

LISTADO DE VISTAS:

-- Vista de precios por provincia (vista_precios_provincia)

```
CREATE OR REPLACE VIEW vista_precios_provincia AS
```

```
SELECT
```

```
    Provincia,
```

```
    COUNT(*) AS Cantidad_Autos,
```

```
    AVG(Precio_ARS) AS Precio_Promedio,
```

```
    SUM(Precio_ARS) AS Precio_Total
```

```
FROM autos
```

```
GROUP BY Provincia;
```

	Provincia	Cantidad_Autos	Precio_Promedio	Precio_Total
►	Neuquén	12527	41512278.217450	520024309230.00
	Mendoza	10323	41742228.272208	430905022454.00
	Buenos Aires	9685	41807728.207744	404907847692.00
	Córdoba	9430	41823821.217815	394398634084.00
	CABA	8210	41569737.037272	341287541076.00
	Santa Fe	5490	41772943.568670	229333460192.00
	San Juan	3626	41805581.383894	151587038098.00
	San Luis	3494	42381372.123068	148080514198.00
	Formosa	412	40523054.733010	16695498550.00
	Tucumán	410	42477360.795122	17415717926.00
	Corrientes	401	41250963.596010	16541636402.00
	Chaco	398	42498105.336683	16914245924.00
	Santiago del Estero	397	40902640.831234	16238348410.00
	Salta	396	41636632.146465	16488106330.00
	Chubut	396	42211441.550505	16715730854.00
	Tierra del Fuego	395	41361397.437975	16337751988.00
	Catamarca	393	42433652.152672	16676425296.00
	Entre Ríos	388	41760175.417526	16202948062.00
	Río Negro	386	40953113.134715	15807901670.00
	La Pampa	377	43055702.021220	16231999662.00
	Jujuy	375	40801725.562667	15300647086.00
	La Rioja	364	41615842.472527	15148166660.00
	Santa Cruz	363	42046195.272727	15262768884.00
	Misiones	352	41348435.482955	14554649290.00

Descripción:

Muestra, por cada provincia, el número de autos vendidos y su precio promedio o total, agrupados a partir de los datos de la tabla autos.

Objetivo:

Permitir un análisis rápido de los costos y ventas en cada provincia, útil para comparaciones regionales y elaborar estrategias de mercado.

Tablas involucradas:

- autos (se agrupa por Provincia y se calcula SUM(Precio_ARS) o AVG(Precio_ARS)).

-- Vista de ventas por concesionario (vista_ventas_concesionario)

```
CREATE OR REPLACE VIEW vista_ventas_concesionario AS
```

```
SELECT
```

```

Concesionario,

COUNT(*) AS Cantidad_Ventas,

SUM(Precio_ARS) AS Total_Ventas

FROM autos

GROUP BY Concesionario;

```

	Concesionario	Cantidad_Ventas	Total_Ventas
►	Peugeot Autos S.A.	11373	357269190000.00
	VW Patagonia	4990	222388795974.00
	Chevrolet Norte	5953	204810379700.00
	Fiat Motor Sur	8076	224185576000.00
	Ford Sur S.A.	10829	639738080650.00
	Nissan Global	3444	226252202400.00
	VW Federal	5000	221819529294.00
	Toyota Center	9796	436975311000.00
	Toyota Express	6021	262379345000.00
	Renault Futuro	3506	83238500000.00

Descripción:

Agrupar la información de venta de autos por concesionario, mostrando la cantidad de vehículos vendidos y el monto total facturado (suma de precios).

Objetivo:

Identificar los concesionarios con mayor volumen de ventas y facturación, para evaluar su rendimiento y planificar incentivos o mejoras.

Tablas involucradas:

- autos (usa Concesionario, Precio_ARS para agrupar y sumar).

```
-- Vista de conteo por tipo de energía (vista_energia_autos)
```

```
CREATE OR REPLACE VIEW vista_energia_autos AS
```

```
SELECT
```

```
    Energia,
```

```
    COUNT(*) AS Cantidad
```

```
FROM autos
```

```
GROUP BY Energia;
```

	Energia	Cantidad
►	Gasolina	44468
	Diesel	13789
	Eléctrico/Híbrido	10731

Descripción:

Presenta un desglose de cuántos autos corresponden a cada tipo de energía (Gasolina, Diésel, Eléctrico/Híbrido, etc.).

