

# BASES DE DATOS TEMPORALES

## *Implementaciones y Standard*

“Cualquier Tiempo Pasado Fue Anterior”  
Les Luthiers

Docentes : Ing. Fernando Sato  
A.S. Sebastian Trossero

# RESUMEN

---

- Propuesta soporte temporal como parte de la aplicación.
- Standard SQL:2011
- Propuesta SQLServer
- Propuesta de Oracle - Informativo
- Propuesta de PostgreSQL - Informativo
- Instalación PostgreSQL - Informativo

# Bases de Datos Temporales

## Solución sin Soporte Temporal

---

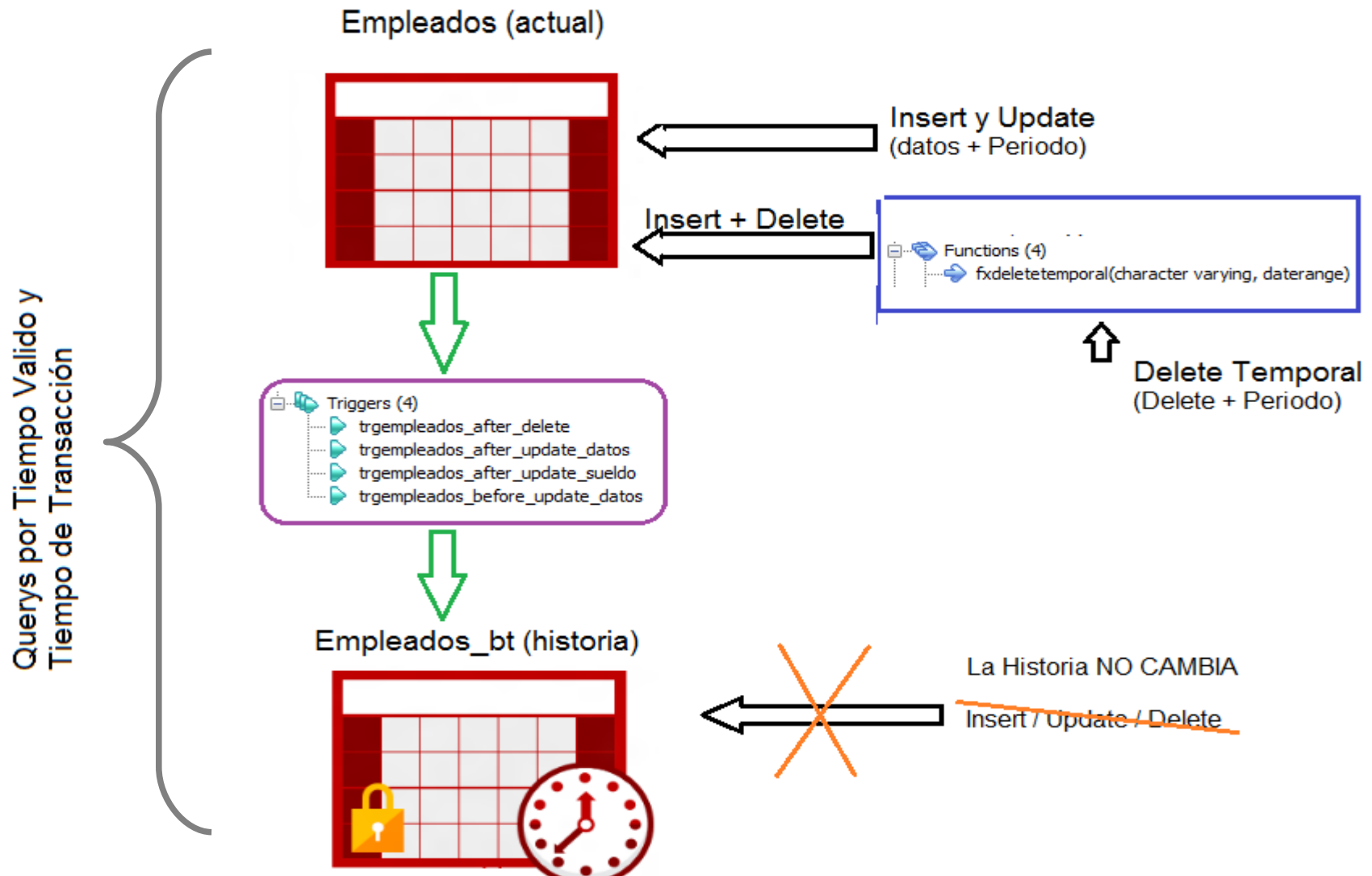
La necesidad de soporte temporal no es nueva.

Las bases de datos activas otorgan el soporte básico para tratar la temporalidad.

Dicho de otra forma, podemos dar tratamiento temporal usando Sql 2003, concretamente usando ***tipos de datos de tiempo, funciones de tiempo, trigger y procedimientos almacenados*** y utilizar restricciones en los queries, y actualizaciones que “filtren” los datos afectados.

# Bases de Datos Temporales

## Solución sin Soporte Temporal - Idea



# Bases de Datos Temporales


## Solución sin Soporte Tiempo - Tablas

---

**Tabla Principal:** Contendrá datos actuales y los triggers necesarios.

```
CREATE TABLE empleados (  
  dni integer,  
  nombre character varying(50),  
  fecha_nacimiento date,  
  sueldo numeric(12,2),  
  tt tstzrange DEFAULT  
    tstzrange(current_timestamp , null)  
);
```

Empleados (actual)



# Bases de Datos Temporales

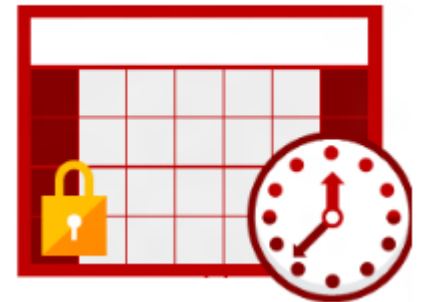
## Solución sin Soporte Tiempo - Tablas

---

**Tabla Histórica:** Contendrá el pasado.

```
CREATE TABLE empleados (  
dni integer,  
nombre character varying(50),  
fecha_nacimiento date,  
sueldo numeric(12,2),  
tt tstzrange  
);
```

Empleados\_bt (historia)



*Este modelo se basa en Fragmentación ¿?*



*Fragmentación Horizontal*

# RESUMEN

---

- **Propuesta soporte temporal como parte de la aplicación.**

## **Tratamiento Tiempo de Transacción**

- Standard SQL:2011
- Propuesta SQLServer
- Propuesta de Oracle - Informativo
- Propuesta de PostgreSQL - Informativo
- Instalación PostgreSQL - Informativo

# Bases de Datos Temporales

## Tratamiento Tiempo de Transacción - Insert

---

Para hacer este análisis nos olvidamos del *tiempo valido*.

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

*¿Que debería hacer el INSERT?*



# Bases de Datos Temporales

## Tratamiento Tiempo de Transacción - Insert

---

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

La función debe cambiar el inicio del tiempo de transacción para Insert, pero no así para Update.

En otras palabras:

```
IF tg_op = 'UPDATE' THEN           new.tt = old.tt;  
  
IF lower(new.tt) is null THEN  
    new.tt = tstzrange(current_timestamp ,null);
```

# Bases de Datos Temporales

## Tratamiento Tiempo de Transacción - Insert

---

```
CREATE FUNCTION fxemp_before_insert_update_tt()  
  RETURNS trigger  
  AS $$  
  BEGIN  
    IF tg_op = 'UPDATE' THEN  
      new.tt = old.tt;  
      RETURN new;  
    END IF;  
  
    IF lower(new.tt) is null THEN  
      new.tt = tstzrange(current_timestamp , null);  
    END IF;  
    RETURN new;  
  END;  $$ LANGUAGE 'plpgsql';
```

# Bases de Datos Temporales

## Tratamiento Tiempo de Transacción - Delete

---

```
DELETE FROM empleados WHERE dni = 10;
```

*¿Que debería hacer?*

*La fila deja de existir en el presente ( Empleados ) y guardar el hecho en el pasado.*



# Bases de Datos Temporales

## Tratamiento Tiempo de Transacción – Delete

---

```
CREATE OR REPLACE FUNCTION fxempleados_after_delete()  
RETURNS trigger AS $$  
BEGIN  
    INSERT INTO empleados_bt  
        SELECT old.dni,                old.nombre,  
               old.fecha_nacimiento, old.sueldo,  
               tstzrange(lower(old.tt), current_timestamp, '[]');  
    RETURN null;  
END  $$ LANGUAGE 'plpgsql'
```

```
CREATE TRIGGER trgempleados_after_delete  
    AFTER DELETE    ON empleados  
    FOR EACH ROW  
    EXECUTE PROCEDURE fxempleados_after_delete();
```

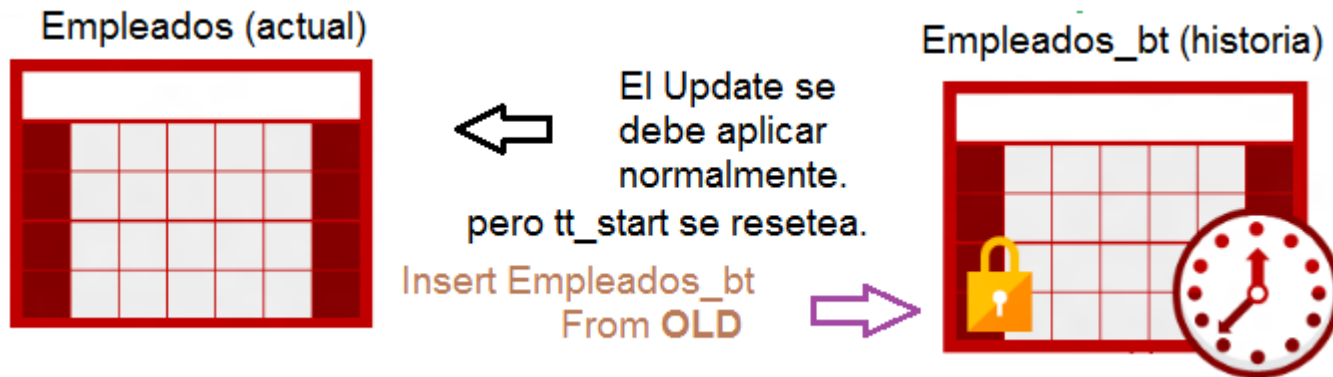
# Bases de Datos Temporales

## Tratamiento Tiempo de Transacción - Update

```
UPDATE empleados SET nombre='Juan' WHERE dni=10;
```

*¿Que debería hacer?*

*La fila cambia en el presente ( Empleados ) y guardar el estado anterior del hecho en el pasado.*



# Bases de Datos Temporales

## Tratamiento Tiempo de Transacción - Update

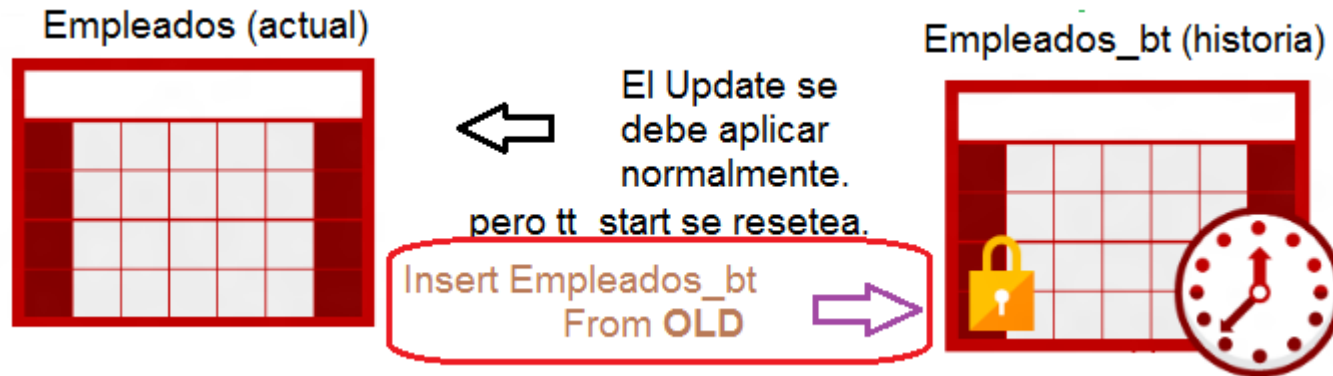
---

```
CREATE OR REPLACE FUNCTION
fxempleados_before_update_datos () RETURNS trigger
AS $$
BEGIN
    new.tt =      tstzrange(current_timestamp ,null);
    RETURN new;
END;
$$           LANGUAGE 'plpgsql';
```

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

# Bases de Datos Temporales

## Tratamiento Tiempo de Transacción - Update



Pero ademas digimos que el copiado debe hacer la misma lógica que el copiado despues de un delete.

```
CREATE TRIGGER trgempleados_after_update_datos
  AFTER UPDATE OF dni, nombre, fecha_nacimiento,
  sueldo
  ON empleados      FOR EACH ROW
EXECUTE PROCEDURE fxempleados_after_delete(); --
Función ya existente.
```

# RESUMEN

---

- Propuesta soporte temporal como parte de la aplicación.

