Introducción

MPBP Eventos & Catering es una empresa especializada en la organización integral de eventos sociales y corporativos. Bajo el lema **"Mesa Perfecta, Banquetes Premium"**, ofrece soluciones completas que incluyen servicios de banquete, música, ambientación y personal calificado, adaptados a las necesidades específicas de cada cliente.

La compañía busca garantizar experiencias memorables, combinando calidad gastronómica, planificación logística y atención personalizada. Para lograrlo, requiere un sistema centralizado que permita gestionar clientes, eventos, servicios contratados y asignación de empleados, optimizando los procesos internos y reduciendo errores operativos.

El presente proyecto propone el diseño e implementación de una base de datos relacional que soporte dichas operaciones, mantenga la integridad de la información y facilite consultas para la toma de decisiones comerciales y logísticas.

Enlace del proyecto:
El siguiente proyecto está en el siguiente repositorio
https://github.com/fer0o/Comision-81830-SQL
precisamente en la carpeta de proyecto
https://github.com/fer0o/Comision-81830-SQL/tree/master/Proyecto
La información del Drive con la información escrita está en la siguiente dirección
https://docs.google.com/document/d/1csgdAPqTOh1XFJWOHlR-cDjo9nDksQ0WFCDLSCuyk_k/edit?usp=sharing

Objetivo del proyecto

- Diseñar un esquema de base de datos normalizado que registre información de clientes, eventos y servicios.

- Modelar las relaciones entre eventos, servicios y empleados para asegurar una correcta asignación de recursos.

- Permitir el registro y consulta de variantes de servicios según el tipo de evento.

- Facilitar la generación de reportes para análisis de ventas, planificación de personal y control de inventario de servicios.

- Mantener la integridad y consistencia de los datos mediante claves primarias, foráneas y restricciones adecuadas.

Situación Problemática

MPBP Eventos & Catering busca optimizar la gestión de clientes, eventos y servicios mediante un sistema centralizado que permita almacenar y relacionar la información de manera eficiente.

Si bien actualmente las operaciones pueden gestionarse con herramientas básicas como hojas de cálculo, esta metodología no resulta ideal para una empresa en crecimiento, ya que presenta limitaciones en el manejo de grandes volúmenes de datos, la integridad de la información y la automatización de procesos.

Para evitar depender de hojas de cálculo como base de datos y con el fin de mejorar la organización interna, se propone el desarrollo de una **base de datos relacional** que permita:

- Mantener registros confiables y actualizados.
- Relacionar clientes, eventos, servicios y empleados de forma estructurada.
- Facilitar consultas rápidas y generación de reportes estratégicos.
- Garantizar la integridad y consistencia de la información.

Con esta solución, la empresa podrá sostener un crecimiento ordenado, optimizar recursos y ofrecer un mejor servicio a sus clientes.

Modelo de Negocio

1) Propuesta de valor

MPBP ofrece organización integral de eventos sociales y corporativos: **banquete, música, ambientación y personal** bajo el lema “Mesa Perfecta, Banquetes Premium”. Se adapta al **tipo de evento** (boda, graduación, cumpleaños, corporativo) con paquetes y variantes específicas.

2) Actores principales

- **Clientes**: personas o empresas que contratan.
- **Empleados**: meseros, chefs, DJs, decoradores, etc.
- **Catálogo de servicios**: comida, música, ambientación y personal (con detalles/variantes por tipo de evento).
- **Eventos**: unidad central de operación (fecha, lugar, tipo, cliente).

3) Flujo operativo (de punta a punta)

1. **Alta de cliente** y registro de datos de contacto.
2. **Creación del evento** (tipo, fecha, lugar, cliente).
3. **Selección de servicios** desde el catálogo:
    - Variantes por **tipo de evento** (p. ej., menú de boda vs. graduación).
    - Definición de **cantidad** y **precio aplicado** (permite ajustes/promos).
4. **Asignación de empleados** al evento (roles y horas estimadas).
5. **Cálculo económico básico**:
    - Subtotales por servicio (cantidad × precio).
    - Total del evento (suma de subtotales).
6. **Ejecución y cierre**:
    - Ajustes finales (horas reales, cambios de servicios si aplica).
    - Reportes operativos y de ventas.

