

Bases de Datos Temporales, Espaciales y Espacio-Temporales

Nicolás Del Piano

Índice general

Resumen	2
Introducción	3
Bases de Datos Relacionales	4
Bases de Datos Temporales	5
Bases de Datos Espaciales	20
Bases de Datos Espacio-Temporales	39

Resumen

Este trabajo presenta una introducción hacia las Bases de Datos Temporales, Espaciales y Espacio-Temporales. Las Bases de Datos Temporales (*Temporal Databases*) están diseñadas para la captura de información que varía en el tiempo. Las Bases de Datos Espaciales (*Spatial Databases*) fueron concebidas por la necesidad de registrar el cambio geográfico y físico de cierta información. Por último, las Bases de Datos Espacio-Temporales (*Spatio-Temporal Databases*) son el resultado de la unión de las capacidades y propiedades ofrecidas por ambos tipos de Bases de Datos. Primero se presentarán conceptos de Bases de Datos clásicas, para luego abordar más claramente los temas centrales de esta monografía. El segundo capítulo presenta de una manera detallada las Bases de Datos Temporales, el tercero lo hace para las Espaciales, y el cuarto para las Espacio-Temporales. Por último se abordan tópicos generales relacionados con estos tipos de Bases de Datos.

Introducción

Hoy en día, la cantidad de información que manejan las corporaciones y empresas es gigantesca. Por esta razón, es necesario el uso de una herramienta que provea una forma de gestionar adecuadamente esta información. Este es el propósito de las Bases de Datos: brindar al usuario una forma de controlar el acceso, almacenamiento y administración de los datos de la entidad en cuestión.

Con la aparición de nuevas tecnologías, el surgimiento de nuevas necesidades fue inevitable, implicando que las Bases de Datos Relacionales no sean una bala de plata (aunque sean las más usadas actualmente) para resolver todos los problemas de gestión de datos. Surgieron conceptos como Minería de Datos, Data Warehouse y Big Data: la información ya no tiene la misma dimensión que antes. Fue entonces cuando el Modelo Relacional clásico necesitaba extenderse para representar eficientemente datos que varíen en tiempo y espacio.

Las Bases de Datos Temporales se encargan del dominio del tiempo y su relación con los datos, permite analizar la historia y controlar la validez temporal de los mismos. Una gran variedad de aplicaciones del mundo real manejan datos variables en el tiempo: control de inventario, registros médicos, operaciones bancarias, sistemas de información geográfica, gestión de reservas, aplicaciones científicas, etcétera. Esta necesidad de referencias temporales justifica la creación de un modelo de datos temporal.

Las Bases de Datos Espaciales extienden el modelo para representar el dominio espacial, con estructuras que puedan identificar un objeto en el espacio. Deben permitir la descripción de objetos espaciales mediante tres características: atributos, localización y topología. Además deben proveer tipos de datos espaciales para estructurar entidades geométricas en el espacio. Existen diversas áreas donde la gestión de información geométrica, geográfica o espacial es crucial: Sistemas de Información Geográfica, Bases de Datos multimedia, imágenes satelitales, ciencias ambientales, astronomía.

El objetivo de las Bases de Datos Espacio-Temporales es extender los modelos de información espacial para incluir el tiempo y describir en forma más dinámica la realidad que se quiere representar. El modelo espacio-temporal abarca aplicaciones demográficas, ecológicas, relacionadas con marketing, militares, urbanísticas y de fenómenos naturales, entre otras.

Bases de Datos Relacionales

Bases de Datos Temporales

El tiempo es un aspecto importante para los fenómenos del mundo real: los eventos ocurren en momentos de tiempo específicos.

A veces nos interesa saber con cierta certeza cuándo ocurrió tal evento, y poder compararlo con otros para obtener información de interés.

Muchas de las áreas donde se aplican las Bases de Datos tienen naturaleza temporal:

- Control de inventario.
- Registros médicos.
- Sistemas de información geográfica.
- Operaciones bancarias.
- Data Warehousing.
- Sistemas de control de reservas (aerolíneas, hoteles, etc).
- Aplicaciones científicas.

Relaciones no temporales

En la Figura 3.1 puede observarse una tabla relacional no temporal. Cada tupla

Id	Nombre	Estado	Sueldo
1	Juan	Activo	5700
2	Manuel	Activo	2300

Figura 3.1: Tabla relacional no temporal.

representa un hecho verdadero *ahora*. Solo hay un estado representable de la Base de Datos: *el actual* (*current snapshot*). A medida que el tiempo transcurre, los datos se van actualizando y modificando. Con este modelo, perdemos información. Las Bases de Datos convencionales representan el estado de la información en un instante de tiempo dado. Aunque la Base de Datos es actualizada, estos cambios son vistos como modificaciones del estado actual y los datos obsoletos son borrados. Por lo tanto, solo podemos utilizar la información actual de la Base de Datos. Esto genera un problema cuando queremos responder preguntas involucradas a intervalos de tiempo: *¿Cuáles empleados percibieron un aumento el mes pasado?*

Bases de Datos Temporales: Definición

Un *DBMS Temporal* es un Sistema de Gestión de Bases de Datos que proporciona herramientas para el manejo y control de Bases de Datos Temporales.

Una *Base de Datos Temporales* es una Base de Datos que tiene dimensión del tiempo a través del almacenamiento de datos temporales.

Proporcionan un marco que mantiene la historia de los cambios que se produjeron en la fuente de datos. Están diseñadas para la captura de la información que varía en el transcurso del tiempo (puede apreciarse esta relación en la Figura 3.2).

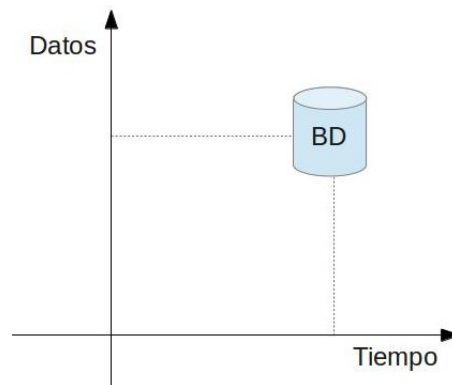


Figura 3.2: Relación tiempo y datos.

Datos Temporales

Un *Dato Temporal* es un dato convencional al que se le asocia un período de tiempo para expresar valores temporales en la Base de Datos.

Este agregado de información temporal se denomina *time-stamping*. Al asociar el tiempo con la información, es posible almacenar diferentes estados de una base de datos.

Dimensión del Tiempo

El tiempo es infinito, continuo y multidimensional [1]. Las computadoras no pueden representar información continua, de hecho, se aproximan discretamente. Así, para representar una noción del tiempo, se lo transforma en un conjunto discreto con una cierta granularidad. Por ejemplo, la sentencia *Homero Simpson nació el 12 de Mayo de 1956* tiene una granularidad de días.

Las Bases de Datos Temporales almacenan dos dimensiones de tiempo:

- Tiempo Válido
- Tiempo Transaccional

El *Tiempo Válido* representa cuándo un hecho tiene validez, es decir, es verdadero en el mundo real. El Tiempo Válido de un evento es el tiempo de un reloj en el que ese evento ocurrió [1]. Es independiente de si dicho evento fue registrado o no en la Base de Datos. Los Tiempos Válidos pueden encontrarse en el pasado, presente o futuro. Una de las características es que todos los eventos tienen asociado un Tiempo Válido, pero no necesariamente son registrados. Además brindan

la capacidad de gestionar la historia de la Base de Datos.

El *Tiempo Transaccional* registra el período de tiempo donde un hecho fue almacenado en la Base de Datos. Permiten realizar consultas que muestren el estado de la Base de Datos en un tiempo específico. Este tiempo está acotado en ambos extremos; la creación de la Base de Datos y el tiempo presente, es decir, los Datos Transaccionales viven solamente dentro de la vida de una Base de Datos. Una de las capacidades interesantes es que permiten volver hacia un estado anterior (*roll-back*), ya que almacenan datos de las operaciones que se fueron haciendo.

Estas dos dimensiones son ortogonales. Un Modelo de Datos que no soporte ninguno de estas dimensiones se denomina *snapshot*, ya que captura solamente una imagen de la Base de Datos. Si se brinda soporte para Tiempo Válido, entonces se cuenta con una Modelo de Datos *histórico*, mientras que uno que soporte Tiempo Transaccional solamente se denomina *rollback*. En caso de que ambas dimensiones estén soportadas, se denomina *bitemporal*.