## Tratamiento Tiempo Valido

Agregamos los atributos en ambas tablas.

```
ALTER TABLE empleados      ADD tv daterange;  
ALTER TABLE empleados_bt  ADD tv daterange;
```



# Bases de Datos Temporales

## Tratamiento Tiempo Valido - 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.

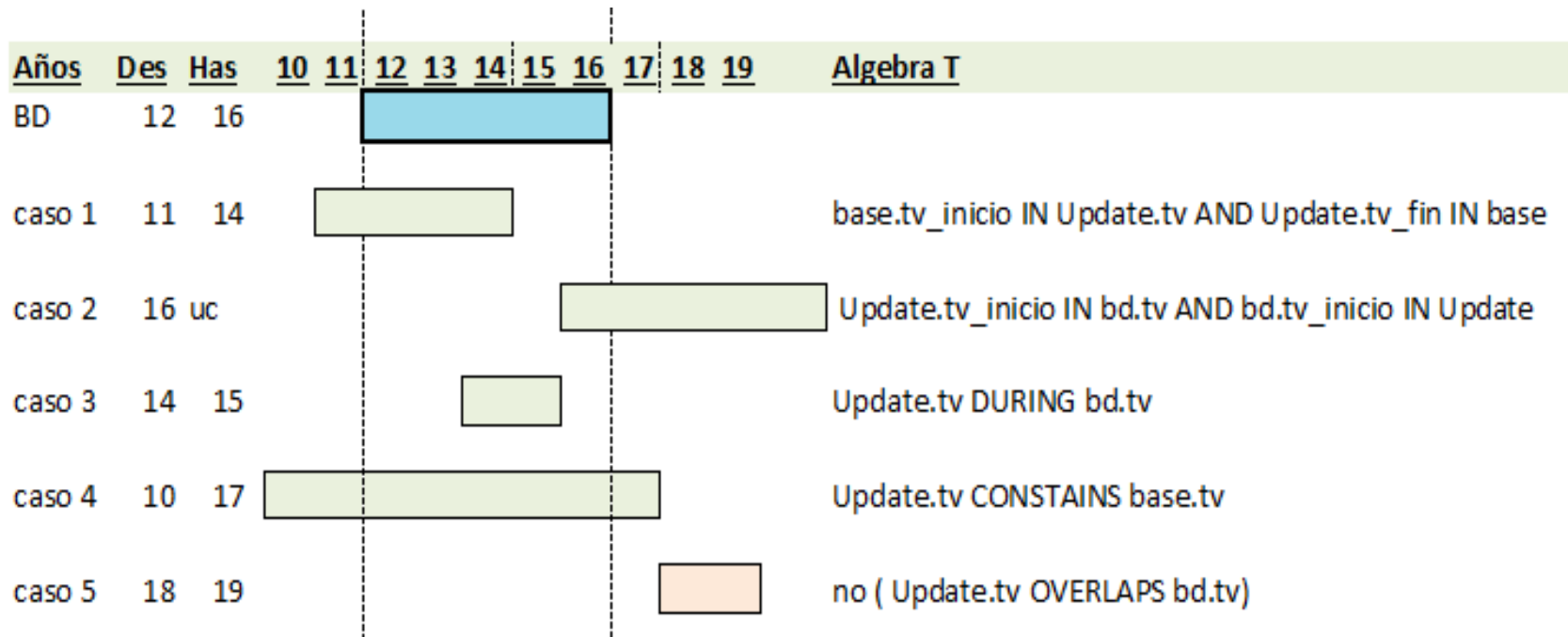
```
INSERT INTO empleados (dni, nombre, fec_nac, sueldo,  
                        tv)  
VALUES (10, 'Juan', '1999-06-02', 50000,  
        Daterange('2012-04-14', '2016-11-8', '[]'));
```

*¿Pero que pasa con los Update y Delete?*

# Bases de Datos Temporales

## Tratamiento Tiempo Valido

Comenzamos analizando los casos posibles Modificaciones,  
¿Como detectamos cada situación?



# Bases de Datos Temporales

## Tratamiento Tiempo Valido

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

Estrategía 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) > inicio(tramo1)

Estrategia 2:

Not(Inicio(tramo2) > Fin(tramo1) OR Inicio(tramo1) > Fin(tramo2))

# Bases de Datos Temporales

## Tratamiento Tiempo Valido

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

Estrategía 1:

inicio(tramo2) <@ tramo1 OR -- IN  
 fin(tramo2) <@ tramo1 OR -- IN  
 Tramo2 <@ tramo1 --DURING

Estrategia 2:

tramo1 && tramo2 --OVERLAPS

# Bases de Datos Temporales

## Tratamiento Tiempo Valido – Caso 1

Caso 1 de Modificación

¿Como tratamos esta situación?

<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
Tambien en actual:	13	16				50 k							Insert Actual From New

Tengamos en cuenta que la condición:

"**lower (base)** <@ **update** **and upper (update)** <@ **base**"

Para el Trigger se debe interpretar:

Como base = old y lo que se corresponde con update será lo que ingresa por new.

$X <@ Y$ : X incluido en Y, donde X es un TimeStamp e y es un Periodo.

# Bases de Datos Temporales

## Tratamiento Tiempo Valido – Caso 1

---

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

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

RETURN null;
END
$BODY$ LANGUAGE plpgsql
```

# Bases de Datos Temporales

## Tratamiento Tiempo Valido – Caso 2

Caso 2 de Modificación

¿Como tratamos esta situación?

<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>
BD	12	16			50 k							
Caso 2	14	17					60 k					
Queda en Actual:	14	17					60 k					
Tambien en actual:	12	13			50 k							

Update Temporal Propio  
Insert Actual From New

Tengamos en cuenta que la condición:

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

Para el Trigger se debe interpretar: base = old y update = new.

$X <@ Y$ : X incluido en Y, donde X es un TimeStamp e y es un Periodo.

# Bases de Datos Temporales

## Tratamiento Tiempo Valido – Caso 2

---

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

```
CREATE OR REPLACE FUNCTION public.fxemp_after_update_sueldo()
RETURNS trigger AS $BODY$
BEGIN-- caso 2 /*****
IF lower(new.tv) <@ old.tv and
  (upper(old.tv) is null or upper(old.tv) <@ new.tv) THEN
  INSERT INTO empleados (dni,...,tt, tv,)
    SELECT old.dni,..., tstzrange(lower(old.tt),null),
      daterange(lower(old.tv),lower(new.tv));
END IF;

RETURN null;
END
$BODY$ LANGUAGE plpgsql
```



# Bases de Datos Temporales

## Tratamiento Tiempo Valido – Caso 3

Caso 3 de Modificación

¿Como tratamos esta situación?

Años	Des	Has	10	11	12	13	14	15	16	17	18	19
BD	12	16			50 k							
Caso 2	13	14				60 k						
Tambien en actual:	12	13			50k							
Queda en Actual:	13	14				60 k						
Tambien en actual:	15	16						50k				

Insert Actual From New

Update Temporal Propio

Insert Actual From New

Tengamos en cuenta que la condición:

"**new.tv**) <@ **old.tv**"

Para el Trigger se debe interpretar: base = old y update = new.

X <@ Y: X incluido en Y, donde X e Y son Periodos.

# Bases de Datos Temporales

## Tratamiento Tiempo Valido – Caso 3

---

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

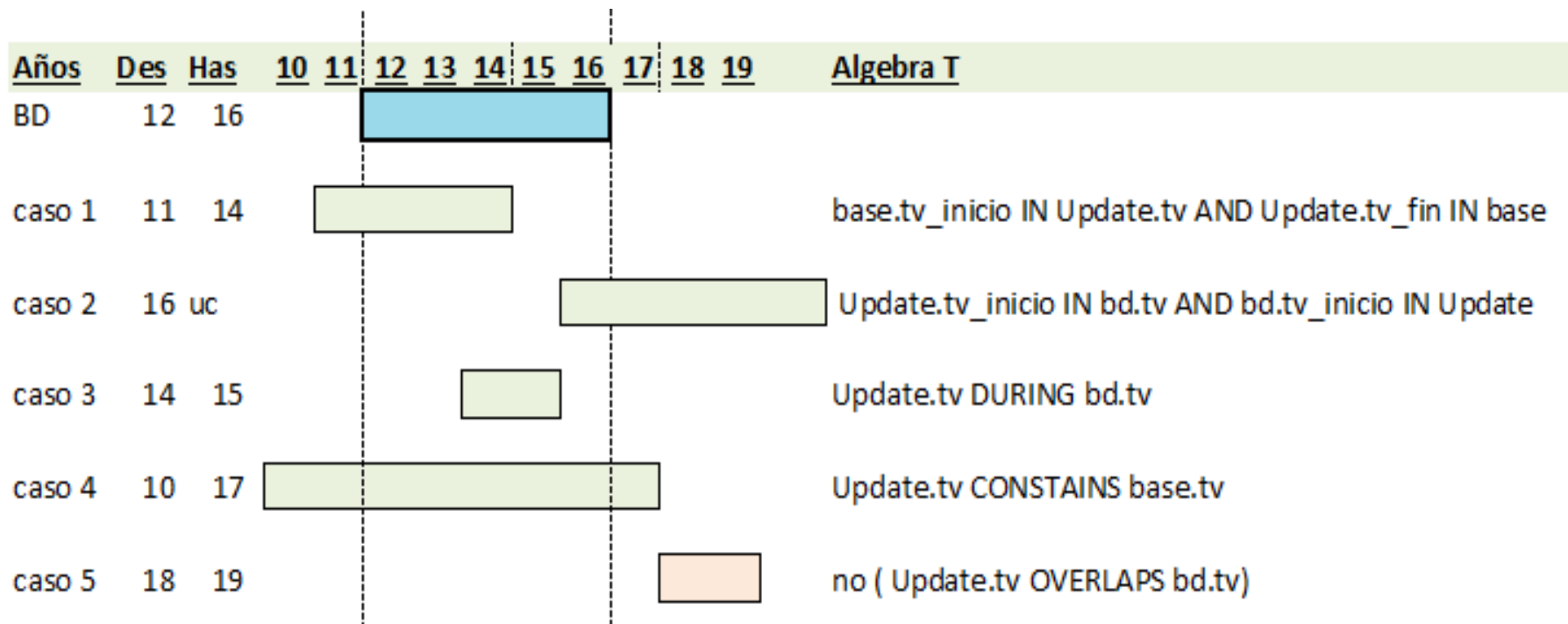
```
CREATE OR REPLACE FUNCTION public.fxemp_after_update_sueldo()
RETURNS trigger AS $BODY$
BEGIN-- caso 3 /*****/
IF new.tv <@ old.tv THEN
    INSERT INTO empleados (dni,...,tt, tv,)
        SELECT old.dni,..., tstzrange(lower(old.tt),null),
            daterange(lower(old.tv),lower(new.tv));
    INSERT INTO empleados (dni,...,tt, tv,)
        SELECT old.dni,..., tstzrange(lower(old.tt),timestamp),
            daterange(upper(new.tv),upper(old.tv));
END IF;

RETURN null;
END
$BODY$ LANGUAGE plpgsql
```

# Bases de Datos Temporales

## Tratamiento Tiempo Valido – Borrado

**Delete:** Las situaciones aplicadas al UPDATE son aplicables también al tratamiento de DELETE.



# Bases de Datos Temporales

## Tratamiento Tiempo Valido – Borrado

---

**Delete:** Pero la DML Delete Sql 2003/2008 no tiene posibilidades de “Portar” el período de tiempo valido.

