Instituto Tecnológico de Costa Rica

Administración de Tecnologías de Información

TI-3600 Bases de datos

Grupo 02

Proyecto 2 - Entrega 3

Docente

Ing Jacqueline Solís

Estudiante

Nelson Chavarría Aragón

I Semestre

Marzo, 2025

Contenido

Descripción del proyecto:

El proyecto se basará en diseñar e implementar un sistema que provea escalabilidad en la gestión de reservas de hoteles para una cadena internacional, esta misma debe de ofrecer la posibilidad de realizar diferentes operaciones, la misma debe de garantizar una alta disponibilidad, seguridad, y capacidad de crecimiento proyectado a cinco años.

Alcance del proyecto:

Este proyecto busca diseñar e implementar una plataforma centralizada de reservas para una cadena hotelera con presencia internacional, capaz de manejar múltiples propiedades de forma unificada. La solución priorizará la escalabilidad, la seguridad y la integración con sistemas existentes, garantizando operaciones fluidas incluso ante un crecimiento estimado del 10% anual. Entre los objetivos críticos destacan la tolerancia a fallos, con un RPO (pérdida máxima de datos) de 5 minutos y un RTO (tiempo de recuperación) de 30 minutos, asegurando una disponibilidad del 99.98%.

El sistema cubrirá todas las etapas del ciclo de reservas: gestión de clientes (incluyendo preferencias e historial), control de inventario de habitaciones en tiempo real, procesamiento de reservas individuales y grupales, así como integración con pasarelas de pago. Para agilizar procesos, se incorporarán automatizaciones mediante *triggers* y procedimientos almacenados, enfocados en confirmaciones, cancelaciones y políticas personalizadas.

Arquitectura y Requerimientos Técnicos

Se adoptará un diseño modular multiplataforma (Linux, Windows, macOS) con capacidad para soportar al menos 200 usuarios concurrentes. Se evaluarán dos modelos de alta disponibilidad (activo-activo o activo-pasivo), incluyendo redundancia en servidores, almacenamiento SAN/NAS y *backups* en cintas LTO. La seguridad será transversal, con:

- Cifrado de datos sensibles.
- Control de accesos por roles (RBAC).
- Auditoría detallada de operaciones críticas.

Entregables y Exclusiones

Se proporcionará documentación técnica completa, scripts SQL para despliegue y pruebas, una API funcional para demostración, y un análisis costo-beneficio de las arquitecturas propuestas. Quedan fuera del alcance:

- Interfaces de usuario final (solo se entregarán APIs para integración).
- Adquisición física de hardware (aunque se definirán especificaciones técnicas).

Este enfoque garantiza una solución robusta, adaptable y alineada con las necesidades operativas y normativas de la industria hotelera.

Roles del equipo:

Básicamente trabajaré de forma individual en este proyecto y por ende se dejarán de lado ciertos aspectos como lo son los roles de equipo.

Requerimientos:

Tabla de terminología de la matriz de requerimientos:

Columna	Descripción
Categoría	La agrupación general del requerimiento (ej. Seguridad, API, Disponibilidad).
Requerimiento	Nombre breve y claro del requerimiento.
Tipo	Clasificación del requerimiento: F Funcional / NF No funcional / T Técnica / N Negocio.
Descripción Detallada	Explicación específica del requerimiento.
Prioridad	Nivel de importancia: Crítica / Alta / Media / Baja.
Justificación	Razón por la cual el requerimiento es necesario.
Fuente	Origen del requerimiento: normativas, usuarios, equipo interno, etc.
Criterio de Aceptación	Condición para considerar que el requerimiento está cumplido.

Tabla de requerimientos de sistema

Categoría	Requerimiento	Tipo	Descripción Detallada	Prioridad	Justificación	Criterio de Aceptación
Accessibility	Cumplir WCAG 2.1 AA	NF	El sistema debe de establecer un contraste adecuado, lectores de pantalla, etc. De forma predeterminada	Baja	Accesibilidad para huéspedes discapacitados.	Testado con herramientas como axe.
API	Endpoints RESTful para CRUD de reservas	F	El sistema debe permitir crear, consultar, modificar y cancelar reservaciones vía API RESTful.	Alta	Facilita integración con canales de venta.	API documentada con Swagger y 100% cobertura Postman.
API	Soportar versionado de API (v1, v2)	T	El sistema debe de mantener compatibilidad con versiones anteriores del mismo durante 6 meses.	Media	Permite actualizaciones sin romper integraciones.	Requests a /v1/reservations funcionan post-implementación.
API	Limitar rate limiting	T	En el sistema se debe implementar throttling por API key.	Media	Previene DDoS y abuse.	Requests sobre límite reciben HTTP 429.
Auditoría	Registrar cambios en reservas	F	El sistema debe de guardar información sobre loggear usuario, timestamp y cambios en tabla de auditoría.	Media	Cumple con SOX para trazabilidad.	Bitácora contiene 100% de modificaciones.
Auditoría	Registro de accesos, consultas y modificaciones	F	El sistema debe registrar toda actividad sobre datos sensibles en la base de datos.	Crítica	Permite rastrear incidentes y auditoría.	Log de auditoría activo con revisiones periódicas
Backup	Ejecutar respaldos incrementales cada 4h	T	El sistema de forma automática debe generar backups diferenciales entre respaldos completos diarios.	Alta	Cumple RPO de 5 minutos.	Logs muestran backups exitosos cada 4h ±2 min.
Base de Datos	Implementar replicación sincrónica	T	Configurar réplica en caliente con retraso máximo de 5 segundos.	Alta	Cumple RPO de 5 minutos en fallos.	Test de failover valida sincronización.
Base de Datos	Normalizar modelo a 3FN	F	El sistema debe de implementar una forma solvente de normalización de datos.	Alta	Garantiza integridad referencial.	Diseño ER correcto.

Categoría	Requerimiento	Tipo	Descripción Detallada	Prioridad	Justificación	Criterio de Aceptación
BI	Exportar datos a Power BI	T	El sistema debe de mostrar un feed diario a data warehouse para análisis.	Baja	Habilita reporting avanzado.	ETL ejecutado diariamente sin errores.
Cache	Implementar Redis para consultas	T	El sistema debe cachear resultados de búsqueda de habitaciones por 5 min.	Media	Reduce carga en BD.	Hit rate >80% en producción.
Clientes	Registrar preferencias históricas	F	El sistema debe almacenar tipo de habitación favorita, alergias y solicitudes especiales.	Media	Personaliza servicio.	Dashboard muestra datos históricos por cliente.
Clientes	Integrar con programa de fidelización	F	El sistema debe de sincronizar puntos y beneficios con sistema de membresía existente.	Alta	Incentiva recompensas.	API consume/actualiza puntos correctamente.
Cloud	Desplegar en multi-AZ	T	El sistema puede dDistribuir carga en 2 zonas de disponibilidad.	Alta	Aumenta resiliencia ante fallos.	Simulacro de caída de AZ no afecta servicio.
Compliance	Auditoría trimestral de seguridad	NF	Debe de existir un Pen testing interno + informe de vulnerabilidades.	Media	Cumple con ISO 27001.	Reporte con findings corregidos en <7 días.
Concurrencia	Bloquear habitaciones durante checkout	F	El sistema tiene que implementar SELECT FOR UPDATE en transacciones.	Crítica	Evita doble asignación.	Test de estrés con 200 usuarios simultáneos.
Costos	Consultoría externa en seguridad	N	El sistema debe de establecer el p resupuesto para expertos externos si no se cuenta con personal calificado interno.	Alta	Asegura diseño adecuado de políticas de seguridad.	Contrato de consultoría aprobado e incluido en plan de trabajo
CRM	Marcar clientes frecuentes	F	Etiquetar usuarios con +5 estancias en últimos 12 meses.	Baja	Facilita ofertas personalizadas.	Segmentación visible en dashboard.

Categoría	Requerimiento	Tipo	Descripción Detallada	Prioridad	Justificación	Criterio de Aceptación
DevOps	Implementar CI/CD	T	Se debe de conta con Pipeline automatizado con tests unitarios/integración.	Media	Agiliza despliegues seguros.	Deploys en <15 min con rollback automático.
Disponibilidad	Garantizar 99.98% uptime anual	NF	El sistema debe de ser escalable al uso de clustering activo-activo con balanceo de carga.	Crítica	Asegura continuidad operativa en horario comercial.	Monitoreo muestra <1.75h de downtime/año.
Documentación	Generar Swagger/Open API	T	Ss debe de documentar endpoints con ejemplos request/response.	Media	Facilita integración por terceros.	95% de endpoints documentados.
Facturación	Exportar datos contables	F	El sistema debe de generar un archivo SAF-T automático para contabilidad mensual.	Baja	Obligatorio en jurisdicciones europeas.	Archivo pasa validación oficial.
Habitaciones	Gestionar estados (limpia/mantenimiento)	F	El sistema deba actualizar estados mediante interfaz housekeeping.	Alta	Optimiza asignación de habitaciones.	Reporte diario de estados coincide con físico.
Habitaciones	Aplicar tarifas dinámicas	F	El sistema debe de calcular precios según temporada, ocupación y demanda. Probablemente signifique implementar algunos trigger.	Media	Maximiza ingresos.	Reglas de pricing validadas con históricos.
Hardware	Módulos HSM opcionales	T	El sistema de ofrecer la posibilidad de uso de Hardware Security Modules para gestión de claves.	Baja	Mejora seguridad en infraestructuras críticas.	Evaluación de viabilidad completada

Categoría	Requerimiento	Tipo	Descripción Detallada	Prioridad	Justificación	Criterio de Aceptación
I18n	Soportar múltiples monedas	F	Se deben mostrar precios en USD, EUR, GBP según ubicación.	Media	Mejora experiencia internacional.	Conversión actualizada diariamente via API.
Infraestructura	Soportar 10K reservas/mes por hotel	T	Se debe escalar horizontalmente con auto-balancing.	Alta	Prepara para crecimiento.	Load testing con JMeter valida capacidad.
Integración	Conectar con CRM corporativo	T	Debe de implementarse el API REST para sincronizar datos de clientes con Salesforce.	Media	Unifica visión del cliente.	Test end-to-end valida flujo bidireccional.
Licencias	Licencias de software de cifrado	T	Se debe contar con licenciamiento de herramientas adicionales si no están incluidas en el SGBD.	Media	Garantiza cumplimiento sin depender del motor de BD.	Licencias activas y legalmente válidas
Licencias	Licencias de herramientas SIEM y auditoría	N	Se debe contar con licenciamiento adecuado para herramientas de monitoreo y auditoría.	Alta	Evita sanciones por uso no autorizado y garantiza soporte.	Herramientas licenciadas correctamente con soporte vigente
Localización	Soportar multi-idioma (EN/ES/FR)	NF	Se puede traducir interfaz y correos automáticos.	Baja	Necesidad de hoteles internacionales.	UI muestra idioma según preferencia usuario.
Logística	Alertar housekeeping al check-out	F	Notificar a sistema de limpieza al liberar habitación.	Media	Reduce tiempo entre ocupaciones.	Notificaciones push recibidas en tablets.
Logs	Centralizar registros en ELK	T	El sistema debe enviar logs a stack Elasticsearch para análisis.	Baja	Facilita troubleshooting.	Logs disponibles por 90 días.
Mobile	Notificar confirmación vía SMS	F	El sistema puede enviar SMS con código de reserva al completar booking.	Media	Reduce no-shows.	95% de SMS entregados en <2 min.
Monitoreo	Alertar SLA breaches	NF	El sistema puede notificar en Slack/Email cuando uptime <99.9%.	Alta	Permite acción correctiva inmediata.	Alertas probadas en simulacros.