Ejemplo

Análizamos en forma no temporal y temporal el siguiente ejemplo:

- El Señor X nace en Springfield el 12 de Mayo de 1956.



- Su padre lo registra el 13 de Mayo de 1956.
- Se muda a Arroyos Cipreses el 3 de Agosto de 1980, pero olvida registrarse; lo hace el 16 de Agosto del mismo año.
- Muere el 20 de Abril de 2004.

Ejemplo: no temporal

Nombre	ViveEn
Señor X	Springfield

⇓ Update

Nombre	ViveEn
Señor X	Arroyos Cipreses

⇓ Delete

Nombre	ViveEn
Señor X	Arroyos Cipreses

Ejemplo: Tiempo Válido

Nombre	ViveEn	Valid-From	Valid-To
Señor X	Springfield	12-May-1956	∞

⇓ Update

Nombre	ViveEn	Valid-From	Valid-To
Señor X	Springfield	12-May-1956	2-Aug-1980

⇓ Insert

Nombre	ViveEn	Valid-From	Valid-To
Señor X	Springfield	12-May-1956	2-Aug-1980
Señor X	Arroyos Cipreses	3-Aug-1980	∞

⇓ Update

Nombre	ViveEn	Valid-From	Valid-To
Señor X	Springfield	12-May-1956	2-Aug-1980
Señor X	Arroyos Cipreses	3-Aug-1980	20-Apr-2004

Ejemplo: Tiempo Transaccional

Nombre	ViveEn	Transaction-From	Transaction-To
Señor X	Springfield	13-May-1956	∞

⇓ Insert

Nombre	ViveEn	Transaction-From	Transaction-To
Señor X	Springfield	13-May-1956	16-Aug-1980
Señor X	Arroyos Cipreses	16-Aug-1980	∞

⇓ Update

Nombre	ViveEn	Transaction-From	Transaction-To
Señor X	Springfield	13-May-1956	16-Aug-1980
Señor X	Arroyos Cipreses	16-Aug-1980	20-Apr-2004

Base de Datos Bitemporal

Incluyen ambos tiempos (Válido y Transaccional) lo que les permite proveer información histórica, a la vez que brindan la capacidad de hacer roll-back de los datos. En la siguiente figura se puede apreciar un ejemplo:

Nombre	ViveEn	Valid-From	Valid-To	Transaction-From	Transaction-To
Señor X	Springfield	12-May-1956	∞	13-May-1956	16-Aug-1980
Señor X	Springfield	12-May-1956	2-Aug-1980	16-Aug-1980	∞
Señor X	Arroyos Cipreses	3-Aug-1980	∞	16-Aug-1980	20-Apr-2004
Señor X	Arroyos Cipreses	3-Aug-1980	20-Apr-2004	20-Apr-2004	∞

Figura 3.3: Ejemplo de una relación bitemporal.

Extensiones Temporales

Hay dos formas de extender el modelo relacional para especificar requisitos temporales.

La forma mostrada en los ejemplos antes mencionados se denomina **marcaje de tuplas**. Este método es muy común en el modelo relacional. Se utiliza un atributo especial para indicar la validez de una tupla: se indica *desde* y *hasta* para representar intervalos de tiempo.

$$(attr_1, \dots, attr_n) \rightarrow (attr_1, \dots, attr_n, temp_attr_1, \dots, temp_attr_m)$$

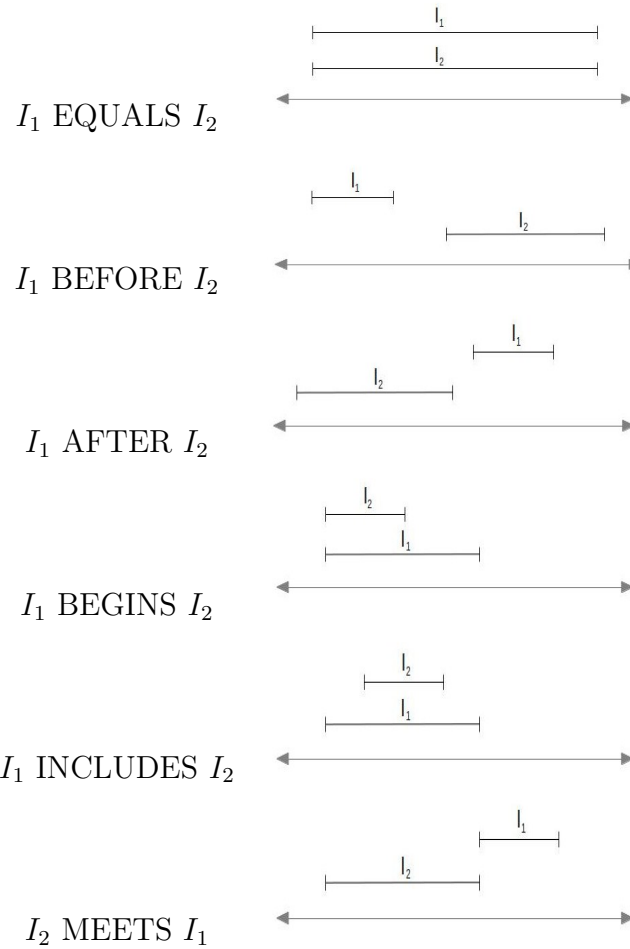
Una de las desventajas que tiene esta forma es que una entidad puede estar representada por varias tuplas, por lo que no puede lograrse una representación 1:1 de la realidad lo que podría generar información redundante.

La segunda forma se denomina **marcaje de atributos** y usa atributos multivaluados. La idea es que la marca de tiempo y la entrada referenciada se almacenen en el mismo atributo de forma anidada. Al mismo tiempo que permite la correspondencia 1:1 entre entidades y hechos reales, dificulta las actualizaciones y no cumple la 1NF.

Operadores de Allen

Necesitamos una forma de comparar datos temporales. Allen (1983) propone un conjunto de operadores temporales lógicos para comparar intervalos de tiempo. El operador es una función de tipo: $I_t \times I_t \rightarrow \{True, False\}$.

Algunos ejemplos de estos operadores:



Modelo Relacional Temporal

El modelo relacional temporal incorpora la semántica temporal en el modelo relacional utilizando marcaje de tuplas como extensión temporal.

Alguna de las características que debe ofrecer son:

- Un tipo de datos para períodos de tiempo, incluyendo la habilidad de representar períodos infinitos.
- Soporte para tiempo válido, transaccional y tablas bitemporales.
- Tiempo transaccional controlado por el sistema.
- Consultas temporales.
- Predicados y operadores que actúen sobre intervalos de tiempo.

Lenguaje de consulta temporales

Una Base de Datos Temporal es un repositorio de información temporal. Un lenguaje de consulta temporal es cualquier lenguaje de consulta para bases de datos temporales.

Las propiedades de un lenguaje de consulta temporal son:

- Semántica declarativa

- Implementación eficiente.
- Independencia en la representación.
- Expresividad en la consulta.

Algunos ejemplos importantes son: TSQL, TQuel, HRDM, Backlog. Muchos otros son derivados de éstos.

DBMS Temporal

Los DBMS Temporales deben respaldar un lenguaje de definición de datos temporales, un sistema de restricciones temporales, un lenguaje de manipulación de datos y un lenguaje de consulta temporal.

Algunos ejemplos de DBMS Temporales son: TimeDB, Oracle Workspace Manager, PostgreSQL, IBM DB2, entre otros.

Historia y estandarización

Hacia un modelo de datos temporal

La comunidad de las Bases de Datos Temporales fue prolífica en la producción de modelos de datos temporales y lenguajes de consulta.

En los últimos 20 años, docenas de modelos relacionales de datos temporales fueron propuestos. *Richard Snodgrass* propuso en 1992 que la comunidad de Bases de Datos Temporales realice extensiones a SQL.

