

Bases de Datos Temporales

DOCENTES:

- TROSSERO, SEBASTIÁN
- SCHMUCKLER, JORGE

Resumen

- ❑ Introducción
- ❑ Tratamientos de tiempos
- ❑ Standard SQL:2011
- ❑ Propuesta SQLServer
- ❑ Propuesta de Oracle
- ❑ Propuesta de PostgreSQL

Introducción

Conceptos básicos

- ❑ **Base de datos temporal:** es un SGBD que implementa un tratamiento específico para datos que son variantes en el tiempo. Debe permitir un correcto almacenamiento y acceso a los datos con sus respectivos cambios de valores en el tiempo.
- ❑ **Tiempo de validez:** es un atributo que se refiere al periodo de tiempo que un hecho es válido en el mundo real.
- ❑ **Tiempo de transacción:** indica el periodo de tiempo en el cual un hecho está guardado en la base de datos.
- ❑ **Dato Bitemporal:** es la combinación del tiempo de validez y el tiempo de transacción.

Introducción

Conceptos básicos

□ **Partición Horizontal:** Consiste en una estrategia dónde separamos las filas correspondientes a una tabla de la base de datos en distintas tablas.

Ejemplo:

Tabla: Empleados

ID	nombre	fecha_nacimiento	sueldo
1001	Ana García	1985-06-15	5000
1002	Carlos Sánchez	1990-09-23	3000
1003	María Rodríguez	1982-04-08	3500
1004	Juan Pérez	1978-11-12	5000

Tabla: Empleados A-L

ID	nombre	fecha_nacimiento	sueldo
1001	Ana García	1985-06-15	5000
1002	Carlos Sánchez	1990-09-23	3000

Tabla: Empleados M-Z

ID	nombre	fecha_nacimiento	sueldo
1003	María Rodríguez	1982-04-08	3500
1004	Juan Pérez	1978-11-12	5000

Introducción

Conceptos básicos

❑ **Partición Vertical:** Consiste en una estrategia dónde cada partición contiene un subconjunto de columnas de las que componen una tabla.

Ejemplo

Tabla: Empleados

ID	nombre	fecha_nacimiento	sueldo
1001	Ana García	1985-06-15	5000
1002	Carlos Sánchez	1990-09-23	3000
1003	María Rodríguez	1982-04-08	3500
1004	Juan Pérez	1978-11-12	5000

Tabla: Empleados

ID	nombre	fecha_nacimiento
1001	Ana García	1985-06-15
1002	Carlos Sánchez	1990-09-23
1003	María Rodríguez	1982-04-08
1004	Juan Pérez	1978-11-12



Tabla: Empleados Sueldo

ID	sueldo
1001	5000
1002	3000
1003	3500
1004	5000

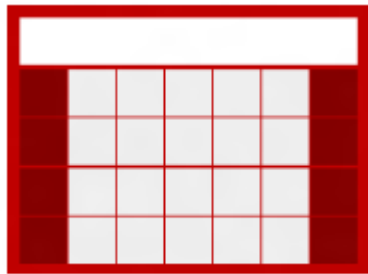
*PK: se repite en ambas particiones

Introducción

Idea sin soporte temporal

Escenario: administrar tiempo de transacción y tiempo de validez (bitemporalidad) en la tabla empleados.

Empleados



```
CREATE TABLE Empleados (  
    n_legajo INTEGER,  
    nombre VARCHAR(60),  
    fecha_nacimiento DATE,  
    sueldo DECIMAL(12,2),  
    tt tstzrange,  
    tv granorange  
);
```

- ❑ ¿Qué datos son variables en el tiempo?
- ❑ ¿Cómo registrar tiempo de validez?
- ❑ ¿Cómo registrar tiempo de transacción?

Introducción

Idea sin soporte temporal

❑ INSERT:

```
INSERT INTO empleados (n_legajo, nombre, fecha_nacimiento, sueldo, tt, tv)
VALUES ( 17, 'Nombre', 'xxxx-xx-xx', 9999, '---- --tt', '---- --tv);
```

❑ UPDATE:

```
UPDATE empleados
  SET sueldo = 99999,
      tt = '----',
      tv = '----'
WHERE n_legajo = 17
  AND tt = '----'
  AND tv = '----');
```

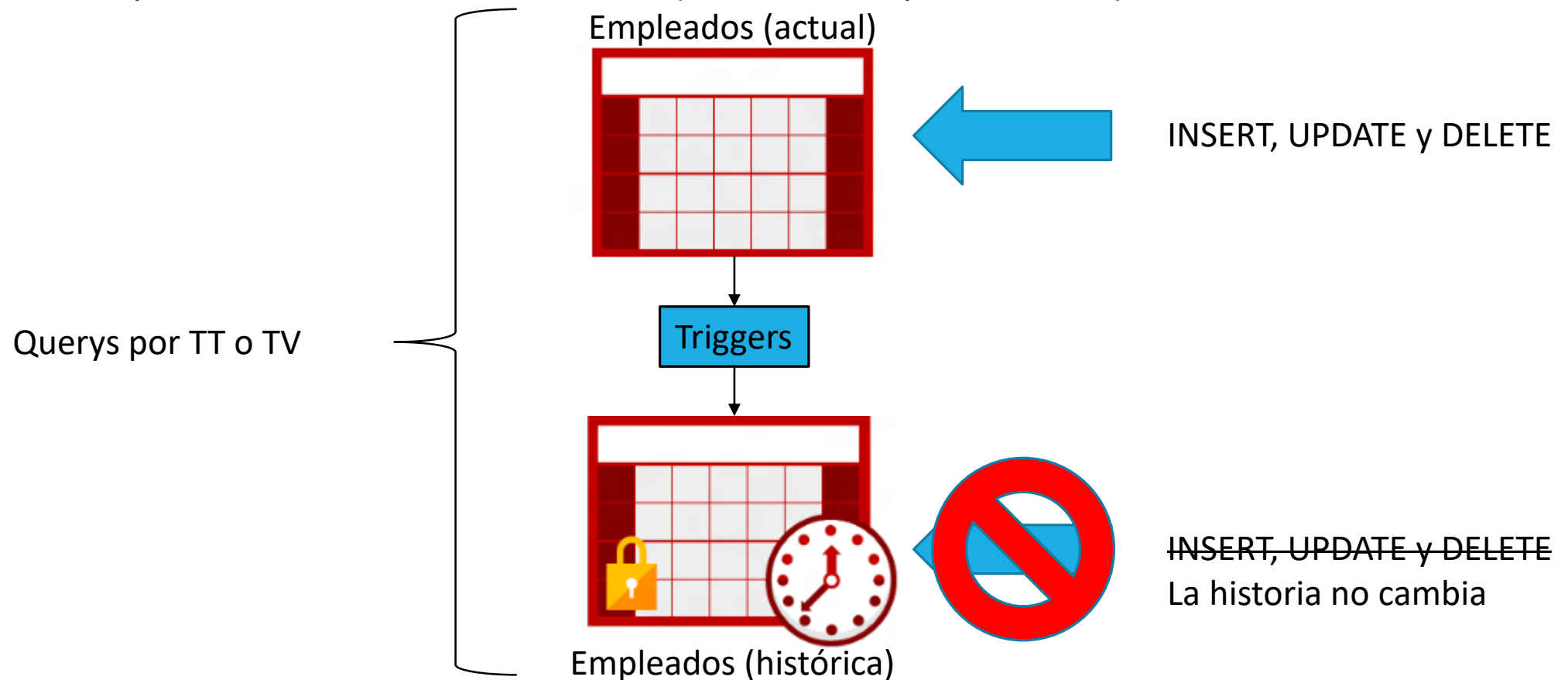
❑ DELETE:

```
DELETE FROM empleados WHERE n_legajo = 17 AND tt = '----' AND tv = '----'
```

