BASES DE DATOS

Clase 18 - Arquitectura y lenguaje formal

Profesora: Erika Gutiérrez Beltrán

Tema 2: Bases de datos relacionales

Objetivo

Transformar los modelos conceptuales a lógicos. Construir bases de datos para datos estructurados

Temario

- Utilización de los modelos en el mundo real ✅
- Transformación del modelado a la base de datos ✅
- Clases de entidades ✅
- Modelo relacional, restricciones de integridad y normalización ✅
- Lenguaje de consulta formal ✅
- Lenguaje de consulta SQL ✅
- Arquitectura de los sistemas gestores de Bases de datos, OLTP, OLAP ✅

Logros

Capturar datos desde una pantalla de usuario real y almacenarlos en una base de datos. Manipulación de estos datos.

Arquitectura OLTP

Un sistema **On-line Transaction Processing** o OLTP, se define como una herramienta que admite aplicaciones enfocadas en el procesamiento de consultas manteniendo la integridad de los datos que son accedidos múltiples veces y en la eficiencia, una medida que se toma del número de transacciones por segundo, además de la administración y procesamiento de las bases de datos ofreciendo altos niveles de disponibilidad, seguridad y confiabilidad.

Los OLTP son bases de datos orientadas al procesamiento de transacciones.

Ejemplos de sistemas OLTP

- Supermercado de cadena
- Reservas áereas
- Tiendas de grandes ventas
- Bancos

Arquitectura OLTP

Características de los OLTP

- Los datos son átomicos (ACID)
- Los datos con los que se interactúan son actuales
- Se procesa un registro a la vez
- Permite ejecutar procesos repetitivos altamente estructurados
- Tiene aplicaciones backend
- Se maneja por filas

Recordemos las propiedades ACID

A = se garantiza que una transacción se realiza correctamente, todo se ejecuta con éxito. Si una parte de la transacción falla, toda falla.

C = todos los datos serán válidos de acuerdo a las reglas definidas (restricciones)

I = Aislamiento, una transacción no se ve afectada por otra. No es posible leer datos de una transacción incompleta

D = Cualquier cambio realizado a través de una transacción se debe almacenar permanentemente

Arquitectura OLAP

Online Analytical Processing se enfoca en grandes cantidades de datos recopilando información de múltiples fuentes y luego guardando esos datos en almacenes. Se requieren grandes recursos computaciones dado la cantidad de datos. Se basa en estructura de columnas.

Un ejemplo de un sistema OLAP, historial clínico de un paciente, aquí no se borrar registros.

OLAP permite hacer reportes, análisis de datos y minería de datos.

Características:

- Analizar datos en múltiples dimensiones
- Cálculos consistentes
- Restricciones para proteger los datos
- Curva de aprendizaje alta

Arquitectura OLAP - OLTP

- ¿Para la base de datos de películas que tipo de arquitectura usarías y por qué?, da varios ejemplos donde sea posible visualizar el uso de OLAP y de OLTP

- ¿Qué tipo crees que hemos utilizado OLAP o OLTP en el corte 1 y 2?, ¿por qué?

Comandos adicionales de SQL

- **LIKE**

Permite buscar en una cadena de texto coincidencias exactas o similares, ejemplo:

WHERE name LIKE ‘nam%’ – Devuelve valores cuyo nombre inician por nam

Para negar el **LIKE** sería **NOT LIKE**

WHERE name LIKE ‘nam’ – Funciona como un igual

WHERE name LIKE ‘_nam_’ – Coincide con cualquier cadena que comience con cualquier carácter seguido del patrón de búsqueda y finalice con cualquier otro

WHERE name LIKE ‘nam_’ – Coincide con cualquier cadena que inicie por nam y sea seguida de un solo carácter

WHERE name LIKE ‘%nam%’ – La cadena contiene nam

Comandos adicionales de SQL

- **LIKE**

Los patrones se pueden combinar, ejemplo:

WHERE name LIKE ‘_nam%’ – Comienza con cualquier carácter, es seguido por nam y termina con cualquier numero de caracteres

- **ILIKE es similar a LIKE**, su variación está en coincidir con el valor sin distinción de mayúsculas y minúsculas

