# LAPORAN TUGAS 1

# Temporal Database

Laporan dibuat untuk memenuhi salah satu tugas mata kuliah

IF4040 Pemodelan Data Lanjut

Disusun oleh:

| | |
|---|---|
| **Hilya Fadhilah Imania** | **13520024** |
| **Adiyansa Prasetya Wicaksana** | **13520044** |
| **Rayhan Kinan Muhannad** | **13520065** |

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2023**

## DAFTAR ISI

## Deskripsi Kasus

Dalam dunia bisnis, terutama di sektor layanan, informasi tentang pelanggan dan staf sangat penting. Untuk memahami dan menganalisis tren, serta untuk membuat keputusan bisnis yang tepat, perusahaan perlu menyimpan data historis. Salah satu cara untuk melakukannya adalah dengan menggunakan basis data temporal.

Dalam kasus ini, kita akan membahas representasi basis data temporal untuk sebuah perusahaan yang memiliki informasi tentang pelanggan dan staf.

**Tabel Pelanggan (customer):**

- Setiap pelanggan memiliki `id` unik yang dihasilkan secara otomatis.
- Nama pelanggan disimpan dalam kolom `name`.
- Periode langganan pelanggan, yaitu waktu dimulai dan berakhirnya langganan, disimpan dalam kolom `subscription_period`. Kolom ini menggunakan tipe data khusus `valid_period_domain` yang merepresentasikan periode waktu tertentu.

**Tabel Staf (staff):**

Setiap anggota staf memiliki `id` unik yang dihasilkan secara otomatis.

- Nama anggota staf disimpan dalam kolom `name`.
- Periode pekerjaan anggota staf, yaitu waktu dimulai dan berakhirnya pekerjaan, disimpan dalam kolom `employment_period`. Kolom ini juga menggunakan tipe data `valid_period_domain`.

Dengan menggunakan basis data temporal seperti ini, perusahaan dapat dengan mudah melacak perubahan sepanjang waktu, baik untuk pelanggan maupun staf. Misalnya, perusahaan dapat mengetahui pelanggan mana yang telah berlangganan selama periode tertentu atau staf mana yang bekerja selama periode tertentu.

Selain itu, representasi basis data temporal yang digunakan dalam kasus ini didasarkan pada konsep "Snodgrass Valid Timestamp (1D)". Konsep ini merupakan salah satu pendekatan dalam pembuatan basis data temporal yang memungkinkan pencatatan waktu validitas dari suatu data. Dengan menggunakan pendekatan ini, kita dapat menentukan kapan suatu data mulai valid dan kapan data tersebut berakhir validitasnya. Dalam implementasinya, kita memanfaatkan PostgreSQL.

## Skema

```
-- Define the temporal model with "id" as primary key
CREATE TABLE customer (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(255) NOT NULL,
    subscription_period valid_period_domain NOT NULL
);

CREATE TABLE staff (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(255) NOT NULL,
    employment_period valid_period_domain NOT NULL
);
```

## Supporting Types

```
-- Define the valid period type to be used in the temporal model
CREATE TYPE valid_period AS (
    start_timestamp BIGINT,
    end_timestamp BIGINT
);

CREATE DOMAIN valid_period_domain AS valid_period
CHECK (
    (VALUE).start_timestamp <= (VALUE).end_timestamp
);
```

## Procedure

1. Temporal Customer Insertion

```
-- Temporal customer insertion
CREATE PROCEDURE customer_insertion(
    input_name VARCHAR(255),
    input_subscription_period valid_period_domain
)
LANGUAGE plpgsql
AS $$
DECLARE
    found BOOLEAN;
    coalesced_period valid_period_domain;
BEGIN
    -- Check if the customer already exists
    SELECT EXISTS(
        SELECT 1
        FROM "customer"
        WHERE "customer"."name" = input_name
        AND temporal_can_merge("customer"."subscription_period",
input_subscription_period)
    ) INTO found;

    IF found THEN
        -- Insert the new period
        INSERT INTO "customer" ("name", "subscription_period")
        VALUES (input_name, input_subscription_period);

        -- Get coalesced period
        WITH "temp_table" AS (
            SELECT
temporal_coalesce_single("customer"."subscription_period") AS "period"
            FROM "customer"
            WHERE "customer"."name" = input_name
            AND temporal_can_merge("customer"."subscription_period",
input_subscription_period)
        )
        SELECT ("temp_table"."period").start_timestamp,
("temp_table"."period").end_timestamp
        INTO coalesced_period
        FROM "temp_table";

        -- Delete all the periods that overlap with the new one
        DELETE FROM "customer"
        WHERE "customer"."name" = input_name
        AND temporal_can_merge("customer"."subscription_period",
input_subscription_period);

        -- Insert the new period
        INSERT INTO "customer" ("name", "subscription_period")
        VALUES (input_name, coalesced_period);
    ELSE
        -- If the customer does not exist, insert it
        INSERT INTO "customer" ("name", "subscription_period")
        VALUES (input_name, input_subscription_period);
    END IF;

    -- Commit the transaction
    COMMIT;
END;
$$;
```