Introducción

Idea sin soporte temporal

❑ ¿Qué implementación de tablas usamos (monolítica o particionada)?



Resumen

- ❑ Introducción
- ❑ Tratamientos de tiempos
- ❑ Standard SQL:2011
- ❑ Propuesta SQLServer
- ❑ Propuesta de Oracle
- ❑ Propuesta de PostgreSQL

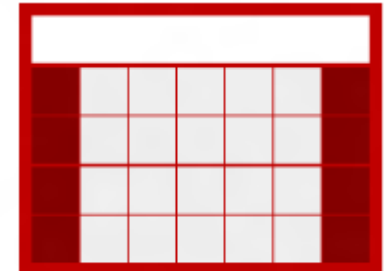
Tratamientos de tiempos

Tiempo de Transacción

Tabla actual o principal: Contendrá datos actuales y los triggers necesarios.

```
CREATE TABLE Empleados (  
    n_legajo INTEGER,  
    nombre VARCHAR(60),  
    fecha_nacimiento DATE,  
    sueldo DECIMAL(12,2),  
    tt tstzrange DEFAULT, tstzrange (current_timestamp, null)  
);
```

Empleados (actual)



* Si el SGBD no permite un tipo de datos “rango”, se debería cambiar por columnas “ttdesde” y “tthasta” y modificar las validaciones que se verán a continuación

Tratamientos de tiempos

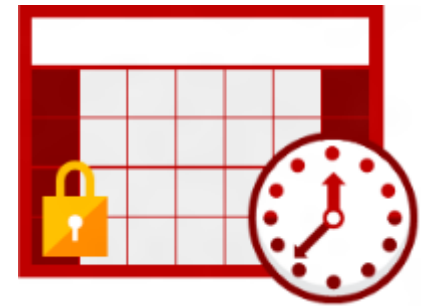
Tiempo de Transacción

Tabla Histórica: Contendrá el pasado.

```
CREATE TABLE EmpleadosHistory (  
    n_legajo INTEGER,  
    nombre VARCHAR(60),  
    fecha_nacimiento DATE,  
    sueldo DECIMAL(12,2),  
    tt tstzrange  
);
```

```
CREATE TABLE EmpleadosHistory LIKE Empleados; --alternativa
```

Empleados (histórica)



*Implementación de partición horizontal, el diseño de columnas es igual a la tabla actual.

Tratamientos de tiempos

Tiempo de Transacción - INSERT

❑ Análisis de INSERT

```
INSERT INTO empleados (n_legajo, nombre, fecha_nacimiento, sueldo)
VALUES (17, 'Juan', '1999-06-02', 8000);
```

¿Qué debería suceder en la tabla actual?

¿Y en la tabla histórica?

Tratamientos de tiempos

Tiempo de Transacción - INSERT

TABLA ACTUAL	
Cambios	Se registran los datos insertados
Tiempo de transacción – Inicio	Momento de ejecución de la instrucción
Tiempo de transacción – Fin	Imposible predecir => NULL, INFINITE, '9999-12-31 23:59:59....'

TABLA HISTÓRICA	
Cambios	No hay acciones en esta tabla

Tratamientos de tiempos

Tiempo de Transacción - INSERT

Aprovechando los valores por default definidos para el campo “tt”, no se necesitaría ningún tipo de administración, pero nada evitaría que se pueda insertar un valor distinto que el default y se insertaría un dato incorrecto (no es el tiempo real de transacción).

Otro punto a tener en cuenta es que no se debería permitir actualizar el valor de “tt” este debería ser manejado solamente de manera **automática**

Tratamientos de tiempos

Tiempo de Transacción - INSERT

```
CREATE FUNCTION fxemp_before_insert_update_tt()  
  RETURNS trigger AS $$  
  BEGIN  
    IF tg_op = 'UPDATE' THEN      --evitar que el update cambie el tt  
      new.tt = old.tt;  
      RETURN new;  
    END IF;  
  
    --seteamos que el valor en un insert sea desde ahora hasta "infinito"  
    new.tt = tstzrange(current_timestamp, null);  
  
    RETURN new;  
  END;  
$$ LANGUAGE 'plpgsql';
```

```
CREATE TRIGGER trgemp_before_insert_update_of_tt  
  BEFORE INSERT OR UPDATE OF tt ON empleados  
  FOR EACH ROW EXECUTE PROCEDURE fxemp_before_insert_update_tt();
```

Tratamientos de tiempos

Tiempo de Transacción - DELETE

❑ Análisis DELETE:

```
DELETE FROM empleados WHERE n_legajo = 17
```

¿Qué debería suceder en la tabla actual?

¿Y en la tabla histórica?

