a ejecutar el bloque A con la vista indexada ya creada, el plan de ejecución utilizó internamente el índice de la vista (ViewClustered), lo que evidenció la capacidad del optimizador para reutilizar automáticamente la vista indexada aún cuando la consulta se formula sobre la tabla base.

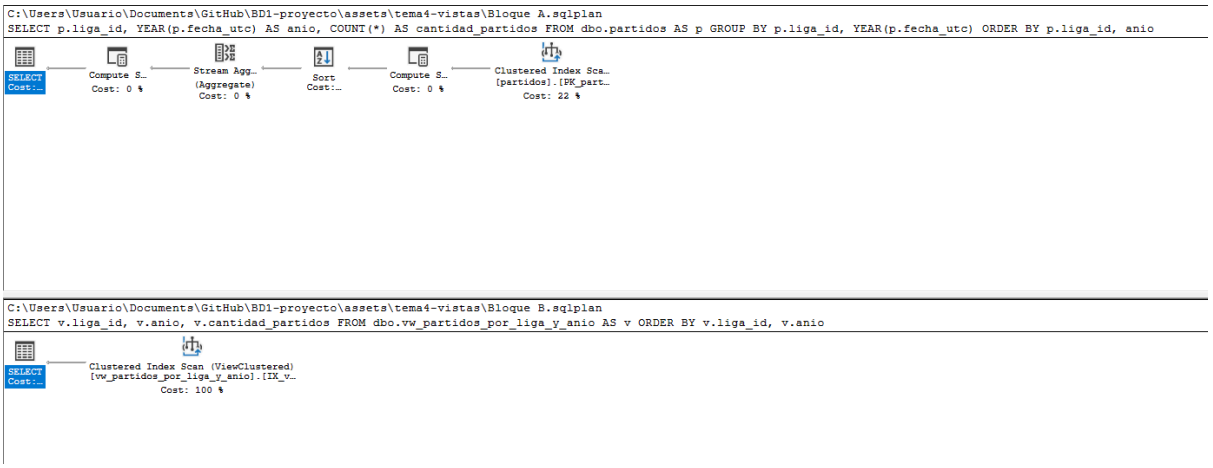


Figura 24: Captura de pantalla sobre comparacion de Plan A vs Plan B

4.6.6 Script de rollback del Tema 4

Para garantizar la reversibilidad de las pruebas, se implementó el script 04_rollback_tema4.sql, cuyo propósito es restaurar el estado inicial del modelo respecto de este tema.

Pasos ejecutados por el script:

1. Eliminación del índice IX_vw_partidos_por_liga_y_anio si existe.
2. Eliminación de las vistas vw_partidos_por_liga_y_anio y vw_partidos_basicos.
3. Borrado de los registros de prueba con IDs entre 9000 y 9009 en dbo.partidos.

Las verificaciones posteriores mostraron que, tras ejecutar el rollback, no permanecían filas dentro del rango de IDs de prueba ni objetos creados por el Tema 4, dejando la base lista para nuevas ejecuciones de los scripts.

Capítulo V — Conclusiones

El desarrollo completo de los cuatro temas permitió no solo aplicar los contenidos teóricos de la materia, sino también **validarlos mediante pruebas reales**, cuyos resultados se observaron directamente en las capturas de pantalla y en las salidas de SQL Server. Las conclusiones que se presentan a continuación se basan exclusivamente en estos resultados prácticos.

5.1. Conclusiones sobre Procedimientos Almacenados (SP) y Funciones (UDF)

A partir de las pruebas realizadas sobre la base *tribuneros_bdi*, se pudo comprobar de manera práctica el rol que cumplen los procedimientos almacenados y las funciones dentro de la lógica de negocio y la integridad de los datos.