```
CREATE OR REPLACE FUNCTION fxdeletetemporal(dml
character varying, ptv daterange)
    RETURNS varchar AS $BODY$
DECLARE
    dml_sentencia varchar;
    dml_where varchar;
    dml_tabla varchar;
BEGIN
    /****** Tratamiento de DELETE *****/
    RETURN 'Informar estadisticos de lo efectuado';
END;
$BODY$ LANGUAGE plpgsql;
```

# Bases de Datos Temporales

## Tratamiento Tiempo Valido – Caso 1

---

**Delete:** Obtenemos las variables dml\_tabla y dml\_where.

```
dml_sentencia = trim(split_part(dml, 'FROM', 1));  
dml_where     = trim(split_part(dml, 'WHERE', 2));  
dml_tabla     =  
trim(split_part(split_part(dml, 'WHERE', 1), 'FROM', 2));  
  
IF dml_sentencia != 'DELETE' THEN  
    RAISE EXCEPTION 'Sentencia debe ser DELETE';  
END IF;
```

**Nota:** Controlamos que realmente dml sea DELETE.

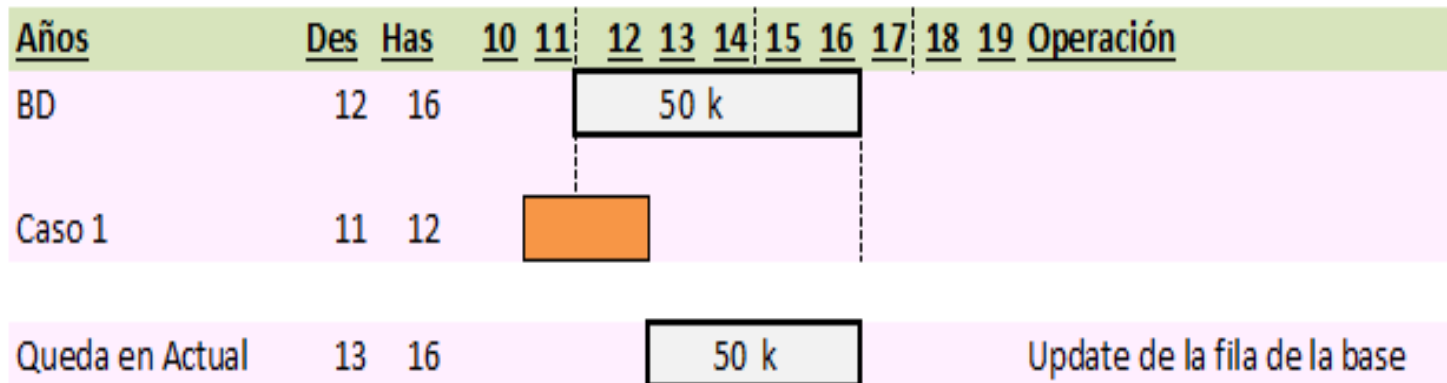
# Bases de Datos Temporales

## Tratamiento Tiempo Valido – Caso 1

**Caso 1 DELETE:** Se crea un cursor para tratar las filas afectadas:

```
FOR cursor IN EXECUTE 'SELECT * FROM ' || dml_tabla
|| ' WHERE ' || dml_where || ' AND tv && ' || ' ' ||
cast(deletetv as varchar) || ' ' ||
```

Graficamente:



Tengamos en cuenta que la condición:

```
"lower(cursor.tv) <@ deletetv and
upper(deletetv) <@ cursor.tv"
```

# Bases de Datos Temporales

## Tratamiento Tiempo Valido – Caso 1

---

**Caso 1 DELETE:** Código realizado por la función:

```
IF      (lower(cursor.tv) <@ deletetv and
upper(deletetv)      <@ cursor.tv)      THEN

DELETE FROM empleados
        WHERE dni              = cursor.dni
        AND nombre             = cursor.nombre
        AND fecha_nacimiento = cursor.fecha_nacimiento
        AND sueldo             = cursor.sueldo;

INSERT INTO empleados (dni, nombre, ...)
        SELECT cursor.dni, cursor.nombre,
        tstzrange(lower(cursor.tt), null),
        daterange(upper(deletetv), upper(cursor.tv));

END IF;
```

# Bases de Datos Temporales

## Tratamiento Tiempo Valido

---

Protección de **Empleados\_bt**: Para completar debemos generar un trigger que no permite cambiar, borrar o insertar en la historia.

```
CREATE TRIGGER trgempleados_bt_before_dml
BEFORE INSERT OR DELETE OR UPDATE
ON public.empleados_bt FOR EACH ROW
EXECUTE PROCEDURE fxempleados_bt_before_dml();

CREATE OR REPLACE FUNCTION fxempleados_bt_before_dml()
RETURNS trigger AS $BODY$
BEGIN
    IF user != 'postgres' THEN
        RAISE EXCEPTION 'Imposible cambiar historia';
    END IF;
END; $BODY$ LANGUAGE plpgsql
```

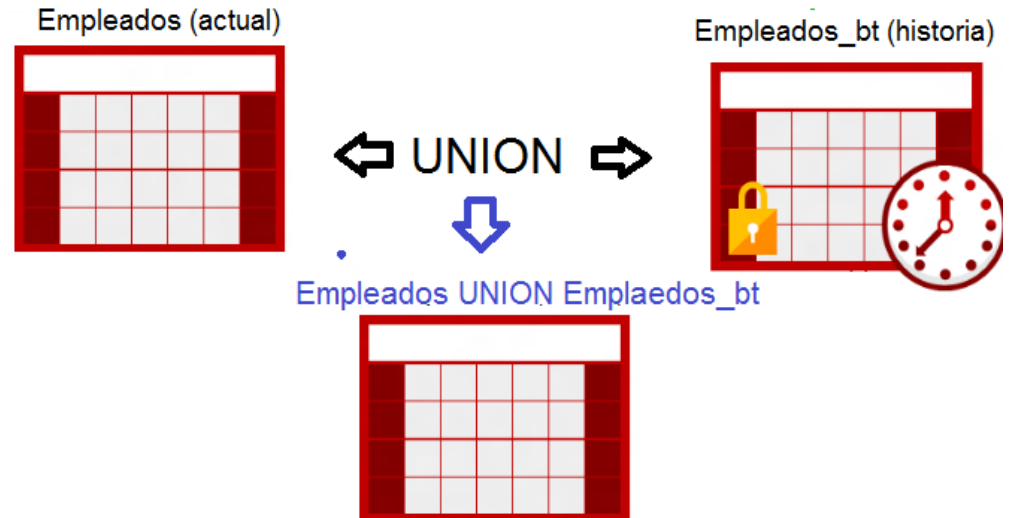


# Bases de Datos Temporales

## Tratamiento TT – Vista Integradora

---

Por comodidad de trabajo  
generamos la vista  
vempleados\_historiatt



```
CREATE VIEW vempleados_historiatt AS
    SELECT * FROM empleados
    UNION
    SELECT * FROM empleados_bt;
```

# RESUMEN

---

- Propuesta soporte temporal como parte de la aplicación.
- **Standard SQL:2011**
- Propuesta SQLServer
- Propuesta de Oracle - Informativo
- Propuesta de PostgreSQL - Informativo
- Instalación PostgreSQL - Informativo

# Bases de Datos Temporales

## Standard – Revisión Historica

---

- **Años 80:** El tratamiento de temporalidad se basa en investigaciones fundamentalmente académicas.
- **Año 95:** Intento Fallido por cubrir con SQL Temporalidad. Nace TSQL2 impulsado por Snodgrass.
- **Próximos Años:** Lenta 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.

# Bases de Datos Temporales

## Interval (SQL1992)

---

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

### Ejemplos

- `2018 year 3 month 12 days`
- `3 hours 12 years`

### 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')`

# Bases de Datos Temporales

## Metadatos SQL2011 - Period

---

**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** y una **duración**, indirectamente un fin de tiempo.

### **Consideraciones del Standard**

- **Period NO es un tipo de dato** nuevo. Motivo dar alta compatibilidad.
- **Period** 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 superior a **INICIO**

# Bases de Datos Temporales

## SQL2011 Period Operadores

---

**PERIOD:** comparación con Algebra Relacional.

**BEGIN** Y **END**, obtienen los extremos (**START** y **END** AR)

### SQL2011

BEGINS

ENDS

PRECEDES

SUCCEEDS

DURING, CONTAINS

OVERLAPS

IMMEDIATELY PRECEDES

IMMEDIATELY SUCCEEDS

UNION

INTERSECT

MINUS

### AR

STARTS

FINISHES

BEFORE

AFTER

DURING, CONTAINS(inverso)

OVERLAPS

MEETS

METBY

Idem

Idem

Idem

# Bases de Datos Temporales

## Tipos de Periodo – TT y TV

---

Tratamiento de los 2 tipos de tiempo:

### 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.

# Bases de Datos Temporales

## Tablas Temporales - TV

---

### Ejemplo de Tablas Temporales con Tiempo de Validez:

```
CREATE TABLE Emp (  
    ENo          INTEGER,  
    EName        VARCHAR(50),  
    Estart       DATE,  
    Eend         DATE,  
    EDept        INTEGER,  
    PERIOD FOR Eperiod (Estart, Eend)  
);
```

### Ejemplo de Insert:

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



# Bases de Datos Temporales

## Tablas Temporales - TV

---

### Ejemplo de Actualización:

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

Inicial

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

Actualizado

# Bases de Datos Temporales

## Tablas Temporales - TV

---

### Ejemplo de Eliminación:

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

Inicial

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

Actualizado

# Bases de Datos Temporales

## 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)
```

# Bases de Datos Temporales

## Tablas Temporales – Primary Key TV

---

**Periodos Superpuestos:** Ahora veamos el siguiente ejemplo.

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

Para evitar Superposiciones usamos el predicado WITHOUT OVERLAPS

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

# Bases de Datos Temporales

## Tablas Temporales – Foreign Key TV

---

**Integridad Referencial Temporal:** Ahora supongamos que también tenemos una tabla temporal Dept:

```
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:

Emp

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

Dpto

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

# Bases de Datos Temporales

## Tablas Temporales – Foreign Key TV

---

**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.

# Bases de Datos Temporales

## Tablas Temporales – Consultas TV

---

### Tablas Temporales admiten 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'
```

¿Podríamos usar el operador BETWEEN?, tambien IN de AR

# Bases de Datos Temporales

## Tablas Temporales – Consultas TV

---

**Tablas Temporales consultas SQL2011:** Una forma mas sencilla es usar:

<i>CONTAINS,</i>	<i>OVERLAPS,</i>
<i>EQUALS,</i>	<i>PRECEDES,</i>
<i>IMMEDIATELY PRECEDES,</i>	<i>IMMEDIATELY SUCCEEDS</i>

-- Donde trabaja 217 02-1-2011

```
SELECT Name, Edept
FROM Emp WHERE ENo = 217
AND EPeriod CONTAINS DATE '2011-01-02'
```

-- Todos Dptos trabajo 217 entre 2-1-2010 y 1-1-2011

```
SELECT Ename, Edept
FROM Emp WHERE ENo = 217 AND
EPeriod OVERLAPS PERIOD (DATE '2010-1-1', DATE '2011-1-1')
```



# Bases de Datos Temporales

## Recordamos para Consultas

---

**Operadores SQL2011:** *Similares Operadores Allen.*

2011: "**OVERLAPS**" superposición periodo completo.

Allen: "**OVERLAPS**" superposición de puntos de periodos.

2011: "X **CONTAINS** Y" equivale a la expresión Allen:

"(X CONTAINS Y) O (X STARTS Y) O (X FINISHES Y) O (X EQUAL Y)".

2011: "X **PRECEDES** Y" equivalente a la expresión Allen:

"(X BEFORE Y) O (X MEETS Y)".

2011: "X **SUCCEEDS** Y" in SQL:2011 es equivalente a la expresión booleana usando Allen operadores "(X AFTER Y) OR (X MET\_BY Y)".