Tratamientos de tiempos

Tiempo de Transacción - DELETE

TABLA ACTUAL	
Cambios	La fila deja de existir en la tabla actual

TABLA HISTÓRICA	
Cambios	Se trasladan los datos de la tabla actual a la tabla histórica
Tiempo de transacción – Inicio	Mismo “tti” que tenía en actual
Tiempo de transacción – Fin	Momento de ejecución de la instrucción - 1 grano de tiempo

Tratamientos de tiempos

Tiempo de Transacción - DELETE

```
CREATE FUNCTION fxemp_after_delete()  
  RETURNS trigger AS $$  
  BEGIN  
  
    INSERT INTO EmpleadosHistory  
      SELECT old.dni, old.nombre, old.fecha_nacimiento, old.sueldo,  
        tstzrange(lower(old.tt),current_timestamp,'[]');  
  
    --postgres nos permite definir intervalo cerrado-cerrado con '[]'  
    --nos evita restarle un grano de tiempo al TTF  
  
    RETURN new;  
  END;  
  $$ LANGUAGE 'plpgsql';
```

```
CREATE TRIGGER trgemp_after_delete  
  AFTER DELETE ON empleados  
  FOR EACH ROW EXECUTE PROCEDURE fxemp_after_delete();
```

Tratamientos de tiempos

Tiempo de Transacción - UPDATE

□ Análisis UPDATE

```
UPDATE empleados SET sueldo = 99999 WHERE n_legajo = 17;
```

¿Qué debería suceder en la tabla actual?

¿Y en la tabla histórica?

Tratamientos de tiempos

Tiempo de Transacción - UPDATE

TABLA ACTUAL	
Cambios	Se actualizan los datos
Tiempo de transacción – Inicio	Momento de ejecución de la instrucción (se “resetea”)
Tiempo de transacción – Fin	Imposible predecir => NULL, INFINITE, ‘9999-12-31 23:59:59....’

TABLA HISTÓRICA	
Cambios	Se trasladan los datos previos de la tabla actual a la tabla histórica
Tiempo de transacción – Inicio	Mismo “tti” que tenía en actual
Tiempo de transacción – Fin	Momento de ejecución de la instrucción - 1 grano de tiempo

Tratamientos de tiempos

Tiempo de Transacción - UPDATE

En resumen, además de ejecutar la instrucción del UPDATE debemos resetear el “TTI”, recordar que pusimos un control para que no se pueda modificar el tt directamente desde un pero si podremos hacerlo desde una función.

Además del control de la tabla actual, debemos enviar los datos anteriores a la tabla histórica y cerrar el periodo de tiempo de transacción. Si lo analizamos la operación es exactamente igual a la del DELETE, podemos reutilizar esta función

Tratamientos de tiempos

Tiempo de Transacción - UPDATE

```
CREATE FUNCTION fxemp_before_update_datos ()  
  RETURNS trigger AS $$  
  BEGIN  
    --Reseteo del tiempo de transacción  
    new.tt = tstzrange(current_timestamp ,null);  
  
    RETURN new;  
  END;  
  $$ LANGUAGE 'plpgsql';
```

```
CREATE TRIGGER trgemp_before_update_datos  
  BEFORE UPDATE OF dni, nombre, fecha_nacimiento, sueldo ON empleados  
  FOR EACH ROW EXECUTE PROCEDURE fxemp_before_update_datos();
```

```
CREATE TRIGGER trgemp_after_update_datos  
  AFTER UPDATE OF dni, nombre, fecha_nacimiento, sueldo ON empleados  
  FOR EACH ROW EXECUTE PROCEDURE fxemp_after_delete(); --reutilizacion
```

Tratamientos de tiempos

Tiempo de Validez

Para dar soporte de tiempo válido a nuestro ejemplo, la propuesta sería agregar a ambas tablas otro rango de tiempo:

```
ALTER TABLE Empleados ADD tv daterange;  
ALTER TABLE EmpleadosHistory ADD tv daterange;
```

En este caso elegimos un grano de tiempo “día”, por eso el tipo se define como “daterange”

Tratamientos de tiempos

Tiempo de Validez - INSERT

El INSERT no exige tratamiento especial, simplemente se insertará la fila con los datos no temporales, mas los temporales de validez aportados por el usuario.

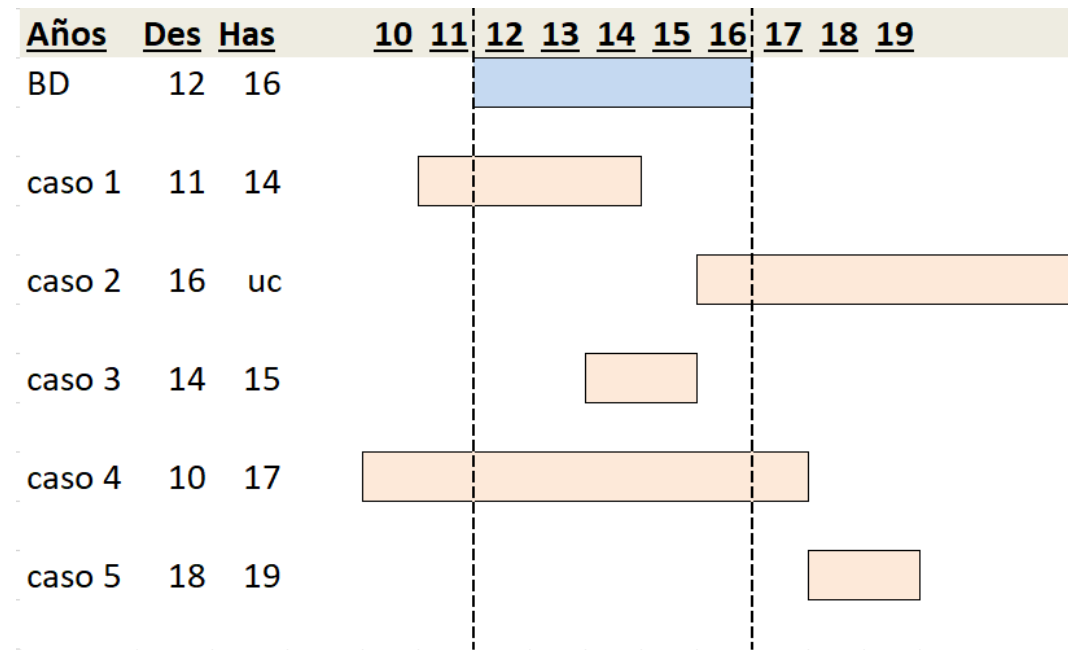
```
INSERT INTO empleados (n_legajo, nombre, fecha_nacimiento, sueldo, tv)
VALUES (17, 'Juan', '1999-06-02', 8000,
        Daterange('2012-04-14', '2016-11-8', '[]')) );
```

¿Pero que sucede con UPDATE y DELETE?

Tratamientos de tiempos

Tiempo de Validez - UPDATE

Comencemos analizando una serie de casos posibles:



Premisa: no debería haber más de un valor para un mismo momento

¿Cómo detectamos cada situación?