Categoría	Requerimiento	Tipo	Descripción Detallada	Prioridad	Justificacion	Criterio de Aceptación
Monitoreo	Integración con SIEM	T	Debe integrarse con herramientas SIEM para alertas y detección de amenazas.	Alta	Mejora la detección temprana de incidentes.	Alertas de prueba generadas y recibidas correctamente
Onboarding	Migrar datos históricos	F	Se debe poder transferir 3 años de reservas activas desde legacy system.	Alta	Garantiza continuidad operativa.	99.9% de registros migrados sin errores.
Onboarding	Validar KYC de clientes	F	Se debe de poder escanear documento de identidad al registrar nuevo cliente.	Alta	Cumple regulaciones antifraude.	OCR detecta 99% de documentos válidos.
Pagos	Procesar transacciones PCI-compliant	F	El sistema debe integrar pasarelas de pago tokenizadas sin almacenar PAN.	Crítica	Cumple estándares de seguridad.	Certificación QSA obtenida.
Pagos	Generar facturas electrónicas	F	El sistema debe emitir CFDI 4.0 con folios autorizados por SAT.	Alta	Requerimiento legal en México.	Facturas pasan validación del PAC.
Reportes	Generar ocupación por segmento	F	El sistema debe de poder exportar CSV con ocupación por tipo de cliente (corporativo/turista).	Media	Optimiza estrategias comerciales.	Reporte generado diariamente a las 6:00 AM.
Reservaciones	Validar disponibilidad en tiempo real	F	El sistema debe verificar inventario antes de confirmar reservas.	Crítica	Evita overbooking.	Test simultáneos muestran consistencia.
Reservaciones	Soportar reservas multi-hotel	F	Se deben permitir reservas consecutivas en distintos hoteles de la cadena.	Alta	Mejora experiencia cliente.	Flujo completo documentado en diagramas UML.
Seguridad	Implementar autenticación multifactor (MFA)	NF	El sistema debe requerir MFA para accesos administrativos y operaciones críticas.	Crítica	Mitiga riesgos de acceso no autorizado.	MFA activado para todos los usuarios con permisos elevados.

Categoría	Requerimiento	Tipo	Descripción Detallada	Prioridad	Justificación	Criterio de Aceptación
Seguridad	Cifrar datos sensibles en tránsito y reposo	T	El sistema debe de emplear AES-256 para datos en reposo y TLS 1.3+ en tránsito.	Crítica	Cumple con GDPR y protege información de clientes.	Auditoría confirma cifrado activo en todas las capas.
Seguridad	Roles de usuario con mínimo privilegio	F	El sistema debe permitir definir y asignar roles con el menor acceso necesario.	Crítica	Minimiza el riesgo de accesos no autorizados.	Accesos están limitados por rol y verificados en pruebas de seguridad
Seguridad	Políticas de contraseñas y MFA	NF	El sistema debe de validar que las contraseñas sean fuertes, con expiración periódica y soporte para autenticación multifactor.	Alta	Fortalece el control de acceso y reduce riesgos de intrusión.	Contraseñas cumplen política y MFA está activa y funcional
Seguridad	Segmentación de red y firewall	T	Se debe de poder tener acceso interno segmentado, uso de DMZ para servicios expuestos, firewall activo.	Crítica	Limita superficie de ataque y controla flujos de red.	Segmentación implementada y validada en pruebas de red
Seguridad	Cifrado en reposo	T	El sistema debe de establecer un cifrado AES-256 de base de datos, respaldos y logs.	Crítica	Protege datos sensibles ante acceso físico o pérdida.	Auditoría muestra cifrado activo en almacenamiento
Seguridad	Cifrado en tránsito	T	El sistema debe de hacer uso de SSL/TLS en todas las comunicaciones con la base de datos.	Crítica	Previene ataques de interceptación de datos.	Certificados válidos y cifrado activo verificado en conexiones
Servicios	Vender paquetes (hospedaje+experiencias)	F	El sistema debe de agrupar habitación, spa y tours en un solo SKU.	Media	Aumenta venta cruzada.	Checkout muestra paquetes con descuento.
Soporte	Ofrecer helpdesk 24/7	NF	El sistema debe de ofrecer soporte telefónico y chat en 3 idiomas.	Media	Mejora experiencia cliente.	Tickets resueltos en <15 min (urgentes).
Training	Capacitar personal en sitio	F	Se debe de ofrecer la posibilidad de 8 horas de training para recepcionistas y administradores.	Alta	Asegura adopción efectiva.	Encuesta post-training con ≥4/5 satisfacción.
UI	Responsive para móviles	F	La interfaz debe de ser adaptable a smartphones (iOS/Android).	Alta	30% de reservas provienen de móviles.	Pruebas en BrowserStack sin defects.

Propuesta de selección de tecnologías

A) Diagrama de arquitectura

En esta fase, se utiliza el modelo Ansi/Sparc de 3 capas.

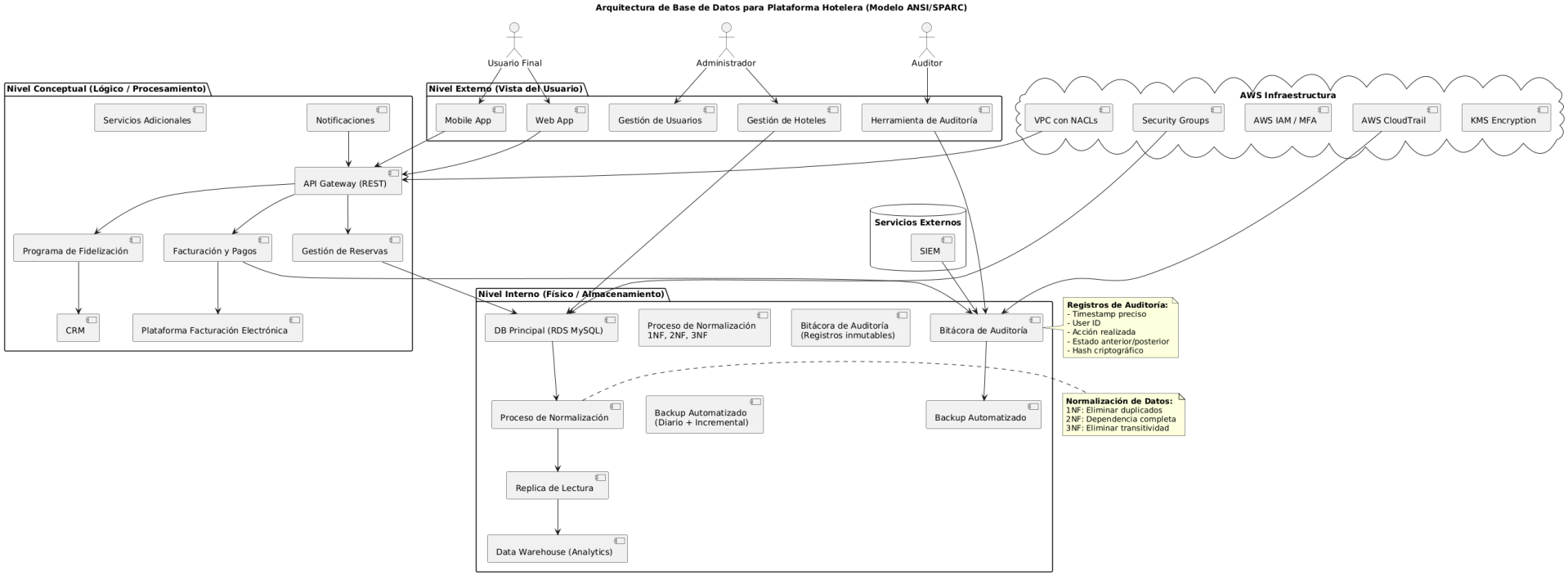


Figura de autoría propia, creada con la herramienta plantUML

B) Arquitecturas propuestas

Para la propuesta de estas arquitecturas, en aws me basaré en experiencia propia sumado a documentación oficial de *Amazon web service.*

- AWS RDS Multi-AZ

Características:

1. Replicación síncrona entre Availability Zones (AZs).
2. Failover automático (30-60 segundos) sin cambios en la conexión de la app.
3. Sin gestión de servidores: AWS maneja parches, backups y escalado.
4. Ideal para: Aplicaciones empresariales que requieren HA sin complejidad.

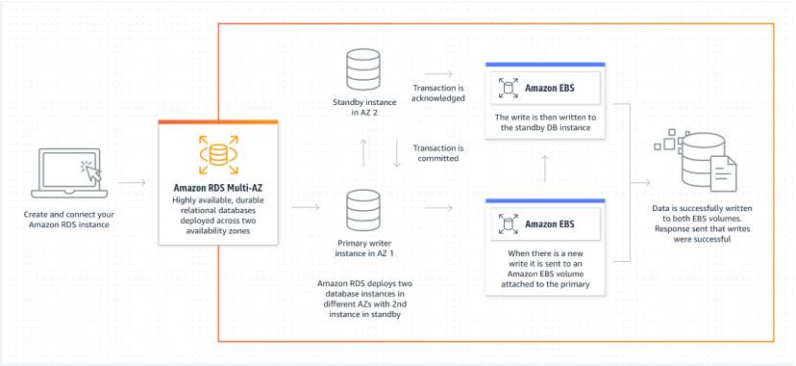
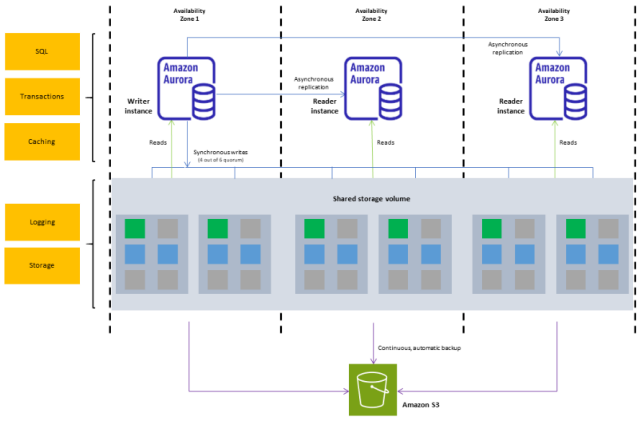


Imagen obtenida de (AWS, 2025) documentación oficial

- AWS Aurora

Características:

1. Hasta 15 réplicas de lectura con consistencia en milisegundos.
2. Almacenamiento autoescalable hasta 128 TB.
3. Failover más rápido que RDS estándar.
4. Ideal para: Cargas de trabajo intensivas en lecturas.