4) Reglas de negocio clave

- Un **cliente** puede tener **muchos eventos**.
- Un **evento** puede contratar **muchos servicios** y un **servicio** puede participar en **muchos eventos** (relación N:M).
- Un **evento** puede tener **muchos empleados** y un **empleado** puede participar en **muchos eventos** (N:M).
- Los servicios se clasifican en **comida, música, personal, ambientación**, con **tablas de detalle** para variantes por tipo de evento.
- El **precio aplicado** al evento puede diferir del precio de referencia del catálogo (promos, negociación).
- Subtotal por servicio = **cantidad × precio_unitario** (se guarda para auditoría de la cotización).

5) Información y reportes esperados

- **Agenda** de eventos por fecha y tipo.
- **Ingresos** por mes/evento/servicio.

Listado de Tablas

1. clientes

Descripción: Almacena la información de los clientes que contratan eventos.

Campos:

- id_cliente **INT** – PK, autoincremental.
- nombre **VARCHAR(120)** – Nombre del cliente.
- apellido **VARCHAR(120)** – Apellido del cliente.
- email **VARCHAR(150)** – Correo electrónico, único.
- telefono **VARCHAR(20)** – Teléfono de contacto.
- direccion **VARCHAR(200)** – Dirección del cliente.

2. servicios

Descripción: Catálogo de servicios disponibles para los eventos.

Campos:

- id_servicio **INT** – PK, autoincremental.
- tipo_servicio **ENUM(…)** – Clasificación del servicio (comida, música, ambientación, etc.).
- nombre_servicio **VARCHAR(100)** – Nombre del servicio.
- activo **BOOLEAN** – Indica si el servicio está activo.

3. empleados

Descripción: Información del personal disponible para trabajar en los eventos.

Campos:

- id_empleado **INT** – PK, autoincremental.
- nombre **VARCHAR(120)** – Nombre del empleado.
- apellido **VARCHAR(120)** – Apellido del empleado.
- email **VARCHAR(150)** – Correo electrónico, único.
- telefono **VARCHAR(20)** – Teléfono de contacto.
- puesto **VARCHAR(50)** – Cargo o rol del empleado.

4. eventos

Descripción: Información general de los eventos programados.

Campos:

- id_evento **INT** – PK, autoincremental.
- nombre_evento **VARCHAR(120)** – Nombre del evento.
- fecha_evento **DATETIME** – Fecha y hora del evento.
- tipo_evento **ENUM(…)** – Tipo (boda, cumpleaños, graduación, corporativo, etc.).
- lugar **VARCHAR(150)** – Lugar del evento.
- id_cliente **INT** – FK hacia clientes.

5. eventos_servicios

Descripción: Relación de servicios contratados para un evento.

Campos:

- id_evento_servicio **INT** – PK, autoincremental.
- id_evento **INT** – FK hacia eventos.
- id_servicio **INT** – FK hacia servicios.
- cantidad **INT** – Cantidad contratada.
- precio_unitario **DECIMAL(10,2)** – Precio por unidad.
- subtotal **DECIMAL(10,2)** – Total calculado.

6. eventos_empleados

Descripción: Asignación de empleados a eventos.

Campos:

- id_evento_empleado **INT** – PK, autoincremental.
- id_evento **INT** – FK hacia eventos.
- id_empleado **INT** – FK hacia empleados.
- rol_en_evento **VARCHAR(50)** – Función específica.
- horas_estimadas **DECIMAL(5,2)** – Horas asignadas.
- notas **VARCHAR(200)** – Observaciones.

7. proveedores

Descripción: Datos de los proveedores que brindan servicios a la empresa.

Campos:

- id_proveedor **INT** – PK, autoincremental.
- nombre_proveedor **VARCHAR(150)** – Nombre del proveedor.
- contacto **VARCHAR(120)** – Persona de contacto.
- telefono **VARCHAR(20)** – Teléfono de contacto.
- email **VARCHAR(150)** – Correo electrónico.
- direccion **VARCHAR(200)** – Dirección.
- tipo_servicio **ENUM(…)** – Categoría de servicio ofrecido.
- activo **BOOLEAN** – Indica si está activo.