2011: "X **EQUALS** Y", "X **IMMEDIATELY PRECEDES** Y", y "X **IMMEDIATELY SUCCEEDS** Y" es equivalentes a los operadores Allen "X **equal** Y", "X **meets** Y", and "X **met\_by** Y", respectivamente.

# Bases de Datos Temporales

## Tablas Temporales - TT

---

Ejemplo Tablas Temporales con Tiempo de Transacción:

```
CREATE TABLE Emp (  
  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 hecho 1-1-2012 9hs:

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

```
INSERT INTO Emp (Eno, Ename) VALUES (217, 'Juan');
```

# Bases de Datos Temporales

## Tablas Temporales - TT

---

### Ejemplo de Actualización:

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

Inicial

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

Actualizado

# Bases de Datos Temporales

## Tablas Temporales - TT

---

### Ejemplo de Eliminación:

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

Inicial

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

Actualizado

# Bases de Datos Temporales

## Tablas Temporales – Primary Key TT

---

### Claves Primarias y Ajenas en Tablas con Versionado:

Son mas simples, puesto que en la tabla principal solo hay filas actuales.

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

## Foreign Key TT

```
ALTER TABLE Emp  
    ADD FOREIGN KEY (Edept) REFERENCES Dept (DNo)
```

# Bases de Datos Temporales

## 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'
```

# Bases de Datos Temporales

## Tablas Temporales – Consultas TT

---

### Consulta **FOR SYSTEM\_TIME BETWEEN**:

En este caso, la consulta recupera las filas de Emp que estaban registradas en el periodo 2-1-2011 y 31-12-2011

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

En cambio si no se hace mención a **FOR SYSTEM\_TIME**, se supone a partir de **CURRENT\_TIMESTAMP** (filas actuales).

```
SELECT Eno, EName, Sys_Start, Sys_End FROM Emp
```

# Bases de Datos Temporales

## Tablas Temporales – Consultas TT

---

**Para recuperar todo (actual e histórico):**

Finalmente, para recuperar tanto las filas actuales como las filas históricas de una tabla versionada por el sistema, se puede usar una consulta de del tipo que se muestra a continuación:

```
SELECT ENo, EName, Sys_Start, Sys_End
FROM Emp FOR SYSTEM_TIME
    FROM TIMESTAMP '0001-01-01 00:00:00' --Mínima
    TO TIMESTAMP '9999-12-31 23:59:59' --Máxima
```



# Bases de Datos Temporales

## Tablas Temporales – Tablas Bitemporales

---

### Tablas BiTemporales:

```
CREATE TABLE Emp (  
  ENo INTEGER,      EName VARCHAR(30),  
  EStart DATE,      EEnd DATE,  
  EDept INTEGER,  
  PERIOD FOR EPeriod (EStart, EEnd),  
  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),  
  PRIMARY KEY (ENo, EPeriod WITHOUT OVERLAPS),  
  FOREIGN KEY (Edept, PERIOD EPeriod)  
    REFERENCES Dept(DNo, PERIOD DPeriod)  
)  
  WITH SYSTEM VERSIONING;
```

# Bases de Datos Temporales

## Tablas Temporales – Tablas Bitemporales

---

### Consultas con ambas Dimensiones:

```
SELECT ENo, EDept FROM Emp WHERE ENo = 22217  
    FOR SYSTEM_TIME AS OF TIMESTAMP '2011-07-01 00:00:00'  
    AND EPeriod CONTAINS DATE '2010-12-01'
```

Por ejemplo, la consulta devuelve el departamento donde trabajaba el empleado 22217 1 de diciembre de 2010, que estaba registrado en la base de datos el 1 de julio, 2011.

# RESUMEN

---

- Propuesta soporte temporal como parte de la aplicación.
- Standard SQL:2011
- **Propuesta SQLServer**
- Propuesta de Oracle - Informativo
- Propuesta de PostgreSQL - Informativo
- Instalación PostgreSQL - Informativo

# Bases de Datos Temporales

## SqlServer – Manifiesto Microsoft

---

<https://msdn.microsoft.com/en-us/library/dn935015.aspx>

“SQL Server 2016 introduces support for system-versioned temporal tables as a database feature that brings built-in support for providing information about data stored in the table at any point in time rather than only the data that is correct at the current moment in time. Temporal is a database feature that was introduced in ANSI SQL 2011 and is now supported in SQL Server 2016.”

*SQL Server 2016 introduce soporte para tablas temporales mediante sistema-de-versionado como una función de base de datos que trae soporte integrado para proporcionar información sobre los datos almacenados en la tabla en cualquier punto en el tiempo en lugar de sólo los datos en el momento actual. Temporal es una característica de base de datos que se introdujo en ANSI SQL 2011 y ahora se admite en SQL Server 2016 .*

# Bases de Datos Temporales

## SqlServer – Manifiesto Microsoft

---

<https://msdn.microsoft.com/en-us/library/dn935015.aspx>

### ***¿Cómo es la propuesta temporal de SqlServer?***

Tiene un Sistema de versionado, lleva el Tiempo de Transacción:

- Tabla histórica: Valor anterior para cada fila. Pasado.

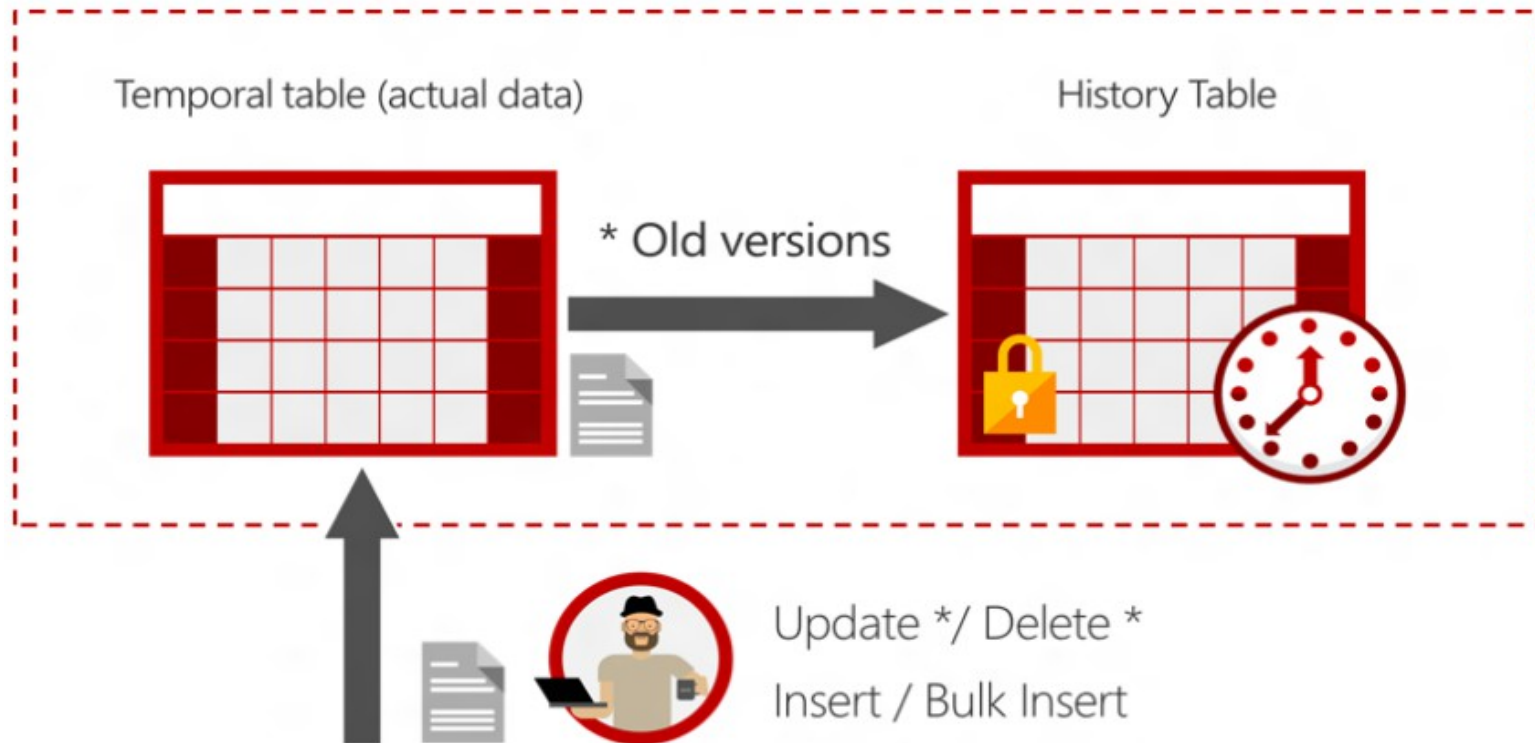
Cada tabla tiene las siguientes columnas adicionales *datetime2* para definir el período de transacción:

- Inicio de Periodo (ttinicio): Instante inicial de transacción "GENERATED ALWAYS AS ROW START".
- Fin de Periodo (ttfinal): Instante final de transacción "GENERATED ALWAYS AS ROW END".
- Y un PERIOD FOR SYSTEM\_TIME (nombre\_col\_ttinicio, nombre\_col\_ttfinal)

# Bases de Datos Temporales

## SqlServer – Manifiesto Microsoft

<https://msdn.microsoft.com/en-us/library/dn935015.aspx>



*Cada vez que hacemos update o delete los datos previos a la actualización se guardaran automáticamente en la tabla de historia (pasado).*

# Bases de Datos Temporales

## SqlServer – Manifiesto Microsoft

---

<https://msdn.microsoft.com/en-us/library/dn935015.aspx>

```
CREATE TABLE dbo.Employee (  
    [EmployeeID] int NOT NULL PRIMARY KEY CLUSTERED,  
    [Name] nvarchar(100) NOT NULL,  
    [Position] varchar(100) NOT NULL,  
    [Department] varchar(100) NOT NULL,  
    [Address] nvarchar(1024) NOT NULL,  
    [AnnualSalary] decimal (10,2) NOT NULL,  
    [TTFrom] datetime2 GENERATED ALWAYS AS ROW START,  
    [TTTo] datetime2 GENERATED ALWAYS AS ROW END,  
    PERIOD FOR SYSTEM_TIME (TTFrom, TTTo)  
) WITH (SYSTEM_VERSIONING  
        = ON (HISTORY_TABLE = dbo.EmployeeHistory));
```

# Bases de Datos Temporales

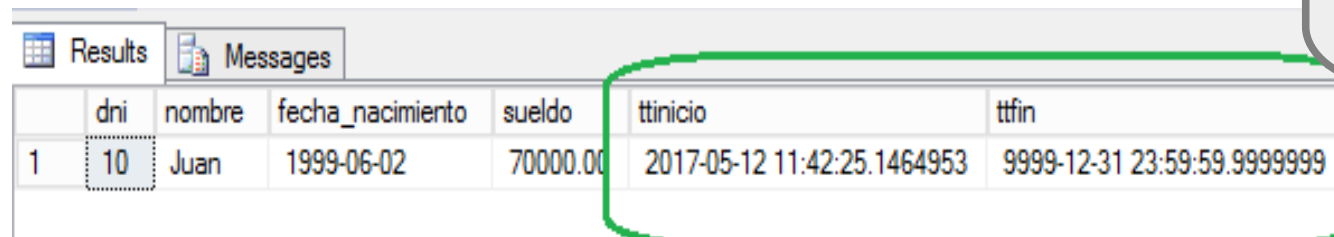
## SqlServer – Manifiesto Microsoft

---

<https://msdn.microsoft.com/en-us/library/dn935015.aspx>

**INSERT:** En un INSERT, el sistema fija en **SysStartTime** ttinicio con la hora de la transacción actual. También fija en **SysEndTime** ttfin al valor máximo de 9999- 12-31. Esto marca la fila como abierta.

```
INSERT INTO empleados
(dni, nombre, fecha_nacimiento, sueldo)
VALUES (30, 'Maria', '1994-05-30', 23000);
```



	dni	nombre	fecha_nacimiento	sueldo	ttinicio	ttfin
1	10	Juan	1999-06-02	70000.00	2017-05-12 11:42:25.1464953	9999-12-31 23:59:59.9999999