Objetivo:

Facilitar la consulta sobre la adopción de tecnologías de combustible alternativas y la comparación entre ellas.

Tablas involucradas:

- autos (agrupa por la columna Energia).

```
-- Vista de autos híbridos/eléctricos (vista_autos_ecologicos_agrupados)
```

```
CREATE OR REPLACE VIEW vista_autos_ecologicos_agrupados AS
```

```
SELECT
```

```
    Marca,
```

```
    Modelo,
```

```
    Version,
```

```
    Motor,
```

```
    Energia,
```

```
    COUNT(*) AS Cantidad
```

```
FROM autos
```

```
WHERE Energia = 'Eléctrico/Híbrido'
```

```
GROUP BY
```

```
    Marca,
```

```
    Modelo,
```

```
    Version,
```

```
    Motor,
```

```
    Energia;
```

	Marca	Modelo	Version	Motor	Energia	Cantidad
▶	Toyota	Corolla Cross HEV	SEG	1.8 L + 53 kW e-Motor	Eléctrico/Híbrido	3842
	Ford	Maverick	XLT H	2.5 L + 94 kW e-Motor	Eléctrico/Híbrido	3383
	Renault	Kwid E-Tech	Iconic	48 kW PMSM	Eléctrico/Híbrido	1770
	Renault	Kwid E-Tech	Zen	48 kW PMSM	Eléctrico/Híbrido	1736

Descripción:

Filtra todos los vehículos que utilizan energías limpias o combinadas (Eléctrico, Híbrido) para ver únicamente estos registros.

Objetivo:

Proporcionar un listado enfocado en la categoría ecológica, permitiendo analizar su adopción e impacto en el mercado.

Tablas involucradas:

- autos (se filtra por la columna Energia = 'Eléctrico/Híbrido').

```
-- Vista de participación de mercado por marca (vista_participacion_marca)
```

```
CREATE OR REPLACE VIEW vista_participacion_marca AS
```

```
SELECT
```

```
    Marca,
```

```
    COUNT(*) AS Cantidad_Patentamientos,
```

```
    ROUND(
```

```
        COUNT(*) * 100.0 / (SELECT COUNT(*) FROM autos),
```

```
        2
```

```
    ) AS Porcentaje_Mercado
```

FROM autos

GROUP BY Marca

ORDER BY Porcentaje_Mercado DESC;

	Marca	Cantidad_Patentamientos	Porcentaje_Mercado
▶	Toyota	15817	22.93
	Peugeot	11373	16.49
	Ford	10829	15.70
	Volkswagen	9990	14.48
	Fiat	8076	11.71
	Chevrolet	5953	8.63
	Renault	3506	5.08
	Nissan	3444	4.99

Descripción:

Agrupar los patentamientos por marca, calculando el porcentaje de participación que tiene cada una en el total de ventas.

Objetivo:

Ver el "market share" de cada marca de forma rápida y en un solo vistazo.

Tablas involucradas:

- autos (agrupa por Marca, calcula COUNT(*), y un sub-SELECT para el total o se hace una ventana de análisis para el total general).

FUNCIONES :

Función que valida si un auto es híbrido o eléctrico (retorna 1/0)

```
sql
CopiarEditar
DELIMITER $$

CREATE FUNCTION fn_es_ecologico(
    p_energia VARCHAR(20)
)
RETURNS TINYINT
DETERMINISTIC
BEGIN
    -- Retorna 1 si la energía es 'Eléctrico', 'Híbrido' o 'Eléctrico/Híbrido', y 0 en
    caso contrario
    IF p_energia IN ('Eléctrico', 'Híbrido', 'Eléctrico/Híbrido') THEN
        RETURN 1;
    ELSE
        RETURN 0;
    END IF;
END $$

DELIMITER ;
```

- **Descripción:** Determina si un tipo de energía corresponde a vehículo ecológico (1) o no (0).
- **Objetivo:** Tener rápidamente una columna booleana en consultas, sin escribir la lógica repetida.
- **Tablas:** No accede a ninguna; se basa en el parámetro p_energia.