En primer lugar, los **procedimientos almacenados demostraron ser esenciales para asegurar el control sobre cómo se insertan, actualizan o eliminan los datos**. En las pruebas del módulo de opiniones, el SP aplicó correctamente las validaciones de negocio: evitó la duplicidad (un usuario no puede opinar dos veces sobre el mismo partido), verificó la existencia de las claves foráneas, y bloqueó operaciones incompletas o inconsistentes. Todo esto contrastó claramente con la inserción directa, donde fue posible generar errores, cargar datos inválidos o romper la integridad referencial.

Asimismo, el uso de **transacciones** confirmó en la práctica el principio de **atomicidad**: ante cualquier error interno, el SP revirtió la operación completa, dejando la tabla en un estado consistente. Esto evidenció por qué las sentencias sueltas no garantizan el mismo nivel de seguridad ni confiabilidad que un procedimiento bien estructurado.

Por otra parte, las **funciones definidas por el usuario (UDF)** permitieron encapsular y reutilizar lógica de obtención de datos. Funciones como *fn_ObtenerNombreUsuario* o la función de promedio del partido facilitaron la escritura de consultas más claras, reduciendo la repetición de JOINs y garantizando uniformidad en el formato de los resultados. Además, se comprobó que las UDF solo retornan valores y no modifican datos, cumpliendo con las restricciones del motor y evitando efectos secundarios.

Al comparar **insertar directo vs. insertar vía SP**, se observó con claridad la diferencia entre “poder insertar” y “insertar correctamente”. Mientras que el INSERT directo permitió datos duplicados o inválidos, el SP solo aceptó información válida según las reglas del modelo. Esto demuestra que, en un proyecto real, las operaciones críticas deben canalizarse siempre mediante SP para preservar la coherencia del sistema.

Finalmente, los **mensajes PRINT** utilizados durante las pruebas ayudaron a visualizar el flujo interno de ejecución, identificar el punto exacto donde ocurrían fallos y comprender mejor la secuencia lógica del procedimiento.

En síntesis, se cumplió el objetivo del tema: se comprobó que los SP permiten garantizar integridad, consistencia y control de negocio, mientras que las funciones mejoran la claridad y reutilización de lógica dentro de las consultas. Ambos componentes aportan robustez y organización al diseño de la base de datos.

5.2 Conclusiones sobre Optimización de Consultas a Través de Índices

Durante el desarrollo del Tema 2 se realizaron pruebas de rendimiento sobre la tabla *partidos* con más de un millón de registros, lo que permitió observar de manera concreta el impacto real del uso de índices en SQL Server. Las consultas ejecutadas sin ningún índice obligaron al motor a realizar *table scans* completos, generando tiempos de ejecución elevados y un costo estimado muy alto en los planes de ejecución. Esto confirmó que en tablas de gran volumen, incluso filtros simples como rangos de fechas requieren recorrer millones de filas, afectando gravemente el rendimiento.

La creación de un índice no agrupado sobre *fecha_utc* produjo una mejora inmediata. Las mismas consultas pasaron de *table scan* a *index seek*, reduciendo el costo del plan hasta casi 3 veces del escenario sin índice. Esta diferencia demostró que un solo índice bien elegido es capaz de transformar el comportamiento del optimizador y acelerar significativamente la lectura de datos filtrados por una columna con alta selectividad.

Posteriormente, al reemplazarlo por un **índice compuesto** en *(fecha_utc, estado)* con columnas incluidas (*liga_id, equipo_local, equipo_visitante, goles_local, goles_visitante, estadio*), las consultas quedaron totalmente cubiertas por el índice. El plan de ejecución mostró una reducción adicional del costo, alcanzando la mejor performance del conjunto (84,7% más eficiente respecto al escenario sin índice). Esto confirmó que los índices compuestos y las columnas incluídas permiten evitar accesos adicionales a la tabla (lookups), optimizando tanto las consultas filtradas como las agregaciones.

Las pruebas también permitieron observar que los índices tienen un costo: su creación sobre un millón de filas incurre en tiempo de procesamiento y el mantenimiento del índice implica más trabajo en operaciones DML. Sin embargo, en escenarios de consulta intensiva como los de este proyecto, los beneficios superan ampliamente los costos.