Solución  
a UC



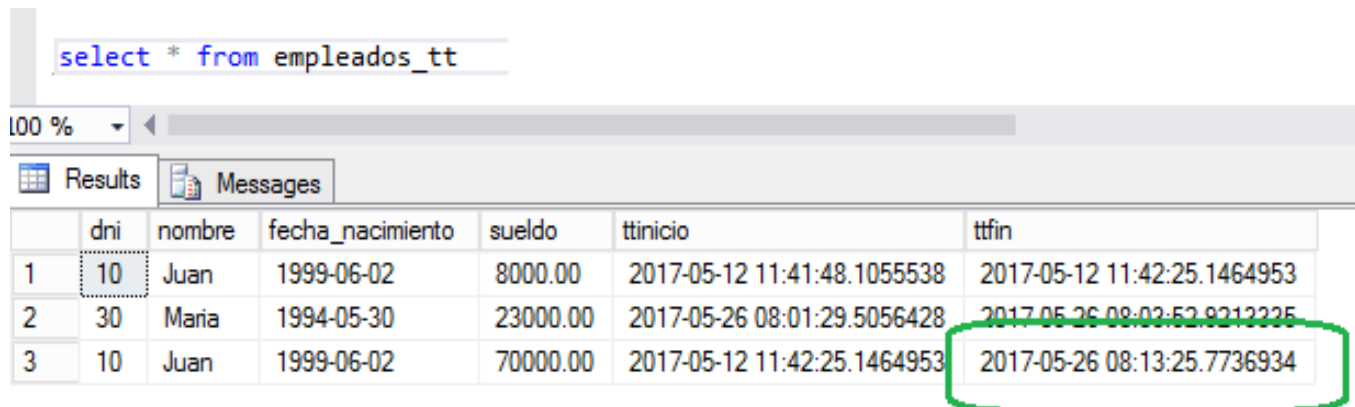
# Bases de Datos Temporales

## SqlServer – Manifiesto Microsoft

<https://msdn.microsoft.com/en-us/library/dn935015.aspx>

```
UPDATE EMPLEADOS SET SUELDO = 18200 WHERE DNI = 10;
```

**UPDATE:** En un update, el sistema almacena el valor anterior de la fila en la tabla de historial y establece el valor de la columna SysEndTime a la hora de inicio de la transacción actual. Esto marca la fila como cerrada, con un tiempo de registro para el que la fila era válida.



select \* from empleados\_tt

	dni	nombre	fecha_nacimiento	sueldo	ttinicio	ttfin
1	10	Juan	1999-06-02	8000.00	2017-05-12 11:41:48.1055538	2017-05-12 11:42:25.1464953
2	30	Maria	1994-05-30	23000.00	2017-05-26 08:01:29.5056428	2017-05-26 08:02:52.8212225
3	10	Juan	1999-06-02	70000.00	2017-05-12 11:42:25.1464953	2017-05-26 08:13:25.7736934

# Bases de Datos Temporales

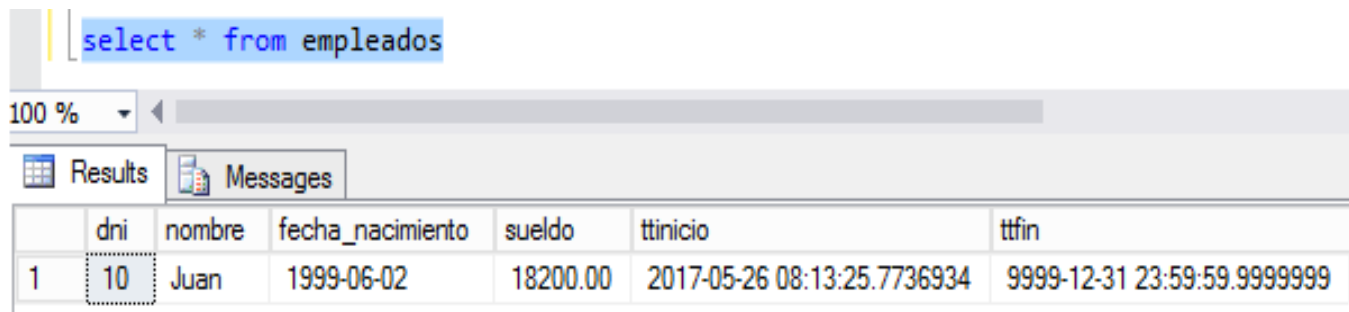
## SqlServer – Manifiesto Microsoft

---

<https://msdn.microsoft.com/en-us/library/dn935015.aspx>

```
UPDATE EMPLEADOS SET SUELDO = 18200 WHERE DNI = 10;
```

En la tabla actual, se actualiza la fila con su nuevo valor y el sistema fija el valor de SysStartTime a la hora de inicio de la operación. El valor de la fila actualizada en la tabla actual para SysEndTime sigue siendo el valor máximo de 9999-12-31.



The screenshot shows a SQL query window with the command `select * from empleados`. Below the query, the 'Results' tab is active, displaying a single row of data from the 'empleados' table. The columns are: `dni`, `nombre`, `fecha_nacimiento`, `sueldo`, `ttinicio`, and `ttfin`. The row contains the values: 1, 10, Juan, 1999-06-02, 18200.00, and 2017-05-26 08:13:25.7736934. The `ttfin` column is not visible in the screenshot, but the text indicates it is 9999-12-31 23:59:59.9999999.

	dni	nombre	fecha_nacimiento	sueldo	ttinicio	ttfin
1	10	Juan	1999-06-02	18200.00	2017-05-26 08:13:25.7736934	9999-12-31 23:59:59.9999999

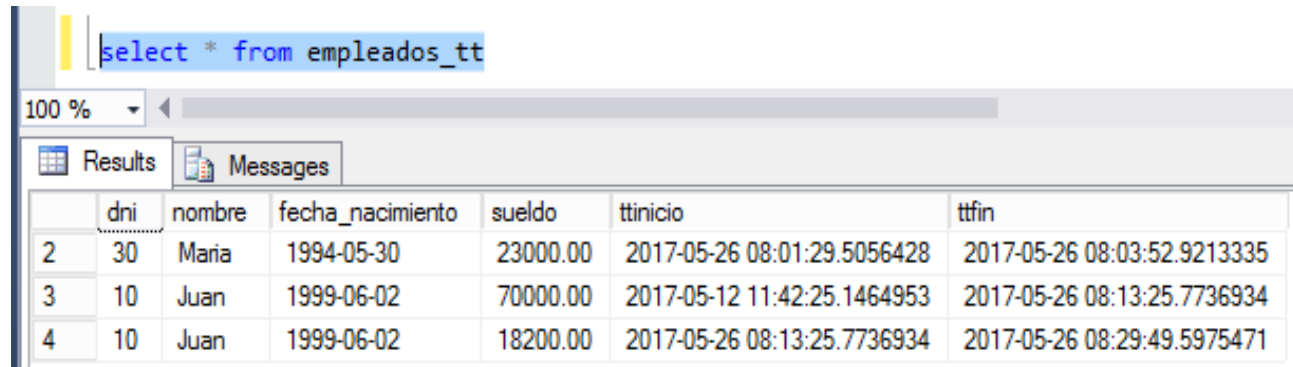
# Bases de Datos Temporales

## SqlServer – Manifiesto Microsoft

---

<https://msdn.microsoft.com/en-us/library/dn935015.aspx>

**DELETE:** En un DELETE, el sistema almacena el valor anterior de la fila en la **tabla de historial** y establece el valor de **SysEndTime** a la hora de finalización de la transacción actual. Esto marca la fila como cerrada, con un tiempo de registro para el que la fila anterior era válida.



The screenshot shows a SQL query window with the command `select * from empleados_tt`. Below the query, the 'Results' tab is active, displaying a table with 7 columns: `dni`, `nombre`, `fecha_nacimiento`, `suelo`, `ttinicio`, and `tfin`. The table contains three rows of data.

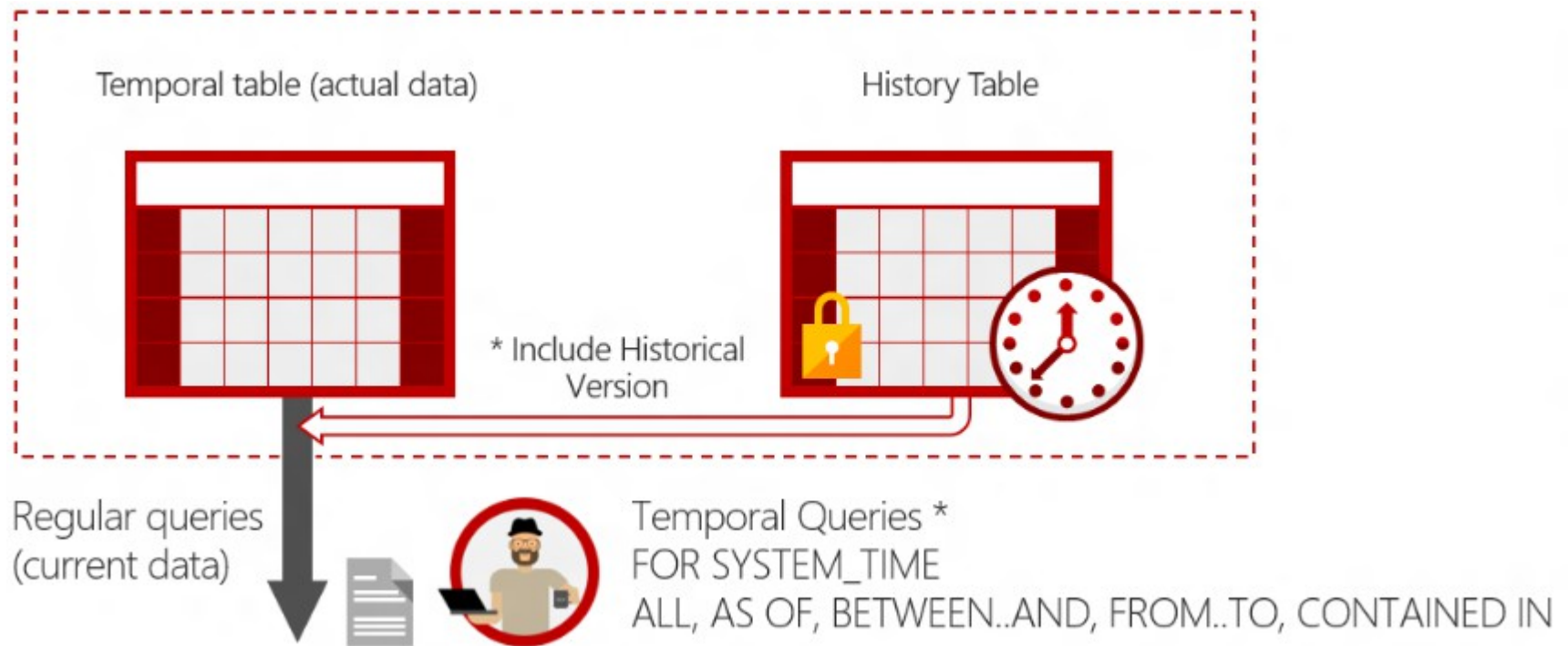
	dni	nombre	fecha_nacimiento	suelo	ttinicio	tfin
2	30	Maria	1994-05-30	23000.00	2017-05-26 08:01:29.5056428	2017-05-26 08:03:52.9213335
3	10	Juan	1999-06-02	70000.00	2017-05-12 11:42:25.1464953	2017-05-26 08:13:25.7736934
4	10	Juan	1999-06-02	18200.00	2017-05-26 08:13:25.7736934	2017-05-26 08:29:49.5975471

En la **tabla actual**, se elimina la fila. Las consultas de la tabla actual no devolverán esta fila. Solamente las consultas que tienen que ver con los datos de la **historia** devuelven datos para las filas cerradas.

# Bases de Datos Temporales

## SqlServer – Manifiesto Microsoft

<https://msdn.microsoft.com/en-us/library/dn935015.aspx>



*Cuando hacemos una consulta con `SYSTEM_TIME` el sistema "sabe" de que tabla tomar los datos, actual o de versiones (historia).*

# Bases de Datos Temporales

## SqlServer – Consultas

---

Para esto se extiende select con  
FOR SYSTEM\_TIME AS OF fecha.