Uso:

```

sql
CopiarEditar
SELECT
    ID_Patente,
    Energia,
    fn_es_ecologico(Energia) AS EsEcologico
FROM autos;

```

Función que estandariza (formatea) una patente a mayúsculas

```

sql
CopiarEditar
DELIMITER $$

CREATE FUNCTION fn_formato_patente_simple(
    p_patente VARCHAR(10)
)
RETURNS VARCHAR(10)
DETERMINISTIC
BEGIN
    -- Convierte la patente a mayúsculas y quita espacios en blanco
    RETURN UPPER(TRIM(p_patente));
END $$

DELIMITER ;

```

- **Descripción:** Devuelve la patente en **mayúsculas** y sin espacios al inicio o fin.
- **Objetivo:** Evitar inconsistencias en reportes y búsquedas, donde "abc123" y "ABC123" deben considerarse iguales.
- **Tablas:** No usa tablas, solo formatea la cadena recibida.

Uso:

```

sql
CopiarEditar
SELECT
    ID_Patente,
    fn_formato_patente_simple(ID_Patente) AS Patente_Normalizada
FROM autos;

```

Función que retorna el rango de precio (simplificado)

```

DELIMITER $$

CREATE FUNCTION fn_rango_precio_simple(
    p_precio DECIMAL(10,2)
)
RETURNS VARCHAR(15)
DETERMINISTIC
BEGIN
    -- Clasifica en 3 categorías sencillas
    IF p_precio < 25000000 THEN
        RETURN 'Económico';
    ELSEIF p_precio < 40000000 THEN
        RETURN 'Mediano';
    ELSE
        RETURN 'Alto';
    END IF;
END

```

```

ELSE
    RETURN 'Alto';
END IF;
END $$

```

DELIMITER ;

- **Descripción:** Devuelve una etiqueta (Económico, Mediano, Alto) basada en un corte de precio muy simple.
- **Objetivo:** Tener una clasificación rápida en reportes o vistas sin escribir CASE en cada query.
- **Tablas:** No accede a tablas, solo el valor p_precio.

Uso:

```

SELECT
    Precio_ARS,
    fn_rango_precio_simple(Precio_ARS) AS Rango
FROM autos;

```

Stored Procedures:

1. sp_insertar_nuevo_patentamiento

- **Descripción:** Inserta un nuevo registro de auto patentado, validando que se cumplan ciertas condiciones (por ejemplo, la existencia de la marca, el modelo, etc.).
- **Objetivo:**
 - Centralizar la lógica de inserción para evitar inconsistencias.
 - Asegurar la integridad referencial (que la MarcaID o el ModeloID existan).
 - Facilitar que los usuarios (o sistemas externos) no tengan que conocer la estructura detallada de la tabla autos.
- **Tablas que interactúa:**
 - autos (inserta el registro principal).
 - Opcionalmente, marcas, modelos, motores (si validas FK antes de insertar).
 - Puede registrar en una tabla de **bitácora** o **logs** quién y cuándo hizo la inserción.

Ejemplo de definición (pseudocódigo MySQL):

```

sql
CopiarEditar
DELIMITER $$

```

```

CREATE PROCEDURE sp_insertar_nuevo_patentamiento (
    IN p_patente VARCHAR(10),
    IN p_marcaID INT,
    IN p_modeloID INT,
    IN p_versionID INT,
    IN p_motorID INT,
    IN p_color VARCHAR(50),
    IN p_origen VARCHAR(50),
    IN p_fecha DATE,
    IN p_concesionario VARCHAR(100),
    IN p_precio DECIMAL(10,2),
    IN p_provincia VARCHAR(50),
    IN p_energia VARCHAR(20),

```

```

    IN p_forma_compra VARCHAR(20)
)
BEGIN
    -- (Validaciones opcionales si se desea)

    INSERT INTO autos (
        ID_Patente, MarcaID, ModeloID, VersionID, MotorID,
        Color, Origen, Fecha_Patentamiento, Concesionario,
        Precio_ARS, Provincia, Energia, Forma_Compra
    )
    VALUES (
        p_patente, p_marcaID, p_modeloID, p_versionID, p_motorID,
        p_color, p_origen, p_fecha, p_concesionario,
        p_precio, p_provincia, p_energia, p_forma_compra
    );
END $$

DELIMITER ;

```

2. sp_actualizar_precio_vehiculo

- **Descripción:**
Actualiza el precio de un vehículo en la tabla autos, por ejemplo al subir la lista de precios o al aplicarse una promoción.
- **Objetivo:**
 - o Mantener la información de precios al día sin permitir actualizaciones directas sin control.
 - o Registrar en una tabla de histórico, si es necesario, las variaciones de precio.
 - o Evitar errores en la sintaxis de actualización cuando se hace a mano.
- **Tablas que interactúa:**
 - o autos (actualiza la columna Precio_ARS).
 - o Opcional: historico_precios si se desea mantener registro de los cambios con su fecha.