Proceso de estandarización de SQL

El responsable del estándar SQL en Estados Unidos es INCITS(ANSI) DM32.2. Internacionalmente, es responsable el comité ISO/IEC JTC 1/SC 32 Data Management and Interchange/WG3. Muchas de las capacidades del estándar SQL fueron originadas en Estados Unidos. Usualmente la aprobación de nuevos estándares tiene un ciclo de 3 a 5 años. Actualmente, hay 7 versiones del Standard SQL: 86(SQL-87), 89, 92(SQL2), SQL:1999(SQL3), SQL:2003, SQL:2008, SQL:2011.

SQL Temporal: primer intento (1995-2001)

X3H2(ahora conocido como DM32.2) y WG 3 aprobaron el trabajo SQL/Temporal en 1995. Estados Unidos fue el primero en hacer la propuesta, añadiendo nuevas extensiones a SQL, basadas en el trabajo de Richard Snodgrass. La propuesta de USA estaba basada en TSQL2, una extensión de SQL-92. En relación al trabajo realizado por Estados Unidos, el Reino Unido lanzó una propuesta muy similar, pero debido a conflictos entre estas propuestas, ANSI e ISO decidieron cancelar en el año 2001 el proyecto SQL/Temporal.

SQL Temporal: segundo intento (2008-2011)

En 2008, un segundo intento se hizo presente. Empezó con la aceptación de la propuesta *system-versioned tables* llevada a cabo por INCITS DM32.2 y ISO/IEC JTC1 SC32 WG3. La idea no fue resucitar SQL/Temporal, sino que se añadieron

las ideas a SQL/Foundation. Una de las características interesantes son las *application-time period tables*.

Las características temporales en SQL:2011 están inspiradas en las anteriores propuestas hechas en SQL/Temporal, pero con una sintaxis un poco diferente.

Características temporales en SQL-92

Inclusión de tipos de datos temporales:

- DATE (10 posiciones) = YEAR, MONTH, DAY (yyyy-mm-dd)
- TIME (8 posiciones) = HOUR, MINUTE, SECOND (hh:mm:ss)
- TIMESTAMP (DATE, TIME, fracciones de segundo y desplazamiento de acuerdo al huso horario estándar)
- INTERVAL: período de tiempo. Para incrementar/decrementar el valor actual

TSQL2

TSQL2 [2] es un lenguaje de consulta temporal consensuado por un comité de grupos de investigación en bases de datos temporales. Originalmente apareció como una extensión para SQL-92. La especificación incluye las ideas y conceptos de Tiempo Válido, Tiempo Transaccional y tablas bitemporales. Este lenguaje es conocido también como extensiones ANSI X3.135-1992 y ISO/IEC 9075:1992.

Algunas de las características principales:

- Incluye cuatro tipos de marcas de tiempo válido: intervalos, instantes, períodos y elementos.
- Asociadas a estas marcas, existen tres categorías de operadores: extractores, constructores y comparadores.
- Tiene una variable denominada NOW que indica el momento actual (es usada como tiempo de referencia).

Extractores

Operación	Operador
Extractor de eventos	BEGIN(event) END(period) BEGIN(element) END(event) END(period) END(element)
Extractor de períodos	FIRST(period) FIRST(element) LAST(period) LAST(element)

Constructores

Operación	Operador
Constructor de eventos	FIRST(event, event) LAST(event, event)
Constructor de períodos	PERIOD(event, event) INTERSECT(period, period)
Constructor de elementos	INTERSECT(element, element) element + element element – element

Comparadores

Operación	Operador
Comparador de eventos	event PRECEDES event event = event event OVERLAPS event event MEETS event event CONTAINS event
Comparador de períodos	period PRECEDES period period = period period OVERLAPS period period MEETS period period CONTAINS period
Comparador de elementos	element PRECEDES element element = element element OVERLAPS element element CONTAINS element

Operadores de Allen

Operador de Allen	Operador TSQL2
a BEFORE b	a PRECEDES b
a EQUAL b	a = b
a OVERLAPS b	a OVERLAPS b AND END(a) PRECEDES END(b)
a MEETS b	END(a) = BEGIN(b)
a DURING b	BEGIN(b) PRECEDES BEGIN(a) AND END(a) PRECEDES END(b)
a START b	BEGIN(a) = BEGIN(b) AND END(a) PRECEDES END(b)
a FINISH b	BEGIN(b) PRECEDES BEGIN(a) AND END(a) = END(b)

Ejemplos de consulta en TSQL2: Tiempo Válido

```
> CREATE TABLE empleado(ename VARCHAR(12), eno INTEGER PRIMARY KEY, cumple DATE);
> CREATE TABLE salario(enno INTEGER REFERENCES empleado(enno), sueldo INTEGER);
```

```
> INSERT INTO empleado
VALUES('Homero', 1, DATE '1955-03-21');
> INSERT INTO empleado
VALUES('Lenny', 2, '1956-09-18');
```

```
> INSERT INTO salario VALUES(1, 2000);
> INSERT INTO salario VALUES(2, 4000);
> ALTER TABLE salario ADD VALIDTIME PERIOD(DATE);
> ALTER TABLE empleado ADD VALIDTIME PERIOD(DATE);
```

```
> INSERT INTO empleado
VALUES('Carl', 3, DATE '1955-08-10');
> INSERT INTO salario VALUES(3, 3500);
> COMMIT;
```

ename	eno	cumple	Válido
Homero	1	1955-03-21	[1995-02-01 - 9999-12-31)
Lenny	2	1956-09-18	[1995-02-01 - 9999-12-31)
Carl	3	1955-08-10	[1995-02-01 - 9999-12-31)

eno	suelo	Válido
1	2000	[1995-02-01 - 9999-12-31)
2	4000	[1995-02-01 - 9999-12-31)
3	3500	[1995-02-01 - 9999-12-31)

Cambiar el nombre de 'Carl' por 'Carlos'.

```
> VALIDTIME UPDATE empleado
SET ename = 'Carlos'
WHERE ename = 'Carl';
> COMMIT;
```

ename	eno	cumple	Válido
Homero	1	1955-03-21	[1995-02-01 - 9999-12-31)
Lenny	2	1956-09-18	[1995-02-01 - 9999-12-31)
Carlos	3	1955-08-10	[1995-02-01 - 9999-12-31)

¿Quién percibió un aumento de sueldo?

```
> UPDATE salario
SET sueldo = 1.05 * amount
WHERE eno = 3;
> COMMIT;
```

```
> NONSEQUENCED VALIDTIME SELECT ename
FROM empleado AS E, salario AS S1, salario AS S2
WHERE E.eno = S1.eno AND E.eno = S2.eno
AND S1.sueldo < S2.sueldo AND
VALIDTIME(S1) MEETS VALIDTIME(S2);
```

ename
Carlos

Ejemplos de consulta: Tiempo Transaccional

ename	eno	cumple	Válido
Homero	1	1955-03-21	[1995-02-01 - 9999-12-31)
Lenny	2	1956-09-18	[1995-02-01 - 9999-12-31)
Carlos	3	1955-08-10	[1995-02-01 - 9999-12-31)

```
> ALTER TABLE empleado ADD TRANSACTIONTIME;
> COMMIT;
```

ename	eno	cumple	Válido	Transacción
Homero	1	1955-03-21	[1995-02-01 - 9999-12-31)	[1995-07-01 - 9999-12-31)
Lenny	2	1956-09-18	[1995-02-01 - 9999-12-31)	[1995-07-01 - 9999-12-31)
Carlos	3	1955-08-10	[1995-02-01 - 9999-12-31)	[1995-07-01 - 9999-12-31)

```
> UPDATE empleado
SET ename = 'Homero J.'
WHERE ename = 'Homero';
> COMMIT;
```

ename	eno	cumple	Válido	Transacción
Homero	1	1955-03-21	[1995-02-01 - 9999-12-31)	[1995-07-01 - 2014-04-23)
Homero J.	1	1955-03-21	[1995-02-01 - 9999-12-31)	[2014-04-23 - 9999-12-31)
Lenny	2	1956-09-18	[1995-02-01 - 9999-12-31)	[1995-07-01 - 9999-12-31)
Carlos	3	1955-08-10	[1995-02-01 - 9999-12-31)	[1995-07-01 - 9999-12-31)

¿Cuándo trabajó algún empleado por más de seis meses?