Estado del empleado 10 al 28-2-2017

```
SELECT * FROM empleados
    FOR SYSTEM_TIME AS OF '2017-02-28'.
WHERE DNI = 10
```

Estado actual del empleado 10

```
SELECT * FROM empleados
WHERE DNI = 10
```

Historia del empleado 10 incluyendo actualidad

```
SELECT * FROM empleados
    FOR SYSTEM_TIME ALL
WHERE DNI = 10
```

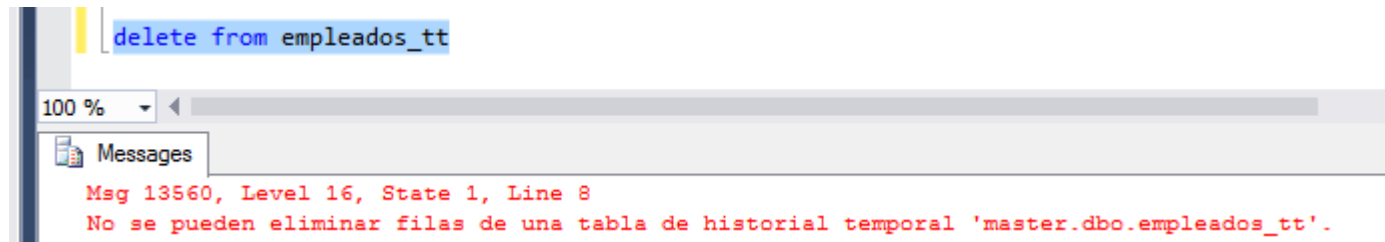
# Bases de Datos Temporales

## Tabla de Versiones

---

Hay posibilidades de modificar la tabla de versiones?

```
DELETE FROM empleados_tt
```



Tampoco se puede updatear.

Para poder realizar mantenimiento sobre esta se debe “apagar” el versionado.

```
ALTER TABLE empleados  
SET (SYSTEM_VERSIONING = Off);
```

# Bases de Datos Temporales

## Conversión a Tablas Temporales

---

Supongamos que Empleados existe como tabla SQL 2008

```
CREATE TABLE empleados (  
dni integer not null primary key,  
nombre Varchar(50),  
fecha_nacimiento date,  
Sueldo numeric(12,2));
```

Lo primero que haremos es agregar 2 atributos datetime2 y constituir el periodo.

```
ALTER TABLE empleados ADD  
    ttinicio    DATETIME2 GENERATED ALWAYS AS ROW START  
                NOT NULL CONSTRAINT EMPLEADOS_TTINICIO  
                DEFAULT ('19000101'),  
    ttfin       DATETIME2  GENERATED ALWAYS AS ROW END NOT  
                NULL CONSTRAINT EMPLEADOS_TTFIN  
                DEFAULT ('19001231'),  
    PERIOD FOR SYSTEM_TIME (TTINICIO, TTFIN);
```

# Bases de Datos Temporales

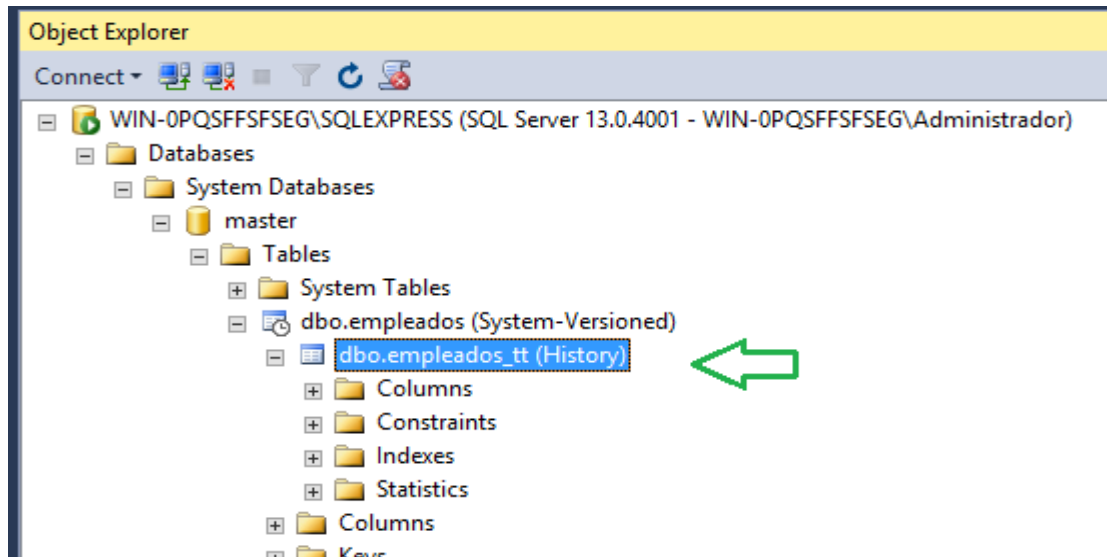
## Convención a Tablas Temporales

---

Luego “prendemos” el versionado.

```
ALTER TABLE empleados  
SET (SYSTEM_VERSIONING = ON  
    (HISTORY_TABLE = dbo.empleados_tt));
```

Esto crea automáticamente history\_table, el object explorer nos muestra esta situación.





# Bases de Datos Temporales

## Tiempo de Validez en SqlServer 2016

---

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;~~

En el DELETE este problema se agudiza.

Igual situación sufren los operadores temporales del standard.

Esperemos mas cobertura de SQL2011 en futuras versiones.

# RESUMEN

---

- Propuesta soporte temporal como parte de la aplicación.
- Standard SQL:2011
- Propuesta SQLServer
- **Propuesta de Oracle** - **Informativo**
- Propuesta de PostgreSQL - **Informativo**
- Instalación PostgreSQL - **Informativo**

# Bases de Datos Temporales

## Propuesta de ORACLE

---

- Según la documentación oficial: **Oracle Compliance To Core SQL:2011**

[https://docs.oracle.com/database/121/SQLRF/ap\\_standard\\_sql003.htm#SQLRF55516](https://docs.oracle.com/database/121/SQLRF/ap_standard_sql003.htm#SQLRF55516)

### T180, System-versioned tables

Oracle's **Flashback** capability is substantially the same as the standard's system-versioned tables. Some key differences are:

- In Oracle you do not need to designate particular tables for journaling; **all tables are journaled**.
- In Oracle, **LOB columns need to be individually designated for journaling**, because of the potential for large amounts of data. The standard has no analogous provision.
- In Oracle you **need a privilege** in order to **read historical data**.
- In Oracle, the retention **period is specified through DBA actions**, rather than DDL at the table level.
- In the standard, journaled tables have columns to record the start and end timestamps for the row. **In Oracle, this is provided through pseudocolumns.**

# Bases de Datos Temporales

## Propuesta de ORACLE

---

- ORACLE aborda la implementación con un Wrapper (capa intermedia).
- Trabaja como una caja negra, prácticamente poco adaptable a las necesidades de los usuarios.
- No es compatible con TSQL2, pero de alguna forma aborda la gestión del tiempo:

```
EXECUTE dbms_wm.GoToDate('06-APR-2010');  
SELECT number, geom  
FROM parcel_history  
WHERE number = 1490;
```

# Bases de Datos Temporales

## Propuesta de ORACLE

---

Permite mediante el Procedimiento EnableVersioning activar la gestión del tiempo válido, el atributo WM\_VALID de tipo WM\_PERIOD es añadido por este procedimiento automática/e.

```
CREATE TABLE employee (  
  id INTEGER PRIMARY KEY,  
  name CHAR(20),  
  surname CHAR(20),  
  salary NUMBER);
```

```
EXECUTE DBMS_WM.EnableVersioning  
( 'employee', 'VIEW_WO_OVERWRITE', FALSE, TRUE);
```

```
INSERT INTO employee VALUES  
(101, 'Petar', 'Petrović', 8000,  
WMSYS.WM_PERIOD(TO_DATE('12-06-2001', 'MM-DD-YYYY'),  
TO_DATE('01-01-2013', 'MM-DD-YYYY'))  
);
```

# Bases de Datos Temporales

## Propuesta de ORACLE

---

Actualización de Tiempo de Validez.

<u>id</u>	name	surname	salary	<u>VM_VALID</u>	<u>VM_VALID</u>
101	Petar	Petrović	8000	6.12.2001.	1.1.2013.

```
EXECUTE DBMS_WM.SetValidTime( TO_DATE('01-01.2005', 'MM-DD-YYYY'), DBMS_WM.UNTIL_CHANGED);
```

```
UPDATE employee SET salary = 10000  
WHERE id = 101;
```

<u>id</u>	name	surname	salary	<u>VM_VALID</u>	<u>VM_VALID</u>
101	Petar	Petrović	8000	6.12.2001.	1.1.2005.
101	Petar	Petrović	10000	1.1.2005.	NULL

El tiempo de validez se establece mediante procedimiento SetValidTime , después se ejecuta la actualización utilizando este tiempo de validez.

# Bases de Datos Temporales

## Propuesta de ORACLE

---

### **Sentencia UPDATE** (dos formas de trabajo)

#### **Secuenciado**

- Establecer el tiempo de validez utilizando el procedimiento SetValidTime
- El atributo VM\_VALID no se cambia explícitamente.
- Oracle se encarga de que periodos de tiempo no se solapan y , si es necesario, inserta tuplas adicionales en relación.

#### **No Secuenciado**

- El atributo VM\_VALID se cambia explícitamente en la sentencia.
- El usuario tiene que cuidar de que los períodos de tiempo válidos no se superpongan (OVERLAP).

### **Sentencia INSERT**

- Los períodos de tiempo válidos han de ser introducidos sin solapamiento (OVERLAP) u Oracle lanza una excepción.

# Bases de Datos Temporales

## Propuesta de ORACLE

---

### **Sentencia DELETE** (dos formas de trabajo)

#### **Secuenciado**

- Establecer el tiempo de validez utilizando el procedimiento SetValidTime
- Los datos se actualizan según valid time.
- Como resultado se pueden producir nuevas tuplas de una relación

#### **No Secuenciado**

- Si todas las tuplas para un determinado empleado necesitan ser eliminadas, se debe definir el tiempo válido para el delete con ( SetValidTime )



# Bases de Datos Temporales

## Propuesta de ORACLE

---

El tipo de dato WM\_PERIOD se define de la siguiente manera:

```
CREATE TYPE WM_PERIOD AS OBJECT (  
    validfrom TIMESTAMP WITH TIME ZONE,  
    validtill TIMESTAMP WITH TIME ZONE);
```

Limite inferior **validFrom** es inclusivo, mientras que limite superior **validTill** es exclusivo.

Las operaciones sobre valid time son ( subconjunto de operadores Allens):

WM_OVERLAPS	Overlaps
WM_CONTAINS	Constains
WM_MEETS	Meets
WM_EQUALS	=
WM_LESSTHEN	Less que
WM_GREATHERTHEN	Greather que

# Bases de Datos Temporales

## Propuesta de ORACLE

---

Las operaciones sobre periodos son:

WM_INTERSECTION	Intersección
WM_LDIFF	Diferencia desde el left
WM_RDIFF	Diferencia desde el right

Ejemplo: Selecciona todos los empleados cuyo salario fue cambiado al menos una vez entre 1.1.2004. y 31.12.2005.

```
SELECT id, name, surname
FROM employee
WHERE WM_INTERSECTS (employee.WM_VALID,
                     WM_PERIOD (TO_DATE ('01-01.2004.',
                     'MM-D.D.YYYY'), TO_DATE ('12-31.2005.',
                     'MM-D.D.YYYY'))
                     ) = 1
GROUP BY id, name, surname
HAVING COUNT (*) > 1;
```

# Bases de Datos Temporales

## Propuesta de ORACLE

---

Manejando el tiempo de transacción.

- Uso de Vistas Especiales.
- Para la relación de empleados una vista empleados\_historia es creada.
- Contiene todos los atributos de empleados mas algunos otros.
- Atributos adicionales:

WM_WORKSPACE	Workspace de la tupla
WM_VERSION	Numero de versión de la tupla.
WM_USERNAME	Usuario que ingreso la tupla
WM_OPTYPE	Operación realizada en la tupla: D (delete), I (insert) ili U (update)
WM_CREATETIME	Timestamp en el que se crea.
WM_RETIRETIME	Timestamp de eliminacion o actualizacion.