2. Temporal Staff Insertion

```
CREATE PROCEDURE staff_insertion(
    input_name VARCHAR(255),
    input_employment_period valid_period_domain
)
LANGUAGE plpgsql
AS $$
DECLARE
    found BOOLEAN;
    coalesced_period valid_period_domain;
BEGIN
    -- Check if the staff already exists
    SELECT EXISTS(
        SELECT 1
        FROM "staff"
        WHERE "staff"."name" = input_name
        AND temporal_can_merge("staff"."employment_period",
input_employment_period)
    ) INTO found;

    IF found THEN
        -- Insert the new period
        INSERT INTO "staff" ("name", "employment_period")
        VALUES (input_name, input_employment_period);

        -- Get coalesced period
        WITH "temp_table" AS (
            SELECT temporal_coalesce_single("staff"."employment_period") AS
"period"
            FROM "staff"
            WHERE "staff"."name" = input_name
            AND temporal_can_merge("staff"."employment_period",
input_employment_period)
        )
        SELECT ("temp_table"."period").start_timestamp,
("temp_table"."period").end_timestamp
        INTO coalesced_period
        FROM "temp_table";

        -- Delete all the periods that overlap with the new one
        DELETE FROM "staff"
        WHERE "staff"."name" = input_name
        AND temporal_can_merge("staff"."employment_period",
input_employment_period);

        -- Insert the new period
        INSERT INTO "staff" ("name", "employment_period")
        VALUES (input_name, coalesced_period);
    ELSE
        -- If the staff does not exist, insert it
        INSERT INTO "staff" ("name", "employment_period")
        VALUES (input_name, input_employment_period);
    END IF;

    -- Commit the transaction
    COMMIT;
END;
$$;
```

3. Temporal Customer Deletion

```
-- Temporal customer deletion
CREATE PROCEDURE customer_deletion(
    input_name VARCHAR(255)
)
LANGUAGE plpgsql
AS $$
BEGIN
    DELETE FROM "customer" WHERE "customer"."name" = input_name;
    COMMIT;
END;
$$;
```

4. Temporal Staff Deletion

```
-- Temporal staff deletion
CREATE PROCEDURE staff_deletion(
    input_name VARCHAR(255)
)
LANGUAGE plpgsql
```

```
AS $$
BEGIN
    DELETE FROM "staff" WHERE "staff"."name" = input_name;
    COMMIT;
END;
$$;
```

5. Temporal Customer Modification

```
-- Temporal customer modification
CREATE PROCEDURE customer_modification(
    input_name VARCHAR(255),
    input_subscription_period valid_period_domain
)
LANGUAGE plpgsql
AS $$
BEGIN
    -- Call the deletion and insertion procedures
    CALL customer_deletion(input_name);
    CALL customer_insertion(input_name, input_subscription_period);

    -- Commit the transaction
    COMMIT;
END;
$$;
```

6. Temporal Staff Modification

```
-- Temporal staff modification
CREATE PROCEDURE staff_modification(
    input_name VARCHAR(255),
    input_employment_period valid_period_domain
)
LANGUAGE plpgsql
AS $$
BEGIN
    -- Call the deletion and insertion procedures
    CALL staff_deletion(input_name);
    CALL staff_insertion(input_name, input_employment_period);

    -- Commit the transaction
    COMMIT;
END;
$$;
```

## Allen’s 13 Relation

1. Before

```
CREATE FUNCTION temporal_before_than(p1 valid_period_domain, p2
valid_period_domain)
RETURNS BOOLEAN
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN p1.end_timestamp + 1 < p2.start_timestamp;
END;
$$;
```

2. After

```
CREATE FUNCTION temporal_after_than(p1 valid_period_domain, p2
valid_period_domain)
RETURNS BOOLEAN
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN p1.start_timestamp > p2.end_timestamp + 1;
END;
$$;
```

3. Meet

```
CREATE FUNCTION temporal_meets(p1 valid_period_domain, p2
valid_period_domain)
RETURNS BOOLEAN
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN p1.end_timestamp + 1 = p2.start_timestamp;
END;
$$;
```

4. Met-By

```
CREATE FUNCTION temporal_meets_inverse(p1 valid_period_domain, p2
valid_period_domain)
RETURNS BOOLEAN
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN p1.start_timestamp = p2.end_timestamp + 1;
END;
$$;
```

5. Overlap

```
CREATE FUNCTION temporal_overlaps(p1 valid_period_domain, p2
valid_period_domain)
RETURNS BOOLEAN
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN p1.start_timestamp < p2.start_timestamp AND p1.end_timestamp <
p2.end_timestamp AND p1.end_timestamp >= p2.start_timestamp;
END;
$$;
```

6. Overlapped-By

```
CREATE FUNCTION temporal_overlaps_inverse(p1 valid_period_domain, p2
valid_period_domain)
RETURNS BOOLEAN
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN p1.start_timestamp > p2.start_timestamp AND p1.end_timestamp >
p2.end_timestamp AND p1.start_timestamp <= p2.end_timestamp;
END;
$$;
```

7. Start

```
CREATE FUNCTION temporal_starts(p1 valid_period_domain, p2
valid_period_domain)
RETURNS BOOLEAN
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN p1.start_timestamp = p2.start_timestamp AND p1.end_timestamp <
p2.end_timestamp;
END;
$$;
```