Documentación oficial (AWS, 2025)

C) Estimación de costes de implementación

Para la primera propuesta en AWS, RDS multi A-Z, la estimación de costes se basará en las siguientes suposiciones, se tomará la región de OHIO como base, se tomará en cuenta que la base de datos será de normal-alto consumo y que se partirá de una instancia en large dado a que el proyecto es medio-grande, añadido a lo anterior, el coste mensual es de 372 dólares por mes.

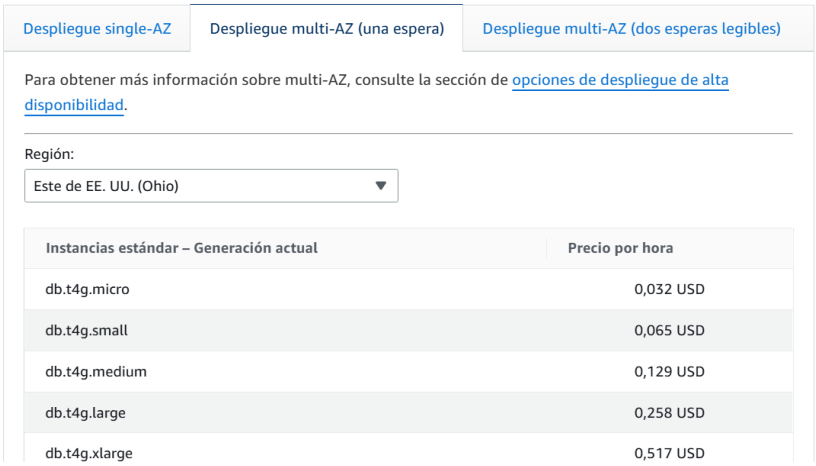
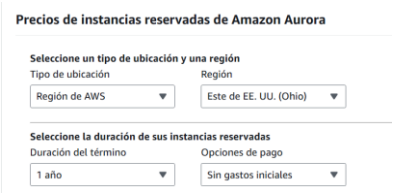


Imagen obtenida de la documentación oficial (AWS, 2025)

Para la segunda propuesta, me basaré en AWS Aurora, utilizaré la calculadora de costes que ofrece AWS de la siguiente forma:



Partiendo de que se buscan los costes de un negocio promedio, el coste mensual es de 352 dólares. Se utilizó el db.x2g.xlarge.

Nombre de la instancia	Cuota inicial de IR	Cuotas mensuales de IR*	Cuota por hora efectiva de IR**	comparación con modalidad bajo demanda	Tarifa bajo demanda
db.t4g.medium	0 USD	42,12 USD	0,058 USD	21 %	0,0730 US
db.t4g.large	0 USD	84,17 USD	0,115 USD	21 %	0,1460 US
db.t3.small	0 USD	24,82 USD	0,034 USD	17 %	0,0410 US
db.t3.medium	0 USD	47,45 USD	0,065 USD	21 %	0,0820 US
db.t3.large	0 USD	99,28 USD	0,136 USD	17 %	0,1640 US
db.x2g.large	0 USD	178,78 USD	0,245 USD	35 %	0,3770 US
db.x2g.xlarge	0 USD	357,55 USD	0,490 USD	35 %	0,7540 US

Imagen obtenida de la documentación oficial (AWS, 2025)

Comparativa de arquitecturas

1. Propuesta 1: AWS RDS (PostgreSQL/MySQL) - Instancia db.xlarge (372 USD/mes)

Arquitectura Propuesta:

- Capa de Presentación: Aplicación web/móvil hospedada en AWS Elastic Beanstalk o EC2 Auto Scaling.
- Capa de Lógica de Negocio: Microservicios en AWS Lambda o ECS (Docker) con API Gateway.
- Capa de Base de Datos: AWS RDS (PostgreSQL/MySQL) en configuración Multi-AZ (alta disponibilidad).
- Redes: Conexiones seguras mediante AWS Site-to-Site VPN o AWS Direct Connect.
- Almacenamiento: EBS gp3 con snapshots automatizados.
- Backup: AWS Backup + Amazon S3 Glacier para retención a largo plazo.
- Monitoreo: Amazon CloudWatch + AWS Config para auditoría.

Fortalezas:

Alta Disponibilidad Automatizada:

- RDS Multi-AZ garantiza failover automático (< 2 min de downtime).
- Réplica sincrónica en otra AZ.

Escalabilidad Horizontal:

- Lecturas escalables con réplicas de lectura (hasta 5).
- Escalado vertical sin downtime (cambio de instancia).

Seguridad Integrada:

- Cifrado en tránsito (SSL/TLS) y en reposo (KMS).
- IAM para gestión de accesos.

Soporte Multiplataforma:

- Compatibilidad con aplicaciones en Linux, Windows y macOS.
- Cumplimiento Normativo:
- Certificaciones SOC, ISO, PCI DSS, GDPR.

Debilidades:

Costo Operativo Elevado:

- El precio aumenta con réplicas y almacenamiento adicional.

Limitaciones en Customización:

- No se puede modificar el kernel de la base de datos.

Dependencia de AWS:

- Lock-in tecnológico.

2. Propuesta 2: Amazon Aurora PostgreSQL/Mysql - Instancia db.x2g.xlarge (352 USD/mes)

Arquitectura Propuesta:

- Capa de Presentación: AWS AppSync (GraphQL) + Amplify para frontend.
- Capa de Lógica de Negocio: AWS Fargate (ECS sin servidor) con balanceo de carga.
- Capa de Base de Datos: Aurora PostgreSQL con almacenamiento distribuido (6 copias en 3 AZs).
- Redes: AWS PrivateLink para conexiones internas seguras.
- Almacenamiento: Aurora Storage Auto-Scaling (hasta 128 TB).
- Backup: Puntos de recuperación continuos (Continuous Backup) + Cross-Region Replication.
- Monitoreo: Amazon RDS Performance Insights + AWS GuardDuty.

Fortalezas:

- Alta Disponibilidad Superior, Aurora replica datos en 3 AZs (RPO ≈ 0, RTO < 1 min). Posee un clúster Aurora con hasta 15 réplicas de lectura.
- Rendimiento Optimizado, perfectamente hasta 5x mejor rendimiento que RDS PostgreSQL. Con un escalado automático de almacenamiento.
- Respaldo Continuo: Puntos de recuperación con granularidad de segundos.
- Reducción de Costos: Menor costo por IOPS (no se paga por operaciones de disco).
- Compatibilidad con PostgreSQL/MySQL: Migración sencilla desde bases existentes.

Debilidades:

- Costo Inicial de Migración, esto parte de que se requieren ajustes en *queries* para aprovechar Aurora.
- Limitaciones en Customización, pueden haber problemas de escalabilidad dado a que no se soporta todas las extensiones de PostgreSQL.
- Dependencia de AWS, menos portabilidad que soluciones on-premise. Aunque una total cohesión en el proyecto.

A continuación, se muestra una tabla comparando el desempeño a alto nivel ofrecido por las tecnologías elegidas, he aquí una tabla de terminología:

Tabla de terminología de la tabla comparativa de desempeño de arquitecturas	
Columnas	**Justificación**
Requisitos/Requerimientos	Define el requerimiento o la información a lo que apela la fila.
AWS RDS	Define como el requisito se aplica en base a la tecnología de AWS RDS.
Amazon Aurora	Define como el requisito se aplica en base a la tecnología de Amazon Aurora

Tabla comparativa de desempeño de arquitectura		
Resquisistos	**AWS RDS**	**Amazon Aurora**
Alta Disponibilidad (RTO ≤ 30 min, RPO ≤ 5 min)	(Multi-AZ)	(Clúster Multi-AZ)
Soporte 200+ usuarios concurrentes	(Depende de instancia)	(Mejor rendimiento)
Escalado Horizontal	(Réplicas de lectura)	(Hasta 15 réplicas)
Redundancia de Hardware	(Multi-AZ)	(6 copias en 3 AZs)
Respaldo Automatizado	(Snapshots)	(Backup continuo)
Soporte Multiplataforma		
Conexiones Seguras (VPN/VPC)		
Recuperación ante borrado accidental	(PITR)	(PITR con segundos)

Estimación de costos totales de proyectos

Para realizar una correcta estimación de los costes mensuales de tecnología de base de datos en el primer año en las diferentes arquitecturas, se plantea una tabla comparativa de las arquitecturas, he aquí de la tabla de terminología de estas.

Tabla de terminología de la tabla de comparativa de costes por arquitectura utilizada	
Termino	**Explicación**
Componente	Define los servicios que poseen un coste asociado en el sistema.
AWS RDS (PostgreSQL/MySQL)	Detalla el coste mensual del servicio especificado al mes en AWS RDS.
Amazon Aurora (Postgresql/MySQL)	Detalla el coste mensual del servicio especificado al mes en Amazon Aurora.

Tabla comparativa de costes por arquitectura utilizada			
Componente	**AWS RDS (PostgreSQL/MySQL)**	**Amazon Aurora PostgreSQL**	**Notas**
Costo Base (Instancia Principal)	$372/mes (db.xlarge)	$352/mes (db.x2g.xlarge)	Precio por nodo primario (Multi-AZ incluido).
Réplicas de Lectura	$372/mes cada una (hasta 5)	$352/mes cada una (hasta 15)	Aurora permite más réplicas sin penalización de IOPS.
Almacenamiento (GB/mes)	$0.115/GB (gp3)	$0.10/GB (Aurora Auto-Scaling)	Aurora optimiza almacenamiento sin costos adicionales por IOPS.
Backup Automatizado	$0.095/GB-mes (Snapshots)	$0.021/GB-mes (Backup Continuo)	Aurora incluye backups granulares sin costo adicional.
Conexión VPN (Site-to-Site)	0.05/hora+0.05/*hora*+0.02/GB	0.05/hora+0.05/*hora*+0.02/GB	Igual en ambas opciones.
Monitoreo (CloudWatch)	$30/mes (métricas básicas)	$30/mes (métricas básicas)	Costo similar.
Licencias de HA	Incluido en RDS	Incluido en Aurora	No requiere licencias adicionales.
Costo Estimado (1 Año)	6,500−6,500−10,000	6,000−6,000−9,000	Depende de réplicas y almacenamiento.

Por otro lado, también se detallan los diferentes costes adicionales no tan importantes pero que si pueden dar un beneficio relativamente importante al proyecto. Se detalla el nombre del servicio, el coste, a que arquitectura se aplica y comentarios breves sobre la importancia en el proyecto.