# RESUMEN

---

- Propuesta SQLServer
- Propuesta de Oracle
- **Propuesta de PostgreSQL**
- Instalación PostgreSQL

# Bases de Datos Temporales


## Propuesta de PostgreSQL

---

- PostgreSQL soporta tipos de datos rango desde versión 2.0
- Tipos Rango

- `int4range` – rango de `integer`
- `int8range` – rango de `bigint`
- `numrange` – rango de `numeric`

- `tsrange` – rango de `timestamp sin timezone`
- `tstzrange` – rango de `timestamp con timezone`
- `daterange` – rango de `date`



Periodos  
De Tiempo

No posee funciones específicas para gestionar **tiempo valido y de transacción**.

# Bases de Datos Temporales

## Propuesta de PostgreSQL

---

- Intervalos Abiertos y Cerrados

Todas las definiciones de rango nos permite definir intervalos abiertos '(' o cerrados '[', ejemplo de uso con `daterange`:

- `daterange` – rango de `date`

`Daterange ( inicio date, fin date, 'propiedades de abierto-cerrado')`

Ej:

`Daterange( '2016-01-30', '2016-02-15','(]') – ( '30-1-16', '15-2-16' ]`

Mismo uso con:    `int4range`    `int8range`    `numrange`  
                  `tstzrange`    `tsrange`

# Bases de Datos Temporales

## PostgreSQL

---

El límite superior puede ser definido como abierto o cerrado.

```
'[2010-01-01 11:30, 2010-01-01 15:00)::tsrange  
daterange('2001-12-06', '2005-01-01', '[')')
```

Pero PostgreSQL siempre lo expresa  
'[]' → Inicio Cerrado, fin abierto.

Definamos:

('30-1-2012', '15-2-2016' ] → Ver propiedades abier/cerr!!!!

[ '30-1-2012', '15-2-2016' ]



# Bases de Datos Temporales

## PostgreSQL

---

COMO definimos UC o "Hasta ahora" ?

- Usamos NULL en la función de rango, ej:

```
daterange('2016-3-2', null, '[]')
```

Y entonces el NULL → intervalo Empty?

- Usamos la Constante figurativa empty.

```
daterange('empty')
```

**Nota: ver tb función** `isempty`

Definamos:

Empty → Tramo no definido o vacío

( uc , uc ) → es lo mismo que empty?





# Bases de Datos Temporales

## PostgreSQL

---

### INFORMATIVO

Indices de tipo GiST o SP-GiST se pueden utilizar con tipos range para mejorar la performance de los operadores de comparación (operadores Allens).

```
CREATE TABLE empleados (  
    nombre CHAR(20),  
    salario NUMBER,  
    tv tsrange);          -- tv: Tiempo Valido  
  
CREATE INDEX emp_idx ON empleados USING gist(tv);
```

# Bases de Datos Temporales

## Propuesta de PostgreSQL

---

- Definamos los siguientes rangos:

( '23-1-2010', '15-4-2016' ] → Ver propiedades abier/cerr!!!!

[ '31-1-2009', '20-12-2016' ]

[ '11-7-2021', uc / null ] → Como se genera rango abierto  
→ Esta bien la propiedad abier/cerr?

Empty → Tramo no definido o vacío

( uc , uc ) → es lo mismo que empty?

# Bases de Datos Temporales

## Propuesta de PostgreSQL

---

- Definamos los siguientes rangos:

( '23-1-2010', '15-4-2016' ] → Ver propiedades abier/cerr!!!!

[ '31-1-2009', '20-12-2016' ]

[ '11-7-2021', uc / null ] → Como se genera rango abierto  
→ Esta bien la propiedad abier/cerr?

Empty → Tramo no definido o vacío

( uc , uc ) → es lo mismo que empty?

# Bases de Datos Temporales

## Propuesta de PostgreSQL

---

Como implementa los operadores begin y end Allen?

- **Begin** → **lower()**
- **End** → **upper()**

Obtengamos begin y end de los intervalos:

`'[2016-03-13 , 2017-03-16]'`

`Empty` → Tramo no definido o vacío

`( uc , uc )` → ?

Que relación tienen estos resultados con el nulo?

# Bases de Datos Temporales

## Propuesta de PostgreSQL

---



### PROBLEMA

Que sucede si queremos manejar la asignación de recursos físicos en periodos excluyentes?

```
CREATE TABLE librosreservas (  
    ejemId    INTEGER,           cliente_id INTEGER,  
    tramoreserva DATERANGE  
);
```

```
INSERT INTO librosreservas VALUES  
    (1108, 12, '[2016-05-23, 2016-05-26)');  
INSERT INTO librosreservas VALUES  
    (1108, 3, '[2016-05-25, 2016-05-30)');
```

# Bases de Datos Temporales

## PostgreSQL

---

Para evitar situaciones de OVERLAP en los periodos podemos usar una TConstraint `EXCLUDE` constraint e indice `GIST`.

```
ALTER TABLE librosreservas ADD CONSTRAINT lrcexclude  
    EXCLUDE USING gist (ejemid WITH =, tramoreserva  
                        WITH &&);
```

```
CREATE EXTENSION btree_gist; --Cree extensión si es neces.
```

Esta constraint de tabla “tconstraint” no permite que se realice overlap para el mismo libro en la fecha de reserva.

# Bases de Datos Temporales

## PostgreSQL – Operadores (I)

---

Operadores sobre tipos rango:

Operador	Descripción
=	Igual
<>	distinto

Ejemplo:

```
select
  Daterange('2014-04-20','2014-04-24','[]') =
  Daterange('2014-04-20','2014-04-24','()')
```

Resultado: False

# Bases de Datos Temporales

## PostgreSQL – Operadores (II)

---

Operadores sobre tipos rango:

Operador	Descripción
<, <=	Menor, Menor o igual
>, >=	Mayor, Mayor o igual

Primero compara el limite inferior, y si solo si este es igual compara el limite superior. Ejemplo:

```
select
  Daterange('2014-04-19', '2023-04-24', '[') <
  Daterange('2014-04-20', '2016-04-24', '(')
```

Resultado: True

**Nota:** Funciona distinto a **Before** de TSQL2 (Verdadero:  $f1 < i2$  es verdadero  $[i1, f1][i2, f2]$ )



# Bases de Datos Temporales

## PostgreSQL – Operadores (III)

---

Operadores sobre tipos rango:

Operador	Descripción
<, <=	Menor, Menor o igual
>, >=	Mayor, Mayor o igual

Otro Ejemplo:

```
select
  Daterange('2014-04-20','2014-04-24','(]') >
  Daterange('2014-04-20','2014-04-24','[[]')
```

**Resultado:** True

→ True: Debido a la propiedad abierto del inicio de rango 2.

# Bases de Datos Temporales

## PostgreSQL – Operadores (IV)

---

Operadores sobre tipos rango:

Operador	Descripción
@>	Contains de Tsql2
<@	Contenido en

Ejemplo:

```
select
  Daterange('2014-04-01', '2014-04-24', '[]') @>
  Daterange('2014-04-11', '2014-04-12', '[]')
```

Resultado: True

**Nota:** @> Idem contains de TSQL2.

# Bases de Datos Temporales

## PostgreSQL – Operadores (IV)

---

Ejemplo de aplicación de CONTAINS: @>:

Se tiene la relación de alquileres:

```
CREATE TABLE AlquilerAutos(  
    patente    VARCHAR(8),  
    Dni INTEGER,  
    tramoalquiler DATERANGE  
);
```

Y se recibieron multas para el auto AUY850 los días 15 y 16 de mayo de este año.

Se desea saber si las multas corresponden a un mismo inquilino.

**Nota:** La tabla AlquilerAutos no posee registros de tramos consecutivos (**MEETS**) para un mismo auto y mismo inquilino.

# Bases de Datos Temporales

## PostgreSQL – Operadores (V)

---

Operadores sobre tipos rango:

Operador	Descripción
&&	Overlap de Tsql2
True	[i2 [i1, f1] f2] o [i1[i2, f1] f2]

Ejemplo:

```
select
  Daterange('2014-04-01','2014-04-24','()') &&
  Daterange('2014-04-11','2014-04-12','[]')
```

Resultado: True

**Notas:** && Idem Overlap de TSQL2.

Tiene en cuenta la propiedad de abierto o cerrado de los limites.

# Bases de Datos Temporales

## PostgreSQL – Operadores (IV)

---

Ejemplo de aplicación de OVERLAP: &&:

Se tiene la relación de alquileres:

```
CREATE TABLE AlquilerAutos (  
    patente    VARCHAR(8),  
    Dni        INTEGER,  
    tramoalquiler DATERANGE  
);
```

Y se recibieron multas para el auto AUY850 los días 15 y 16 de mayo de este año.

Se desea saber si el vehículo estuvo alquilado alguno de esos días.

**Nota:** La tabla AlquilerAutos no posee registros de tramos consecutivos (**MEETS**) para un mismo auto y mismo inquilino.

# Bases de Datos Temporales

## PostgreSQL – Operadores (VI)

---

Operadores sobre tipos rango:

Operador	Descripción
<<	<b>BEFORE</b> de Tsql2. Estrictamente menor.
>>	Estrictamente mayor.

Ejemplo:

```
select
```

```
Daterange('2014-04-01','2014-04-15','[]') <<  
Daterange('2014-04-15','2014-04-26','[]')
```

Resultado: False

```
select
```

```
Daterange('2014-04-01','2014-04-15','[]') <<  
Daterange('2014-04-15','2014-04-26','[]')
```

Resultado: True

**Notas:** Tiene en cuenta la propiedad de abierto o cerrado de los límites, si  $f1 = i2$  y alguno es abierto el resultado será true.

# Bases de Datos Temporales

## PostgreSQL – Operadores (VII)

---

Operadores sobre tipos rango:

Operador	Descripción
- -	Operador MEETS de TSQL2

Ejemplo:

```
select
  Daterange('2014-04-01', '2014-04-24', '[') -|-
  Daterange('2014-04-25', '2014-04-29', '[')
```

Resultado: True

**Nota:** tenga en cuenta que " $i2 = f1+1$  ó  $i1 = f2+1$ " da true.

# Bases de Datos Temporales

## PostgreSQL – Operadores (X)

---

Operadores sobre tipos rango:

Operador	Descripción
+	Union
*	Intersección
-	Diferencia

**Nota:** Si se realiza union sobre conjuntos disjuntos nos dará error.



# Bases de Datos Temporales

## PostgreSQL

---

### Ejemplos de Operador union +:

```
select
  daterange('2014-04-2','2014-04-24') +
  daterange('2014-04-17','2015-01-08')
```

Data Output		Explain	Messages
	union daterange		
1	[2014-04-02,2015-01-08)		

```
select
  daterange('2013-04-2','2013-04-24') +
  daterange('2014-04-17','2015-01-08')
```

Data Output	Explain	Messages	History
ERROR: el resultado de la unión de rangos no sería contiguo			
***** Error *****			
ERROR: el resultado de la unión de rangos no sería contiguo			
SQL state: 22000			

# Bases de Datos Temporales

## PostgreSQL

---

### Ejemplos de Operador intersección \*:

```
select
  daterange('2014-04-2','2014-04-24') *
  daterange('2014-04-17','2015-01-08')
```

Data Output		Explain	Messages	Hi
	intersección daterange			
1	[2014-04-17, 2014-04-24)			

```
select
  daterange('2013-04-2','2013-04-24') *
  daterange('2014-04-17','2015-01-08')
```

Data Output		Explain
	intersección daterange	
1	empty	

# Bases de Datos Temporales

## PostgreSQL

---

### Ejemplos de Operador diferencia -:

```
select
  daterange('2014-04-2','2014-04-24') -
  daterange('2014-04-17','2015-01-08')
```

Data Output	Explain	Messages
	<b>diferencia daterange</b>	
1	[2014-04-02,2014-04-17)	

```
select
  daterange('2013-04-2','2013-04-24') -
  daterange('2014-04-17','2015-01-08')
```

Data Output	Explain	Messages
	<b>diferencia daterange</b>	
1	[2013-04-02,2013-04-24)	

# Bases de Datos Temporales

## PostgreSQL

---

Funciones sobre tipos rango:

Función	Descripción
lower	Limite superior del rango (begin)
upper	Limite superior del rango (end)
isempty	Rango vacio
lower_inc	Booleana: si ext_inferior incluye el limite.
upper_inc	Booleana: si ext_superior incluye el limite.
lower_inf	Booleana: si ext_inferior es infinite o null
upper_inf	Booleana: si ext_sup es infinite o null

# Bases de Datos Temporales

## PostgreSQL

---

### Ejemplo de uso de funciones sobre tipos rango:

```
select lower_inf(daterange('2014-04-20',null)),
       upper_inf(daterange('2014-04-20',null,'()')),
       lower_inc(daterange('2014-04-20',null,'[]')),
       upper_inc(daterange('2014-04-20',null,'()')),
       isempty(daterange(null,null))
```

	Data Output	Explain	Messages	History	
	lower_inf boolean	upper_inf boolean	lower_inc boolean	upper_inc boolean	isempty boolean
1	f	t	f	f	f

# Bases de Datos Temporales

## PostgreSQL

---

### Malas noticias para UNFOLD y COALESCE

A la versión 9.5 de PostgreSQL no existe implementación de UNFOLD y COALESCE para relaciones Temporales.