ename	eno	cumple	Válido	Transacción
Homero	1	1955-03-21	[1995-02-01 - 9999-12-31)	[1995-07-01 - 2014-04-23)
Homero J.	1	1955-03-21	[1995-02-01 - 9999-12-31)	[2014-04-23 - 9999-12-31)
Lenny	2	1956-09-18	[1995-02-01 - 9999-12-31)	[1995-07-01 - 9999-12-31)
Carlos	3	1955-08-10	[1995-02-01 - 1995-03-31)	[1995-07-01 - 9999-12-31)

```
> VALIDTIME AND TRANSACTIONTIME SELECT ename, eno
FROM empleado
WHERE INTERVAL(VALIDTIME(empleado) MONTH) >
INTERVAL '6' MONTH;
```


ename	eno	cumple	Válido	Transacción
Homero	1	1955-03-21	[1995-02-01 - 9999-12-31)	[1995-07-01 - 2014-04-23)
Homero J.	1	1955-03-21	[1995-02-01 - 9999-12-31)	[2014-04-23 - 9999-12-31)
Lenny	2	1956-09-18	[1995-02-01 - 9999-12-31)	[1995-07-01 - 9999-12-31)

Estado del Arte: Las BDT en la actualidad

Actualmente se pueden manipular los datos temporales de las siguientes formas:

- Usar un tipo de datos temporal integrado al DBMS y brindar soporte temporal con aplicaciones.
- Implementar un tipo de dato abstracto para el tiempo.
- Extender el modelo de datos no temporal a uno temporal.
- Generalizar un modelo de datos no temporal en uno temporal.

SQL:2011

El tiempo transaccional es manejado con *system-versioned tables*, que contienen el período de tiempo del sistema, y el tiempo válido es manejado con tablas que contienen *application-time period* (en la Figura 3.4 se puede apreciar esta relación). Las técnicas e ideas de TSQL se tuvieron en cuenta por el comité, pero las exten-

transaction time → system time
valid time → application time

Figura 3.4: Relación entre los diferentes tiempos.

siones sintácticas que se hicieron difirieron considerablemente de aquellas propuestas en TSQL.

SQL:2011 Application-Time tables

El tipo de datos PERIOD para intervalos de tiempo, sigue no disponible. Es simulado usando pares de instantes (con la semántica [cerrado,abierto)). Application-time period tables son tablas que contienen una cláusula PERIOD, con un nombre del período definido por el usuario. Estas tablas contienen también dos columnas adicionales para almacenar el tiempo de inicio y fin de un período de un dato temporal.

SQL:2011 System-Versioned tables

Son tablas que contienen una cláusula PERIOD con un nombre de período predefinido (SYSTEM_TIME). Contienen dos columnas adicionales que se refieren a el inicio y fin de una transacción. Ambos valores son seteados por el sistema. También preservan las versiones antiguas de las filas.

SQL:2011 Ejemplo

Se tiene la siguiente tabla:

emp_name	dept_id	start_date	end_date
John	M24	1998-01-31	9999-12-31
John	J13	1995-11-15	1998-01-31
Tracy	K25	1996-01-01	2000-03-31

¿En cuál departamento estuvo John el 1 de Diciembre de 1997?

```
SELECT dept_id
FROM empleados
WHERE emp_name = 'John' AND start_date ≤ DATE '1997-12-01' AND end_date
> DATE '1997-12-01';
```

¿En qué departamento está John ahora?

```
SELECT dept_id
FROM empleados
WHERE emp_name = 'John' AND start_date ≤ CURRENT_DATE AND end_date
> CURRENT_DATE;
```

Se borra la 3 fila el 16/4/2014:

```
DELETE FROM empleados
WHERE emp_name = 'Tracy';
```

La tabla queda:

emp_name	dept_id	system_start	system_end
John	M24	1998-01-31	9999-12-31
John	J13	1995-11-15	1998-01-31
Tracy	K25	1995-11-15	2014-04-16

SQL:2011 vs TSQL

La siguiente tabla comparativa muestra las diferencias entre ambos estándares:

TSQL	SQL:2011
valid time transaction time	application time system time
timestamping	versioning
validtime table transactiontime table bitemporal table	application time period table system-versioned table system-versioned application time period table

TimeDB

TimeDB es un DBMS Bitemporal basado en SQL [?]. Soporta un lenguaje de consulta, un lenguaje de manipulación de datos y un lenguaje de definición de datos. TimeDB brinda ATSQL2, un lenguaje de consulta basado en TSQL2, ChronoLog [?] y Bitemporal ChronoLog. Las características principales son que manipula los datos temporales buscando extender el modelo de datos relacional a uno temporal y traduce sentencias TSQL en sentencias SQL estándar para luego ser ejecutadas en DBMS como Oracle, Sybase, etcétera.

TimeDB: TDDL

En TimeDB, una tabla bitemporal puede crearse de esta manera:

```
CREATE TABLE empleados (EmpID INTEGER, Name CHAR(30), Department
CHAR(40),
Salary INTEGER)
AS VALIDTIME AND TRANSACTIONTIME;
```

TimeDB: TDML

Las siguientes sentencias insertan datos temporales a una tabla:

```
VALIDTIME PERIOD '1985-1990'
INSERT INTO empleados VALUES (10,'John','Research',11000);
```

```
VALIDTIME PERIOD '1990-1993'
INSERT INTO empleados VALUES (10,'John','Sales',11000);
```

```
VALIDTIME PERIOD '1993-forever'
INSERT INTO empleados VALUES (10,'John','Sales',12000);
```

TimeDB: TQL

Para hacer consultas en TimeDB:

```
VALIDTIME
SELECT * FROM empleados;
```

```
TRANSACTIONTIME
SELECT * FROM empleados;
```

```
VALIDTIME AND TRANSACTIONTIME
SELECT * FROM empleados;
```

Oracle Workspace Manager

Es una herramienta de Oracle Database [?] que brinda a los desarrolladores la capacidad de manejar distintas versiones temporales de la base de datos.

Se puede habilitar el soporte de Tiempo Válido al momento de creación de una tabla:

- Crear la tabla de empleados y sus salarios:

```
CREATE TABLE empleados (  
  name VARCHAR(16) PRIMARY KEY,  
  salary NUMBER  
);
```
- Versionar la tabla. Especificar TRUE para el soporte de tiempo válido:

```
EXECUTE DBMS_WM.EnableVersioning('empleados','VIEW_WO_OVERWRITE',FALSE,T
```
- Insertar las filas:

```
INSERT INTO empleados VALUES(  
  'Adams',  
  30000,  
  WMSYS.WM_PERIOD(TO_DATE('01-01-1990','MM-DD-YYYY'),  
    TO_DATE('01-01-2005','MM-DD-YYYY'))
```

El tipo de datos WM_PERIOD es usado para especificar el rango del tiempo válido.

Las constantes del tiempo válido son:

DBMS_WM.MIN_TIME \approx 01-Jan-(-4712)

DBMS_WM.MAX_TIME \approx 31-Dec-9999

DBMS_WM.UNTIL_CHANGED es un TIMESTAMP que se comporta como MAX_TIME hasta que es modificado.

Algunos operadores:

WM_OVERLAPS, WM_CONTAINS, WM_MEETS, WM_EQUALS, WM_INTERSECTION, etcétera.

Bases de Datos Espaciales

En muchas áreas de gestión de información existe la necesidad de administrar datos *espacial*, *geométricos* o *geográficos*: datos relacionados con el *espacio*. El espacio de interés puede ser, por ejemplo, la representación 2-d de la superficie de la tierra, un volumen que contiene el modelo de un cerebro humano, o un espacio 3-d representando la disposición de las cadenas de moléculas de proteínas. Con el advenimiento de los DBMS relacionales, hubieron intentos por modelar esta información en los sistemas de bases de datos convencionales. El problema: el soporte no es muy bueno para manejar *grandes cantidades de datos espaciales*.

Pensando espacialmente

¿Dónde está esto? ¿Cuánto tardo para llegar a X lugar?

Hoy en día tenemos grandes herramientas para responder este tipo de preguntas: Google Maps, Virtual Earth, MapQuest, Yahoo, etc...

Yendo más allá, las organizaciones han descubierto que estas herramientas son un gran recurso para analizar patrones de datos. Por ejemplo, una popular empresa de pizzas, trazando los recorridos de las direcciones de los *pizza lovers*, puede encontrar fácilmente dónde inaugurar su nuevo local.