Tratamientos de tiempos

Tiempo de Validez - UPDATE

❑ Caso 1:

Años	Des	Has	10	11	12	13	14	15	16	17	18	19
BD	12	16										
caso 1	11	14										

bd.tv_inicio IN update.tv AND update.tv_fin IN bd.tv

❑ Caso 2:

Años	Des	Has	10	11	12	13	14	15	16	17	18	19
BD	12	16										
caso 2	16	uc										

update.tv_inicio IN bd.tv AND bd.tb_fin IN update.tv

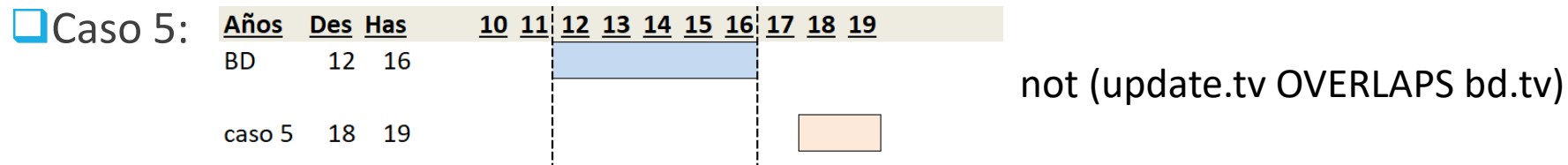
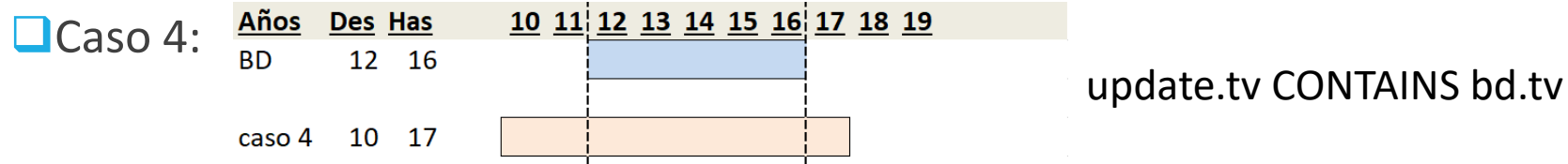
❑ Caso 3:

Años	Des	Has	10	11	12	13	14	15	16	17	18	19
BD	12	16										
caso 3	14	15										

update.tv DURING bd.tv

Tratamientos de tiempos

Tiempo de Validez - UPDATE



Los casos 4 y 5 no incurren en ningún conflicto, el primero reemplaza el valor anterior por toda la duración original y el segundo no tiene overlaps directamente

Tratamientos de tiempos

Tiempo de Validez - UPDATE

Analicemos la situación con plpgsql :

BdaAnalisisSuperposicionTramos.sql

```
Tramos -----Representacion ----- Rango --- ResultadoEst1 ResultadoEst2
-----
BD:          #####          [8,19)
Caso1:  *****          [2,14)  Superpuesto Superpuesto
Caso2:          *****          [10,23) Superpuesto Superpuesto
Caso3:  *****          [3,29)  Superpuesto Superpuesto
Caso4:          ***          [11,14) Superpuesto Superpuesto
Caso5:          *****          [22,27) No No
Caso6:  ***          [1,4) No No
```

- ❑ Estrategia 1: inicio(tramo2) BETWEEN inicio(tramo1) AND fin(tramo1) OR
 fin(tramo2) BETWEEN inicio(tramo1) AND fin(tramo1) OR
 inicio(tramo2) < inicio(tramo1) AND fin(tramo2) > fin(tramo1)
- ❑ Estrategia 2: Not(Inicio(tramo2) > Fin(tramo1) OR Inicio(tramo1) > Fin(tramo2))

Tratamientos de tiempos

Tiempo de Validez - UPDATE

Usando el operador Overlaps(2011) && y in <@: *BdaAnalisisSuperposicionTramosConOverlaps2011.sql*

Tramos	-----Representacion	-----	Rango ---	ResultadoEst1	ResultadoEst2
BD:	#####		[8,19)		
Caso1:	*****		[2,14)	Superpuesto	Superpuesto
Caso2:	*****		[10,23)	Superpuesto	Superpuesto
Caso3:	*****		[3,29)	Superpuesto	Superpuesto
Caso4:	***		[11,14)	Superpuesto	Superpuesto
Caso5:		*****	[22,27)	No	No
Caso6:	***		[1,4)	No	No

- ❑ Estrategia 1: inicio(tramo2) <@ tramo1 OR -- IN
 fin(tramo2) <@ tramo1 OR -- IN
 Tramo2 <@ tramo1 --DURING
- ❑ Estrategia 2: tramo1 && tramo2 --OVERLAPS

Tratamientos de tiempos

Tiempo de Validez - UPDATE - Caso 1

<u>Años</u>	<u>Des</u>	<u>Has</u>	<u>10</u>	<u>11</u>	<u>12</u>	<u>13</u>	<u>14</u>	<u>15</u>	<u>16</u>	<u>17</u>	<u>18</u>	<u>19</u>	<u>Operación</u>
BD	12	16			50 k								
caso 1	11	12		60 k									
Queda en actual	11	12		60 k									update temporal propio
También actual	13	16				50 k							insert actual from new

Tengamos en cuenta que la condición:

"**lower**(BD) <@ **UPDATE** **AND** **upper**(**UPDATE**) <@ BD"

Para el Trigger se debe interpretar:

Como **BD** => **old**

Como **UPDATE** será lo que ingresa por **new**

"**inicio**(old.tv) <@ new.tv **AND** **fin**(new.tv) <@ old.tv"

Tratamientos de tiempos

Tiempo de Validez - UPDATE - Caso 1

```

CREATE FUNCTION fxemp_after_update_sueldo()
  RETURNS trigger AS $$
  BEGIN -- /***** caso 1 *****/
    --base=old,update = new
    IF (lower(old.tv) <@ new.tv AND upper(new.tv) <@ old.tv) THEN
      INSERT INTO empleados (dni,...,sueldo, tt, tv)
        SELECT old.dni,...,old.sueldo,
          tstzrange(current_timestamp, null),
          daterange(upper(new.tv), upper(old.tv));

    END IF;

    RETURN new;
  END;
  $$ LANGUAGE 'plpgsql';