8. contratos

Descripción: Contratos firmados con proveedores.

Campos:

- id_contrato **INT** – PK, autoincremental.
- id_proveedor **INT** – FK hacia proveedores.
- fecha_inicio **DATE** – Inicio del contrato.
- fecha_fin **DATE** – Fin del contrato.
- monto **DECIMAL(10,2)** – Valor acordado.
- descripcion **VARCHAR(300)** – Detalles del contrato.

- activo **BOOLEAN** – Estado del contrato.

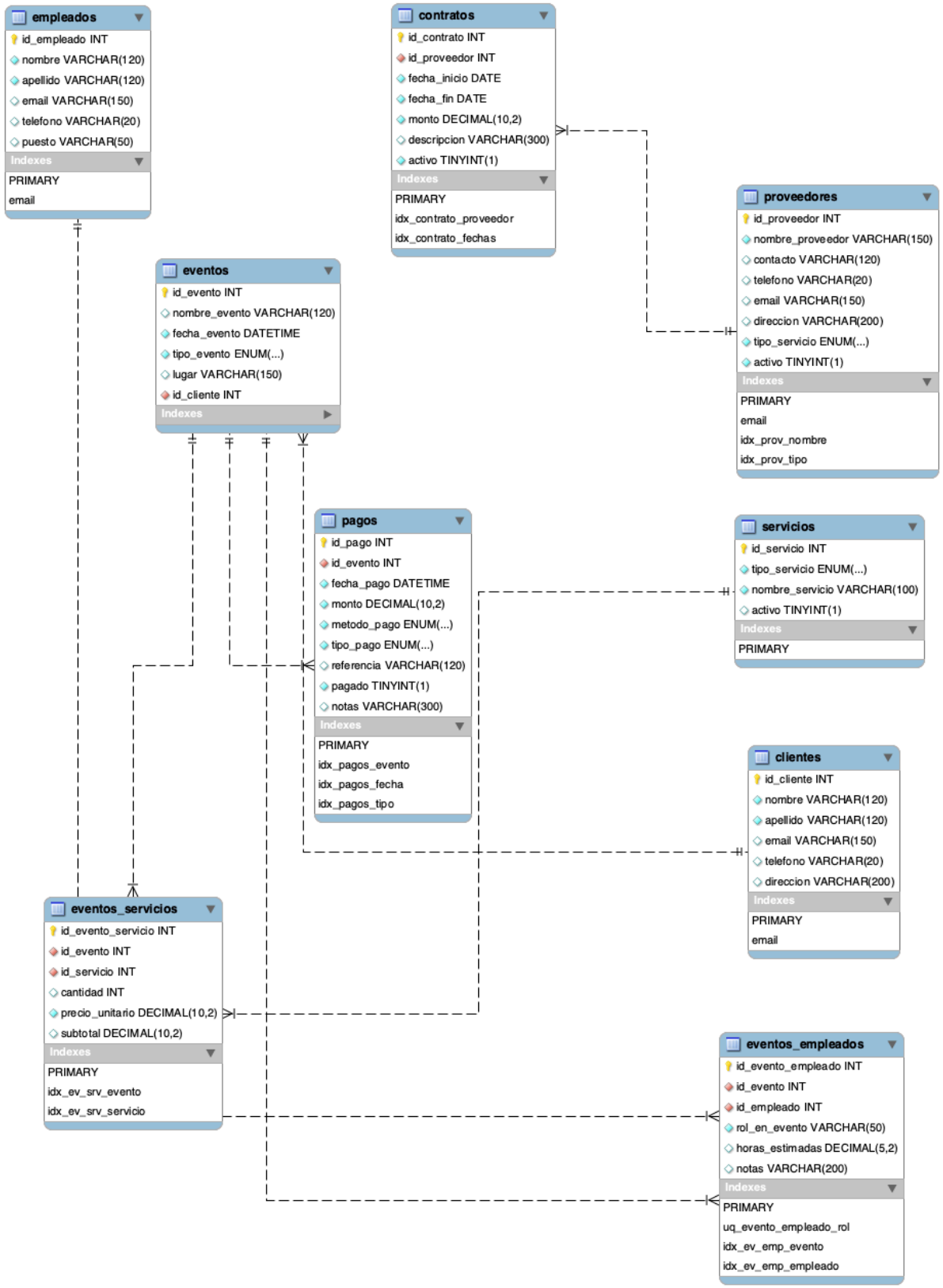
9. pagos

Descripción: Pagos realizados para los eventos.