Pensando espacialmente, ¿en un DBMS relacional?

Si bien los DBMS relacionales son muy populares y responden a la mayoría de las consultas, la visión real es que muchas de ellas son incapaces de manejar datos espaciales, o son poco amigables. Esto se debe a que el rol tradicional de un DBMS es y ha sido almacenar información relacionada al mundo de los negocios: los datos que residen en esas grandes bases de datos es simple (nombres, saldos, direcciones, etcétera).

Por ejemplo, una consulta como *listar todos los empleados cuyo sueldo es mayor a X* puede ser simple y eficientemente respondida, aún en bases de datos gigantes. Pero, ¿qué sucede si quiero saber: *listar todos los empleados que residan a menos de 10 km de la compañía*? En los DBMS relacionales, es posible contestar estas preguntas pero es totalmente *ineficiente*.

En base a esta problemática nacen las Bases de Datos Espaciales.

Aplicaciones de las BD Espaciales

- Sistemas de Información Geográfica.
- Bases de Datos Multimedia.
- Imágenes satelitales.

- Modelo de circuitos integrados de gran tamaño.
- Censos.
- Ciencias Ambientales.
- Astronomía.

¿Quién se beneficia?

Usuario de celular: ¿Dónde está la siguiente estación de servicio? ¿Hay una camino a casa?

Médico: Basado en el siguiente MRI, ¿hemos tratado a alguien con el mismo resultado?


Biólogo molecular: ¿Está la topología de la biosíntesis del gel aminoácido en el genoma descubierto en otra secuencia de la base de datos?

Astrónomo: Buscar todas las galaxias dentro de dos minutos de arcos de quásares.

Climatólogo: Testear y verificar mi nuevo modelo climático.

¿Qué se busca?

Tener soporte para representar este tipo de tablas:

Id	País	Geografía
1	Argentina	
2	Brasil	

Y además brindar herramientas que permitan comparar eficientemente datos como los de la tercer columna.

Bases de Datos Espaciales

Definición 1: Es una Base de Datos que es optimizada para almacenar y consultar información relacionada a objetos en el espacio, incluyendo puntos, líneas y polígonos.

Definición 2: Es una colección de datos espaciales y no espaciales que están interrelacionados.

Deben permitir la descripción de los objetos espaciales mediante tres características:

- **Atributos:** qué es un objeto de acuerdo a sus características.
- **Localización:** conocer dónde está el objeto y qué lugar ocupa.
- **Topología:** mejorar la interpretación semántica del contexto y establecer jerarquías.

DBMS Espacial

- *Un SDBMS es un DBMS.*
- *Ofrece datos espaciales (SDTs) en su modelo de datos y su lenguaje de consulta.*
- *Soporta tipos de datos espaciales en su implementación, proveyendo al menos indexación espacial y algoritmos eficientes para joins espaciales.*

Un *SDBMS* es un *DBMS*. Suena trivial; refleja el hecho de que la información espacial o geométrica está, en práctica, conectada con la información *no-espacial*. El *SDBMS* tiene propiedades adicionales para manejar datos espaciales.

Tipos de datos espaciales: POINT, LINE, REGION, etcétera.

Proveen una abstracción fundamental para modelar la estructura de entidades geométricas en el espacio, así como sus relaciones, propiedades y operaciones.

Objetivos de un SDBMS

Deben integrar la representación y manipulación de datos geométricos y espaciales con los datos no espaciales en un nivel lógico y además proveer un soporte eficiente para almacenar y procesar datos a nivel físico.

Requerimientos de un SDBMS

- El lenguaje de consulta debe incorporar nuevas funciones sobre componentes geométricos.
- Utilizar métodos de acceso eficientes.
- Incorporar nuevos algoritmos para procesar consultas que satisfacen combinaciones de restricciones.
- La representación lógica debe ser extendida a datos geométricos.
- Capacidad de manejar grandes colecciones de objetos geométricos.

Hacia un modelo de datos espacial

Asumiendo aplicaciones 2-d y GIS, existen dos aspectos importantes que necesitan ser representados:

- **Objetos en el Espacio:** distinguir entre entidades espaciales de acuerdo a su representación geométrica.

- **Espacio:** describir cada punto del espacio, es decir, tener una idea de la representación del espacio mismo.

Modelando Datos Espaciales

Los datos espaciales tienen las siguientes desventajas:

- Estructura compleja.
- Las SBD tienden a ser muy grandes.
- No existe un álgebra estándar.
- Operadores no cerrados.
- Costo computacional elevado.

Es por ello que hay que implementar algoritmos y estructuras de datos que optimicen la manipulación de los mismos.

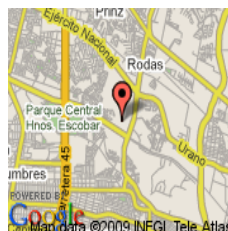
Objetos en el Espacio

Son objetos únicos e individuales. Para modelarlos, las abstracciones fundamentales son el *punto*, la *línea* y la *región* (polígono).



Punto

Un punto representa el aspecto geométrico de un objeto para el cual lo único relevante es su localización en el espacio. Por ejemplo, una ciudad puede ser modelada como un punto en un mapa.



Línea

Una línea representa es la abstracción básica del movimiento a través del espacio, o conexiones en espacio. Por ejemplo, ríos, carreteras, cables de teléfono, etcétera. Usualmente representadas como *polilíneas*, i.e, secuencias de segmentos.

Región o polígono

Una región es una abstracción para algo que tenga una extensión en el espacio 2-d. Por ejemplo un país, un lago, un parque nacional, etcétera. Una región puede tener huecos y puede ser conformada por distintos objetos del espacio.

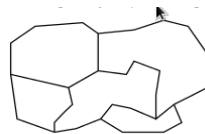


Espacio

Hay que caracterizar los puntos del espacio. De esta forma, podemos modelar mapas temáticos que indiquen por ejemplo, la división de un país en provincias. Existen dos importantes instancias en las colecciones de objetos relacionadas espacialmente (espacio):son las *particiones* y las *redes*.

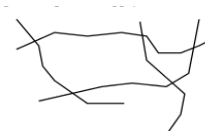
Particiones

Una *partición* puede ser vista como un conjunto de regiones que son disjuntas. La adyacencia en este caso tiene un interés particular; pares de regiones que tienen frontera común. Pueden ser usadas para representar mapas.



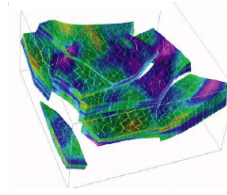
Redes

Una red puede ser vista como un grafo embebido en el plano. Está compuesta por un conjunto de objetos puntos (nodos) y un conjunto de objetos líneas (aristas). Pueden ser usadas para representar elementos geográficos como carreteras, ríos, transportes públicos, etcétera.



Otras abstracciones

Hemos mencionado hasta aquí las más comunes. Obviamente, existen otras como las *particiones anidadas* (e.g. un país dividido en provincias, y a su vez cada provincia en estados), o los modelos de terrenos digitales, para representar relieves.



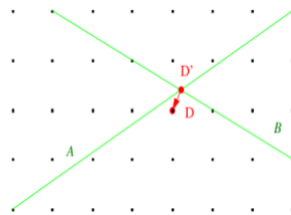
Organizando el Espacio

La geometría Euclidiana no es adecuada como una base para modelar en las bases de datos espaciales.

- El espacio Euclídeo es **continuo**, i.e., los puntos son representados como un par de números reales: $p = (x, y) \in \mathbf{R}^2$.
- Pero la computadora trabaja en forma **discreta**.
 - \Rightarrow el espacio es representado por un *raster discreto*.

Ejemplo: Intersección de dos líneas

- El punto de intersección es redondeado al punto de la grilla más cercano.
- Una prueba para determinar si el punto de intersección está en una de las líneas determina un resultado falso.



Solución: Definir una base geométrica discreta para modelar el espacio, así también como para lidiar con su implementación.

Dos intentos para una base geométrica:

- **Simplicial complexes** (Frank & Kuhn)
- **Realms** (Güting & Schneider)

Organizando el Espacio: Simplicial Complexes

Conceptos básicos de **simplicial complexes**

- **d-simplex:** Un objeto de dimensión d (ver Figura 4.5).
- **Simplicial Complex:** Un conjunto finito de simplices tales que la intersección de dos simplices cualquiera en el conjunto es un componente usado en los simplices (ver Figura 4.6).