8. Started-By

```
CREATE FUNCTION temporal_starts_inverse(p1 valid_period_domain, p2
valid_period_domain)
RETURNS BOOLEAN
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN p1.start_timestamp = p2.start_timestamp AND p1.end_timestamp >
p2.end_timestamp;
END;
$$;
```

9. During

```
CREATE FUNCTION temporal_during(p1 valid_period_domain, p2
valid_period_domain)
RETURNS BOOLEAN
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN p1.start_timestamp > p2.start_timestamp AND p1.end_timestamp <
p2.end_timestamp;
END;
$$;
```

10. Contain

```
CREATE FUNCTION temporal_during_inverse(p1 valid_period_domain, p2
valid_period_domain)
RETURNS BOOLEAN
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN p1.start_timestamp < p2.start_timestamp AND p1.end_timestamp >
p2.end_timestamp;
END;
$$;
```

11. Finish

```
CREATE FUNCTION temporal_finishes(p1 valid_period_domain, p2
valid_period_domain)
RETURNS BOOLEAN
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN p1.start_timestamp > p2.start_timestamp AND p1.end_timestamp =
p2.end_timestamp;
END;
$$;
```

12. Finished-By

```
CREATE FUNCTION temporal_finishes_inverse(p1 valid_period_domain, p2
valid_period_domain)
RETURNS BOOLEAN
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN p1.start_timestamp < p2.start_timestamp AND p1.end_timestamp =
p2.end_timestamp;
END;
$$;
```

13. Equal

```
CREATE FUNCTION temporal_equal(p1 valid_period_domain, p2
valid_period_domain)
RETURNS BOOLEAN
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN p1.start_timestamp = p2.start_timestamp AND p1.end_timestamp =
p2.end_timestamp;
END;
$$;
```

## User-Defined Function

1. Merge Function

```
CREATE FUNCTION temporal_can_merge(p1 valid_period_domain, p2
valid_period_domain)
RETURNS BOOLEAN
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN NOT temporal_before_than(p1, p2) AND NOT temporal_after_than(p1,
p2);
END;
$$;

CREATE FUNCTION temporal_merge(p1 valid_period_domain, p2
valid_period_domain)
RETURNS valid_period_domain
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN (LEAST(p1.start_timestamp, p2.start_timestamp),
GREATEST(p1.end_timestamp, p2.end_timestamp))::valid_period_domain;
END;
$$;

CREATE FUNCTION temporal_merge_to_array(pArr valid_period_domain[], pNew
valid_period_domain)
RETURNS valid_period_domain[]
LANGUAGE plpgsql
AS $$
DECLARE
    pArrResult valid_period_domain[] := '{}';
    pIter valid_period_domain;
```

```
BEGIN
    FOREACH pIter IN ARRAY pArr LOOP
        -- If the new period can be merged with the current one, merge them
        IF temporal_can_merge(pIter, pNew) THEN
            pNew := temporal_merge(pIter, pNew);
        ELSE -- If not, append the current period to the result
            pArrResult := pArrResult || pIter;
        END IF;
    END LOOP;

    -- Append the new period to the result
    RETURN pArrResult || pNew;
END;
$$;

CREATE FUNCTION temporal_merge_to_array(pArr valid_period_domain[], pArrNew
valid_period_domain[])
RETURNS valid_period_domain[]
LANGUAGE plpgsql
AS $$
DECLARE
    pArrResult valid_period_domain[];
    pIter valid_period_domain;
BEGIN
    -- Initialize the result array
    pArrResult := pArr;

    -- Merge the new periods with the current ones
    FOREACH pIter IN ARRAY pArrNew LOOP
        pArrResult := temporal_merge_to_array(pArrResult, pIter);
    END LOOP;

    -- Return the result
    RETURN pArrResult;
END;
$$;
```

2. Intersect Function

```
CREATE FUNCTION temporal_can_intersect(p1 valid_period_domain, p2
valid_period_domain)
RETURNS BOOLEAN
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN NOT temporal_before_than(p1, p2)
    AND NOT temporal_after_than(p1, p2)
    AND NOT temporal_meets(p1, p2)
    AND NOT temporal_meets_inverse(p1, p2);
```

```
END;
$$;

CREATE FUNCTION temporal_intersection(p1 valid_period_domain, p2
valid_period_domain)
RETURNS valid_period_domain
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN (GREATEST(p1.start_timestamp, p2.start_timestamp),
LEAST(p1.end_timestamp, p2.end_timestamp))::valid_period_domain;
END;
$$;

CREATE FUNCTION temporal_intersect_to_array(pArr valid_period_domain[], pNew
valid_period_domain)
RETURNS valid_period_domain[]
LANGUAGE plpgsql
AS $$
DECLARE
    pArrResult valid_period_domain[] := '{}';
    pIter valid_period_domain;
BEGIN
    -- If the current array is empty, return the new period
    IF pArr IS NULL THEN
        RETURN ARRAY[pNew];
    END IF;

    FOREACH pIter IN ARRAY pArr LOOP
        -- If the new period can be intersected with the current one,
intersect them
        IF temporal_can_intersect(pIter, pNew) THEN
            pArrResult := pArrResult || temporal_intersection(pIter, pNew);
        END IF;
    END LOOP;

    -- Return the result
    RETURN pArrResult;
END;
$$;

CREATE FUNCTION temporal_intersect_to_array(pArr valid_period_domain[],
pArrNew valid_period_domain[])
RETURNS valid_period_domain[]
LANGUAGE plpgsql
AS $$
DECLARE
    pArrResult valid_period_domain[] = '{}';
    pIter valid_period_domain;
BEGIN
    -- If the current array is empty, return the new period
    IF pArr IS NULL THEN
        RETURN pArrNew;
    END IF;

    -- Intersect the new periods with the current ones
    FOREACH pIter IN ARRAY pArrNew LOOP
        pArrResult := pArrResult || temporal_intersect_to_array(pArr,
pIter);
    END LOOP;

    -- Return the result
    RETURN pArrResult;
END;
$$;
```