En síntesis, las pruebas realizadas demostraron que:

- Los índices son fundamentales para mejorar el rendimiento en tablas de gran tamaño.

- Un índice simple puede optimizar mucho una consulta, pero un índice compuesto correctamente diseñado puede llevar el rendimiento a su nivel óptimo.

- Analizar los planes de ejecución es clave para tomar decisiones de optimización basadas en evidencia real.

Los objetivos del tema se cumplieron plenamente: se comprobó el impacto del uso de índices, se compararon escenarios con y sin optimización, y se validaron los beneficios de elegir correctamente las columnas que conforman un índice según los patrones reales de uso.

5.3 Conclusiones sobre Transacciones y Transacciones Anidadas

A lo largo del desarrollo del Tema 3 se implementaron y probaron procedimientos almacenados que utilizan transacciones explícitas, transacciones anidadas y puntos de guardado. Las pruebas permitieron analizar de forma práctica cómo SQL Server garantiza la atomicidad, la consistencia y el control de errores en operaciones complejas que involucran múltiples inserciones y validaciones.

En primer lugar, el uso de transacciones explícitas en procedimientos como *sp_Registrar_Usuario_Completo* y *sp_Calificar_y_Opinar* demostró que es posible asegurar que un conjunto de operaciones DML se ejecute como una unidad indivisible: o bien todas se completan, o ninguna afecta la base de datos. Las pruebas exitosas mostraron que, ante operaciones correctas, los COMMIT consolidan los datos, mientras que ante errores —como claves duplicadas, referencias inválidas o violaciones de reglas de negocio— el bloque TRY/CATCH ejecutó el ROLLBACK apropiado, evitando estados parciales o inconsistentes.

Por otra parte, las pruebas con *sp_Seguir_Equipo_Con_Recordatorios* permitieron observar el funcionamiento de transacciones anidadas y el uso de SAVE TRANSACTION para lograr rollbacks parciales. En este procedimiento, algunos recordatorios inválidos se revirtieron selectivamente utilizando ROLLBACK TO SavePoint, mientras que las demás operaciones exitosas (incluyendo el seguimiento del equipo y otros recordatorios válidos) se conservaron. Este comportamiento confirmó que SQL Server no soporta verdaderas transacciones independientes en niveles anidados, pero sí permite control granular mediante savepoints, lo cual es útil para escenarios de procesamiento por lotes donde algunos elementos pueden fallar sin invalidar el resto.

Las pruebas de rollback completo evidenciaron también la importancia de patrones robustos como SET XACT_ABORT ON junto con TRY/CATCH. En casos de errores críticos —como división por cero— SQL Server revirtió toda la transacción correctamente, impidiendo que

quedaran datos huérfanos o parciales en distintas tablas relacionadas. Esto demuestra que el manejo de errores y el diseño de procedimientos transaccionales bien estructurados son fundamentales para mantener integridad y coherencia.

Finalmente, el análisis de rendimiento mostró que el costo adicional de abrir y cerrar transacciones es mínimo para cargas OLTP típicas. El beneficio obtenido en términos de seguridad, integridad y control de flujo supera ampliamente ese pequeño overhead.

En síntesis, las pruebas del Tema 3 permitieron verificar que las transacciones explícitas, los savepoints y el manejo adecuado de errores son herramientas esenciales para implementar lógica de negocio compleja de forma segura. Se cumplió el objetivo del tema: demostrar mediante casos reales cómo SQL Server gestiona commits, rollbacks y puntos de guardado, y cómo estos mecanismos garantizan que la base de datos mantenga un estado siempre consistente incluso ante fallos parciales o totales.

5.4 Conclusiones sobre Vistas y Vistas Indexadas

El desarrollo del Tema 4 permitió verificar de manera práctica cómo las vistas y, especialmente, las vistas indexadas pueden mejorar la organización del modelo y optimizar consultas de uso frecuente en la base tribuneros_bdi.