Ejemplo:

```

sql
CopiarEditar
DELIMITER $$

CREATE PROCEDURE sp_actualizar_precio_vehiculo (
    IN p_patente VARCHAR(10),
    IN p_nuevo_precio DECIMAL(10,2)
)
BEGIN
    -- Registrar cambio en historico (opcional)
    -- INSERT INTO historico_precios (ID_Patente, PrecioAntiguo, PrecioNuevo,
    FechaCambio)
    -- SELECT a.ID_Patente, a.Precio_ARS, p_nuevo_precio, NOW()
    -- FROM autos a
    -- WHERE a.ID_Patente = p_patente;

    -- Actualizar el precio
    UPDATE autos
    SET Precio_ARS = p_nuevo_precio
    WHERE ID_Patente = p_patente;
END $$

DELIMITER ;

```

3. sp_eliminar_registro_patentamiento

- **Descripción:**
Elimina (o marca como eliminado) un registro de la tabla autos. Útil si se detecta un auto duplicado o un dato cargado por error.
- **Objetivo:**
 - Tener un punto central para las bajas, permitiendo agregar validaciones o auditoría (por ejemplo, no permitir eliminar si el auto ya fue facturado).
 - Asegurar consistencia de datos y no depender de borrados manuales.
- **Tablas que interactúa:**
 - autos.
 - Opcional: bitacora o registro_bajas (para anotar quién y cuándo eliminó).

Ejemplo:

```
sql
CopiarEditar
DELIMITER $$

CREATE PROCEDURE sp_eliminar_registro_patentamiento (
    IN p_patente VARCHAR(10)
)
BEGIN
    -- (Validaciones opcionales: chequear si se puede borrar)
    DELETE FROM autos
    WHERE ID_Patente = p_patente;
END $$

DELIMITER ;
```

O, en vez de DELETE, se podría hacer un UPDATE autos SET Activo = 0 si prefieres conservar los datos pero marcarlos como inactivos.

4. sp_generar_reporte_mensual

- **Descripción:**
Genera un reporte de patentamientos en un mes y año específico, agrupando por provincia, marca, o cualquier dimensión. Puede guardar los resultados en una tabla temporal o devolver un SELECT.
- **Objetivo:**
 - Automatizar la obtención de un informe resumido de ventas/patentamientos, sin que el usuario deba escribir la lógica de agrupar y filtrar.
 - Posibilidad de programar la ejecución del reporte (por ejemplo, cada inicio de mes) y guardar datos en una tabla histórica.
- **Tablas que interactúa:**
 - autos (fuente principal de datos).
 - Podrías usar marcas, modelos, etc. si necesitas la descripción textual.
 - Opcional: una tabla de **reportes** si deseas guardar los resultados.

Ejemplo (versión que hace un SELECT final):

```
sql
CopiarEditar
DELIMITER $$

CREATE PROCEDURE sp_generar_reporte_mensual (
    IN p_anio INT,
    IN p_mes INT
)
BEGIN
    SELECT
        MarcaID,
        COUNT(*) AS Cantidad,
        SUM(Precio_ARS) AS TotalVentas
    FROM autos
```



```
WHERE YEAR(Fecha_Patentamiento) = p_anio
AND MONTH(Fecha_Patentamiento) = p_mes
GROUP BY MarcaID;
END $$
```

DELIMITER ;

Luego lo llamas con:

```
sql
CopiarEditar
CALL sp_generar_reporte_mensual(2025, 3);
```

Te traerá la data del mes 3 (marzo) de 2025.

5. sp_sincronizar_precios

- **Descripción:**
Sincroniza los precios entre la tabla autos y la tabla precios, donde precios podría ser una tabla histórica o de referencia.
- **Objetivo:**
 - Mantener la coherencia de datos de precios en dos tablas.
 - Prevenir valores obsoletos en la tabla precios si la principal es autos, o viceversa.
- **Tablas que interactúa:**
 - autos.
 - precios.