3. Difference Function

```
CREATE FUNCTION temporal_difference(p1 valid_period_domain, p2
valid_period_domain)
RETURNS valid_period_domain[]
LANGUAGE plpgsql
AS $$
BEGIN
    IF temporal_can_intersect(p1, p2) THEN
        IF temporal_overlaps(p1, p2) THEN
            RETURN ARRAY[(p1.start_timestamp, p2.start_timestamp -
1)::valid_period_domain];
        ELSIF temporal_overlaps_inverse(p1, p2) THEN
            RETURN ARRAY[(p2.end_timestamp + 1,
p1.end_timestamp)::valid_period_domain];
        ELSIF temporal_starts_inverse(p1, p2) THEN
            RETURN ARRAY[(p2.end_timestamp + 1,
p1.end_timestamp)::valid_period_domain];
        ELSIF temporal_during_inverse(p1, p2) THEN
            RETURN ARRAY[(p1.start_timestamp, p2.start_timestamp -
1)::valid_period_domain, (p2.end_timestamp + 1,
p1.end_timestamp)::valid_period_domain];
        ELSIF temporal_finishes_inverse(p1, p2) THEN
            RETURN ARRAY[(p1.start_timestamp, p2.start_timestamp -
1)::valid_period_domain];
        ELSE
            RETURN ARRAY[]::valid_period_domain[];
        END IF;
    ELSE
        RETURN ARRAY[p1];
    END IF;
END;
$$;
```

4. Coalesce Aggregation Function

```
CREATE AGGREGATE temporal_coalesce_single(valid_period_domain) (
    sfunc = temporal_merge,
    stype = valid_period_domain
);

CREATE AGGREGATE temporal_coalesce_multiple(valid_period_domain) (
    sfunc = temporal_merge_to_array,
    stype = valid_period_domain[],
    initcond = '{}'
);

CREATE AGGREGATE temporal_coalesce_multiple(valid_period_domain[]) (
    sfunc = temporal_merge_to_array,
    stype = valid_period_domain[],
    initcond = '{}'
);

CREATE AGGREGATE temporal_section_single(valid_period_domain) (
    sfunc = temporal_intersection,
    stype = valid_period_domain
);

CREATE AGGREGATE temporal_section_multiple(valid_period_domain) (
    sfunc = temporal_intersect_to_array,
    stype = valid_period_domain[]
);

CREATE AGGREGATE temporal_section_multiple(valid_period_domain[]) (
    sfunc = temporal_intersect_to_array,
    stype = valid_period_domain[]
);
```

5. Slice Function

```
CREATE FUNCTION temporal_slice(p valid_period_domain, input_timestamp
BIGINT)
RETURNS BOOLEAN
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN p.start_timestamp <= input_timestamp AND p.end_timestamp >=
input_timestamp;
END;
$$;
```

## Relational Algebra Queries

1. Temporal Projection

```
-- Temporal Projection: \pi_{name}^{B}(customer)
SELECT "customer"."name",
UNNEST(temporal_coalesce_multiple("customer"."subscription_period")) AS
subscription_period -- Merge all the periods
FROM "customer"
GROUP BY "customer"."name";
```

2. Temporal Selection

```
-- Temporal Selection: \sigma_{name = 'Anca'}^{B}(staff)
SELECT "staff"."id", "staff"."name",
UNNEST(temporal_coalesce_multiple("staff"."employment_period")) AS
employment_period -- Merge all the periods
FROM "staff"
WHERE "staff"."name" = 'Anca'
GROUP BY "staff"."id";
```

3. Temporal Join

```
-- Temporal Join: \pi_{name}^{B}(customer \bowtie^{B} staff)
SELECT "customer"."name" AS "customer_name", "staff"."name" AS "staff_name",
UNNEST(temporal_coalesce_multiple(temporal_intersection("customer"."subscrip
tion_period", "staff"."employment_period"))) AS period -- Merge all the
periods
FROM "customer"
JOIN "staff" ON temporal_can_intersect("customer"."subscription_period",
"staff"."employment_period")
GROUP BY "customer"."name", "staff"."name";
```