Como solución de Minima podemos plantear una función o procedimiento almacenado que implemente estas funciones.



# Bases de Datos Temporales

## Propuesta de PostgreSQL

---

Ejemplo: Selecciona todos los empleados cuyo salario fue cambiado al menos una vez entre el 1-1-2004. y 31-12-

2

employee

id	name	surname	salary	valid:daterange
100	Ivan	Ivkošić	5000	('1.1.2010.', infinite)
101	Petar	Petrović	8000	('6.12.2001.', '1.1.2005')
101	Petar	Petrović	10000	('2.1.2005.', infinite)
102	Marko	Marković	6000	('1.7.2009.', '1.7.2011.')
105	Ante	Antić	6000	('20.5.1998.', '19.5.2008.')
105	Ante	Antić	7000	('20.5.2008.', infinite)

result

id	name	surname
101	Petar	Petrović

```
SELECT id, name, surname FROM employee
WHERE valid && daterange('2004-01-01', '2005-12-31', '[')
GROUP BY id, name, surname
HAVING COUNT(*) > 1;
```

# RESUMEN

---

- Propuesta soporte temporal como parte de la aplicación.
  - Standard Sql 2011
  - Propuesta SQLServer
  - Propuesta de Oracle
  - Propuesta de PostgreSQL
- Instalación ExtensiónPostgreSQL**
- Ejercicio Propuesto



# Bases de Datos Temporales

## Implementación PostgreSQL

Para configurar el Servidor PostgreSQL de trabajo de la materia utilizamos el artículo de "PostgreSQL Extension Network (PGXN)" titulado **temporal\_tables versión 1.0.2**, el cual se alcanza desde:

[http://pgxn.org/dist/temporal\\_tables/](http://pgxn.org/dist/temporal_tables/)

El mismo describe la instalación de extensión antes mencionada, "temporal\_tables\_1.0.2":

- Requerimientos
- Instalación
- Uso



# Bases de Datos Temporales

## Implementación PostgreSQL

---

[http://pgxn.org/dist/temporal\\_tables/](http://pgxn.org/dist/temporal_tables/)

Se utilizo un servidor PostgreSQL 9.4 sobre un servidor Open Suse Linux Leap 42.1 en un servicio virtualizado con Virtual Box 5.

El mismo se puede utilizar importando el Servicio Virtualizado:

BDA Uader (Open Suse Leap 4.21).vmdk

- Usuario: alumno, password: alumno
- Password de Root: alumno
- Schema: Bda

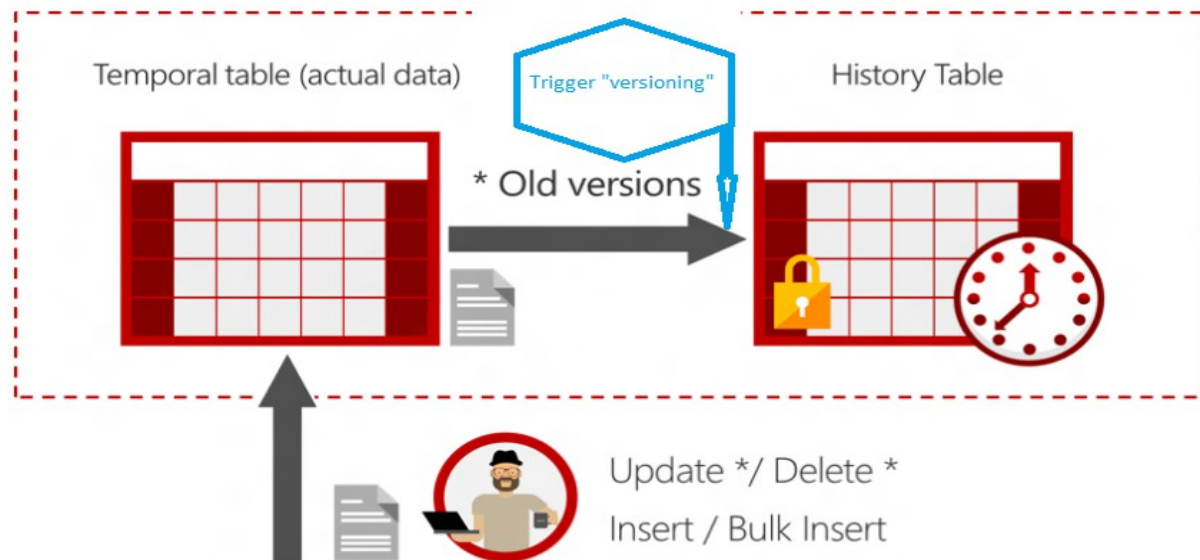
# Bases de Datos Temporales

## Implementación PostgreSQL

---

Modelo de la Propuesta:

[http://pgxn.org/dist/temporal\\_tables/](http://pgxn.org/dist/temporal_tables/)



El usuario debe crear la tabla de historia, y generar un trigger usando una función genérica versioning que realiza la autoadministración de tiempo de transacción.

# Bases de Datos Temporales

## Implementación PostgreSQL

---

Modelo de la Propuesta:

[http://pgxn.org/dist/temporal\\_tables/](http://pgxn.org/dist/temporal_tables/)

Creación de las tablas:

```
CREATE TABLE employees (                                -- Actual
    name text NOT NULL PRIMARY KEY,
    department text,
    salary numeric(20, 2)

ALTER TABLE employees -- periodo para transact-time Actual
ADD COLUMN sys_period tstzrange NOT NULL;

-- Tabla historia a partir de la actual
CREATE TABLE employees_history (LIKE employees);
```

# Bases de Datos Temporales

## Implementación PostgreSQL

---

Modelo de la Propuesta:

[http://pgxn.org/dist/temporal\\_tables/](http://pgxn.org/dist/temporal_tables/)

Creación del Trigger para tiempo de transacción:

```
CREATE TRIGGER versioning_trigger
BEFORE INSERT OR UPDATE OR DELETE ON employees
FOR EACH ROW
EXECUTE PROCEDURE
    versioning('sys_period', 'employees_history', true);
```

# Bases de Datos Temporales

## Implementación PostgreSQL

---

Modelo de la Propuesta [http://pgxn.org/dist/temporal\\_tables/](http://pgxn.org/dist/temporal_tables/)

Insertando filas: se realiza el 08-08-2006

```
INSERT INTO employees (name, department, salary)
VALUES ('Bernard Marx', 'HyC Centre', 10000);
```

```
INSERT INTO employees (name, department, salary)
VALUES ('Lenina Crowne', 'HyC Centre', 7000);
```

```
INSERT INTO employees (name, department, salary)
VALUES ('Helmholtz Watson', 'CIE', 18500);
```

### Tabla actual

Name	department	salary	sys_period
-----	-----	-----	-----
Bernard Marx	HyC Centre	10000	[2006-08-08, )
Lenina Crowne	HyC Centre	7000	[2006-08-08, )
Helmholtz Watson	CIE	18500	[2006-08-08, )

Nota: La tabla historia queda sin datos.

# Bases de Datos Temporales

## Implementación PostgreSQL

---

Modelo de la Propuesta [http://pgxn.org/dist/temporal\\_tables/](http://pgxn.org/dist/temporal_tables/)

Update de una Fila: se realiza el 27-02-2007

```
UPDATE employees
    SET salary = 11200          WHERE name = 'Bernard Marx';
```

### Tabla actual

Name	department	salary	sys_period
-----	-----	-----	-----
Bernard Marx	HyC Centre	11200	[2007-02-27, )
Lenina Crowne	HyC Centre	7000	[2006-08-08, )
Helmholtz Watson	CIE	18500	[2006-08-08, )

### Tabla Histórica

Name	department	salary	sys_period
-----	-----	-----	-----
Bernard Marx	HyC Centre	10000	[2006-08-08, 2007-02-27)

# Bases de Datos Temporales

## Implementación PostgreSQL

---

Modelo de la Propuesta [http://pgxn.org/dist/temporal\\_tables/](http://pgxn.org/dist/temporal_tables/)

Update de una Fila: se realiza el 24-12-2012

```
DELETE FROM employees WHERE name = 'Helmholtz Watson';;
```

### Tabla actual

Name	department	salary	sys_period
-----	-----	-----	-----
Bernard Marx	HyC Centre	10000	[2007-02-27, )
Lenina Crowne	HyC Centre	7000	[2006-08-08, )

### Tabla Histórica

Name	department	salary	sys_period
-----	-----	-----	-----
Bernard Marx	HyC Centre	10000	[2006-08-08,2007-02-27)
Helmholtz Watson	CIE	18500	[2006-08-08,2012-12-24)



# RESUMEN

---

- Propuesta soporte temporal como parte de la aplicación.
- Standard Sql 2011
- Propuesta SQLServer
- Propuesta de Oracle- **Informativo**
- Propuesta de Postgre- **Informativo**
- **Ejercicio Propuesto** - **Informativo**

# Bases de Datos Temporales

## Ejercicio Propuesto

---

Implemente una relación temporal con tiempo de validez y tiempo de transacción para el registro de la ciudad de residencia de una persona.

Tome como base para su estudio el caso de Bernardo desarrollado en clases anteriores, utilice los mismos atributos para calificar la relación.

- Bernardo Sabina nació un soleado 6/3/1985 en Zaragoza. Su madre registró su nacimiento al día siguiente.
- Tras terminar sus estudios de Ingeniería el 15/6/2007, Bernardo se mudó ese mismo día a Torre vieja a vender hamburguesas con queso.
- Sin embargo, no registró su mudanza hasta el 25 de junio, ya que tenía una competición nacional de ping pong.
- Pese a tener un futuro prometedor, Bernardo murió el 20/09/2012 de un ladrillazo en la cabeza, cuando iba a ver la presentación de su hermano en un partido de futbol. El equipo forense registró su muerte el mismo día.

# Bases de Datos Temporales

## Ejercicio Propuesto

---

Implemente una relación temporal con tiempo de validez y tiempo de transacción para el registro de la ciudad de residencia de una persona.

Tome como base para su estudio el caso de Bernardo desarrollado en clases anteriores, utilice los mismos atributos para calificar la relación.

Restricciones: Para la resolución puede usar un servidor de base de datos relacional tradicional o un servidor con extensión para manejo de relaciones temporales.

Nota: Cree los objetos necesarios para su correcto funcionamiento, trigger, vistas, etc.

Realice un informe de observaciones y consideraciones a tener en cuenta en su propuesta de solución.

# Bases de Datos Temporales

## Fuentes

---

Libro: Introducción a los SISTEMAS DE BASES DE DATOS

Autor: C.J. Date

Editorial: Addison Wesley

Libro: Fundamentos de Sistemas de Bases de Datos

Autor: Elmasri / Navathe

Editorial: Pearson

SqlServer: <https://msdn.microsoft.com/en-us/library/dn935015.aspx>

Oracle: <https://www.salvis.com/blog/2014/01/04/multi-temporal-database-features-in-oracle-12c/>