Ejemplo:

```
sql
CopiarEditar
DELIMITER $$

CREATE PROCEDURE sp_sincronizar_precios ()
BEGIN
    -- Ejemplo: Actualizar la tabla precios a partir de autos
    UPDATE precios p
    JOIN autos a ON p.AutoID = a.AutoID
    SET p.Precio_ARS = a.Precio_ARS
    WHERE p.Precio_ARS <> a.Precio_ARS;

    -- Podrías agregar más lógica, por ejemplo, insertar registros nuevos en 'precios'
END $$

DELIMITER ;
```

6. sp_normalizar_autos

- **Descripción:**
Para el caso en que tengas datos duplicados o con información en texto (Marca, Modelo, etc.) y quieras establecer los IDs correctos en la tabla autos_normalizados, o viceversa.
- **Objetivo:**
 - Reducir manualidades al hacer un UPDATE masivo o varios INSERT.
 - Garantizar que la relación con las tablas marcas, modelos, motores, versiones quede bien establecida (usando sus IDs).
- **Tablas que interactúa:**
 - autos y autos_normalizados.
 - marcas, modelos, versiones, motores para localizar los IDs correctos según la descripción textual.

Ejemplo (versión de UPDATE simplificado):

```
sql
CopiarEditar
DELIMITER $$

CREATE PROCEDURE sp_normalizar_autos ()
BEGIN
    UPDATE autos_normalizados an
    JOIN autos a ON an.AutoID = a.AutoID
    JOIN motores m ON a.Motor = m.Descripcion
    SET an.MotorID = m.MotorID
    WHERE an.MotorID IS NULL;
END $$

DELIMITER ;

TRIGGERS
```

1. Trigger de Auditoría de Inserción

- **Descripción:** Al **insertar** un nuevo registro en la tabla autos (o cualquier tabla que quieras auditar), este trigger guarda un registro en la tabla bitacora con datos como la patente, la fecha y el usuario que hizo el insert.
- **Objetivo:**
 - Mantener un **histórico** de quién y cuándo insertó un nuevo auto.
 - Facilitar la trazabilidad si se requiere.
- **Tablas involucradas:**
 - **Tabla principal:** autos.
 - **Tabla de auditoría:** bitacora (o logs).

Ejemplo de código (MySQL):

```
sql
CopiarEditar
CREATE TRIGGER trg_auditoria_autos_insert
AFTER INSERT
ON autos
FOR EACH ROW
BEGIN
    INSERT INTO bitacora (Tabla, Operacion, Descripcion, Fecha)
    VALUES (
        'autos',
        'INSERT',
        CONCAT('Se insertó la patente: ', NEW.ID_Patente),
        NOW()
    );
END;
```

(Adapta campos de bitacora a tu realidad; podrías incluir el usuario si tu sistema de seguridad lo provee.)

2. Trigger para Actualizar Fecha de Última Modificación

- **Descripción:** Al **actualizar** un registro en la tabla autos, se modifica o llena automáticamente la columna Fecha_Ultima_Modificacion.
- **Objetivo:**
 - Saber cuándo fue la última vez que se cambió un registro, sin depender de la intervención manual.
 - Útil para auditoría ligera o para detectar registros obsoletos.
- **Tablas involucradas:**

- o autos (que debe tener una columna Fecha_Ultima_Modificacion de tipo DATETIME o TIMESTAMP).

Ejemplo de código (MySQL):

```
sql
CopiarEditar
CREATE TRIGGER trg_autos_update_fecha
BEFORE UPDATE
ON autos
FOR EACH ROW
BEGIN
    SET NEW.Fecha_Ultima_Modificacion = NOW();
END;
```

(Se ejecuta antes de la actualización, para que la fila se guarde con la fecha/hora nueva.)

3. Trigger de Prevención o Validación de Datos

- **Descripción:** Al **insertar** o **actualizar** un registro en autos, verifica que ciertos campos tengan valores válidos (por ejemplo, que Precio_ARS sea mayor a cero, que Energia no sea un valor desconocido, etc.). Si se incumple, **lanza un error**.
- **Objetivo:**
 - o Prevenir la inserción de datos corruptos o inconsistentes.
 - o Asegurar la calidad de la información en la base.
- **Tablas involucradas:**
 - o autos (o la que quieras proteger).