4. Temporal Union

```
-- Temporal Union: \pi_{name}^{B}(customer \cup^{B} staff)
WITH "union_data" AS (
    SELECT "customer"."name", "customer"."subscription_period" AS "period"
    FROM "customer"
    UNION
    SELECT "staff"."name", "staff"."employment_period" AS "period"
    FROM "staff"
)
SELECT "union_data"."name",
UNNEST(temporal_coalesce_multiple("union_data"."period")) AS period -- Merge
all the periods
FROM "union_data"
GROUP BY "union_data"."name";
```

5. Temporal Set Difference

```
-- Temporal Set Difference: \pi_{name}^{B}(staff) -^{B}
\pi_{name}^{B}(customer)
WITH "difference_data" AS (
    SELECT "staff"."name"
    FROM "staff"
    EXCEPT
    SELECT "customer"."name"
    FROM "customer"
)
SELECT "staff"."name",
UNNEST(temporal_coalesce_multiple("staff"."employment_period")) AS period--
Merge all the periods
FROM "staff"
JOIN "difference_data" ON "staff"."name" = "difference_data"."name"
GROUP BY "staff"."name"
UNION
SELECT "staff"."name",
UNNEST(temporal_section_multiple(temporal_difference("staff"."employment_per
iod", "customer"."subscription_period"))) AS period -- Find intersection
between differing periods
FROM "staff"
JOIN "customer" ON "staff"."name" = "customer"."name"
GROUP BY "staff"."name";
```

6. Temporal Time Slice

```
-- Temporal Time Slice: \tau_{3}^{B}(staff)
SELECT "staff"."name"
FROM "staff"
WHERE temporal_slice("staff"."employment_period", 3)
GROUP BY "staff"."name";
```

## Data Sample

### Insertion

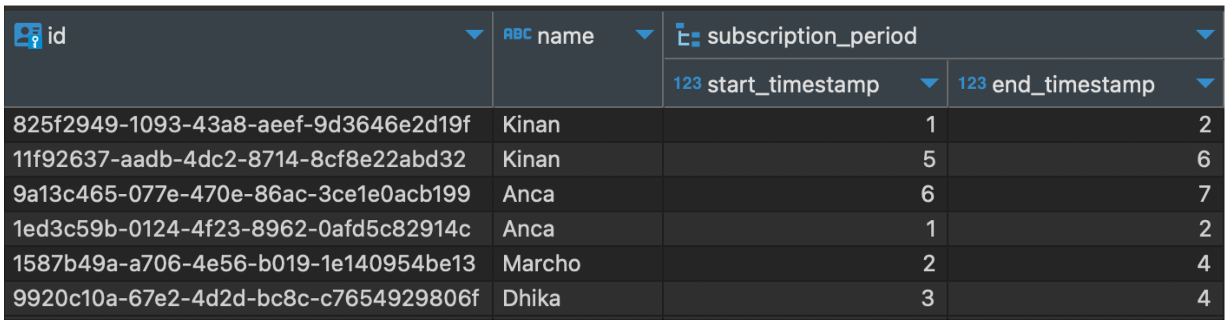
Query:

```
CALL customer_insertion('Kinan', (1, 2)::valid_period_domain);
CALL customer_insertion('Kinan', (5, 6)::valid_period_domain);
CALL customer_insertion('Anca', (6, 7)::valid_period_domain);
CALL customer_insertion('Anca', (1, 2)::valid_period_domain);
CALL customer_insertion('Marcho', (2, 4)::valid_period_domain);
CALL customer_insertion('Dhika', (3, 4)::valid_period_domain);

CALL staff_insertion('Anca', (5, 6)::valid_period_domain);
CALL staff_insertion('Anca', (9, 10)::valid_period_domain);
CALL staff_insertion('Anca', (7, 8)::valid_period_domain);
CALL staff_insertion('Dipa', (3, 4)::valid_period_domain);
CALL staff_insertion('Januar', (1, 2)::valid_period_domain);
CALL staff_insertion('Januar', (5, 6)::valid_period_domain);
```

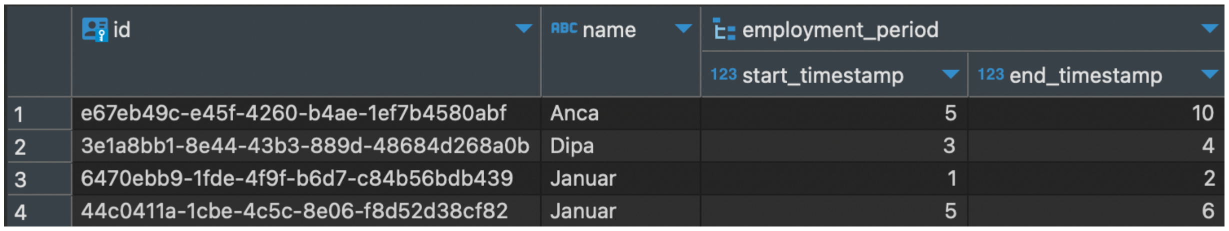
Result:

**Customer**



| id | name | subscription_period | |
|---|---|---|---|
| | | start_timestamp | end_timestamp |
| 825f2949-1093-43a8-aeef-9d3646e2d19f | Kinan | 1 | 2 |
| 11f92637-aadb-4dc2-8714-8cf8e22abd32 | Kinan | 5 | 6 |
| 9a13c465-077e-470e-86ac-3ce1e0acb199 | Anca | 6 | 7 |
| 1ed3c59b-0124-4f23-8962-0afd5c82914c | Anca | 1 | 2 |
| 1587b49a-a706-4e56-b019-1e140954be13 | Marcho | 2 | 4 |
| 9920c10a-67e2-4d2d-bc8c-c7654929806f | Dhika | 3 | 4 |