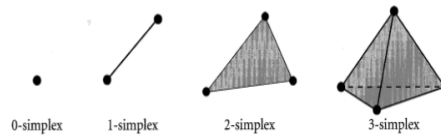


Figura 4.5: Simplicies de diferentes dimensiones

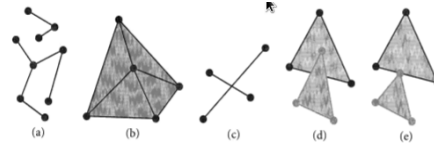


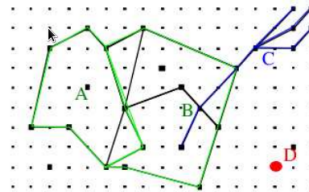
Figura 4.6: Intersección de simplicies formando componentes

Organizando el Espacio: Realm

Realm: Un conjunto finito de puntos y segmentos de líneas definidos sobre una grilla tal que:

- cada punto o punto final de los segmentos es un punto en la grilla
- cada punto final de un segmento es un punto del realm
- ningún punto del realm se encuentra dentro de un segmento
- dos segmentos no se intersecan salvo en sus puntos finales

Realm provee una descripción completa de la geometría (puntos y líneas), como puede observarse en la siguiente figura:



Tipos de Datos y Álgebras Espaciales

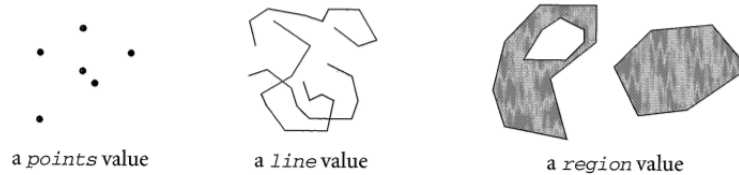
Las básicas abstracciones espaciales pueden ser embebidas en un DBMS existente usando tipos abstractos de datos.

- **Tipos de Datos Espaciales** encapsulan
 - la estructura de un objeto espacial, e.g., región
 - operaciones sobre esas estructuras
- **Álgebra Espacial**, es una colección de datos espaciales con operaciones relacionadas
 - Completitud y clausura deben ser propiedades del álgebra

Álgebra ROSE

El álgebra de ROSE (RObust Spatial Extension, Güting & Schneider) es un álgebra espacial con tipos de datos espaciales basados en el modelo Realm (esto es, objetos compuestos por elementos Realm). Este álgebra utiliza como tipos de datos puntos, líneas y regiones.

Figura 4.7: Tipos de datos del álgebra de ROSE



Álgebra ROSE: Operaciones

Teniendo en cuenta los siguientes conjuntos:

$\text{EXT} = \{\text{líneas, regiones}\}$ $\text{GEO} = \{\text{puntos, líneas, regiones}\}$

podemos definir las operaciones sobre el álgebra de ROSE de la siguiente manera:

- Predicados espaciales para relaciones topológicas:
 - **inside**: $\text{geo} \times \text{regions} \rightarrow \text{bool}$
 - **intersect**, **meets**: $\text{ext1} \times \text{ext2} \rightarrow \text{bool}$
 - **adjacent**, **encloses**: $\text{regions} \times \text{regions} \rightarrow \text{bool}$
- Operaciones que devuelven tipos de datos espaciales atómicos:
 - **intersection**: $\text{lines} \times \text{lines} \rightarrow \text{points}$
 - **intersection**: $\text{regions} \times \text{regions} \rightarrow \text{regions}$
 - **plus**, **minus**: $\text{geo} \times \text{geo} \rightarrow \text{geo}$
 - **contour**: $\text{regions} \rightarrow \text{lines}$
- Operadores espaciales que devuelven números:
 - **dist**: $\text{geo1} \times \text{geo2} \rightarrow \text{real}$
 - **perimeter**, **area**: $\text{regions} \rightarrow \text{real}$
- Operadores espaciales en conjuntos de objetos:
 - **sum**: $\text{set}(\text{obj}) \times (\text{obj} \rightarrow \text{geo}) \rightarrow \text{geo}$
 - Alguna función espacial agregada como por ejemplo la suma de las áreas de las provincias determinan el área de un país.
 - **closest**: $\text{set}(\text{obj}) \times (\text{obj} \rightarrow \text{geo1}) \times \text{geo2} \rightarrow \text{set}(\text{obj})$
 - Determina dentro de un conjunto de objetos aquellos cuyo atributo espacial tiene mínima distancia.
 - Otras operaciones complejas...

Álgebra ROSE: Propiedades

Estos ejemplos son suficientes para ver qué tipo de operaciones están disponibles en un álgebra espacial.

Algunas propiedades que debe tener el álgebra espacial son:

- Extensibilidad
- Completitud
- ¿Uno o más tipos?
- Operaciones de conjuntos

Relaciones Espaciales

Entre las operaciones espaciales, podemos distinguir las *relaciones*.

- Topológicas: adjacent, inside, disjoint. Son invariantes dentro de transformaciones topológicas como scaling, rotation, translation.
- De dirección: above, below, north_of.
- Métricas: distance < 100.

Seis relaciones topológicas válidas entre dos regiones simples (sin huecos, conexas): *disjoint*, *in*, *touch*, *equal*, *cover*, *overlap*.

Integrando la Geometría en el Modelo de Datos

Los tipos de datos espaciales pueden ser embebidos en cualquier modelo de datos, e.g., el modelo relacional. El modelo de datos del DBMS debe ser extendido al nivel de tipos de datos atómicos, como strings y enteros. La idea básica es representar **objetos espaciales** con objetos (del modelo de datos del DBMS) con al menos un atributo espacial.

Los objetos espaciales son tuplas con al menos un atributo espacial.

```
relation estados (ename: STRING; area:REGION; epop: INTEGER)
relation ciudades (cname: STRING; centro:POINT; ext:REGION; cpop: INTEGER)
relation rios (rname: STRING; ruta:LINE)
```

Consultas

Dos problemas principales:

- Conectar las operaciones del álgebra espacial (incluyendo los predicados de relaciones espaciales) a las facilidades de un lenguaje de consulta de un DBMS.

Los operadores fundamentales son:

- Spatial selection
- Spatial join

– Overlay, fusion, ...

- Proveer una representación gráfica de la información espacial (o sea, los resultados de la consulta) y además una entrada gráfica para los valores en las consultas.

Consultas: Selección Espacial

Selección Espacial: retorna aquellos objetos que satisfacen un predicado espacial en la consulta.

Todas las ciudades de Argentina

```
SELECT cname FROM ciudades c
WHERE c.centro inside Argentina.area
```

Todas las grandes ciudades en un radio de 100km de Rosario

```
SELECT cname FROM ciudades c
WHERE dist(c.centro, Rosario.centro) <100
AND c.pop >500k
```

Consultas: Join Espacial

Join Espacial: un join que compara cualesquiera dos objetos que fueron *joinados* basado en un predicado que toma los valores espaciales de ambos.

Para cada río de Argentina, encontrar todas las ciudades dentro de 50 kms

```
SELECT r.rname, c.cname, length(intersection(r.ruta, c.area))
FROM rios r, ciudades c
WHERE r.ruta intersects Argentina.area
AND dist(r.ruta, c.area) <50
```

Consultas: Otras operaciones

Overlay: retorna las regiones que son el resultado de sobreponer dos particiones.

Fusion: es una forma especial de agrupación. Para cada grupo obtenido, se forma la unión de todos los valores de un atributo espacial.

Voronoi: computa de una colección de objetos puntos S la correspondiente colección de objetos región. Para cada punto p , la región consiste de los puntos del plano más cercanos a p , y que no estén más cercanos a otro punto de S .

Consultas: I/O Gráfica

¿Cómo determinar *Argentina* en los ejemplos anteriores(input), o cómo mostrar *intersection(ruta,Argentina.area)* o *rio.ruta(output)*?

Requerimientos para consultas espaciales:

- Tipos de datos espaciales.
- Muestreo gráfico de los resultados de consulta.

- Combinación gráfica de varios resultados de consultas.
- Herramientas para interactuar con gráficos de entrada y resultados.
- Facilidad para verificar el contenido de una muestra.