Tabla de posibles costes adicionales			
Servicio Adicional	**Costo (USD/mes)**	**Aplicable a**	**Notas**
AWS Direct Connect	300−300−1,500+	Ambas	Conexión dedicada (reduce latencia).
AWS Backup (Políticas Avanzadas)	5−5−50	Ambas	Gestión centralizada.
Amazon GuardDuty (Seguridad)	10−10−500	Ambas	Detección de amenazas.
Personal Especializado (DBA/Arquitecto)	3,000−3,000−8,000	Ambas	Salarios externos (si se contrata).

Costos anuales esperados en el proyecto

En la esta sección se establecerá una cantidad de costes de la infraestructura utilizada y los costes de personal del proyecto a lo largo del primer año de desarrollo del proyecto. A continuación, se detallan los costes asociados al personal del proyecto, se detallará en una tabla que se basará en la siguiente tabla de terminología.

Término	Explicación
Rol	Rango del personal que participará en la elaboración del proyecto.
Salario Mensual	Se calcula a partir de los salarios promedios de profesionales TI.
Salario Anual	Se multiplica el mensual por 12.
Notas	Importancia en el proyecto, se omite el detallado completo de responsabilidades del personal.

Costo personal primer año			
Rol	**Salario Mensual (USD)**	**Salario Anual (USD)**	**Notas**
Arquitecto de BD	7,000−9,000	84,000−108,000	Diseño HA, replicación, optimización.
Administrador de BD (DBA)	5,000−7,000	60,000−84,000	Mantenimiento, tuning, respaldos.
Ingeniero de Infraestructura	6,500−8,500	78,000−102,000	AWS, redes, redundancia.
Especialista en Backup	4,500−6,000	54,000−72,000	Veeam, cintas LTO, recuperación.
Especialista en Seguridad	6,000−8,000	72,000−96,000	CISSP, cifrado, IAM.
Coordinador de Proyecto	5,500−7,500	66,000−90,000	PMP, Scrum, gestión de equipo.
Auditor de TI (Opcional)	6,000−8,000	72,000−96,000	GDPR, PCI DSS, normativas.
Total Personal (Anual)	40,500-54,000	486,000−648,000	

En la siguiente parte se detallarán específicamente los costes relacionados con la infraestructura tecnológica que se utilizará en el presente proyecto durante un año, a continuación la tabla de terminología.

Tabla de terminología de la tabla de costo Anual de infraestructura tecnológica planteada	
Termino	**Explicación**
Componente	Define los servicios que poseen un coste asociado en el sistema.
AWS RDS (PostgreSQL/MySQL)	Detalla el coste anual del servicio especificado al mes en AWS RDS.
Amazon Aurora (Postgresql/MySQL)	Detalla el coste anual del servicio especificado al mes en Amazon Aurora.
Notas	Se da información relacionada a cada componente.

Costo Anual de infraestructura tecnológica planteada			
Componente	**AWS RDS (PostgreSQL/MySQL)**	**Amazon Aurora PostgreSQL**	**Notas**
Instancia Principal + Multi-AZ	$4,464/año (db.xlarge)	$4,224/año (db.x2g.xlarge)	Alta disponibilidad incluida.
Réplicas de Lectura (2)	$8,928/año (2 x db.xlarge)	$8,448/año (2 x db.x2g.xlarge)	Escalabilidad para lecturas.
Almacenamiento (500 GB)	$690/año (gp3)	$600/año (Aurora)	Aurora optimiza costos.
Backup Automatizado	$114/año (100 GB)	Incluido	Aurora tiene backup continuo sin costo extra.
VPN (Site-to-Site)	$438/año	$438/año	Conexión segura entre sedes.
Monitoreo (CloudWatch)	$360/año	$360/año	Alertas y métricas.
Total, Infraestructura (Anual)	$14,994/año	$14,070/año	Aurora es $924 más económico.

A continuación, se detallan algunos costes adicionales a la infraestructura tecnológica y que se podría incurrir en ellos si es del gusto de los gestores de costo, de detallan nombre, precio e importancia de cada uno.

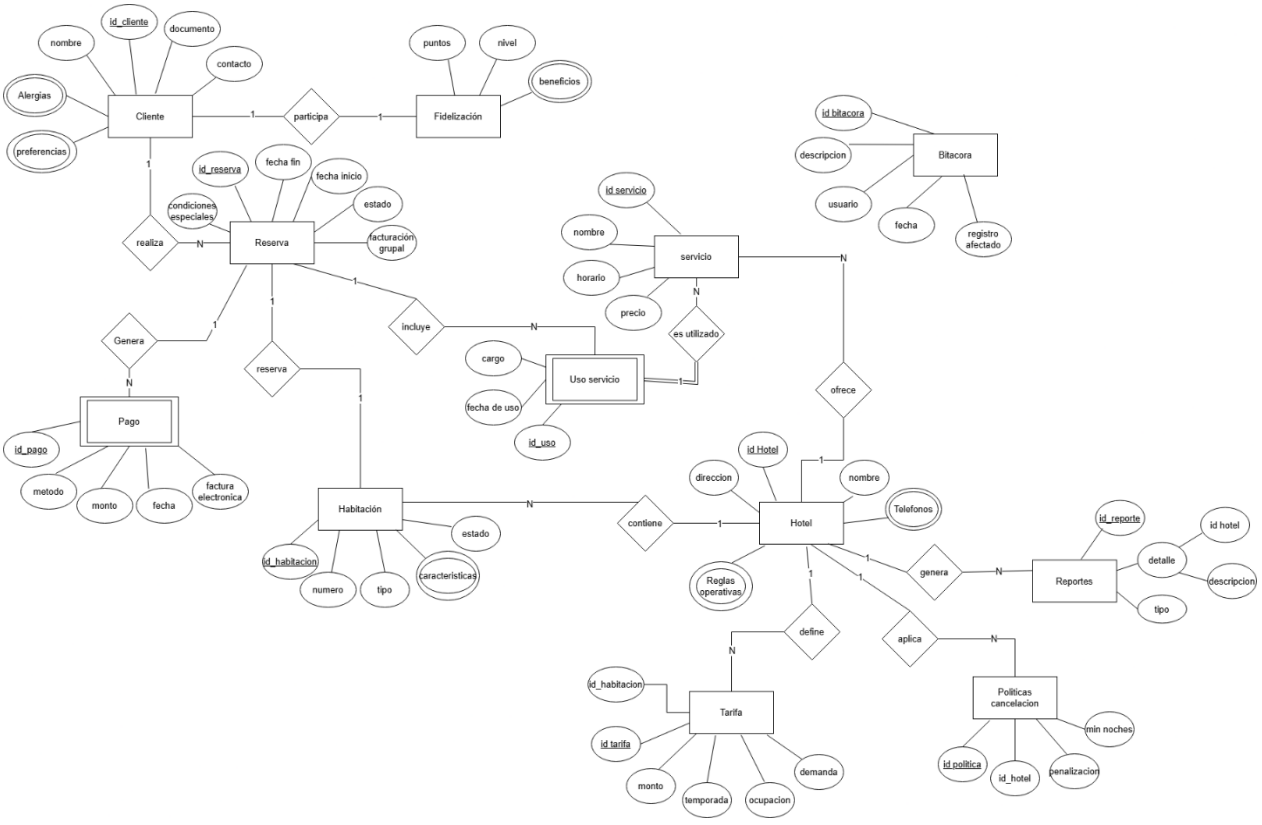
Costes adicionales de infraestructura tecnológica		
Concepto	**Costo (USD)**	**Notas**
Unidad de Cinta LTO-8	3,000−5,000	Drive de respaldo físico.
Cintas LTO-8 (10 unidades)	1,000−2,000	Rotación y almacenamiento offsite.
Software de Backup (Veeam)	$1,500/año	Licencia empresarial.
Servidores Locales (Opcional)	10,000−20,000	Si se requiere hibridación.
UPS/Generador (Opcional)	5,000−15,000	Para infraestructura on-premise.
Total Adicional (Inicial)	10,500−42,000	Depende de necesidades.

Costos totales en el primer año

A continuación, se detallan los costes anuales del proyecto, en dependencia de la tecnología utilizada.

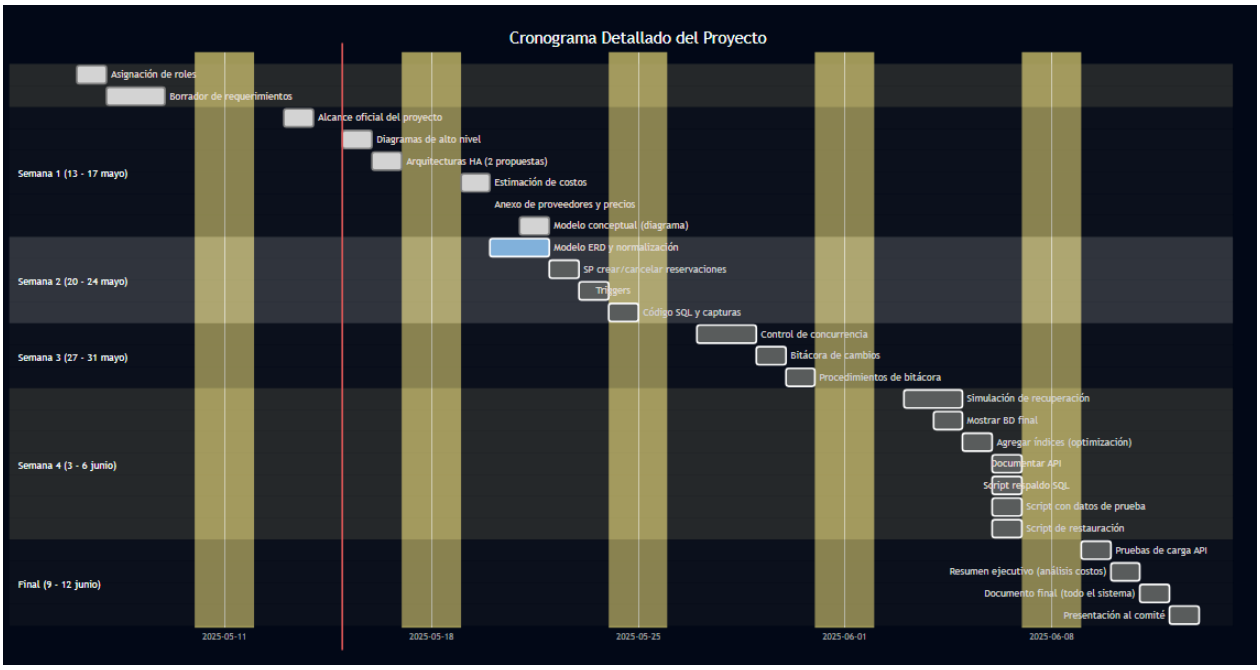
Costos totales del primer año		
Arquitectura	AWS RDS	Amazon Aurora
Personal (Promedio)	$567,000	$567,000
Infraestructura (AWS)	$14,994	$14,070
Hardware/Software Extra	$20,000*	$20,000*
Total Estimado (1er Año)	$601,994	$601,070

Diagrama ER



Cronograma con diagrama de Gantt

He de aclarar que al yo realizar el proyecto de manera individual, pues todo lo efectuaré yo.