```

Años	Des	Has	10	11	12	13	14	15	16	17	18	19	Operación
BD	12	16							50 k				
caso 1	11	12			60 k								
Queda en actual	11	12			60 k								update temporal propio
También actual	13	16							50 k				insert actual from new

```

CREATE TRIGGER trgempleados_after_update_sueldo
  AFTER UPDATE OF sueldo ON empleados
  FOR EACH ROW EXECUTE PROCEDURE fxemp_after_update_sueldo();

```

Tratamientos de tiempos

Tiempo de Validez - UPDATE - Caso 2

Años	Des	Has	10	11	12	13	14	15	16	17	18	19	Operación
BD	12	16			50 k								
caso 2	14	17					60k						
Queda en actual	14	17					60k						update temporal propio insert actual from new
También actual	12	13			50 k								

Tengamos en cuenta que la condición:

"**lower**(new) <@ old and (**upper**(old) is NULL or **upper**(old) <@ new)"

Para el Trigger se debe interpretar:

BD = old

UPDATE = new

Tratamientos de tiempos

Tiempo de Validez - UPDATE - Caso 2

```

CREATE FUNCTION fxemp_after_update_sueldo()
—— RETURNS trigger AS $$
BEGIN -- /***** caso 2 *****/

    IF (lower(new.tv) <@ old.tv AND upper(old.tv) <@ new.tv) THEN
        INSERT INTO empleados (dni,...,sueldo, tt, tv)
            SELECT old.dni,...,old.sueldo,
                tstzrange(lower(old.tt), null),
                daterange(lower(old.tv), loewr(new.tv));

    END IF;

—— RETURN new;
END;
$$ LANGUAGE 'plpgsql';
    
```

Años	Des	Has	10	11	12	13	14	15	16	17	18	19	Operación
BD	12	16					50 k						
caso 2	14	17						60k					
Queda en actual	14	17						60k					update temporal propio
También actual	12	13			50 k								insert actual from new

```

CREATE TRIGGER trgempleados_after_update_sueldo
—— AFTER UPDATE OF sueldo ON empleados
—— FOR EACH ROW EXECUTE PROCEDURE fxemp_after_update_sueldo();
    
```

Tratamientos de tiempos

Tiempo de Validez - UPDATE - Caso 3

Años	Des	Has	10	11	12	13	14	15	16	17	18	19	Operación
BD	12	16			50 k								
caso 3	14	15					60k						
También actual	12	14			50 k								insert actual from new update temporal propio insert actual from new
Queda en actual	14	17					60k						
También actual	12	13							50k				

Tengamos en cuenta que la condición:

`"new.tv <@ old.tv"`

Para el Trigger se debe interpretar:

BD = old

UPDATE = new.

Tratamientos de tiempos

Tiempo de Validez - UPDATE - Caso 3

```

CREATE FUNCTION fxemp_after_update_sueldo()
— RETURNS trigger AS $$
BEGIN -- /***** caso 3 *****/
    IF new.tv <@ old.tv THEN
        INSERT INTO empleados (dni,..., tt, sueldo, tv)
            SELECT old.dni,..., tstzrange(lower(old.tt),null),
                old.sueldo, daterange(lower(old.tv),lower(new.tv));

        INSERT INTO empleados (dni,..., tt, sueldo, tv)
            SELECT old.dni,..., tstzrange(lower(old.tt), current_timestamp),
                old.sueldo, daterange(upper(new.tv), upper(old.tv));

    END IF;
— RETURN new;
END;
$$ LANGUAGE 'plpgsql';

```

Años	Des	Has	10	11	12	13	14	15	16	17	18	19	Operación
BD	12	16					50 k						
caso 3	14	15					60k						
También actual	12	14			50 k								insert actual from new
Queda en actual	14	17				60k							update temporal propio
También actual	12	13					50k						insert actual from new

```

CREATE TRIGGER trgempleados_after_update_sueldo
— AFTER UPDATE OF sueldo ON empleados
— FOR EACH ROW EXECUTE PROCEDURE fxemp_after_update_sueldo();

```

Tratamientos de tiempos

Tiempo de Validez - DELETE

Las situaciones aplicadas al UPDATE (5 casos) son también aplicables al tratamiento del DELETE.

Pero las DMLs definidas en los estándares SQL 2003 y 2008 no tienen la posibilidad “indicar” o “portar” el periodo de tiempo válido. Por lo que sería solamente aplicable por medio de procedimientos almacenados dónde se pasa como parámetro un periodo y con cursores buscamos los elementos a eliminar.

Tratamientos de tiempos

Vista integradora

Generalmente por comodidad se suele crear una tabla que integre las tablas actuales e históricas.

```
CREATE VIEW vw_EmpleadosHistorial AS
SELECT * FROM Empleados
UNION
SELECT * FROM EmpleadosHistory;
```

Resumen

- ❑ Introducción
- ❑ Tratamientos de tiempos
- ❑ **Standard SQL:2011**
- ❑ Propuesta SQLServer
- ❑ Propuesta de Oracle
- ❑ Propuesta de PostgreSQL

Standard

Revisión histórica

- ❑ **Años 80:** El tratamiento de temporalidad se basa en investigaciones fundamentalmente académicas.
- ❑ **Años 90:** Luego de una publicación científica en 1992 de Richard Snodgrass “TSQL: A Design Approach”, se trabaja en TSQL2, una extensión temporal al standard SQL:92. Se la considera el primer acercamiento a un manejo temporal.
- ❑ **Próximos años:** Lento desarrollo e incorporación comercial del tema.
- ❑ **Año 2011:** Standarización, recién en diciembre de 2011 ISO publica SQL 2011. Agrega la posibilidad de manipular y crear tablas temporales y extender las DMLs para tratar precisamente estos datos temporales.
- ❑ **Años posteriores:** Oracle(2013), PostgreSQL(2016), Microsoft(2016) lanzan productos con incorporación parcial del standard SQL2011.

Standard Interval (SQL:92)

INTERVAL: Representa una “duración” en el tiempo, una cantidad de tiempo.

Ejemplos

18 year 3 month 2 days	--(edad)
3 hours 12 minutes	--(duración int.quirurgica)
14 days	--(vacaciones)

Operadores y Funciones

- ❑ Redefinición de la suma y resta (+ y -) para intervalos

`'12 hours 10 minutes' + '3 days'`

- ❑ Redefinición de extract

`extract(year from '12 year 24 month')`

Standard

Period (SQL:2011)

PERIOD: La piedra angular de SQL2011 es “la posibilidad de asociar periodos de tiempo a una tabla”.

Definición: Un Period de tiempo es un *Intervalo de tiempo posicionado en la línea de tiempo*, es decir, tiene un **inicio**, una **duración** e indirectamente un **fin** de periodo.