**Staff**



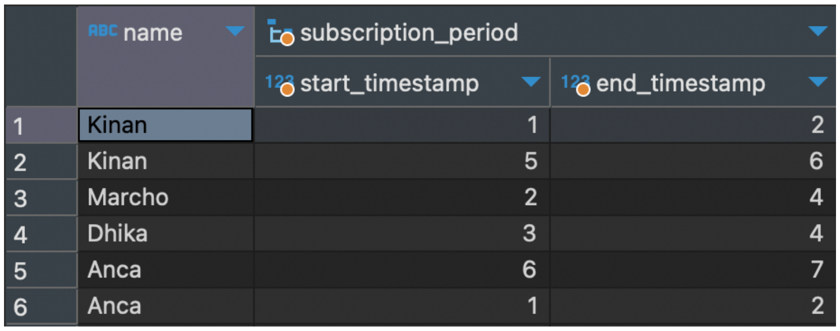
| | id | name | employment_period | |
|---|---|---|---|---|
| | | | start_timestamp | end_timestamp |
| 1 | e67eb49c-e45f-4260-b4ae-1ef7b4580abf | Anca | 5 | 10 |
| 2 | 3e1a8bb1-8e44-43b3-889d-48684d268a0b | Dipa | 3 | 4 |
| 3 | 6470ebb9-1fde-4f9f-b6d7-c84b56bdb439 | Januar | 1 | 2 |
| 4 | 44c0411a-1cbe-4c5c-8e06-f8d52d38cf82 | Januar | 5 | 6 |

## Projection

Query:

```
-- Temporal Projection: \pi_{name}^{B}(customer)
SELECT "customer"."name",
UNNEST(temporal_coalesce_multiple("customer"."subscription_period")) as
subscription_period -- Merge all the periods
FROM "customer"
GROUP BY "customer"."name";
```

Result:



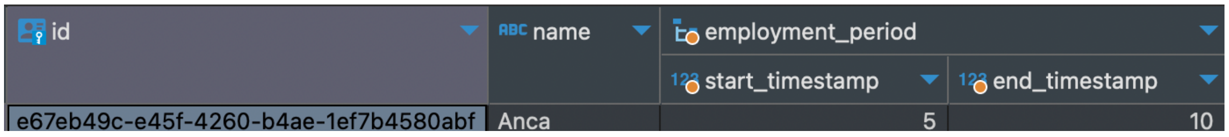
| | name | subscription_period | |
|---|---|---|---|
| | | start_timestamp | end_timestamp |
| 1 | Kinan | 1 | 2 |
| 2 | Kinan | 5 | 6 |
| 3 | Marcho | 2 | 4 |
| 4 | Dhika | 3 | 4 |
| 5 | Anca | 6 | 7 |
| 6 | Anca | 1 | 2 |

## Selection

Query:

```
-- Temporal Selection: \sigma_{name = 'Anca'}^{B}(staff)
SELECT "staff"."id", "staff"."name",
UNNEST(temporal_coalesce_multiple("staff"."employment_period")) AS
employment_period -- Merge all the periods
FROM "staff"
WHERE "staff"."name" = 'Anca'
GROUP BY "staff"."id";
```

Result:



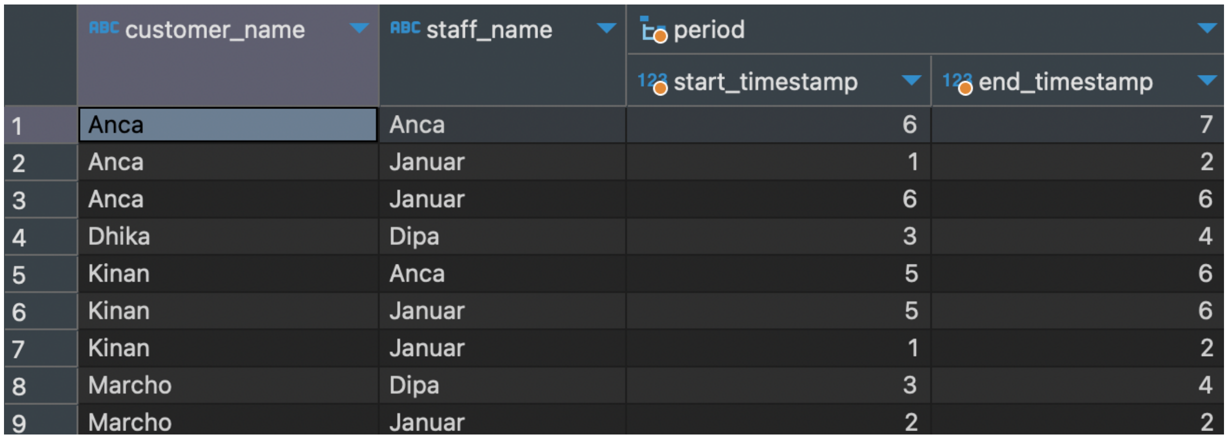
| id | name | employment_period | |
|---|---|---|---|
| | | start_timestamp | end_timestamp |
| e67eb49c-e45f-4260-b4ae-1ef7b4580abf | Anca | 5 | 10 |