Formato de Datos Espaciales

Existen dos estándares de representación:

- **WKT** (Well Known Text)
- **WKB** (Well Known Binary)

Formato de Datos Espaciales: WKT

Esta representación es una codificación en formato ASCII para describir objetos espaciales. Es usada por *PostgreSQL* en *PostGIS*, y también en Google Maps. Podemos describir puntos, multipuntos, líneas, polígonos, multipolígonos, puntos en 3 dimensiones, etcétera.

Sintaxis:

- Punto: `POINT(30 50)`
- Línea: `LINESTRING(1 1, 5 5, 10 10)`
- Polígono: `POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0),(20 20, 20 40, 40 40, 40 20, 20 20))`

Formato de Datos Espaciales: WKB

WKB utiliza enteros sin signo de un byte, enteros sin signo de cuatro bytes, y números de ocho bytes de doble precisión (formato IEEE 754). Un byte son ocho bits. Por ejemplo, un valor WKB que corresponde a un `POINT(1 1)` consiste en esta secuencia de 21 bytes (cada uno representado aquí por dos dígitos hexadecimales):

0101000000000000000000F03F000000000000F03F

Orden de byte	01 → LE o BE
Tipo WKB	01000000 → tipo de objeto
X	000000000000F03F
Y	000000000000F03F

Más rápida que WKT, pero menos entendible para el usuario.

Indexación Espacial

Organiza el espacio y los objetos en él de forma tal que sólo son considerados la parte del espacio y los objetos que sean relevantes a una consulta. Básicamente, es un mecanismo para decrementar el número de búsquedas. El objetivo principal es tratar de no visitar y tocar cada vez a todos los n elementos en la BD. Los índices

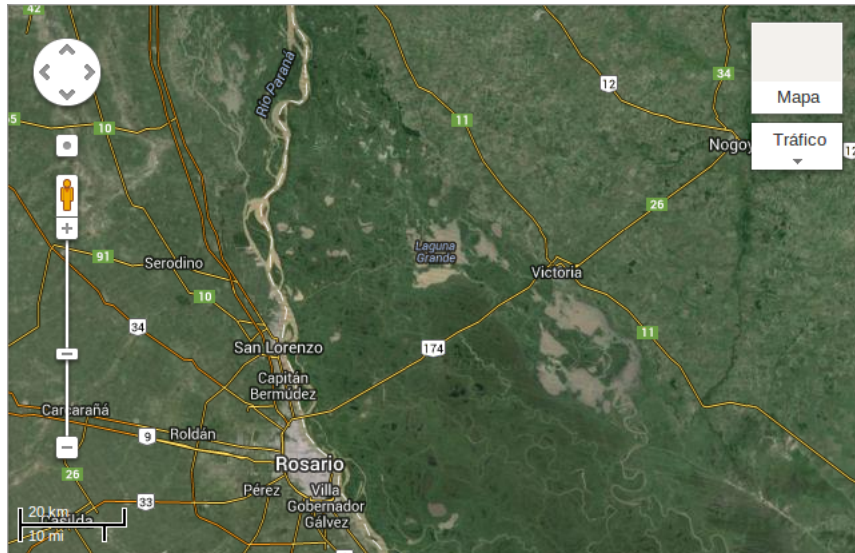


Figura 4.8: Imágen de Google Maps donde se utilizan métodos de indexación espacial

espaciales nos permiten optimizar la recuperación.

El uso principal de la indexación es para la selección espacial, pero es usado también para otras operaciones como join espacial o encontrar el objeto más cercano. Dos enfoques principales de la indexación espacial:

- Mapear objetos espaciales al espacio 1-D y utilizar técnicas estándar de indexación, como B-trees
- Estructuras de indexación espacial dedicadas, como R-trees

Indexación Espacial: R-Tree

R-Tree es una estructura de datos similar a los *B-Trees*, con la diferencia que se utilizan para métodos de acceso espacial. Es decir, podemos indexar coordenadas por ejemplo.

Cada nodo tiene un número variable de entradas, y cada entrada almacena dos datos: una forma de identificar a un nodo hijo y el conjunto límite de todas las entradas de ese nodo hijo.

Búsqueda: recursiva desde la raíz, se verifica si el rectángulo de la consulta se solapa con algún rectángulo del nodo.

Inserción: partiendo de la raíz, se usa una heurística para seleccionar el nodo candidato.

La idea fundamental de la indexación espacial es la aproximación.

- Aproximación continua
- Aproximación por grilla

Para procesar una consulta, se procede a la estrategia de **filtrado y refinamiento**.

1. Filtrado: retorna un conjunto (donde está la solución) de los valores posibles que satisfacen un predicado

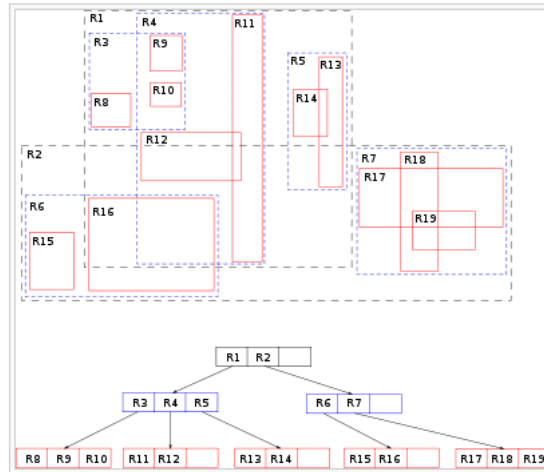


Figura 4.9: Estructura del R-Tree

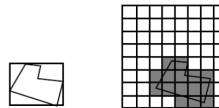


Figura 4.10: Aproximación continua a la izquierda, aproximación por grilla a la derecha

2. Refinado: Para cada candidato, se refina la búsqueda chequeando la geometría

Las estructuras espaciales son diseñadas para almacenar puntos (para objetos puntos) o rectángulos (líneas y regiones). Las operaciones sobre esas estructuras son inserción, borrado y pertenencia.

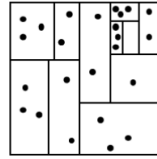
Consultas típicas:

- Puntos
 - Consulta de rango: todos los puntos dentro de un rectángulo
 - Vecino más cercano: el punto más cercano a un punto consultado
 - Distancia: desde un punto, enumerar los puntos que están a cierta distancia
- Rectángulos
 - Intersección
 - Contenido

Indexación Espacial: Estructuras

- Una estructura de indexación espacial dedicada organiza los objetos en **cubetas**.
- Cada cubeta tiene asociada una **región**, que es, una parte del espacio que contiene todos los objetos almacenados en esa cubeta.

- Para estructuras de datos de puntos, las regiones son disjuntas
 - el espacio es particionado y cada punto pertenece a solo una partición (cubeta)

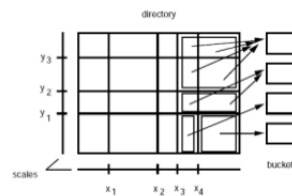


- Para estructuras con rectángulos, las cubetas pueden solaparse

Indexación Espacial: Estructura de Puntos

Estructuras espaciales para puntos: en k dimensiones son representadas como una tupla $t = (x_1, \dots, x_k)$

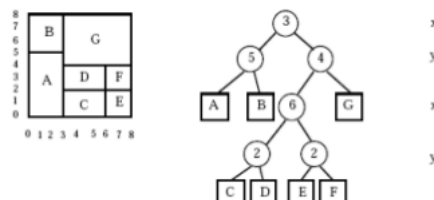
Índice de Grilla: Estructura de indexación espacial para puntos (Nievergelt, Hinterberger y Sevcik 84)



- El directorio es un arreglo k -dimensional cuyas entradas son punteros a las cubetas.
- Todos los puntos en las celdas son almacenados en cubetas apuntando a la entrada de directorio correspondiente.