- **BETWEEN**

La consulta hará una búsqueda entre un rango de valores

WHERE age BETWEEN 18 AND 30

<\>

Comandos adicionales de SQL

- **IS NULL**

Busca que el valor de una columna especifica tenga como valor null o este vacía

WHERE phone IS NULL

- **IS NOT NULL**

Busca entre valores de la columna específica que no sean vacíos

WHERE pone IS NOT NULL

- **IN**

Verifica si un valor coincide con una lista de valores

WHERE id_cliente IN (123, 125)

Comandos adicionales de SQL

- **AND**

Valida que un valor de búsqueda sea uno y otro (expresiones booleanas)

- **OR**

Busca valores donde coincida un campo o el otro

- **<>**

Busca elementos que sean distintos de lo que se indica

WHERE name <> ‘nam’

!= es equivalente a <>

Natural JOIN

Un natural JOIN puede ser un INNER, LEFT o RIGHT JOIN. Remueve las columnas redundantes y se puede utilizar en entidades que han sido normalizadas previamente

```
SELECT * FROM cd.facilities JOIN cd.bookings
    USING (facid);
```

```
SELECT * FROM cd.facilities NATURAL JOIN cd.bookings;
```

Para que esto sea posible de hacer, es importante que las tablas a consultar tengan campos que coinciden en tipo y nombre

Natural JOIN

Teniendo en cuenta la explicación cómo sería un NATURAL LEFT JOIN y un NATURAL RIGHT JOIN.
Intentemos representar las consultas en de los JOINS tradicionales que ya hemos visto, en NATURAL.
Usemos por ahora las tablas del parcial 2

<\>

DIVIDE

R (A	B)
p	1
p	2
p	3
q	1
r	1
r	3

S (C)
1
3

$R[B \div C]S$ (A)

p
r

Permite obtener la información de 2 entidades a través de la selección de filas de una relación que corresponden y coinciden a las filas de otra relación

ANTI JOIN

R (A	B)
p	1
p	2
p	3
q	1
r	1
r	3

S (C)
1
3

$R[B \div C]S$ (A)

p
r

Permite obtener la información de 2 entidades a través de la selección de filas de una relación que corresponden y coinciden a las filas de otra relación

Resumen –
Recordando conceptos

R		S		$R \cup S$		$R \cap S$		$R - S$	
ω	ω	ω	ω	ω	ω	ω	ω		
u	ω	u	ω	u	ω	u	ω		
u	1	u	1	u	1	u	1		
ω	1			ω	1			ω	1

UNIÓN INTERSECCIÓN DIFERENCIA

Resumen –
Recordando conceptos

R		
A	B	C
u	ω	ω
v	1	ω
w	ω	1
x	1	ω
y	ω	1

$R[B, C]$	
B	C
ω	ω
1	ω
ω	1

$R[C]$
C
ω
1

PROYECCIÓN

Recordando conceptos

THETA JOIN != EQUI JOIN

THETA JOIN

Permite unir dos tablas en función de una condición. En la mayoría de los casos a la unión theta se le denomina unión interna. Devolverá todas las filas donde se cumpla la condición (INNER)

EQUI JOIN

Permite unir dos tablas en función de la relación clave primaria y clave foránea.

Recordando conceptos

R

A	B
u	ω
ω	2
w	1

S

C
ω
2

$R[B=C]S$

A	B	C
ω	2	2

$R[B<C]S$

A	B	C
w	1	2

EQUI-JOIN

Recordando conceptos

```sql
-- Theta join
SELECT pe.nombre_pelicula, pe.fecha_estreno
FROM "Grupo 5".pelicula as pe
INNER JOIN "Grupo 5"."peliculasFavoritas" as pef
ON pe.id_pelicula = pef.id_pelicula
WHERE pe.fecha_estreno > '1994-06-07';

-- Equi join
SELECT *
FROM "Grupo 5".pelicula as pe
JOIN "Grupo 5"."peliculasFavoritas" as pef
ON pe.id_pelicula = pef.id_pelicula;
```

Recordando conceptos

```sql
-- Theta join
SELECT pe.nombre_pelicula, pe.fecha_estreno
FROM "Grupo 5".pelicula as pe
INNER JOIN "Grupo 5"."peliculasFavoritas" as pef
ON pe.id_pelicula = pef.id_pelicula
WHERE pe.fecha_estreno > '1994-06-07';

-- Equi join
SELECT *
FROM "Grupo 5".pelicula as pe
JOIN "Grupo 5"."peliculasFavoritas" as pef
ON pe.id_pelicula = pef.id_pelicula;
```