## Join

Query:

```
-- Temporal Join: \pi_{name}^{B}(customer \bowtie^{B} staff)
SELECT "customer"."name" AS "customer_name", "staff"."name" AS "staff_name",
UNNEST(temporal_coalesce_multiple(temporal_intersection("customer"."subscrip
tion_period", "staff"."employment_period"))) AS period -- Merge all the
periods
FROM "customer"
JOIN "staff" ON temporal_can_intersect("customer"."subscription_period",
"staff"."employment_period")
GROUP BY "customer"."name", "staff"."name";
```

Result:



| | customer_name | staff_name | period | |
|---|---|---|---|---|
| | | | start_timestamp | end_timestamp |
| 1 | Anca | Anca | 6 | 7 |
| 2 | Anca | Januar | 1 | 2 |
| 3 | Anca | Januar | 6 | 6 |
| 4 | Dhika | Dipa | 3 | 4 |
| 5 | Kinan | Anca | 5 | 6 |
| 6 | Kinan | Januar | 5 | 6 |
| 7 | Kinan | Januar | 1 | 2 |
| 8 | Marcho | Dipa | 3 | 4 |
| 9 | Marcho | Januar | 2 | 2 |

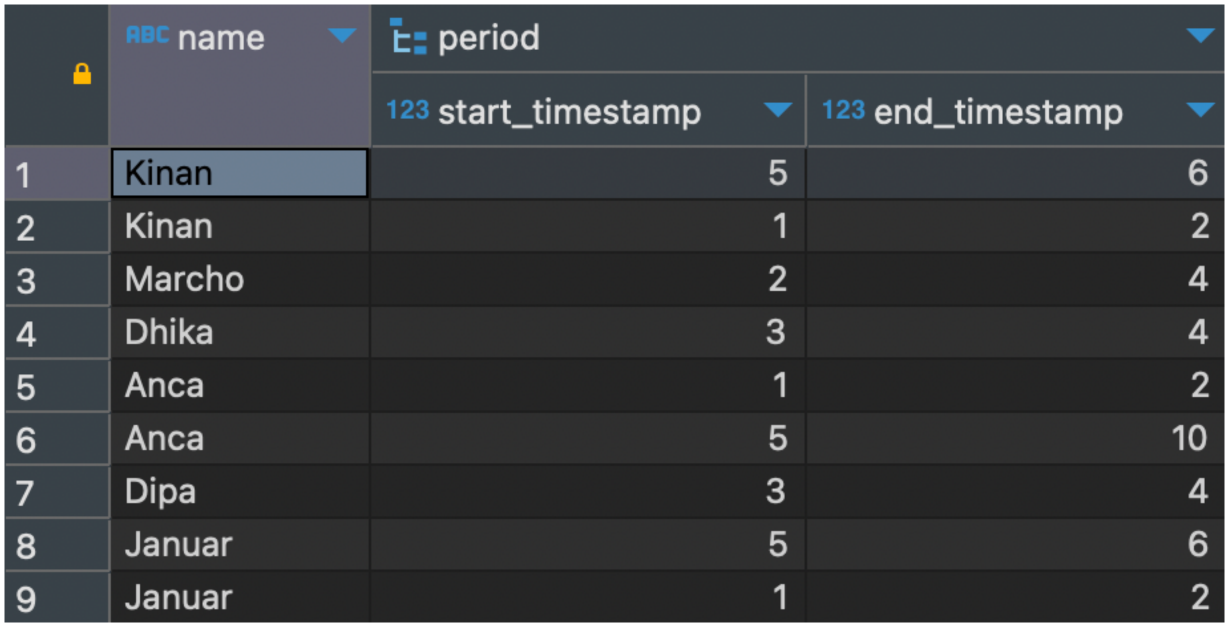
## Union

Query:

```
-- Temporal Union: \pi_{name}^{B}(customer \cup^{B} staff)
WITH "union_data" AS (
    SELECT "customer"."name", "customer"."subscription_period" AS "period"
    FROM "customer"
    UNION
    SELECT "staff"."name", "staff"."employment_period" AS "period"
    FROM "staff"
)
SELECT "union_data"."name",
UNNEST(temporal_coalesce_multiple("union_data"."period")) AS period -- Merge
all the periods
FROM "union_data"
```

```
GROUP BY "union_data"."name";
```

Result:



| | name | period | |
|---|---|---|---|
| | | start_timestamp | end_timestamp |
| 1 | Kinan | 5 | 6 |
| 2 | Kinan | 1 | 2 |
| 3 | Marcho | 2 | 4 |
| 4 | Dhika | 3 | 4 |
| 5 | Anca | 1 | 2 |
| 6 | Anca | 5 | 10 |
| 7 | Dipa | 3 | 4 |
| 8 | Januar | 5 | 6 |
| 9 | Januar | 1 | 2 |