Campos:

- id_pago **INT** – PK, autoincremental.
- id_evento **INT** – FK hacia eventos.
- fecha_pago **DATETIME** – Fecha del pago.
- monto **DECIMAL(10,2)** – Cantidad pagada.
- metodo_pago **ENUM('efectivo','transferencia','tarjeta','cheque')** – Medio de pago.
- tipo_pago **ENUM('anticipo','parcial','liquidacion')** – Etapa del pago.
- referencia **VARCHAR(120)** – Código o nota de referencia.
- pagado **BOOLEAN** – Si está pagado.
- notas **VARCHAR(300)** – Comentarios adicionales.

Modelo Entidad relación.



Entrega #2
Descripción de las Vistas

Vista #1**vista_eventos_por_cliente**

Esta vista muestra de forma clara y ordenada todos los eventos contratados por cada cliente. Facilita la consulta de información clave de un evento: quién es el cliente, el tipo de evento, la fecha, el lugar, la cantidad de personas y el nombre del evento. Además, incluye el costo total de los servicios asociados al evento, calculado a partir de la tabla de relación eventos_servicios.

Tablas que la componen:

- eventos → información del evento (id, fecha, lugar, tipo, nombre, cantidad de personas)
- clientes → información del cliente (nombre y apellido).
- eventos_servicios → detalle de servicios contratados y sus subtotales.

Campos resultantes:

- id_evento → identificador único del evento.
- cliente → nombre completo del cliente.
- tipo_evento → boda, graduación, cumpleaños, corporativo, etc.
- fecha_evento → fecha programada del evento.
- lugar → lugar donde se llevará a cabo.
- cantidad_personas → asistentes previstos.
- nombre_evento → título o descripción del evento.
- costo_total → suma de los subtotales de los servicios contratados.

Ejemplo de uso:
SELECT * FROM vista_eventos_por_cliente

WHERE cliente LIKE 'Ana%';
Esto mostrará todos los eventos contratados por clientes cuyo nombre comience con “Ana”.

Vista #2 **vista_servicios_mas_contratados**

Descripción / Objetivo:

Esta vista permite identificar los servicios más solicitados por los clientes en los diferentes eventos. Muestra cuántas veces se ha contratado cada servicio y la cantidad total de unidades. Es útil para el área operativa y de ventas, ya que permite detectar cuáles son los servicios más populares y planear inventarios o promociones.

Tablas que la componen:

- servicios → catálogo de servicios (comida, música, personal, ambientación)
- eventos_servicios → detalle de los servicios contratados en los eventos.

Campos resultantes:

- tipo_servicio → categoría del servicio (comida, música, personal, ambientación).
- id_servicio → identificador único del servicio.
- nombre_servicio → nombre del servicio.
- veces_contratado → número de eventos distintos en los que se ha contratado.
- cantidad_total → suma de las unidades contratadas (ejemplo: 10 meseros, 2 DJs).

Ejemplos de uso:

```
SELECT * FROM vista_servicios_mas_contratados

WHERE tipo_servicio = 'comida'

LIMIT 5;
```

esto debería mostrar los 5 servicios de comida más contratados.

Vista #3: vista_total_gastado_por_cliente

Descripción / Objetivo:

Esta vista calcula cuánto ha gastado cada cliente en total en la empresa, sumando los servicios contratados en todos sus eventos. También incluye el número de eventos realizados por cada cliente. Es útil para identificar a los mejores clientes y analizar patrones de consumo.

Tablas que la componen:

- clientes → datos de los clientes (id, nombre, apellido).
- eventos → eventos contratados por cada cliente.
- eventos_servicios → servicios y costos asociados a cada evento.

Campos resultantes:

- id_cliente → identificador único del cliente.
- cliente → nombre completo del cliente.
- total_eventos → número de eventos organizados por ese cliente.
- total_gastado → suma total de los costos de los servicios contratados en todos los eventos del cliente.