Comandos adicionales de SQL

- **COUNT**

Permite contar un número especifico registros:

```
SELECT COUNT(*)
FROM "Grupo 5".pelicula;
```

- **MAX**

Devuelve el valor máximo de los registros de una tabla

```
SELECT
    MAX(membercost)
FROM
    cd.facilities;
```

Comandos adicionales de SQL

- **MIN**

Devuelve el valor mínimo entre los registros de una tabla

```
SELECT
    MIN(membercost)
FROM
    cd.facilities;
```

- **SUM**

Permite calcular por medio de la suma de una columna especifica los valores de una tabla

```
SELECT
    SUM( membercost )
FROM
    cd.facilities;
```

<\/>

Comandos adicionales de SQL

- **AVG**

Calcula el promedio de una columna de una tabla

```
SELECT
    AVG( membercost )
FROM
    cd.facilities;
```

- **ROUND**

Permite redondear los resultados de las operaciones a 2 decimales o los que sean indicados

```
SELECT
    ROUND( AVG( membercost ), 2 ) avg_members_cost
FROM
    cd.facilities;
```

Práctica

- Crea una nueva columna edad en la tabla perfil creada para la práctica de películas y luego calcula la edad máxima, identifica el usuario mayor.
- Identifica el usuario menor de los que insertaste

Recuerda utilizar

```
ALTER TABLE nombre_tabla
ADD COLUMN nombre_columna tipo;
```

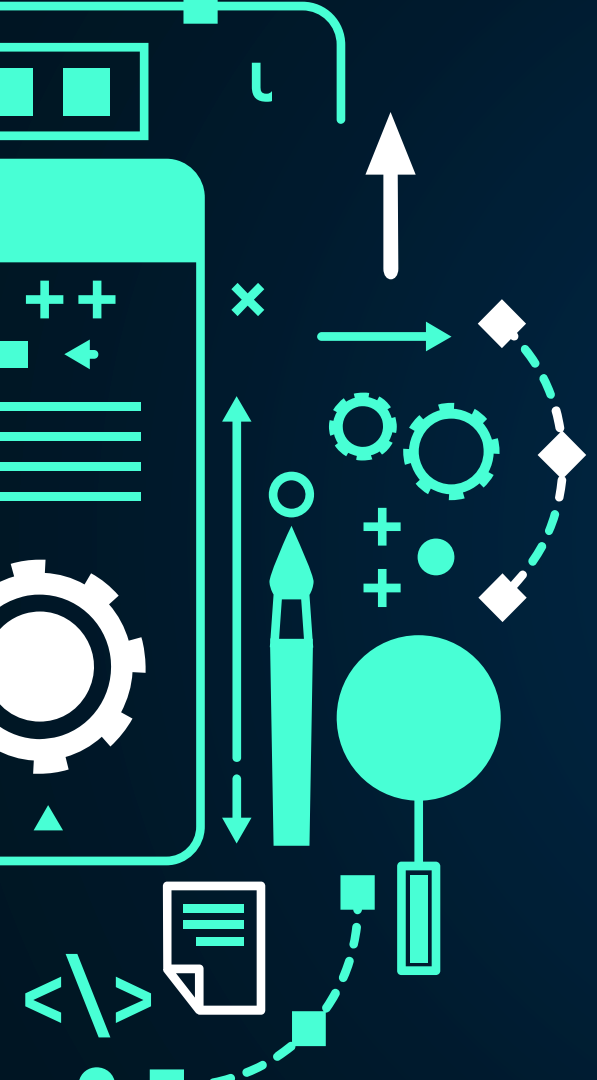
- Crea una nueva tabla de suscripciones con los planes que ofrecerá tu aplicación (mínimo 5) y crea la relación de muchos a muchos (suscripciones_usuario) donde se tenga el id del usuario y la suscripción.

- Crea un inner join para obtener los usuarios que tienen suscripciones y suma los pagos realizados por los usuarios para conocer los ingresos mensuales.

- Calcula la media de las edades de los usuarios, para así conocer cuales son las edades que más hacen uso de la plataforma

REFERENCIAS

- Libro: Fundamentos de base de datos 5 edición, Silberschatz
- Tutorial de PostgreSQL: https://www.postgresqltutorial.com/postgresql-joins/
- Extending the database relational model to capture more meaning, IBM Research Laboratory
- JOIN operations: https://docs.oracle.com/javadb/10.6.2.1/ref/rrefsqlj29840.html



Gracias!