
# Trabajo Práctico Nº 2

**CARRERA**: Ingeniería en Sistemas de Información

**MATERIA**: Paradigmas y Lenguajes de Programación III

**COMISIÓN**: “A”

**PROFESOR**: Mgst. Ing. Gonzalo Pallotta

**ESTUDIANTES**: José Fernando Usui, Ezequiel Neyen Troche y Luciana Nadine Rojas

**FECHA**: 09-10-2025

# TABLA DE CONTENIDO

# Resumen

El presente documento describe el diseño e implementación de un sistema de base de datos relacional orientado a la gestión integral de servicios de movilidad urbana. La arquitectura propuesta soporta operaciones complejas relacionadas con la administración de conductores, pasajeros, viajes, sistemas de pago y evaluaciones bidireccionales. El diseño privilegia la integridad referencial, el rendimiento optimizado mediante estrategias de indexación y la extensibilidad del sistema. Se empleó MySQL como gestor de base de datos, implementando restricciones de validación a nivel de esquema mediante constraints y configuraciones que garantizan la consistencia transaccional. El análisis comparativo de alternativas de diseño fundamenta cada decisión arquitectónica, considerando criterios de normalización, eficiencia computacional y mantenibilidad a largo plazo.

# Introducción

## Contexto y Justificación

Los sistemas de movilidad urbana compartida han experimentado un crecimiento exponencial en las últimas décadas, transformando radicalmente los patrones de transporte en entornos metropolitanos. Plataformas como Uber, Lyft y servicios similares han demostrado la necesidad de arquitecturas de datos robustas capaces de gestionar millones de transacciones diarias mientras mantienen altos estándares de integridad y disponibilidad. El diseño de bases de datos para estos sistemas presenta desafíos particulares relacionados con la concurrencia, la consistencia transaccional y la optimización de consultas complejas que involucran múltiples entidades relacionadas.

La presente documentación analiza exhaustivamente la arquitectura de base de datos desarrollada para un sistema de gestión de movilidad urbana, justificando cada decisión de diseño mediante el análisis comparativo de alternativas y la evaluación de su impacto en dimensiones críticas como rendimiento, escalabilidad e integridad de datos.

## Objetivos del Sistema

El sistema de base de datos propuesto tiene como objetivo principal proporcionar una infraestructura sólida para la gestión integral de operaciones de transporte compartido. Los objetivos específicos incluyen la administración eficiente de entidades fundamentales (conductores, pasajeros, viajes), la implementación de mecanismos de evaluación bidireccional que fomenten la calidad del servicio, el soporte para sistemas de pago múltiples con trazabilidad completa, y la capacidad de realizar análisis temporal sobre estados de viajes y transacciones financieras.

# Marco Teórico

## Principios de Diseño de Bases de Datos Relacionales

El diseño de bases de datos relacionales se fundamenta en principios establecidos por Codd que buscan minimizar la redundancia y maximizar la integridad de los datos. La normalización constituye el proceso sistemático mediante el cual se organizan las tablas y sus relaciones para reducir anomalías de actualización, inserción y eliminación. El presente diseño alcanza predominantemente la tercera forma normal, equilibrando los beneficios de la normalización con consideraciones pragmáticas de rendimiento.

## Estrategias de Indexación

La indexación representa un mecanismo fundamental para optimizar el rendimiento de consultas en sistemas de bases de datos relacionales. Los índices funcionan como estructuras de datos auxiliares que permiten la localización rápida de registros sin necesidad de escanear secuencialmente tablas completas. La selección apropiada de columnas a indexar requiere analizar patrones de consulta frecuentes, considerando el balance entre la aceleración de lecturas y el overhead introducido en operaciones de escritura.

## Integridad Referencial y Constraints

La integridad referencial garantiza que las relaciones entre tablas permanezcan consistentes, previniendo la existencia de referencias huérfanas. Los constraints de tipo CHECK permiten implementar reglas de negocio a nivel de base de datos, proporcionando una capa adicional de validación que complementa las verificaciones realizadas en la capa de aplicación. Esta estrategia de defensa en profundidad resulta particularmente relevante en sistemas distribuidos donde múltiples aplicaciones pueden interactuar con la base de datos.