Ejemplo de uso:

SELECT * FROM vista_total_gastado_por_cliente

WHERE total_gastado > 50000;

Esto debería mostrar a los clientes que han gastado más de $50,000 en la empresa.

Vista #4: vista_estado_pago_evento

Descripción / Objetivo:

Esta vista muestra el estado financiero de cada evento, indicando si está completamente pagado o si aún tiene un saldo pendiente. Permite a la empresa hacer un seguimiento claro de los eventos que ya fueron liquidados y los que todavía requieren cobros.

Tablas que la componen:

- eventos → información general del evento.
- clientes → datos del cliente que contrató el evento.
- eventos_servicios → cálculo del costo total de los servicios contratados.
- pagos → registros de los pagos realizados.

Campos resultantes:

- id_evento → identificador único del evento.
- cliente → nombre completo del cliente.
- nombre_evento → nombre o título del evento.
- total_servicios → suma de los servicios contratados.
- total_pagado → suma de los pagos realizados (solo los confirmados con pagado = TRUE).
- estado_pago → indica si el evento está **"Pagado"** (total_pagado ≥ total_servicios) o **"Pendiente"**.

Ejemplo de uso:
SELECT * FROM vista_estado_pago_evento

WHERE estado_pago = 'Pendiente';

Muestra todos los eventos que aún tienen pagos pendientes.

Vista #5: vista_carga_empleado

Descripción / Objetivo:

Esta vista permite consultar la carga de trabajo de cada empleado, mostrando cuántos eventos tiene asignados y el total de horas estimadas. Es útil para la planificación de personal y para evitar sobrecarga de trabajo en determinados roles (meseros, DJs, coordinadores, etc.).

Tablas que la componen:

- empleados → catálogo de empleados con su información básica y puesto.
- eventos_empleados → asignación de empleados a eventos, con horas estimadas de trabajo.

Campos resultantes:

- id_empleado → identificador único del empleado.
- nombre → nombre del empleado.
- apellido → apellido del empleado.
- puesto → cargo o rol (chef, mesero, DJ, coordinador, etc.).
- eventos_asignados → número de eventos distintos en los que participa el empleado.
- horas_asignadas → suma total de las horas estimadas de trabajo asignadas al empleado.

Ejemplo de uso:

SELECT * FROM vista_carga_empleado

WHERE puesto = 'Mesero'

ORDER BY horas_asignadas DESC;

Muestra la carga de trabajo de todos los meseros, ordenados por las horas asignadas.

Función #1: fn_total_pagado_evento(p_evento INT)

Descripción / Objetivo:

Esta función devuelve el monto total pagado por un evento específico. Se utiliza para saber cuánto dinero ha ingresado ya en relación con un evento contratado, considerando únicamente los pagos confirmados (pagado = TRUE).

Tablas que utiliza:

- pagos → consulta todos los registros de pagos relacionados con el evento.

Parámetros:

- p_evento → identificador del evento (id_evento) sobre el cual se desea calcular el total pagado.

Retorno:

- DECIMAL(10,2) → cantidad total pagada por el evento.

Ejemplo de uso:

-- consulta para funcion de saber el total pagado por evento

SELECT fn_total_pagado_evento(1) AS total_pagado_evento_1;

-- consulta del nombre del evento y el total pagado por cada uno de los eventos registrados

SELECT

  e.id_evento,

  e.nombre_evento,

  fn_total_pagado_evento(e.id_evento) AS total_pagado

FROM eventos e;

Función #2: fn_total_por_pagar_evento(p_evento INT)

Descripción / Objetivo:

Esta función calcula cuánto dinero queda pendiente de pago en un evento específico. Es decir, la diferencia entre el total de servicios contratados y los pagos confirmados registrados.

Tablas que utiliza:

- pagos → para obtener el total de pagos confirmados (pagado = TRUE).
- eventos_servicios → para obtener el costo total de los servicios contratados.

Parámetros

- p_evento → identificador del evento (id_evento) sobre el cual se quiere calcular el saldo pendiente.

Retorno

- DECIMAL(10,2) → saldo por pagar del evento (puede ser 0 si ya está liquidado).