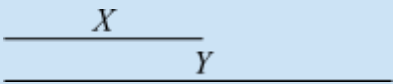
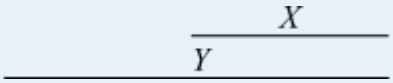
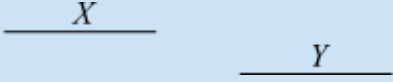
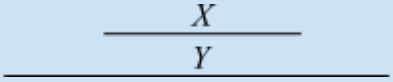
Standard Period (SQL:2011)

Consideraciones del Standard

- ☐ NO es un tipo de dato nuevo. Motivo: dar alta compatibilidad.
- ☐ Es un metadato de tabla que asocia dos columnas de tiempo, inicio y fin.
- ☐ SQL2011 toma el modelo de period [Cerrado,abierto)
- ☐ Inicio de un periodo **NO PUEDE SER NULO.**
- ☐ Fin nunca puede ser inferior a INICIO


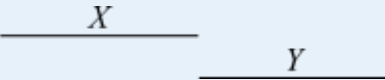
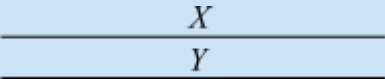
Standard Period (SQL:2011) - Operadores

Predicados temporales

Predicado	Ilustración	Allen's AR	Interpretación AAR
<u>BEGINS</u>		STARTS	X starts Y Y is started by X
<u>ENDS</u>		FINISHES	X finishes Y Y is finished by X
<u>PRECEDES</u>		BEFORE	X precedes Y Y is preceded by X
<u>SUCCEEDS</u>		AFTER	X succeeds Y Y is succeeded by X
<u>DURING,</u> <u>CONTAINS</u>		CONTAINS, DURING (inverso)	X during Y Y contains X

Standard Period (SQL:2011) - Operadores

Predicados temporales (2da parte)

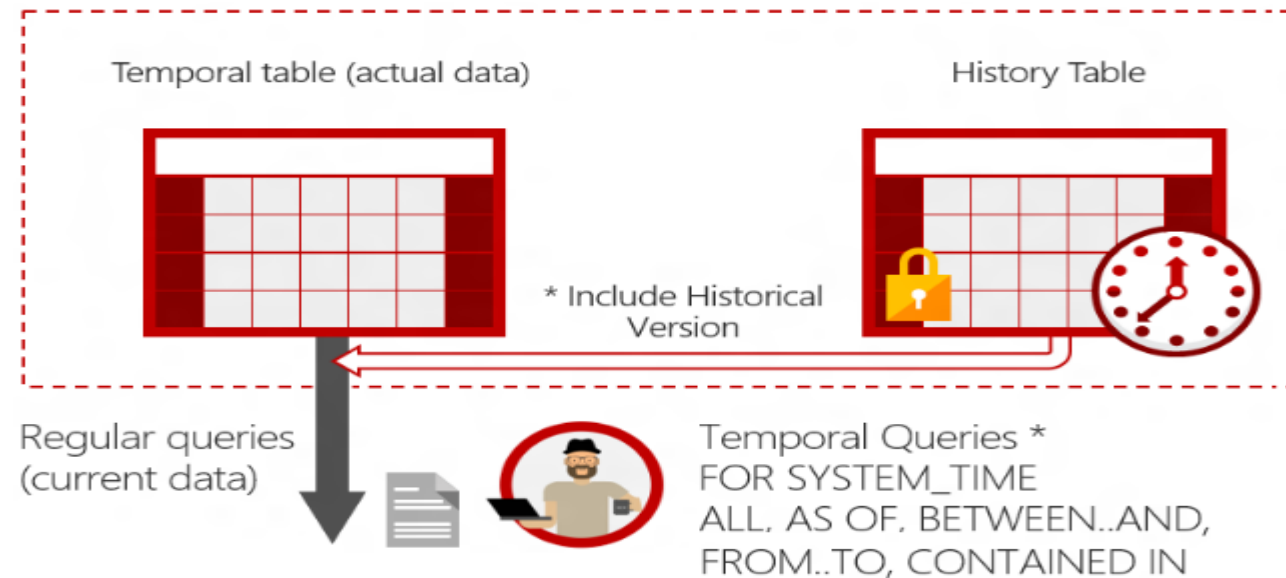
Predicado	Ilustración	Allen's AR	Interpretación AAR
<u>OVERLAPS</u>		OVERLAPS	X overlaps with Y Y is overlapped by X
<u>IMMEDIATELY PRECEDES,</u> <u>IMMEDIATELY SUCCEEDS</u>		MEETS, METBY	X meets Y Y is met by X
<u>EQUALS</u>		EQUALS	X is equal to Y

* Los operadores UNION, INTERSECTS, MINUS, etc. funcionan igual que antes

Standard **SQL:2011**

Implementación de tablas

¿Monolítica o Partición?: No especifica directamente el tipo de implementación, pero refleja la implementación con particionado horizontal.



Standard SQL:2011

Tipos de Periodo – TT y TV

Tiempo de Transacción

- ☐ Soportado por Tablas de Versionado, con **System Time** (Timestamp del sistema).
- ☐ Su nombre es fijo y especificado por SQL2011 ***System_Time***.
- ☐ Uso totalmente declarativo.

Tiempo de Validez

- ☐ Su nombre puede ser definido por el usuario.
- ☐ Usa el mismo espacio de nombres que las columnas.
- ☐ Las columnas de Inicio o Fin pueden ser TIMESTAMP o DATE, ***ambas del mismo tipo.***
- ☐ Restringido a solamente a un Periodo Valido por Tabla.

Standard SQL:2011

Tablas Temporales - TV

CREACIÓN DE TABLAS CON TIEMPO DE VALIDEZ:

```
CREATE TABLE Emp(  
    ENo INTEGER,  
    EName VARCHAR(50),  
    EStart DATE,  
    ESnd DATE,  
    EDept INTEGER  
    PERIOD FOR EPeriod (EStart, EEnd)  
);
```

Ojito:
Recordá que estas son TABLAS
TEMPORALES
de TIEMPO DE VALIDEZ.