En primer lugar, la vista simple vw_partidos_basicos demostró ser útil para encapsular y exponer únicamente un subconjunto controlado de columnas. Las pruebas de inserción, actualización y eliminación realizadas a través de la vista confirmaron que una vista correctamente diseñada puede actuar como “capa lógica” para operaciones CRUD, manteniendo a la aplicación aislada de cambios en la tabla base. No se presentaron restricciones de actualización, ya que la vista no tenía JOINs, funciones ni agregados, y por ello permaneció totalmente actualizable, cumpliendo con el objetivo de simplificar el acceso a la tabla partidos.

Por otro lado, la creación de la vista indexada vw_partidos_por_liga_y_anio permitió evaluar de forma concreta el impacto de materializar datos agregados. Las consultas que originalmente requerían recorrer toda la tabla partidos, agrupar por liga y año y luego ordenar, pasaron a ejecutarse directamente contra un índice clúster único que ya contiene los resultados precalculados. Las pruebas comparativas con SET STATISTICS IO/TIME y los planes de ejecución mostraron una reducción significativa del costo: el plan se simplificó eliminando operadores como Sort y Stream Aggregate, y el motor utilizó directamente el índice materializado, disminuyendo la cantidad de lecturas lógicas.

Además, se verificó que SQL Server es capaz de reutilizar automáticamente una vista indexada incluso cuando la consulta se formula sobre la tabla base, lo que confirma el beneficio práctico de estas estructuras: la aplicación no necesita llamarlas explícitamente para obtener mejoras de rendimiento.

Si bien la vista indexada introdujo un mayor costo en las operaciones DML sobre partidos (debido a que el índice debe mantenerse sincronizado), este impacto es aceptable en

escenarios dominados por consultas estadísticas o de lectura intensiva, como el módulo de análisis de temporadas o reportes de ligas dentro del proyecto.

En síntesis, las pruebas del Tema 4 demostraron que las vistas simplifican la interacción con los datos y que las vistas indexadas pueden mejorar de manera notable el rendimiento de agregaciones frecuentes. Se validaron también los requisitos técnicos exigidos por el motor (SCHEMABINDING, COUNT_BIG, SET options), cumpliendo con lo solicitado por la cátedra. El objetivo del tema se cumplió completamente: se compararon escenarios con y sin optimización, se observaron mejoras reales en los planes de ejecución y se justificó cuándo conviene aplicar este tipo de vistas en sistemas orientados a consultas complejas.

Capítulo VI — Bibliografía

6.1 Referencias citadas

- Elmasri, R., & Navathe, S. (2016). *Fundamentals of Database Systems* (7th ed.). Pearson.
- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database System Concepts* (7th ed.). McGraw-Hill.
- Fowler, M. (2003). *Patterns of Enterprise Application Architecture*. Addison-Wesley.
- ISO/IEC. (2016). *ISO/IEC 9075 — Information technology — Database languages — SQL*. International Organization for Standardization.
- Microsoft. (2024). *Transact-SQL Reference (Database Engine)*. Recuperado de https://learn.microsoft.com/sql/t-sql/language-reference
- SQLShack — Funciones frente a Procedimientos Almacenados en SQL Server: https://www.sqlshack.com/es/funciones-frente-a-los-procedimientos-almacenados-en-sql-server/
- W3Schools — Procedimientos almacenados: https://www.w3schools.com/sql/sql_stored_procedures.asp
- Red Gate - Manejo de transacciones en SQL Server: https://www.red-gate.com/simple-talk/databases/sql-server/t-sql-programming-sql-server/sql-server-transactions/

6.2 Recursos complementarios

- Diagrama entidad-relación: docs/der-tribuneros.png.
- Diccionario de datos: docs/diccionario_datos.md.
- Scripts del proyecto: script/.
- Repositorio en GitHub: BD1-proyecto.