Ejemplo de uso:

```
-- Consultar cuánto falta por pagar en el evento con id = 7

SELECT fn_total_por_pagar_evento(7) AS saldo_pendiente;

-- Ver todos los eventos con su saldo pendiente

SELECT

  e.id_evento,

  e.nombre_evento,

  fn_total_por_pagar_evento(e.id_evento) AS saldo_pendiente

FROM eventos e

ORDER BY saldo_pendiente DESC;
```

Función #3: fn_porcentaje_pagado_evento(p_evento INT)

Descripción / Objetivo:

Esta función devuelve el porcentaje de pago realizado en un evento. Calcula la relación entre el total de pagos confirmados y el total de servicios contratados, expresada en porcentaje. Permite medir el avance del cobro de cada evento.

Tablas que utiliza:

- pagos → obtiene el monto total pagado y confirmado (pagado = TRUE).
- eventos_servicios → obtiene el costo total de los servicios contratados.

Parámetros:

- p_evento → identificador del evento (id_evento) sobre el cual se quiere calcular el porcentaje pagado.

Retorno:

- DECIMAL(5,2) → porcentaje de pago realizado (ejemplo: 75.00 significa 75%).

Ejemplo de uso:

```
-- Porcentaje pagado del evento con id = 3

SELECT fn_porcentaje_pagado_evento(3) AS porcentaje_pagado;



-- Ver todos los eventos con su porcentaje pagado

SELECT

  e.id_evento,

  e.nombre_evento,

  fn_porcentaje_pagado_evento(e.id_evento) AS porcentaje_pagado

FROM eventos e

ORDER BY porcentaje_pagado DESC;
```

Stored Procedure #1: sp_registrar_pago

Objetivo

Registrar un pago para un evento con validaciones de negocio (monto, fechas según tipo de pago, referencias duplicadas y sobrepago). Devuelve el id del pago insertado y el saldo pendiente actualizado del evento.

Parámetros

- IN p_evento INT → ID del evento al que corresponde el pago.
- IN p_fecha DATETIME → Fecha/hora del pago.
- IN p_monto DECIMAL(10,2) → Importe del pago (debe ser > 0).
- IN p_metodo ENUM('efectivo','transferencia','tarjeta','cheque') → Método de pago.
- IN p_tipo ENUM('anticipo','parcial','liquidacion') → Tipo/etapa del pago.
- IN p_referencia VARCHAR(120) → Referencia externa/nota (opcional, si viene se valida unicidad).
- IN p_notas VARCHAR(300) → Observaciones (opcional).
- OUT p_id_pago INT → ID del pago creado.
- OUT p_saldo_nuevo DECIMAL(12,2) → Saldo pendiente del evento después del pago.

Tablas que impacta / consulta

- Lee eventos (verifica existencia y fecha del evento).
- Lee pagos (referencia duplicada y suma de pagos).
- Lee eventos_servicios (total contratado).
- Inserta en pagos.

Validaciones implementadas

1. Monto > 0: bloquea montos nulos/negativos.
2. Evento válido: el id_evento debe existir.
3. Reglas temporales por tipo de pago:
    - anticipo → debe ser antes de la fecha del evento.
    - liquidacion → no puede ser antes del evento (igual o después).
4. Referencia única (si se proporciona): evita referencias duplicadas en la tabla pagos.
5. Sin sobrepago: compara contra fn_total_por_pagar_evento(p_evento) y bloquea si p_monto excede el saldo pendiente.

Transaccionalidad

- Usa START TRANSACTION … COMMIT para garantizar consistencia: o se inserta el pago completo y se devuelve el saldo actualizado, o no se inserta nada si alguna validación falla.