# Metodología

## Configuración del Entorno de Desarrollo

La implementación del sistema requiere la configuración inicial del entorno MySQL mediante la desactivación temporal de verificaciones de integridad. Esta práctica, aunque aparentemente contraintuitiva, optimiza significativamente el proceso de carga inicial de datos relacionados al evitar errores derivados del orden de inserción. El siguiente fragmento ilustra la configuración implementada:

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS,
UNIQUE_CHECKS=0; SET
@OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHEC
KS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES';
```

La configuración del modo SQL incorpora validaciones estrictas mediante STRICT_TRANS_TABLES, que rechaza transacciones que violarían restricciones de tipo de dato, y ONLY_FULL_GROUP_BY, que exige conformidad con el estándar SQL en operaciones de agregación. Estas configuraciones previenen errores sutiles que podrían comprometer la integridad de los datos.

# Selección de Codificación de Caracteres

La creación del esquema emplea utf8mb4 como conjunto de caracteres predeterminado, decisión fundamentada en la necesidad de soportar caracteres Unicode completos, incluyendo emojis y símbolos especiales cada vez más prevalentes en aplicaciones modernas. La collation utf8mb4_0900_ai_ci implementa comparaciones que ignoran acentos y mayúsculas, facilitando búsquedas más flexibles:

```
CREATE SCHEMA IF NOT EXISTS `movilidad_urbana`
DEFAULT CHARACTER SET utf8mb4
COLLATE utf8mb4_0900_ai_ci;
```

Esta elección contrasta con alternativas más restrictivas como latin1, que limitarían significativamente la capacidad de internacionalización del sistema.

# Resultados

## Arquitectura de Entidades Principales

### *Diseño de la Tabla Conductores*

La tabla conductores constituye una entidad fundamental del sistema, almacenando información esencial sobre los operadores de vehículos. La estructura implementada incorpora validaciones a nivel de esquema que garantizan la calidad de los datos:

```
CREATE TABLE `conductores` (
   `id_conductor` INT NOT NULL AUTO_INCREMENT,
   `nombre` VARCHAR(100) NOT NULL,
   `licencia` VARCHAR(50) NOT NULL,
   `fecha_alta` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
   PRIMARY KEY (`id_conductor`),
   CONSTRAINT chk_nombre_conductor CHECK (CHAR_LENGTH(`nombre`) >
   0), CONSTRAINT chk_licencia CHECK (CHAR_LENGTH(`licencia`) > 0)
);
```

La utilización de CHAR_LENGTH en lugar de LENGTH resulta crítica en contextos de codificación multibyte, donde LENGTH retorna el número de bytes mientras que CHAR_LENGTH cuenta caracteres individuales. Esta distinción previene validaciones incorrectas en nombres que contienen caracteres no ASCII.

El índice único sobre la columna licencia implementa una restricción de negocio que

previene el registro duplicado de conductores. Esta decisión de diseño se fundamenta en el análisis de alternativas que incluían el uso de licencia como clave primaria, opción descartada por violar principios de estabilidad de claves primarias ante posibles cambios en documentación legal.

## Estructura de la Tabla Pasajeros

La tabla pasajeros incorpora validación de direcciones de correo electrónico mediante un constraint CHECK que verifica patrones básicos de formato. El constraint implementado utiliza operadores LIKE para garantizar la presencia de elementos estructurales mínimos:

```
CONSTRAINT chk_email CHECK (`email` LIKE '%@%.%')
```

Este enfoque representa un balance entre simplicidad y efectividad. Validaciones más complejas mediante expresiones regulares fueron consideradas pero descartadas por incrementar la complejidad sin aportar beneficios sustanciales, dado que la validación detallada de formatos de email se realiza más apropiadamente en la capa de aplicación.

# Diseño de la Tabla Viajes

La tabla viajes representa la entidad central del sistema, conectando conductores, pasajeros y múltiples entidades relacionadas. La selección del tipo de dato DECIMAL(10,2) para la columna tarifa merece análisis detallado. Este tipo garantiza precisión exacta en cálculos monetarios, evitando errores de redondeo inherentes a tipos de punto flotante como FLOAT o DOUBLE.

La estrategia de indexación implementada responde a patrones de consulta anticipados. Se crearon índices individuales sobre las columnas fecha, id_conductor e id_pasajero, facilitando operaciones frecuentes como la generación de reportes temporales y consultas de historial:

```
CREATE INDEX `idx_viajes_fecha` ON `viajes` (`fecha`);
CREATE INDEX `idx_viajes_conductor` ON `viajes` (`id_conductor`); CREATE INDEX `idx_viajes_pasajero` ON `viajes` (`id_pasajero`);
```

Esta aproximación de índices simples fue preferida sobre índices compuestos tras considerar que estos últimos, aunque potencialmente más eficientes para consultas específicas, introducen mayor overhead de mantenimiento y pueden no ser utilizados eficientemente por el optimizador de consultas en escenarios diversos.

# Sistema de Evaluaciones Bidireccionales

El diseño del sistema de evaluaciones implementa un mecanismo bidireccional mediante el cual tanto pasajeros como conductores pueden calificarse mutuamente. Esta funcionalidad se materializa en la tabla evaluaciones, que emplea un campo tipo para distinguir la dirección de la evaluación:

```
CREATE TABLE `evaluaciones` (
   `id_eval` INT NOT NULL AUTO_INCREMENT,
   `id_viaje` INT NOT NULL,
   `calificacion` TINYINT NULL DEFAULT NULL,
   `comentario` VARCHAR(255) NULL DEFAULT NULL,
   `tipo` TINYINT(1) NOT NULL COMMENT
'1=Pasajero->Conductor, 0=Conductor->Pasajero',
   PRIMARY KEY (`id_eval`),
   CONSTRAINT chk_calificacion CHECK (`calificacion` IS NULL OR
      (`calificacion` BETWEEN 1 AND 5))
);
```

La decisión de implementar ambos tipos de evaluación en una única tabla, en contraposición a crear tablas separadas para cada dirección, simplifica considerablemente las consultas de reporting y análisis agregado. El uso de TINYINT(1) para el campo tipo optimiza el uso de espacio, requiriendo únicamente un byte por registro.

# Gestión de Estados mediante Tablas Temporales

El sistema implementa un patrón de diseño temporal para el seguimiento de cambios de estado tanto en viajes como en pagos. La tabla viaje_estado ejemplifica esta aproximación:

```
CREATE TABLE `viaje_estado` (
  `id_viaje` INT NOT NULL,
  `id_estado` INT NOT NULL,
  `fecha_cambio` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id_viaje`, `id_estado`, `fecha_cambio`)
);
```

La clave primaria compuesta por id_viaje, id_estado y fecha_cambio garantiza la unicidad de cada transición de estado mientras previene duplicados ilógicos. Este diseño permite mantener un historial completo de transiciones, fundamental para auditorías y análisis de patrones operacionales.

# Arquitectura del Sistema de Pagos

La tabla pagos implementa una estructura que soporta múltiples métodos de pago mediante una clave primaria compuesta que incluye tanto un identificador autoincremental como la referencia al método de pago:

```
CREATE TABLE `pagos` (
  `id_pago` INT NOT NULL AUTO_INCREMENT,
  `id_viaje` INT NULL DEFAULT NULL,
  `monto` DECIMAL(10,2) NOT NULL,
  `metodospago_id_metodo` INT NOT NULL,
  PRIMARY KEY (`id_pago`, `metodospago_id_metodo`),
  CONSTRAINT chk_monto CHECK (`monto` >= 0)
);
```

Esta estructura hybrid combina las ventajas de claves surrogadas (simplicidad en joins) con la robustez de claves compuestas que refuerzan la integridad referencial. El diseño permite extensibilidad futura para escenarios donde un viaje podría requerir múltiples transacciones de pago.

# Optimización mediante Vistas

Se implementaron vistas que abstraen la complejidad de joins múltiples y facilitan el acceso a información agregada. La vista vw_viajes_detalle, inicialmente concebida como un join directo, fue refinada para incorporar lógica que selecciona únicamente el estado más reciente de cada viaje:

```
CREATE OR REPLACE VIEW vw_viajes_detalle AS
SELECT v.id_viaje, p.nombre AS pasajero, c.nombre AS conductor, v.origen,
   v.destino, v.fecha, v.tarifa, ev.nombre AS estado_actual
FROM viajes v JOIN pasajeros p ON v.id_pasajero = p.id_pasajero JOIN
conductores c ON v.id_conductor = c.id_conductor JOIN (
   SELECT ve1.id_viaje, ve1.id_estado
   FROM viaje_estado ve1
   JOIN (
      SELECT id_viaje, MAX(fecha_cambio) as ultima_fecha
      FROM viaje_estado
      GROUP BY id_viaje
   ) ve2 ON ve1.id_viaje = ve2.id_viaje
      AND ve1.fecha_cambio = ve2.ultima_fecha
) ve ON v.id_viaje = ve.id_viaje
JOIN estadosviaje ev ON ve.id_estado = ev.id_estado;
```

Esta vista elimina la ambigüedad presente en versiones anteriores que retornaban múltiples filas por viaje cuando existían varias transiciones de estado.

# Discusión

## Análisis de Decisiones de Diseño

El diseño implementado refleja un balance cuidadoso entre múltiples dimensiones de calidad. La normalización alcanzada (predominantemente tercera forma normal) elimina redundancias significativas sin incurrir en la fragmentación excesiva característica de diseños sobre-normalizados. Este equilibrio resulta particularmente evidente en el diseño del sistema de evaluaciones, donde la consolidación de direcciones evaluativas en una única tabla simplifica operaciones de reporting sin comprometer la integridad.

Las estrategias de indexación adoptadas priorizan patrones de consulta identificados como frecuentes en sistemas de movilidad urbana: búsquedas temporales para generación de reportes, consultas de historial por conductor o pasajero, y joins entre entidades relacionadas. La decisión de implementar índices simples en lugar de compuestos complejos responde a consideraciones de mantenibilidad y al reconocimiento de que optimizaciones prematuras pueden resultar contraproducentes.

# Consideraciones de Rendimiento

El rendimiento del sistema se ve influenciado por múltiples factores arquitectónicos. Los índices implementados reducen significativamente la complejidad temporal de consultas frecuentes, transformando operaciones que requerirían escaneos completos de tabla en búsquedas logarítmicas mediante estructuras de árbol B. Sin embargo, esta mejora en rendimiento de lectura introduce overhead en operaciones de escritura, ya que cada inserción, actualización o eliminación requiere actualizar estructuras de índice asociadas.

El uso extensivo de constraints CHECK para validación de datos presenta implicaciones de rendimiento ambivalentes. Si bien estas validaciones introducen overhead computacional en cada operación de escritura, previenen la inserción de datos inválidos que podrían degradar el rendimiento de consultas subsecuentes o requerir operaciones costosas de limpieza de datos.

# Extensibilidad y Mantenibilidad

El diseño prioriza la extensibilidad mediante el uso consistente de claves surrogadas autoincrementales, que facilitan la adición de nuevas relaciones sin modificar estructuras existentes. La separación entre claves técnicas (identificadores autoincrementales) y claves de negocio (como números de licencia) permite que estas últimas evolucionen sin impactar la integridad referencial del sistema.

Los patrones de tablas temporales implementados para estados de viajes y pagos proporcionan trazabilidad completa sin requerir modificaciones estructurales cuando se introducen nuevos estados o transiciones. Esta característica resulta particularmente valiosa en

entornos de producción donde las modificaciones de esquema pueden ser costosas y riesgosas.

# Limitaciones y Trabajo Futuro

El diseño actual presenta limitaciones que podrían abordarse en iteraciones futuras. La ausencia de mecanismos de particionamiento horizontal limita la escalabilidad en escenarios de volúmenes extremadamente altos. La implementación de estrategias de particionamiento por rango temporal en la tabla viajes podría mejorar significativamente el rendimiento de consultas históricas.

El sistema de evaluaciones, aunque funcional, podría beneficiarse de mecanismos más sofisticados de detección de fraude o manipulación de calificaciones. Extensiones futuras podrían incorporar análisis estadísticos de patrones de evaluación para identificar comportamientos anómalos.

# Conclusiones

El sistema de base de datos desarrollado para gestión de movilidad urbana representa una implementación sólida de principios fundamentales de diseño relacional. La arquitectura propuesta equilibra efectivamente requisitos competitivos de normalización, rendimiento e integridad de datos mediante la aplicación juiciosa de constraints, índices y patrones de diseño temporal.

Las decisiones arquitectónicas documentadas reflejan un proceso deliberativo que consideró múltiples alternativas, evaluando cada opción mediante criterios de rendimiento, mantenibilidad y extensibilidad. El resultado es un sistema capaz de soportar operaciones complejas de movilidad urbana mientras mantiene integridad referencial estricta y proporciona mecanismos robustos de auditoría.

La documentación exhaustiva de justificaciones de diseño y análisis comparativos proporciona una base sólida para futuras extensiones y optimizaciones. El sistema implementado constituye una plataforma viable para aplicaciones de movilidad urbana de mediana a gran escala, con capacidad demostrada para evolucionar conforme cambien requisitos operacionales.

# Referencias

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM, 13*(6), 377–387. https://doi.org/10.1145/362384.362685

Date, C. J. (2004). *An introduction to database systems* (8th ed.). Pearson/Addison Wesley.

Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of database systems* (7th ed.). Pearson.

García-Molina, H., Ullman, J. D., & Widom, J. (2009). *Database systems: The complete book* (2nd ed.). Pearson Prentice Hall.

MySQL. (2024). *MySQL 8.0 reference manual*. Oracle Corporation. https://dev.mysql.com/doc/refman/8.0/en/

Ramakrishnan, R., & Gehrke, J. (2003). *Database management systems* (3rd ed.). McGraw-Hill.

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database system concepts* (7th ed.). McGraw-Hill Education.