EJEMPLO DE INSERT:

```
INSERT INTO Emp VALUES (  
    22217, 'Juan',  
    DATE '2010-01-01', DATE '2011-11-12', 3);
```

Eno	EStart	EEnd	EDept
22217	2010-01-01	2011-11-12	3

Standard SQL:2011

Tablas Temporales - TV

EJEMPLO DE UPDATE:

Eno	EStart	EEnd	EDept
22217	2010-01-01	2011-11-12	3

```
UPDATE Emp FOR PORTION OF Eperiod FROM DATE '2011-02-03' TO DATE '2011-09-10'  
SET EDept = 4 WHERE ENo = 22217;
```

ENo	EStart	EEnd	EDept
22217	2010-01-01	2011-02-03	3
22217	2011-02-03	2011-09-10	4
22217	2011-09-10	2011-11-12	3

Standard SQL:2011

Tablas Temporales - TV

EJEMPLO DE DELETE:

Eno	EStart	EEnd	EDept
22217	2010-01-01	2011-11-12	3

DELETE FROM Emp

FOR PORTION OF EPeriod FROM DATE '2011-02-03' TO DATE '2011-09-10'

WHERE ENO = 22217;

ENo	EStart	EEnd	EDept
22217	2010-01-01	2011-02-03	3
22217	2011-09-10	2011-11-12	3

Standard SQL:2011

Tablas Temporales – Primary Key TV

CLAVES CON PERIODOS: Mirando un poco el estado de Emp es obvio que no podemos elegir Eno como Primary key.

ENo	EStart	EEnd	EDept
22217	2010-01-01	2011-02-03	3
22217	2011-02-03	2011-09-10	4
22217	2011-09-10	2011-11-12	3

Usamos entonces la (tripla) => (ENo, EPeriod)

```
ALTER TABLE Emp ADD PRIMARY KEY (ENo, EPeriod)
```

NOTA: agregar simplemente la tripla (ENo, EStart, EEnd) no sería suficiente: una clave (22217, 2010-01-01, 2011-09-10) no sería duplicada pero tiene overlap

Standard SQL:2011

Tablas Temporales – Primary Key TV

EVITAR SUPERPOSICIONES:

Para evitar Superposiciones usamos el predicado WITHOUT OVERLAPS:

```
ALTER TABLE Emp ADD PRIMARY KEY (ENo, EPeriod WITHOUT OVERLAPS)
```

Standard SQL:2011

Tablas Temporales – Foreign Key TV

INTEGRIDAD REFERENCIAL TEMPORAL: Analizando el ejemplo podríamos deducir que puede existir una tabla “Dept” por el campo “Edept” de “Emp”.

```
CREATE TABLE Dept (  
  DNo INTEGER,  
  DName VARCHAR(30),  
  DStart DATE,  
  DEnd DATE,  
  PERIOD FOR DPeriod (DStart, DEnd),  
  PRIMARY KEY (DNo, DPeriod WITHOUT OVERLAPS)  
)
```

Supongamos que queremos asegurarnos que cada valor en Emp.EDept corresponde a algún Dept.DNo y además que este exista en ese punto en el tiempo.

Standard SQL:2011

Tablas Temporales – Foreign Key TV

Emp

ENo	EStart	EEnd	EDept
22218	2010-01-01	2011-02-03	3
22218	2011-02-03	2011-11-12	4

X

Dpto

DNo	DStart	DEnd	DName
3	2009-01-01	2011-12-31	Test
4	2011-06-01	2011-12-31	QA

Propuesta: FOREIGN KEY (EDept, PERIOD EPeriod)....:

```
ALTER TABLE Emp ADD FOREIGN KEY (EDept, PERIOD EPeriod)
REFERENCES Dept (DNo, PERIOD DPeriod)
```

Esta extensión de Foreign Key temporal soluciona el problema de **integridad referencial temporal** en forma declarativa.

Standard SQL:2011

Tablas Temporales – Consultas TV

Las consultas se pueden realizar utilizando SQL regular:

```
SELECT Name, EDept          -- Donde trabaja 217 02-1-2011
FROM Emp
WHERE ENo = 217
      AND EStart <= DATE '2011-01-02'
      AND EEnd   >  DATE '2011-01-02'
```

```
SELECT Ename, EDept          -- Todos Dptos trabajo 217
FROM Emp                     -- entre 2-1-2010 y 1-1-2011
WHERE ENo = 22217
      AND EStart < DATE '2011-01-01'
      AND EEnd >  DATE '2010-01-01'
```

De igual forma se podrían aplicar las sentencias BETWEEN, IN, etc.

Standard SQL:2011

Tablas Temporales – Consultas TV

Desde SQL:2011 se agregan los predicados temporales:

```
SELECT Name, EDept          -- Donde trabaja 217 02-1-2011
FROM Emp
WHERE ENo = 217
      AND DATE '2011-01-02' DURING EPeriod
```

```
SELECT Ename, EDept          -- Todos Dptos trabajo 217
FROM Emp                     -- entre 2-1-2010 y 1-1-2011
WHERE ENo = 217
      AND EPeriod OVERLAPS PERIOD(DATE '2010-01-01', DATE '2011-01-01')
```

Standard SQL:2011

Tablas Temporales – TT

CREACIÓN DE TABLAS CON TIEMPO DE TRANSACCIÓN:

```
CREATE TABLE table
  ENo INTEGER,
  Ename VARCHAR(30)
  Sys_start TIMESTAMP(12) GENERATED ALWAYS AS ROW START,
  Sys_end TIMESTAMP(12) GENERATED ALWAYS AS ROW END,
  PERIOD FOR SYSTEM_TIME (Sys_start, Sys_end)
) WITH SYSTEM VERSIONING
```

EJEMPLO DE INSERT:

```
INSERT INTO Emp (Eno, Ename) VALUES (22217, 'Joe')
--insert realizado el 1 de enero de 2012 a las 9 AM
```

ENo	Sys_Start	Sys_End	EName
22217	2012-01-01 09:00:00	9999-12-31 23:59:59	Joe

Standard SQL:2011

Tablas Temporales – TT

EJEMPLO DE UPDATE:

ENo	Sys_Start	Sys_End	ENAME
22217	2012-01-01 09:00:00	9999-12-31 23:59:59	Joe

```
UPDATE Emp SET EName = 'Tom' WHERE ENo = 22217;
```

```
--Efectuado 3-2-12 10 hs
```

ENo	Sys_Start	Sys_End	ENAME
22217	2012-01-01 09:00:00	2012-02-03 10:00:00	Joe
22217	2012-02-03 10:00:00	9999-12-31 23:59:59	Tom

Standard SQL:2011

Tablas Temporales – TT

EJEMPLO DE DELETE:

ENo	Sys_Start	Sys_End	ENAME
22217	2012-01-01 09:00:00	9999-12-31 23:59:59	Joe

`DELETE FROM Emp WHERE ENo = 22217;`

`--Efectuado el 1-6-2012 0 hs`

ENo	EStart	EEnd	ENAME
22217	2012-01-01 09:00:00	2012-06-01 00:00:00	Joe

Ojito:

Estamos hablando de TABLAS
TEMPORALES

de TIEMPO DE TRANSACCIÓN.