Ejemplo de código (MySQL):

```
sql
CopiarEditar
CREATE TRIGGER trg_autos_insert_valida
BEFORE INSERT
ON autos
FOR EACH ROW
BEGIN
    IF NEW.Precio_ARS <= 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: Precio_ARS no puede ser menor o igual a 0';
    END IF;

    IF NEW.Energia NOT IN ('Gasolina', 'Diesel', 'Eléctrico/Híbrido', 'Eléctrico', 'Híbrido') THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: Tipo de energía inválido.';
    END IF;
END;
```

(Adapta las condiciones según tu catálogo de energías o reglas.)

4. Trigger para Actualizar Tabla Histórica de Precios

- **Descripción:** Cada vez que se **actualiza** la columna Precio_ARS en la tabla autos, se inserta un registro en la tabla historico_precios (guardando la fecha, precio anterior, precio nuevo, etc.).
- **Objetivo:**
 - o Mantener un **historial** de los cambios de precio sin que el usuario deba hacerlo manualmente.

- o Posibilitar reportes de evolución de precios en el tiempo.
- **Tablas involucradas:**
 - o autos.
 - o historico_precios (debe tener campos como ID, Patente o AutoID, PrecioAnterior, PrecioNuevo, FechaCambio...).

Ejemplo de código (MySQL):

```
sql
CopiarEditar
CREATE TRIGGER trg_autos_update_preco
BEFORE UPDATE
ON autos
FOR EACH ROW
BEGIN
    IF NEW.Precio_ARS <> OLD.Precio_ARS THEN
        INSERT INTO historico_precios
            (AutoID, PrecioAnterior, PrecioNuevo, FechaCambio)
        VALUES
            (OLD.AutoID, OLD.Precio_ARS, NEW.Precio_ARS, NOW());
    END IF;
END;
```

(Suponiendo que AutoID sea la llave primaria de autos; si usas ID_Patente, cámbialo en el INSERT.)

5. Trigger para Normalizar Texto en Columnas (Marca, Modelo, Patente...)

- **Descripción:** Al **insertar** o **actualizar** un registro, puede forzar la columna Marca (o Patente, Modelo) a un formato definido (mayúsculas, sin espacios extra, etc.).
- **Objetivo:**
 - o Asegurar **uniformidad** de datos.
 - o Evitar duplicados por diferencias tipográficas ("Toyota" vs "toyota").
- **Tablas involucradas:**
 - o autos, marcas u otra tabla donde quieras mantener un formato uniforme.

Ejemplo de código (MySQL):

```
sql
CopiarEditar
CREATE TRIGGER trg_autos_insert_normaliza
BEFORE INSERT
ON autos
FOR EACH ROW
BEGIN
    SET NEW.Marca = UPPER(TRIM(NEW.Marca));
    SET NEW.Modelo = UPPER(TRIM(NEW.Modelo));
    -- Similar para la patente o versión si manejas texto
END;
```

(Puedes replicar la misma lógica en un BEFORE UPDATE si también quieres normalizar al actualizar.)

6. Trigger para Controlar Eliminaciones (ON DELETE)

- **Descripción:** Antes de eliminar un registro de autos, verifica si el auto ya tiene ciertos vínculos (por ejemplo, facturación). Si no se permite eliminar en tal caso, lanza un error o, en su defecto, mueve la data a una tabla de "bajas".
- **Objetivo:**

- o Evitar eliminaciones de datos que deberían conservarse por requisitos legales o de negocio.
- o Mantener **integridad** y lógica de negocio.
- **Tablas involucradas:**
 - o autos.
 - o Podrías consultar otras tablas (facturas, pagos, etc.) antes de permitir la eliminación.

Ejemplo (MySQL):

```
sql
CopiarEditar
CREATE TRIGGER trg_autos_delete_control
BEFORE DELETE
ON autos
FOR EACH ROW
BEGIN
    -- Ejemplo: no permitir borrar si el auto ya está en una tabla de "Ventas"
    IF EXISTS (SELECT 1 FROM ventas WHERE AutoID = OLD.AutoID) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Error: No se puede eliminar un auto que ya fue
vendido.';
    END IF;
END;
```