Implementación de la base de datos

Capturas de la creación de las tablas

Tabla de clientes

proyecto_2.sql | bases_datos_p2.sql

D: > Personal > tec > universidad > 2025_Semestre_1 > Base de datos > Proyectos > proyecto 2 > bases_datos_p2.sql

```sql
-- Tabla de Hoteles
CREATE TABLE hoteles (
    hotel_id SERIAL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    direccion TEXT NOT NULL,
    ciudad VARCHAR(50) NOT NULL,
    pais VARCHAR(50) NOT NULL,
    telefono VARCHAR(20) NOT NULL,
    email VARCHAR(100) NOT NULL,
    estrellas INTEGER CHECK (estrellas BETWEEN 1 AND 5),
    activo BOOLEAN DEFAULT TRUE,
    fecha_apertura DATE,
    descripcion TEXT
);
```

Tabla de clientes

```sql
CREATE TABLE clientes (
    cliente_id SERIAL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    documento_identidad VARCHAR(20) NOT NULL,
    tipo_documento VARCHAR(20) NOT NULL,
    nacionalidad VARCHAR(50) NOT NULL,
    telefono VARCHAR(20) NOT NULL,
    email VARCHAR(100) NOT NULL,
    fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    activo BOOLEAN DEFAULT TRUE,
    preferencias TEXT,
    alergias TEXT,
    CONSTRAINT unique_document UNIQUE (tipo_documento, documento_identidad),
    CONSTRAINT unique_email UNIQUE (email)
);
```

Tabla de fidelización de clientes

```sql
CREATE TABLE fidelizacion_clientes (
    fidelizacion_id SERIAL PRIMARY KEY,
    cliente_id INTEGER REFERENCES clientes(cliente_id),
    hotel_id INTEGER REFERENCES hoteles(hotel_id),
    puntos_acumulados INTEGER DEFAULT 0,
    nivel_membresia VARCHAR(20),
    beneficios TEXT,
    fecha_actualizacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT unique_cliente_hotel UNIQUE (cliente_id, hotel_id)
);
```

Tabla de tipos de habitación

```sql
CREATE TABLE tipos_habitacion (
    tipo_id SERIAL PRIMARY KEY,
    hotel_id INTEGER REFERENCES hoteles(hotel_id),
    nombre VARCHAR(50) NOT NULL,
    descripcion TEXT,
    capacidad INTEGER NOT NULL,
    tamano INTEGER, -- en m²
    comodidades TEXT,
    precio_base DECIMAL(10, 2) NOT NULL,
    CONSTRAINT unique_tipo_hotel UNIQUE (hotel_id, nombre)
);
```

Tabla de las habitaciones

```sql
CREATE TABLE habitaciones (
    habitacion_id SERIAL PRIMARY KEY,
    hotel_id INTEGER REFERENCES hoteles(hotel_id),
    numero VARCHAR(10) NOT NULL,
    tipo_id INTEGER REFERENCES tipos_habitacion(tipo_id),
    piso INTEGER NOT NULL,
    caracteristicas_especiales TEXT,
    estado VARCHAR(20) NOT NULL CHECK (estado IN ('disponible', 'ocupada', 'mantenimiento', 'limpieza')),
    notas TEXT,
    CONSTRAINT unique_numero_hotel UNIQUE (hotel_id, numero)
);
```

Tablas de las políticas y tarifas por temporada

```sql
-- Tabla de Políticas de Temporada (por hotel)
CREATE TABLE politicas_temporada (
    politica_id SERIAL PRIMARY KEY,
    hotel_id INTEGER REFERENCES hoteles(hotel_id),
    nombre VARCHAR(100) NOT NULL,
    fecha_inicio DATE NOT NULL,
    fecha_fin DATE NOT NULL,
    descripcion TEXT,
    reglas TEXT,
    CONSTRAINT fechas_validas CHECK (fecha_fin > fecha_inicio),
    CONSTRAINT unique_nombre_hotel UNIQUE (hotel_id, nombre)
);

-- Tabla de Tarifas por Temporada
CREATE TABLE tarifas_temporada (
    tarifa_id SERIAL PRIMARY KEY,
    politica_id INTEGER REFERENCES politicas_temporada(politica_id),
    tipo_id INTEGER REFERENCES tipos_habitacion(tipo_id),
    precio DECIMAL(10, 2) NOT NULL,
    descripcion TEXT,
    CONSTRAINT unique_politica_tipo UNIQUE (politica_id, tipo_id)
);
```

Tabla de reservaciones

```sql
-- Tabla de Reservaciones
CREATE TABLE reservaciones (
    reservacion_id SERIAL PRIMARY KEY,
    hotel_id INTEGER REFERENCES hoteles(hotel_id),
    cliente_id INTEGER REFERENCES clientes(cliente_id),
    fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    fecha_entrada DATE NOT NULL,
    fecha_salida DATE NOT NULL,
    adultos INTEGER NOT NULL,
    ninos INTEGER DEFAULT 0,
    estado VARCHAR(20) NOT NULL CHECK (estado IN ('pendiente', 'confirmada', 'cancelada', 'completada'
    tipo_reserva VARCHAR(20) NOT NULL CHECK (tipo_reserva IN ('individual', 'grupo', 'corporativa')),
    solicitudes_especiales TEXT,
    codigo_reserva VARCHAR(20) UNIQUE,
    CONSTRAINT fechas_validas CHECK (fecha_salida > fecha_entrada)
);
```

Tabla de reservación de habitaciones, tabla de detalle

```
-- Tabla de Detalles de Reservación (Habitaciones asignadas)
CREATE TABLE reservacion_habitaciones (
    detalle_id SERIAL PRIMARY KEY,
    reservacion_id INTEGER REFERENCES reservaciones(reservacion_id),
    habitacion_id INTEGER REFERENCES habitaciones(habitacion_id),
    tarifa_aplicada DECIMAL(10, 2) NOT NULL,
    notas TEXT,
    CONSTRAINT unique_reservacion_habitacion UNIQUE (reservacion_id, habitacion_id)
);
```

Tablas de servicios contratados

```
-- Tabla de Detalles de Servicios en Reserva
CREATE TABLE reservacion_servicios (
    detalle_servicio_id SERIAL PRIMARY KEY,
    reservacion_id INTEGER REFERENCES reservaciones(reservacion_id),
    servicio_id INTEGER, -- Referencia flexible (puede ser a servicios_hotel o servicios_externos)
    tipo_servicio VARCHAR(50) NOT NULL, -- 'hotel' o 'externo'
    descripcion VARCHAR(100) NOT NULL,
    fecha_servicio TIMESTAMP NOT NULL,
    cantidad INTEGER DEFAULT 1,
    precio_unitario DECIMAL(10, 2) NOT NULL,
    notas TEXT,
    estado VARCHAR(20) DEFAULT 'pendiente' CHECK (estado IN ('pendiente', 'completado', 'cancelado'))
);
```

Tabla de servicios brindados de hotel o externos

```
-- Tabla de Servicios del Hotel
CREATE TABLE servicios_hotel (
    servicio_id SERIAL PRIMARY KEY,
    hotel_id INTEGER REFERENCES hoteles(hotel_id),
    nombre VARCHAR(100) NOT NULL,
    descripcion TEXT,
    precio_base DECIMAL(10, 2) NOT NULL,
    categoria VARCHAR(50) NOT NULL,
    horario_disponibilidad TEXT,
    activo BOOLEAN DEFAULT TRUE,
    CONSTRAINT unique_servicio_hotel UNIQUE (hotel_id, nombre)
);
```

Tabla de pagos

```
-- Tabla de Pagos
CREATE TABLE pagos (
    pago_id SERIAL PRIMARY KEY,
    reservacion_id INTEGER REFERENCES reservaciones(reservacion_id),
    monto DECIMAL(10, 2) NOT NULL,
    metodo_pago VARCHAR(50) NOT NULL,
    fecha_pago TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    estado VARCHAR(20) NOT NULL CHECK (estado IN ('pendiente', 'completado', 'reembolsado', 'fallido')),
    referencia VARCHAR(100),
    descripcion TEXT
);
```

Tabla de facturas

```sql
-- Tabla de Facturas
CREATE TABLE facturas (
    factura_id SERIAL PRIMARY KEY,
    pago_id INTEGER REFERENCES pagos(pago_id),
    hotel_id INTEGER REFERENCES hoteles(hotel_id),
    numero_factura VARCHAR(50) NOT NULL UNIQUE,
    fecha_emision DATE NOT NULL,
    subtotal DECIMAL(10, 2) NOT NULL,
    impuestos DECIMAL(10, 2) NOT NULL,
    total DECIMAL(10, 2) NOT NULL,
    datos_cliente TEXT NOT NULL,
    detalles TEXT NOT NULL
);
```

Tabla de registro de estadías

```sql
-- Tabla de Histórico de Estadías
CREATE TABLE historico_estadias (
    historico_id SERIAL PRIMARY KEY,
    hotel_id INTEGER REFERENCES hoteles(hotel_id),
    cliente_id INTEGER REFERENCES clientes(cliente_id),
    reservacion_id INTEGER REFERENCES reservaciones(reservacion_id),
    fecha_entrada DATE NOT NULL,
    fecha_salida DATE NOT NULL,
    habitacion_id INTEGER REFERENCES habitaciones(habitacion_id),
    comentarios TEXT,
    calificacion INTEGER CHECK (calificacion BETWEEN 1 AND 5),
    preferencias_registradas TEXT
);
```

Tabla de bitácoras de reservaciones

```sql
CREATE TABLE bitacora_reservaciones (
    bitacora_id SERIAL PRIMARY KEY,
    fecha_hora TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    usuario VARCHAR(50),
    accion VARCHAR(20) NOT NULL,
    tabla_afectada VARCHAR(30) NOT NULL,
    registro_id INTEGER NOT NULL,
    detalles TEXT,
    ip_origen VARCHAR(50)
);
```

Capturas de las funciones, triggers y procedimientos

Función para registrar en bitácora

```sql
-- Función para registrar en bitácora
CREATE OR REPLACE FUNCTION registrar_bitacora(
    p_usuario VARCHAR,
    p_accion VARCHAR,
    p_tabla VARCHAR,
    p_registro_id INTEGER,
    p_detalles TEXT,
    p_ip VARCHAR
) RETURNS VOID AS $$
BEGIN
    INSERT INTO bitacora_reservaciones (usuario, accion, tabla_afectada, registro_id, detalles, ip_origen)
    VALUES (p_usuario, p_accion, p_tabla, p_registro_id, p_detalles, p_ip);
END;
$$ LANGUAGE plpgsql;
```