Como no sabemos cuándo será la
próxima transacción sobre una fila, su
tiempo de finalización es indefinido
(9999999999999).

Standard SQL:2011

Tablas Temporales – Claves TT

CLAVES PRIMARIAS Y FORÁNEAS EN TABLAS DE VERSIONADO:

Son mas simples, puesto que en **la tabla principal solo hay filas actuales**, por lo que las claves “tradicionales” funcionan perfectamente.

Las filas históricas del sistema de una tabla “System_versioned” la forman instantáneas inmutables del pasado. Por lo tanto, cualquier restricción que haya sido válida en el momento que se creó una fila, lo seguirá siendo, por lo que no existiría la necesidad de imponer restricciones para las filas del sistema histórico.

Standard SQL:2011

Tablas Temporales – Consultas TT

Consulta FOR SYSTEM_TIME AS: Por ejemplo, la consulta recupera las filas de Emp que eran actuales (registradas) el 2 de enero de 2011.

```
SELECT ENo, EName, Sys_Start, Sys_End  
FROM Emp  
FOR SYSTEM_TIME AS OF TIMESTAMP '2011-01-02 00:00:00'
```

Idem entre el 2-1-2011 y 31-12-2011

```
SELECT ENo, EName, Sys_Start, Sys_End  
FROM Emp FOR SYSTEM_TIME  
FROM TIMESTAMP '2011-01-02 00:00:00'  
TO TIMESTAMP '2011-12-31 00:00:00'
```

* Para esta última también es válido:

```
FOR SYSTEM_TIME BETWEEN ... AND ...
```

Standard SQL:2011

Tablas Temporales – Consultas TT

CONSIDERACIONES EN CONSULTAS DE TIEMPO DE TRANSACCIÓN:

- ☐ Si no se hace uso del predicado FOR SYSTEM_TIME, las consultas solamente se realizarán sobre las filas actuales.
- ☐ Para recuperar el histórico completo utilizar FOR SYSTEM_TIME FROM '0001-01-01 00:00:00' TO '9999-12-31 23:59:59'

Resumen

- ❑ Introducción
- ❑ Tratamientos de tiempos
- ❑ Standard SQL:2011
- ❑ Propuesta SQLServer
- ❑ Propuesta de Oracle
- ❑ Propuesta de PostgreSQL

Propuesta SQL SERVER

Desde la versión SQL SERVER 2016, se introduce el sistema de versionado de sistema como una función integrada a las bases de datos con el nombre de “Temporal tables”. Basado en el standard SQL:2011.

Documentación:

<https://learn.microsoft.com/en-us/sql/relational-databases/temporal-tables/temporal-tables?view=sql-server-ver16>

Propuesta SQL SERVER

¿Cómo es la propuesta temporal de SqlServer?

Tiene un Sistema de versionado implementado con fragmentación horizontal, administra el Tiempo de Transacción automáticamente:

Tabla histórica: Valor anterior para cada fila. Pasado.

Cada tabla tiene las siguientes columnas adicionales *datetime2* para definir el período de transacción:

Inicio de Periodo: Instante inicial de transacción “GENERATED ALWAYS AS ROW START”.

Fin de Periodo: Instante final de transacción “GENERATED ALWAYS AS ROW END”.

Y un PERIOD FOR SYSTEM_TIME (name_col_ttinicio, name_col_ttfin).

Propuesta SQL SERVER

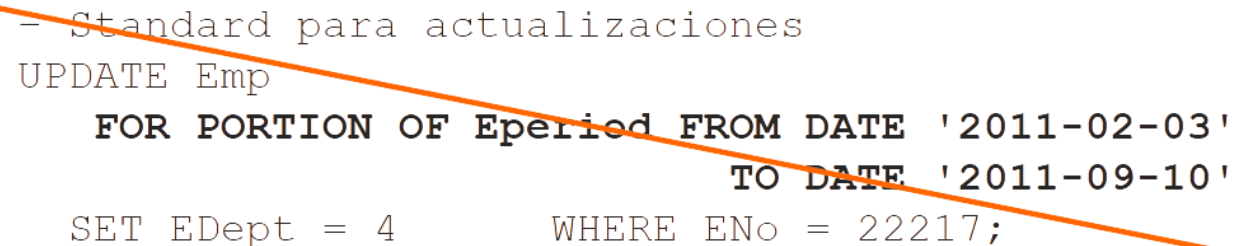
Para revisar la implementación, revisar el archivo de ejemplo_temporales.txt subido a la plataforma Moodle.

```
1 CREATE DATABASE ejemplo;
2
3 USE ejemplo;
4
5 --ejemplo de microsoft
6 CREATE TABLE Department
7 (
8     DepartmentNumber CHAR(10) NOT NULL PRIMARY KEY CLUSTERED,
9     DepartmentName VARCHAR(50) NOT NULL,
10    ManagerID INT NULL,
11    ParentDepartmentNumber CHAR(10) NULL,
12    SysStartTime DATETIME2 GENERATED ALWAYS AS ROW START HIDDEN NOT NULL,
13    SysEndTime DATETIME2 GENERATED ALWAYS AS ROW END HIDDEN NOT NULL,
14    PERIOD FOR SYSTEM_TIME (SysStartTime, SysEndTime)
15 )
16 WITH (SYSTEM_VERSIONING = ON);
17
18 --select de la tabla vacia
19 SELECT * FROM Department;
20
21 --insertamos un departamento
22 INSERT INTO Department VALUES (
23     'ABC-CBA',
24     'prueba',
25     100,
26     NULL
27 )
```

Propuesta SQL SERVER

Si bien el tiempo de transacción esta muy bien cubierto por esta implementación, con mucho ajuste al Standard, No sucede lo mismo con el tiempo de Validez ya que **ni siquiera es posible definir un periodo de validez.**

Tampoco se realiza la gestión automática para UPDATE Y DELETE, con los modificadores del standard FOR PORTION de periodo.



```
- Standard para actualizaciones
UPDATE Emp
    FOR PORTION OF Eperiod FROM DATE '2011-02-03'
                                TO DATE '2011-09-10'
SET EDept = 4      WHERE ENo = 22217;
```

Igual situación sufren los operadores temporales del standard.

Esperemos mas cobertura de SQL2011 en futuras versiones.