Códigos de error (SIGNAL 45000) y mensajes típicos

- 'El monto debe ser > 0'
- 'El evento no existe'
- 'El anticipo debe ser antes del evento'
- 'La liquidación no puede ser antes del evento'
- 'La referencia ya existe'
- 'El monto excede el saldo pendiente'

```
-- Variables de salida

SET @id_pago := NULL; SET @saldo_nuevo := NULL;

-- Registrar un pago parcial de $10,000 por transferencia

CALL sp_registrar_pago(

  16, '2026-04-15 12:00:00', 10000.00,

  'transferencia', 'parcial',

  'REF-16-PARCIAL-01',

  'Pago parcial previo al evento',

  @id_pago, @saldo_nuevo

);

-- Ver resultados devueltos

SELECT @id_pago AS id_pago_creado, @saldo_nuevo AS saldo_pendiente_actualizado;

-- Ver el pago insertado

SELECT * FROM pagos WHERE id_pago = @id_pago;
```

Stored Procedure #2: sp_resumen_evento_sel

Objetivo

Obtener el **expediente completo** de un evento (cabecera, servicios, pagos y totales) de forma flexible. Permite traer **todo** en un solo llamado o **una sección específica** según el parámetro p_seccion.

Parámetros

- IN p_evento INT → ID del evento a consultar.
- IN p_seccion ENUM('todo','cabecera','servicios','pagos','totales')

‘todo’: devuelve 4 results sets (cabecera, servicios, pagos, totales)

‘cabecera’: datos de servicios contratados (cantidad, precio, subtotal)

‘pagos’: historial de pagos (fecha, monto, método, tipo, referencia, notas)

‘totales’: totales consolidados (total_evento, total_pagado, saldo_pendiente, % pagado)

Tablas que consulta

- eventos, clientes → cabecera de evento y datos del cliente.
- eventos_servicios, servicios → detalle y costos de servicios.
- pagos → pagos registrados (considera pagado = TRUE para totales).

Secciones y columnas devueltas

1. **Cabecera** ('cabecera' o 'todo')
    - id_evento, nombre_evento, fecha_evento, tipo_evento, lugar, cantidad_personas, id_cliente, cliente

2. **Servicios** ('servicios' o 'todo')
   tipo_servicio, nombre_servicio, cantidad, precio_unitario, subtotal

3. **Pagos** ('pagos' o 'todo')
    - fecha_pago, monto, metodo_pago, tipo_pago, referencia, pagado, notas

4. **Totales** ('totales' o 'todo')

    - total_evento, total_pagado, saldo_pendiente, porcentaje_pagado
      (usa fn_porcentaje_pagado_evento(p_evento))

Comportamiento

- El SP es **solo lectura** (READS SQL DATA).
- Según p_seccion, puede retornar **uno o varios result sets** en el mismo CALL.
- Emplea COALESCE para evitar NULL en totales cuando no hay registros.

Ejemplos de uso

```
-- 1) Todo el expediente del evento 16 (4 result sets)

CALL sp_resumen_evento_sel(16, 'todo');

-- 2) Solo cabecera

CALL sp_resumen_evento_sel(16, 'cabecera');

-- 3) Solo servicios contratados

CALL sp_resumen_evento_sel(16, 'servicios');

-- 4) Solo pagos

CALL sp_resumen_evento_sel(16, 'pagos');

-- 5) Solo totales (útil para dashboard)

CALL sp_resumen_evento_sel(16, 'totales');
```

Stored Procedure #3: sp_reporte_finanzas_simple()

Objetivo: Generar un **reporte financiero general** de todos los eventos, mostrando para cada uno: total del evento, total pagado, saldo pendiente y un estado (“Pagado”/“Pendiente”). Sirve como tablero rápido para finanzas y seguimiento de cobranza.

Parámetros

- *(Ninguno)*. Devuelve el reporte completo ordenado por fecha de evento.

Tablas que consulta

- eventos (e) → base del listado.
- clientes (c) → nombre del cliente.
- eventos_servicios (subconsulta te) → suma de subtotal por evento (total_evento).
- pagos (subconsulta tp) → suma de monto por evento con pagado = TRUE (total_pagado).

Columnas devueltas

- id_evento
- nombre_evento
- fecha_evento
- tipo_evento
- cliente (nombre + apellido)
  total_evento (Σ subtotales de servicios)
- total_pagado (Σ pagos confirmados)
- saldo_pendiente = total_evento - total_pagado
- estado_pago = 'Pagado' si total_pagado ≥ total_evento, si no 'Pendiente'

Orden de salida

- ORDER BY e.fecha_evento, e.id_evento.