Trigger para cancelar reservación

```sql
CREATE OR REPLACE FUNCTION cancelacion_reservacion_trigger()
RETURNS TRIGGER AS $$
DECLARE
    v_dias_restantes INTEGER;
    v_hotel_nombre VARCHAR(100);
    v_cliente_nombre VARCHAR(100);
    v_habitaciones TEXT;
BEGIN
    -- Solo actuar cuando el estado cambia a 'cancelada'
    IF NEW.estado = 'cancelada' AND OLD.estado ≠ 'cancelada' THEN
        -- Calcular días restantes hasta la fecha de entrada
        v_dias_restantes := NEW.fecha_entrada - CURRENT_DATE;

        -- Registrar en bitácora
        PERFORM registrar_bitacora(
            CURRENT_USER,
            'CANCELACION',
            'reservaciones',
            NEW.reservacion_id,
            'Reserva cancelada. Días restantes: ' || v_dias_restantes,
            inet_client_addr()::TEXT
        );

        -- Enviar alerta si es cancelación de última hora (menos de 3 días)
        IF v_dias_restantes < 3 THEN
            -- Obtener información para la alerta
```

```sql
        -- Enviar alerta si es cancelación de última hora (menos de 3 días)
        IF v_dias_restantes < 3 THEN
            -- Obtener información para la alerta
            SELECT nombre INTO v_hotel_nombre FROM hoteles WHERE hotel_id = NEW.hotel_id;
            SELECT nombre INTO v_cliente_nombre FROM clientes WHERE cliente_id = NEW.cliente_id;

            -- Registrar alerta en bitácora
            PERFORM registrar_bitacora(
                CURRENT_USER,
                'ALERTA',
                'reservaciones',
                NEW.reservacion_id,
                'ALERTA: Cancelación de última hora. Cliente: ' || v_cliente_nombre || ', Hotel: ' || v_hotel_nc
                inet_client_addr()::TEXT
            );

            -- Aquí podrías agregar lógica para enviar email/notificación
            RAISE NOTICE 'ALERTA: Cancelación de última hora. Reservación ID: %, Cliente: %, Hotel: %',
                NEW.reservacion_id, v_cliente_nombre, v_hotel_nombre;
        END IF;
```

```
        -- Liberar habitaciones asociadas
        UPDATE habitaciones h
        SET estado = 'disponible'
        FROM reservacion_habitaciones rh
        WHERE rh.habitacion_id = h.habitacion_id
        AND rh.reservacion_id = NEW.reservacion_id;

        -- Registrar liberación en bitácora
        PERFORM registrar_bitacora(
            CURRENT_USER,
            'ACTUALIZACION',
            'habitaciones',
            NEW.reservacion_id,
            'Habitaciones liberadas por cancelación de reserva',
            inet_client_addr()::TEXT
        );
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
286  CREATE TRIGGER tr_cancelacion_reservacion
287  AFTER UPDATE ON reservaciones
288  FOR EACH ROW
289  EXECUTE FUNCTION cancelacion_reservacion_trigger();
```

Trigger para registrar pago

```
292  CREATE OR REPLACE FUNCTION pago_registrado_trigger()
293  RETURNS TRIGGER AS $$
294  BEGIN
295      -- Solo actuar cuando el pago se marca como completado
296      IF NEW.estado = 'completado' AND OLD.estado ≠ 'completado' THEN
297          -- Actualizar estado de la reservación a 'confirmada'
298          UPDATE reservaciones
299          SET estado = 'confirmada'
300          WHERE reservacion_id = NEW.reservacion_id;
301
302          -- Registrar en bitácora
303          PERFORM registrar_bitacora(
304              CURRENT_USER,
305              'CONFIRMACION',
306              'reservaciones',
307              NEW.reservacion_id,
308              'Reserva confirmada por pago completado. Pago ID: ' || NEW.pago_id,
309              inet_client_addr()::TEXT
310          );
311      END IF;
312
313      RETURN NEW;
314  END;
315  $$ LANGUAGE plpgsql;
```

```
317   -- Asociar trigger a la tabla pagos
318   CREATE TRIGGER tr_pago_registrado
319   AFTER UPDATE ON pagos
320   FOR EACH ROW
321   EXECUTE FUNCTION pago_registrado_trigger();
322
```

Procedimiento para verificar disponibilidad de habitaciones

```
CREATE OR REPLACE FUNCTION verificar_disponibilidad(
    p_hotel_id INTEGER,
    p_tipo_id INTEGER,
    p_fecha_entrada DATE,
    p_fecha_salida DATE
) RETURNS TABLE (
    habitacion_id INTEGER,
    numero VARCHAR,
    piso INTEGER,
    precio_recomendado DECIMAL(10, 2)
) AS $$
BEGIN
    RETURN QUERY
    SELECT h.habitacion_id, h.numero, h.piso,
        COALESCE(
            (SELECT t.precio
             FROM tarifas_temporada t
             JOIN politicas_temporada p ON t.politica_id = p.politica_id
             WHERE t.tipo_id = h.tipo_id
             AND p.fecha_inicio ⩽ p_fecha_entrada
             AND p.fecha_fin ⩾ p_fecha_salida
             LIMIT 1),
            th.precio_base
        ) AS precio_recomendado
    FROM habitaciones h
    JOIN tipos_habitacion th ON h.tipo_id = th.tipo_id
    WHERE h.hotel_id = p_hotel_id
```

```
    JOIN tipos_habitacion th ON h.tipo_id = th.tipo_id
    WHERE h.hotel_id = p_hotel_id
    AND h.tipo_id = p_tipo_id
    AND h.estado = 'disponible'
    AND NOT EXISTS (
        SELECT 1 FROM reservacion_habitaciones rh
        JOIN reservaciones r ON rh.reservacion_id = r.reservacion_id
        WHERE rh.habitacion_id = h.habitacion_id
        AND r.estado NOT IN ('cancelada', 'no-show')
        AND (
            (r.fecha_entrada ⩽ p_fecha_entrada AND r.fecha_salida > p_fecha_entrada) OR
            (r.fecha_entrada < p_fecha_salida AND r.fecha_salida ⩾ p_fecha_salida) OR
            (r.fecha_entrada ⩾ p_fecha_entrada AND r.fecha_salida ⩽ p_fecha_salida)
        )
    );
END;
$$ LANGUAGE plpgsql;
```

Procedimiento para crear reservación

```
CREATE OR REPLACE FUNCTION crear_reservacion(
    p_hotel_id INTEGER,
    p_cliente_id INTEGER,
    p_fecha_entrada DATE,
    p_fecha_salida DATE,
    p_adultos INTEGER,
    p_ninos INTEGER DEFAULT 0,
    p_tipo_reserva VARCHAR(20) DEFAULT 'individual',
    p_solicitudes_especiales TEXT DEFAULT NULL,
    p_usuario VARCHAR(50) DEFAULT CURRENT_USER,
    p_ip_origen VARCHAR(50) DEFAULT inet_client_addr()::TEXT,
    p_tipos_habitaciones JSONB DEFAULT '[{"tipo_id": 1, "cantidad": 1}]'::JSONB
) RETURNS INTEGER AS $$
DECLARE
    v_reservacion_id INTEGER;
    v_codigo_reserva VARCHAR(20);
    v_item JSONB;
    v_habitacion RECORD;
    v_disponibles INTEGER;
    v_tarifa DECIMAL(10, 2);
BEGIN
    -- Validaciones iniciales
    IF p_fecha_salida ⩽ p_fecha_entrada THEN
        RAISE EXCEPTION 'La fecha de salida debe ser posterior a la fecha de entrada';
    END IF;
```

```
    -- Verificar disponibilidad para cada tipo de habitación solicitada
    FOR v_item IN SELECT * FROM jsonb_array_elements(p_tipos_habitaciones) LOOP
        -- Verificar que exista el tipo de habitación en el hotel
        PERFORM 1 FROM tipos_habitacion
        WHERE tipo_id = (v_item->>'tipo_id')::INTEGER AND hotel_id = p_hotel_id;

        IF NOT FOUND THEN
            RAISE EXCEPTION 'El tipo de habitación % no existe en este hotel', (v_item->>'tipo_id');
        END IF;

        -- Contar habitaciones disponibles
        SELECT COUNT(*) INTO v_disponibles
        FROM verificar_disponibilidad(
            p_hotel_id,
            (v_item->>'tipo_id')::INTEGER,
            p_fecha_entrada,
            p_fecha_salida
        );

        IF v_disponibles < (v_item->>'cantidad')::INTEGER THEN
            RAISE EXCEPTION 'No hay suficientes habitaciones disponibles del tipo %', (v_item->>'tipo_id');
        END IF;
    END LOOP;
```

```
    -- Generar código de reserva único
    v_codigo_reserva := 'RES-' || p_hotel_id || '-' ||
                        EXTRACT(YEAR FROM CURRENT_DATE) || '-' ||
                        LPAD(FLOOR(RANDOM() * 10000)::TEXT, 4, '0');

    -- Crear la reservación principal
    INSERT INTO reservaciones (
        hotel_id,
        cliente_id,
        fecha_entrada,
        fecha_salida,
        adultos,
        ninos,
        estado,
        tipo_reserva,
        solicitudes_especiales,
        codigo_reserva
    ) VALUES (
        p_hotel_id,
        p_cliente_id,
        p_fecha_entrada,
        p_fecha_salida,
        p_adultos,
        p_ninos,
        'confirmada', -- Estado inicial
```

```
        p_tipo_reserva,
        p_solicitudes_especiales,
        v_codigo_reserva
    ) RETURNING reservacion_id INTO v_reservacion_id;

    -- Asignar habitaciones específicas
    FOR v_item IN SELECT * FROM jsonb_array_elements(p_tipos_habitaciones) LOOP
        -- Obtener tarifa aplicable
        SELECT COALESCE(
            (SELECT t.precio
             FROM tarifas_temporada t
             JOIN politicas_temporada p ON t.politica_id = p.politica_id
             WHERE t.tipo_id = (v_item->>'tipo_id')::INTEGER
             AND p.fecha_inicio <= p_fecha_entrada
             AND p.fecha_fin >= p_fecha_salida
             LIMIT 1),
            (SELECT precio_base FROM tipos_habitacion WHERE tipo_id = (v_item->>'tipo_id')::INTEGER)
        ) INTO v_tarifa;

        -- Asignar habitaciones disponibles
        FOR v_habitacion IN
            SELECT * FROM verificar_disponibilidad(
                p_hotel_id,
                (v_item->>'tipo_id')::INTEGER,
                p_fecha_entrada,
                p_fecha_salida
            ) LIMIT (v_item->>'cantidad')::INTEGER
        LOOP
            -- Asignar habitación a la reservación
            INSERT INTO reservacion_habitaciones (
                reservacion_id,
                habitacion_id,
                tarifa_aplicada,
                notas
            ) VALUES (
                v_reservacion_id,
                v_habitacion.habitacion_id,
                v_tarifa,
                'Asignación inicial'
            );

            -- Marcar habitación como ocupada
            UPDATE habitaciones
            SET estado = 'ocupada'
            WHERE habitacion_id = v_habitacion.habitacion_id;
        END LOOP;
    END LOOP;

    -- Registrar en bitácora
    PERFORM registrar_bitacora(
        p_usuario,
        'CREACION',
        'reservaciones',
        v_reservacion_id,
        'Reserva creada con código ' || v_codigo_reserva,
        p_ip_origen
    );

    RETURN v_reservacion_id;
END;
$$ LANGUAGE plpgsql;
```