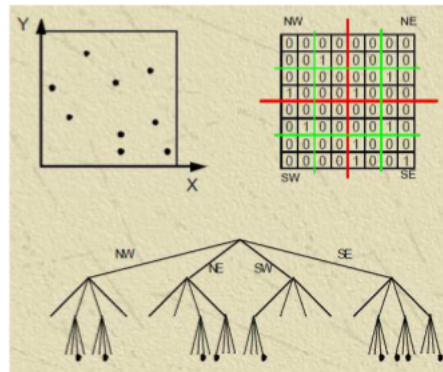
kd-tree:

Árbol binario donde cada nodo interno contiene un valor clave de una dimensión. La clave en el nodo raíz (nivel 0) divide el espacio de datos con respecto a la dimensión 0, las claves en sus hijos (nivel 1) divide los dos subespacios y así sucesivamente hasta que se terminan las dimensiones, luego se reinicia el ciclo.



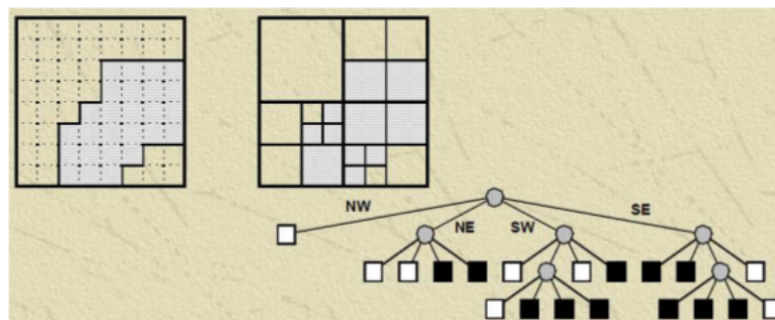
quad-tree:

Divide el espacio de datos en forma recursiva en 4 cuadrantes (NO, NE, SO, SE)



quad-tree:

Hay diferentes algoritmos para procesar puntos, líneas, polígonos (i.e., distintos tipos de nodos, algoritmos de consultas). Es usado muy frecuentemente en GIS comerciales para comprimir, almacenar y manipular imágenes raster.



Indexación Espacial: Estructura de Rectángulo

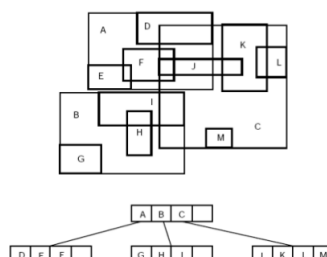
Dos enfoques principales:

- Solapamiento de regiones
- Clipping (Recorte)

Indexación Espacial: Estructura de Rectángulo

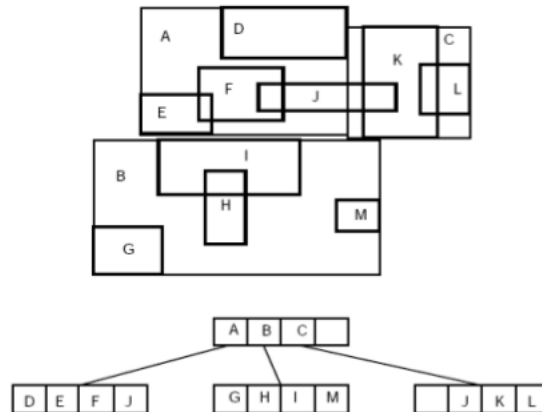
Solapamiento de regiones: se abandona el espacio particionado y las regiones pueden solaparse, e.g., **R-tree**.

- Ventaja: El objeto espacial (clave) está en una sola cubeta.
- Desventaja: Muchos caminos terminan en un solapamiento de cubetas.



Recorte (Clipping): las cubetas son disjuntas, pero los rectángulos son cortados en varias piezas, e.g., **R⁺-tree**.

- Ventaja: Menos ramas en el árbol
- Desventaja: Múltiples entradas para un objeto espacial



Estado del Arte: Las BDE en la actualidad

Spatial SQL

SQL espacial brinda soporte para nuevos tipos de datos como:

- Point
- LineString
- Polygon
- MultiPoint
- etcétera...

Relaciones entre objetos:

- ST_Contains
- ST_Equals
- ST_Intersects
- ST_Touches
- etcétera...

Constructores de objetos:

- ST_MultiLineString
- ST_Point
- ST_MultiPolygon
- ST_MultiCurve
- etcétera...

Spatial SQL: Ejemplo

Crear una tabla nueva:

```
CREATE TABLE Rio(  
Nombre varchar(30),  
Origen varchar(30),  
Longitud number,  
Forma LineString );
```

Consulta: encontrar nombres de países que son vecinos de Estados Unidos en la tabla Países.

```
SELECT P1.Nombre  
FROM Países P1, Países P2  
WHERE Touch(P1.Forma,P2.Forma)=1  
AND P2.Nombre = USA
```

Consulta: encontrar la distancia entre dos puntos, dada una tabla de puntos.

```
SELECT ST_Distance(geometrycolumn,  
ST_GeomFromText('POINT(1 2)',4326)  
FROM tabladepuntos  
ORDER BY  
ST_Distance(geometrycolumn,  
ST_GeomFromText('POINT(1 2)',4326) LIMIT 10)
```

Alternativas Espaciales

Algunas de las elecciones que podemos hacer hoy en día en materia de DBMS espaciales son:

Open Source

- MySQL (actualmente viene con soporte espacial)
- MongoDB (no relacional)
- MariaDB
- PostgreSQL + PostGIS
- SpatialDB
- y más ...

Pagos

- Oracle
- SQL Server

PostgreSQL + PostGIS



PostgreSQL

PostgreSQL es un sistema administrador de bases de datos objeto-relacional basado en POSTGRES, desarrollado en la Universidad de California en Berkeley, en el Departamento de Ciencias de la Computación. POSTGRES fue pionero de muchos conceptos que solo estuvieron disponibles en algunos sistemas de bases de datos comerciales mucho más tarde. PostgreSQL es un proyecto desarrollado con código abierto que soporta el estándar SQL y ofrece muchas características interesantes:

- Consultas complejas
- Claves foráneas
- Disparadores
- Vistas
- Integridad transaccional
- Control de concurrencia

Además PostgreSQL puede ser ampliado por el usuario de muchas formas, mediante la adición de nuevas definiciones de:

- Tipos de datos
- Funciones
- Operadores
- Funciones de agregado
- Métodos indexados
- Lenguajes procedurales

Debido a que es de licencia libre, PostgreSQL puede ser usado, modificado y distribuido por todo el mundo de forma gratuita para cualquier fin, ya sea privado, comercial o académico. Es considerado como uno de los mejores gestores de bases de datos de software libre. Muchas empresas han iniciado el uso de esta herramienta beneficiándose en la reducción de los costos y en el aumento de la fiabilidad.

PostGIS PostGIS es una extensión del sistema de bases de datos objeto-relacional PostgreSQL, que permite el uso de objetos geográficos. Fue creado por Refractor Research Inc, como un proyecto de investigación de tecnología de bases de datos espaciales.

Geometrías básicas soportadas:

- POINT (x y)
- LINESTRING (x₁ y₁, ...,x_n y_n)
- POLYGON (x₁ y₁, ...,x_n y_n)

También brindan soporte para multi-geometrías:

- MULTIPOINT ((POINT₁), ..., (POINT_n))
- MULTILINESTRING ((LINESTRING₁), ..., (LINESTRING_n))
- MULTIPOLYGON ((POLYGON₁), ..., (POLYGON_n))

Los constructores, operadores y relaciones son los que vienen con Spatial SQL, es decir, los mencionados anteriormente. *PostGIS* agrega características interesantes como su propio spatial join, GIS overlay functions, primitivas 2D y 3D, además de un entorno de desarrollo amigable.

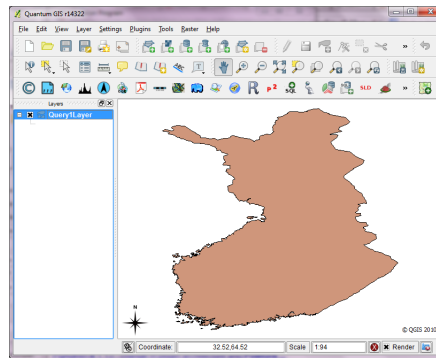


Figura 4.11: Entorno de desarrollo PostGIS

Bases de Datos Espacio-Temporales

Bibliografía

- [1] “Snodgrass R. (1986). Temporal Databases.”
- [2] “TSQL2 Official Webpage.” <http://digital.cs.usu.edu/~cdyreson/pub/temporal/tsql2.htm>.