Ejemplo de uso:

CALL sp_reporte_finanzas_simple();

Trigger #1: bi_pagos_validaciones

Tipo: BEFORE INSERT en la tabla pagos.

Objetivo: Este trigger se ejecuta antes de insertar un nuevo pago y garantiza que se cumplan reglas críticas de negocio:

- Evitar montos inválidos.
- Verificar que el evento exista.
- Validar la coherencia temporal según el tipo de pago.
- Evitar sobrepagos respecto al costo total del evento.

Validaciones implementadas:

1. **Monto positivo** → si NEW.monto <= 0, bloquea el insert.

2. **Evento válido** → consulta en eventos; si no existe, cancela.

3. **Reglas según tipo de pago:**

    - anticipo → debe hacerse **antes** de la fecha del evento.

    - liquidacion → no puede realizarse **antes** de la fecha del evento.

4. **Sobrepago** → suma los pagos confirmados (pagado = TRUE) + el nuevo pago; si excede el total de servicios (eventos_servicios), bloquea.

Tablas que impacta / consulta:

- eventos → para validar existencia y fecha del evento.
- eventos_servicios → para calcular el costo total del evento.
- pagos → para validar pagos confirmados acumulados.

Acción si falla una validación:

- Lanza un error con SIGNAL SQLSTATE '45000' y un mensaje descriptivo (ejemplo: "El monto excede el saldo pendiente").
- Esto cancela automáticamente el INSERT en pagos.

-- prueba exitosa

INSERT INTO pagos (id_evento, fecha_pago, monto, metodo_pago, tipo_pago, referencia, pagado, notas)

VALUES (16, '2026-04-15 12:00:00', 10000.00, 'transferencia', 'parcial', 'TEST-VALIDO-01', TRUE, 'Prueba pago válido');

-- prueba fallida excede el monto que se debe

INSERT INTO pagos (id_evento, fecha_pago, monto, metodo_pago, tipo_pago, referencia, pagado, notas)

VALUES (16, '2026-04-15 12:00:00', 50000.00, 'transferencia', 'parcial', 'TEST-INVALIDO-01', TRUE, 'Prueba sobrepago');

Trigger #2: bi_eventos_servicios_validaciones

Tipo: BEFORE INSERT en la tabla eventos_servicios.

Objetivo: Validar la consistencia y reglas de negocio antes de asignar un servicio a un evento. Previene inserciones erróneas como cantidades nulas, precios inválidos, servicios inexistentes o duplicados.

Validaciones implementadas:

1. **Cantidad válida** → NEW.cantidad debe ser **≥ 1**.
2. **Precio válido** → NEW.precio_unitario debe ser **> 0**.
3. **Servicio existente y activo** →
    - Si el id_servicio no existe en servicios, bloquea el insert.
    - Si existe pero está inactivo (activo = 0), también lo bloquea.
4. **No duplicidad** → evita que un mismo id_servicio se agregue más de una vez al mismo id_evento.

Tablas que impacta / consulta:

- servicios → para validar existencia y estado activo del servicio.
- eventos_servicios → para detectar duplicados.

Acción si falla una validación:

- Lanza un error con SIGNAL SQLSTATE '45000' y un mensaje descriptivo (ejemplo: "El servicio ya está agregado para este evento").
- Esto cancela automáticamente el INSERT en eventos_servicios.

Ejemplo de uso:

-- Intento incorrecto: cantidad inválida

INSERT INTO eventos_servicios (id_evento, id_servicio, cantidad, precio_unitario)

VALUES (16, 1, 0, 45000);

-- Resultado: ERROR -> "La cantidad debe ser >= 1"

```
-- Intento incorrecto: servicio inactivo
UPDATE servicios SET activo = 0 WHERE id_servicio = 5;
INSERT INTO eventos_servicios (id_evento, id_servicio, cantidad, precio_unitario)
VALUES (16, 5, 1, 9000);
-- Resultado: ERROR -> "El servicio está inactivo"

-- Intento correcto
INSERT INTO eventos_servicios (id_evento, id_servicio, cantidad, precio_unitario)
VALUES (16, 2, 1, 30000);
-- Resultado: Servicio agregado al evento.
```