Procedimiento para cancelar reservación

```
CREATE OR REPLACE FUNCTION cancelar_reservacion(
    p_reservacion_id INTEGER,
    p_usuario VARCHAR,
    p_razon TEXT DEFAULT NULL
) RETURNS VOID AS $$
DECLARE
    v_estado_actual VARCHAR;
BEGIN
    -- Obtener estado actual
    SELECT estado INTO v_estado_actual FROM reservaciones WHERE reservacion_id = p_reservacion_id;

    IF v_estado_actual IS NULL THEN
        RAISE EXCEPTION 'Reservación no encontrada';
    END IF;

    IF v_estado_actual = 'cancelada' THEN
        RAISE NOTICE 'La reservación ya está cancelada';
        RETURN;
    END IF;

    -- Actualizar estado a cancelada
    UPDATE reservaciones
    SET estado = 'cancelada'
    WHERE reservacion_id = p_reservacion_id;
```

```
    -- Registrar en bitácora (el trigger manejará la liberación de habitaciones)
    PERFORM registrar_bitacora(
        p_usuario,
        'CANCELACION',
        'reservaciones',
        p_reservacion_id,
        'Reserva cancelada. Razón: ' || COALESCE(p_razon, 'No especificada'),
        inet_client_addr()::TEXT
    );
END;
$$ LANGUAGE plpgsql;
```

Procedimiento para cambiar el estado de pago

```
CREATE OR REPLACE FUNCTION actualizar_estado_pago(
    p_pago_id INTEGER,
    p_estado VARCHAR,
    p_usuario VARCHAR
) RETURNS VOID AS $$
BEGIN
    -- Validar estado
    IF p_estado NOT IN ('pendiente', 'completado', 'reembolsado', 'fallido') THEN
        RAISE EXCEPTION 'Estado de pago no válido';
    END IF;

    -- Actualizar estado del pago (el trigger manejará la actualización de la reserva)
    UPDATE pagos
    SET estado = p_estado
    WHERE pago_id = p_pago_id;

    -- Registrar en bitácora
    PERFORM registrar_bitacora(
        p_usuario,
        'ACTUALIZACION',
        'pagos',
        p_pago_id,
        'Estado de pago actualizado a: ' || p_estado,
        inet_client_addr()::TEXT
    );
END;
$$ LANGUAGE plpgsql;
```

Trigger de registro automático de bitácora sobre cambios de reservaciones:

```sql
CREATE OR REPLACE FUNCTION bitacora_reservaciones_trigger()
RETURNS TRIGGER AS $$
DECLARE
    v_accion VARCHAR;
    v_detalles TEXT;
BEGIN
    IF TG_OP = 'INSERT' THEN
        v_accion := 'CREACION';
        v_detalles := 'Nueva reserva creada. Estado: ' || NEW.estado;
    ELSIF TG_OP = 'UPDATE' THEN
        v_accion := 'ACTUALIZACION';
        v_detalles := 'Estado cambiado de ' || OLD.estado || ' a ' || NEW.estado;

        -- Detalles adicionales para cambios específicos
        IF OLD.fecha_entrada ≠ NEW.fecha_entrada OR OLD.fecha_salida ≠ NEW.fecha_salida THEN
            v_detalles := v_detalles || '. Fechas modificadas.';
        END IF;
    ELSIF TG_OP = 'DELETE' THEN
        v_accion := 'ELIMINACION';
        v_detalles := 'Reserva eliminada';
    END IF;

    PERFORM registrar_bitacora(
        CURRENT_USER,
        v_accion,
        'reservaciones',
        COALESCE(NEW.reservacion_id, OLD.reservacion_id),
        v_detalles,
```

```sql
0       PERFORM registrar_bitacora(
1           CURRENT_USER,
2           v_accion,
3           'reservaciones',
4           COALESCE(NEW.reservacion_id, OLD.reservacion_id),
5           v_detalles,
6           inet_client_addr()::TEXT
7       );
8
9       IF TG_OP = 'DELETE' THEN
0           RETURN OLD;
1       ELSE
2           RETURN NEW;
3       END IF;
4   END;
5   $$ LANGUAGE plpgsql;
6
7   -- Asociar trigger a la tabla reservaciones
8   CREATE TRIGGER tr_bitacora_reservaciones
9   AFTER INSERT OR UPDATE OR DELETE ON reservaciones
0   FOR EACH ROW
1   EXECUTE FUNCTION bitacora_reservaciones_trigger();
```

Trigger para guardar registros de los pagos realizados:

```
624  CREATE OR REPLACE FUNCTION bitacora_pagos_trigger()
625  RETURNS TRIGGER AS $$
626  BEGIN
627      IF TG_OP = 'INSERT' THEN
628          PERFORM registrar_bitacora(
629              CURRENT_USER,
630              'CREACION',
631              'pagos',
632              NEW.pago_id,
633              'Nuevo pago registrado. Monto: ' || NEW.monto || ', Método: ' || NEW.metodo_pago,
634              inet_client_addr()::TEXT
635          );
636      ELSIF TG_OP = 'UPDATE' THEN
637          IF OLD.estado ≠ NEW.estado THEN
638              PERFORM registrar_bitacora(
639                  CURRENT_USER,
640                  'ACTUALIZACION',
641                  'pagos',
642                  NEW.pago_id,
643                  'Estado de pago cambiado de ' || OLD.estado || ' a ' || NEW.estado,
644                  inet_client_addr()::TEXT
645              );
646          END IF;
647      ELSIF TG_OP = 'DELETE' THEN
648          PERFORM registrar_bitacora(
```

```
647      ELSIF TG_OP = 'DELETE' THEN
648          PERFORM registrar_bitacora(
649              CURRENT_USER,
650              'ELIMINACION',
651              'pagos',
652              OLD.pago_id,
653              'Pago eliminado',
654              inet_client_addr()::TEXT
655          );
656      END IF;
657
658      IF TG_OP = 'DELETE' THEN
659          RETURN OLD;
660      ELSE
661          RETURN NEW;
662      END IF;
663  END;
664  $$ LANGUAGE plpgsql;
665
666  -- Asociar trigger a la tabla pagos
667  CREATE TRIGGER tr_bitacora_pagos
668  AFTER INSERT OR UPDATE OR DELETE ON pagos
669  FOR EACH ROW
670  EXECUTE FUNCTION bitacora_pagos_trigger();
```

Mecanismos de control de concurrencia

Se parte de la idea de que el sistema debe de soportar un alto nivel uso, es decir, escalabilidad y para atenderlo, lo ideal es un sistema de colas para manejar la alta concurrencia y en definitiva, crear una tabla y funciones para enrutar los pagos, es lo ideal.

```sql
-- Tabla de cola de pagos
CREATE TABLE cola_pagos (
    pago_queue_id SERIAL PRIMARY KEY,
    reservacion_id INTEGER NOT NULL REFERENCES reservaciones(reservacion_id),
    monto DECIMAL(10, 2) NOT NULL,
    metodo_pago VARCHAR(50) NOT NULL,
    usuario VARCHAR(50) NOT NULL,
    fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    estado VARCHAR(20) DEFAULT 'pendiente' CHECK (estado IN ('pendiente', 'procesando',
                                                             'completado', 'fallido')),
    intentos INTEGER DEFAULT 0,
    ultimo_intento TIMESTAMP,
    mensaje_error TEXT
);
```

Procedimiento para encolar pagos y procedimiento para procesar pagos, es muy importante destacar que utilizando skip locked se bloquean los pagos para manejar la concurrencia.

```sql
CREATE OR REPLACE FUNCTION encolar_pago(
    p_reservacion_id INTEGER,
    p_monto DECIMAL(10, 2),
    p_metodo_pago VARCHAR,
    p_usuario VARCHAR
) RETURNS INTEGER AS $$
DECLARE
    v_pago_queue_id INTEGER;
BEGIN
    INSERT INTO cola_pagos (
        reservacion_id, monto, metodo_pago, usuario
    ) VALUES (
        p_reservacion_id, p_monto, p_metodo_pago, p_usuario
    ) RETURNING pago_queue_id INTO v_pago_queue_id;

    RETURN v_pago_queue_id;
END;
$$ LANGUAGE plpgsql;
```

```sql
CREATE OR REPLACE FUNCTION procesar_cola_pagos()
RETURNS VOID AS $$
DECLARE
    v_pago RECORD;
BEGIN
    -- Se bloquea el procesamiento usando SKIP LOCKED para manejar concurrencia
    SELECT * INTO v_pago FROM cola_pagos
    WHERE estado = 'pendiente'
    ORDER BY fecha_creacion
    FOR UPDATE SKIP LOCKED
    LIMIT 1;

    IF FOUND THEN

        UPDATE cola_pagos
        SET estado = 'procesando',
            intentos = intentos + 1,
            ultimo_intento = CURRENT_TIMESTAMP
        WHERE pago_queue_id = v_pago.pago_queue_id;

        BEGIN

            PERFORM procesar_pago_concurrente(
                v_pago.reservacion_id,
                v_pago.monto,
                v_pago.metodo_pago,
                v_pago.usuario
            );

            UPDATE cola_pagos
            SET estado = 'completado'
            WHERE pago_queue_id = v_pago.pago_queue_id;
        EXCEPTION
            WHEN OTHERS THEN
                UPDATE cola_pagos
                SET estado = CASE WHEN intentos >= 3 THEN 'fallido' ELSE 'pendiente' END,
                    mensaje_error = SQLERRM
                WHERE pago_queue_id = v_pago.pago_queue_id;
        END;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Tablas para la auditoria de conflicto de concurrencia

```
CREATE TABLE conflictos_concurrencia (
    conflicto_id SERIAL PRIMARY KEY,
    fecha_hora TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    tipo_conflicto VARCHAR(50) NOT NULL,
    tabla_afectada VARCHAR(50) NOT NULL,
    id_registro INTEGER,
    usuario VARCHAR(50),
    detalles TEXT,
    resolucion VARCHAR(50)
);
```

Trigger para registrar automáticamente problemas de concurrencia