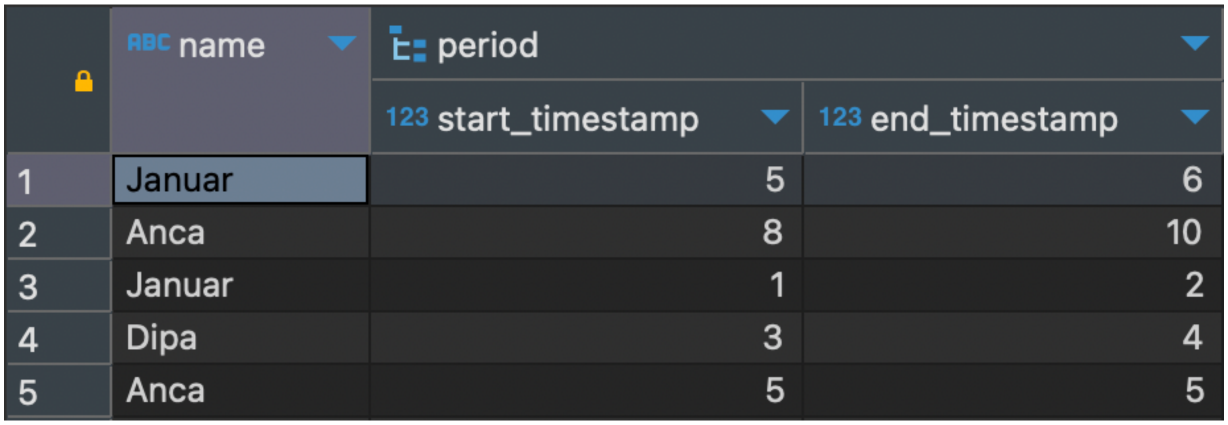
## Set Difference

Query:

```
-- Temporal Set Difference: \pi_{name}^{B}(staff) -^{B}
\pi_{name}^{B}(customer)
WITH "difference_data" AS (
    SELECT "staff"."name"
    FROM "staff"
    EXCEPT
    SELECT "customer"."name"
    FROM "customer"
)
SELECT "staff"."name",
UNNEST(temporal_coalesce_multiple("staff"."employment_period")) AS period--
Merge all the periods
FROM "staff"
JOIN "difference_data" ON "staff"."name" = "difference_data"."name"
GROUP BY "staff"."name"
UNION
SELECT "staff"."name",
UNNEST(temporal_section_multiple(temporal_difference("staff"."employment_per
iod", "customer"."subscription_period"))) AS period -- Find intersection
```

```
between differing periods
FROM "staff"
JOIN "customer" ON "staff"."name" = "customer"."name"
GROUP BY "staff"."name";
```

Result:



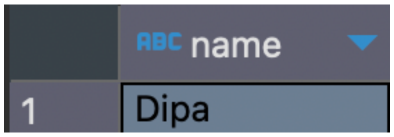
|  | name | period |  |
|---|---|---|---|
|  |  | start_timestamp | end_timestamp |
| 1 | Januar | 5 | 6 |
| 2 | Anca | 8 | 10 |
| 3 | Januar | 1 | 2 |
| 4 | Dipa | 3 | 4 |
| 5 | Anca | 5 | 5 |

## Time Slice

Query:

```
SELECT "staff"."name"
FROM "staff"
WHERE temporal_slice("staff"."employment_period", 3)
GROUP BY "staff"."name";
```

Result:



|  | name |
|---|---|
| 1 | Dipa |

# Analisis

## Hasil

1. Implementasi Temporal Database dengan menggunakan dua skema yaitu **`employee`** dan **`staff`** berhasil dibuat dengan memanfaatkan *type* buatan yaitu **`valid_period_domain`**.
2. Allen's 13 Relation berhasil diimplementasikan dengan membuat *user-defined function* dengan memanfaatkan type **`valid_period_domain`**.
3. *Aggregate coalesce* untuk digunakan dalam *query* juga berhasil diimplementasikan dengan membuat *aggregate* dari *user-defined function* dengan memanfaatkan Allen's 13 Relation dan fungsi-fungsi lainnya.
4. *Relational Algebra Queries* berhasil diimplementasikan sesuai dengan Snodgrass Valid Timestamp (1D).

## Kesimpulan

Dengan skema yang telah dibuat, yaitu tabel pelanggan dan staf, serta penggunaan tipe data khusus **`valid_period_domain`**, perusahaan memiliki alat yang efisien untuk melacak perubahan data sepanjang waktu. Penerapan Allen's 13 Relation dan fungsi-fungsi kustom lainnya memperkaya kemampuan query, memungkinkan analisis data yang lebih mendalam dan presisi. Secara keseluruhan, pendekatan ini memastikan bahwa perusahaan memiliki fondasi data yang kuat untuk mendukung pengambilan keputusan bisnis yang tepat dan responsif terhadap tren yang berubah.

## Pembagian Tugas

| **NIM** | **Nama** | **Tugas** |
|---|---|---|
| 13520024 | Hilya Fadhilah Imania | Kode dan Laporan |
| 13520044 | Adiyansa Prasetya Wicaksana | Kode dan Laporan |
| 13520065 | Rayhan Kinan Muhannad | Kode dan Laporan |