```
CREATE OR REPLACE FUNCTION registrar_conflicto()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO conflictos_concurrencia (
        tipo_conflicto, tabla_afectada, id_registro, usuario, detalles
    ) VALUES (
        TG_OP, TG_TABLE_NAME, COALESCE(NEW.id, OLD.id), CURRENT_USER,
        'Intento de modificación concurrente'
    );

    RETURN NULL; -- La idea es que falle para indicar que no pasó nada
END;
$$ LANGUAGE plpgsql;
```

Corridas de ejemplo de la base de datos

Introducción de registros de prueba, también se adjuntan en el .sql

```
sistema_hotelero=# INSERT INTO hoteles (nombre, direccion, ciudad, pais, telefono, email, estrellas, fecha_apertura, descripcion)
sistema_hotelero-# VALUES
sistema_hotelero-# ('Hotel Central', 'Av. Principal 123', 'San José', 'Costa Rica', '2222-1111', 'central@hotel.com', 4, '2020-01-01', 'Hotel de
 lujo en el centro');
INSERT 0 1
sistema_hotelero=#
sistema_hotelero=# -- Insertar clientes
sistema_hotelero=# INSERT INTO clientes (nombre, documento_identidad, tipo_documento, nacionalidad, telefono, email, preferencias, alergias)
sistema_hotelero-# VALUES
sistema_hotelero-# ('Carlos Pérez', '12345678', 'DNI', 'Costarricense', '8888-9999', 'carlos@example.com', 'Cama King, Piso alto', 'Ninguna');
INSERT 0 1
sistema_hotelero=#
sistema_hotelero=# -- Fidelización
sistema_hotelero=# INSERT INTO fidelizacion_clientes (cliente_id, hotel_id, puntos_acumulados, nivel_membresia, beneficios)
sistema_hotelero-# VALUES
sistema_hotelero-# (1, 1, 500, 'Oro', 'Check-out tardío, upgrades gratuitos');
INSERT 0 1
sistema_hotelero=#
sistema_hotelero=# -- Tipos de habitación
sistema_hotelero=# INSERT INTO tipos_habitacion (hotel_id, nombre, descripcion, capacidad, tamano, comodidades, precio_base)
sistema_hotelero-# VALUES
sistema_hotelero-# (1, 'Estándar', 'Cómoda habitación estándar', 2, 20, 'TV, WiFi, A/C', 70.00),
sistema_hotelero-# (1, 'Suite', 'Suite con sala', 4, 40, 'TV, WiFi, A/C, Sala', 150.00);
INSERT 0 2
sistema_hotelero=#
sistema_hotelero=# -- Habitaciones
sistema_hotelero=# INSERT INTO habitaciones (hotel_id, numero, tipo_id, piso, caracteristicas_especiales, estado)
sistema_hotelero-# VALUES
sistema_hotelero-# (1, '101', 1, 1, 'Vista al jardín', 'disponible'),
sistema_hotelero-# (1, '102', 1, 1, NULL, 'disponible'),
sistema_hotelero-# (1, '201', 2, 2, 'Balcón', 'disponible');
INSERT 0 3
sistema_hotelero=#
sistema_hotelero=# -- Políticas de temporada
sistema_hotelero=# INSERT INTO politicas_temporada (hotel_id, nombre, fecha_inicio, fecha_fin, descripcion, reglas)
sistema_hotelero-# VALUES
sistema_hotelero-# (1, 'Temporada Alta', '2025-12-01', '2026-01-31', 'Altísima demanda', 'Reservas no reembolsables');
INSERT 0 1
sistema_hotelero=#
sistema_hotelero=# -- Tarifas de temporada
sistema_hotelero=# INSERT INTO tarifas_temporada (politica_id, tipo_id, precio, descripcion)
sistema_hotelero-# VALUES
sistema_hotelero-# (1, 1, 100.00, 'Precio temporada alta - estándar'),
sistema_hotelero-# (1, 2, 200.00, 'Precio temporada alta - suite');
INSERT 0 2
```

Función para verificar disponibilidad:

```
sistema_hotelero=# SELECT * FROM verificar_disponibilidad(1, 1, '2025-12-15', '2025-12-20');
 habitacion_id | numero | piso | precio_recomendado
---------------+--------+------+--------------------
             1 | 101    |    1 |             100.00
             2 | 102    |    1 |             100.00
(2 rows)
```

Crear reservación:

```
sistema_hotelero=# SELECT crear_reservacion(
sistema_hotelero(#     1, -- hotel_id
sistema_hotelero(#     1, -- cliente_id
sistema_hotelero(#     '2025-12-15',
sistema_hotelero(#     '2025-12-20',
sistema_hotelero(#     2, -- adultos
sistema_hotelero(#     0, -- niños
sistema_hotelero(#     'individual',
sistema_hotelero(#     'Necesita cuna',
sistema_hotelero(#     '127.0.0.1',
sistema_hotelero(#     '[{"tipo_id": 1, "cantidad": 1}]'
sistema_hotelero(# );
 crear_reservacion
-------------------
                 4
(1 row)

sistema_hotelero=#
```

Cancelar reservación, se denota como se cancela la reservación, en el SELECT * FROM reservación de arriba se ve el único registro en confirmado y abajo en cancelado:

```
sistema_hotelero=# SELECT * FROM reservaciones;
 reservacion_id | hotel_id | cliente_id |       fecha_creacion       | fecha_entrada | fecha_salida | adultos | ninos |   estado   | tipo_reserv
a | solicitudes_especiales | codigo_reserva
----------------+----------+------------+----------------------------+---------------+--------------+---------+-------+------------+------------
--+------------------------+----------------
              4 |        1 |          1 | 2025-05-20 22:23:10.984702 | 2025-12-15    | 2025-12-20   |       2 |     0 | confirmada | individual
  | Necesita cuna          | RES-1-2025-9739
(1 row)


sistema_hotelero=#  SELECT cancelar_reservacion(4, 'admin', 'Cambio de planes');
 cancelar_reservacion
----------------------

(1 row)


sistema_hotelero=#  SELECT * FROM reservaciones;
 reservacion_id | hotel_id | cliente_id |       fecha_creacion       | fecha_entrada | fecha_salida | adultos | ninos |  estado   | tipo_reserva
 | solicitudes_especiales | codigo_reserva
----------------+----------+------------+----------------------------+---------------+--------------+---------+-------+-----------+-------------
-+------------------------+----------------
              4 |        1 |          1 | 2025-05-20 22:23:10.984702 | 2025-12-15    | 2025-12-20   |       2 |     0 | cancelada | individual
 | Necesita cuna          | RES-1-2025-9739
(1 row)


sistema_hotelero=#
```

Realización de pagos, primero crearé una nueva reservación:

```
sistema_hotelero=# SELECT crear_reservacion(
sistema_hotelero(#     1, -- hotel_id
sistema_hotelero(#     1, -- cliente_id
sistema_hotelero(#     '2025-12-13',
sistema_hotelero(#     '2025-12-23',
sistema_hotelero(#     2, -- adultos
sistema_hotelero(#     1, -- niños
sistema_hotelero(#     'grupo',
sistema_hotelero(#     'Necesita cuna',
sistema_hotelero(#     '127.0.3.1',
sistema_hotelero(#     '[{"tipo_id": 1, "cantidad": 1}]' -- JSONB habitaciones
sistema_hotelero(# );
 crear_reservacion
-------------------
                 5
(1 row)
```

Pago y verificación del pago:

```
sistema_hotelero=# INSERT INTO pagos (reservacion_id, monto, metodo_pago, estado)
sistema_hotelero-# VALUES (5, 100.00, 'Tarjeta', 'pendiente');
INSERT 0 1
sistema_hotelero=# SELECT * FROM pagos;
 pago_id | reservacion_id | monto  | metodo_pago |         fecha_pago         |  estado   | referencia | descripcion
---------+----------------+--------+-------------+----------------------------+-----------+------------+-------------
       1 |              5 | 100.00 | Tarjeta     | 2025-05-20 22:30:34.031821 | pendiente |            |
(1 row)
```

Comprobación de los *triggers* de bitácora:

```
sistema_hotelero=# SELECT * FROM bitacora_reservaciones ORDER BY fecha_hora DESC;
 bitacora_id |         fecha_hora         |  usuario  |    accion     | tabla_afectada | registro_id |                      detalles
             |           ip_origen
-------------+----------------------------+-----------+---------------+----------------+-------------+-----------------------------------------
-------------+--------------------------------
          10 | 2025-05-20 22:33:33.331128 | admin     | ACTUALIZACION | pagos          |           5 | Estado de pago actualizado a: completado
             | ::1/128
           9 | 2025-05-20 22:30:34.031821 | postgres  | CREACION      | pagos          |           1 | Nuevo pago registrado. Monto: 100.00, Mét
odo: Tarjeta | ::1/128
           7 | 2025-05-20 22:29:42.72008  | postgres  | CREACION      | reservaciones  |           5 | Nueva reserva creada. Estado: confirmada
             | ::1/128
           8 | 2025-05-20 22:29:42.72008  | 127.0.3.1 | CREACION      | reservaciones  |           5 | Reserva creada con código RES-1-2025-7587
             | [{"tipo_id": 1, "cantidad": 1}]
           6 | 2025-05-20 22:25:49.649673 | admin     | CANCELACION   | reservaciones  |           4 | Reserva cancelada. Razón: Cambio de plane
s            | ::1/128
           3 | 2025-05-20 22:25:49.649673 | postgres  | ACTUALIZACION | reservaciones  |           4 | Estado cambiado de confirmada a cancelada
             | ::1/128
           4 | 2025-05-20 22:25:49.649673 | postgres  | CANCELACION   | reservaciones  |           4 | Reserva cancelada. Días restantes: 209
             | ::1/128
           5 | 2025-05-20 22:25:49.649673 | postgres  | ACTUALIZACION | habitaciones   |           4 | Habitaciones liberadas por cancelación de
 reserva     | ::1/128
           1 | 2025-05-20 22:23:10.984702 | postgres  | CREACION      | reservaciones  |           4 | Nueva reserva creada. Estado: confirmada
             | ::1/128
           2 | 2025-05-20 22:23:10.984702 | 127.0.0.1 | CREACION      | reservaciones  |           4 | Reserva creada con código RES-1-2025-9739
             | [{"tipo_id": 1, "cantidad": 1}]
(10 rows)
```

Confirmación de utilización de habitaciones

```
sistema_hotelero=# SELECT * FROM habitaciones WHERE estado = 'ocupada';
 habitacion_id | hotel_id | numero | tipo_id | piso | caracteristicas_especiales | estado  | notas
---------------+----------+--------+---------+------+----------------------------+---------+-------
             1 |        1 | 101    |       1 |    1 | Vista al jardín            | ocupada |
(1 row)
```

Anexos

Costos de aws

https://aws.amazon.com/es/ec2/pricing/

https://aws.amazon.com/es/rds/aurora/pricing/

https://aws.amazon.com/es/rds/pricing/