

## BINUS University

<b>Academic Career:</b> <i>Undergraduate / <del>Master</del> / <del>Doctoral</del> *)</i>		<b>Class Program:</b> <i>International/Regular/Smart Program/Global Class*)</i>	
<input type="checkbox"/> Mid Exam <input checked="" type="checkbox"/> Final Exam <input type="checkbox"/> Short Term Exam <input type="checkbox"/> Others Exam : _____		<b>Term : Odd/Even/Short *)</b>	
<input checked="" type="checkbox"/> Kemanggisan <input checked="" type="checkbox"/> Alam Sutera <input type="checkbox"/> Bekasi <input type="checkbox"/> Senayan <input type="checkbox"/> Bandung <input type="checkbox"/> Malang		<b>Academic Year :</b> <b>2021 / 2022</b>	
Faculty / Dept. : School of Information Systems / Information Systems		Deadline	Day / Date : Rabu, 16 Februari 2022 Time : 17.00
Code - Course : ISYS6169 – Database Systems		Class : All	
Lecturer : Team		Exam Type : Online	
*) <i>Strikethrough the unnecessary items</i>			
<b>The penalty for CHEATING is DROP OUT!!!</b>			

### FINAL EXAM Database Systems

**Nama : Rio Pramana**

**NIM : 2440016804**

**Kelas : LC01**

**Jawaban:**

1. Diasumsikan list of participants menampilkan semua data participant yang muncul.

Query:

```

1. -- 1
2. -- Show a list of participants who are doing the "SQL Server"
   course with the "Advanced" level.
3. SELECT
4.      *
5. FROM
6.      Participant
7. WHERE
8.      Participant_Id IN (
9.          SELECT
10.             p.Participant_Id
11.          FROM

```

```

12.          Participant p JOIN
13.          OnGoingCourse ogc ON
14.          p.Participant_Id = ogc.Participant_Id
15.          JOIN
16.          Course c ON
17.          c.Course_Id = ogc.Course_Id
18.          WHERE
19.          Course_Name = 'SQL Server'
20.          AND
21.          [Level] = 'Advanced'

```

Screenshot query:

```

-- 1
-- Show a list of participants who are doing the "SQL Server" course with the "Advanced" level.
SELECT
    *
FROM
    Participant
WHERE
    Participant_Id IN (
        SELECT
            p.Participant_Id
        FROM
            Participant p JOIN
            OnGoingCourse ogc ON
            p.Participant_Id = ogc.Participant_Id JOIN
            Course c ON
            c.Course_Id = ogc.Course_Id
        WHERE
            Course_Name = 'SQL Server'
            AND
            [Level] = 'Advanced'
    )

```

2. Diasumsikan list of trainers menampilkan seluruh data trainer yang muncul.

Query:

```

1. -- 2
2. -- Using subquery, show a list of trainers who are responsible
   for the "Project Management dan Leadership" course.
3. SELECT
4.     *
5. FROM
6.     Teacher

```

```

7. WHERE
8.     Teacher_Id IN (
9.         SELECT
10.            t.Teacher_Id
11.        FROM
12.            Teacher t JOIN
13.            Course c ON
14.            t.Teacher_Id = c.Teacher
15.        WHERE
16.            Course_Name = 'Project Management dan
17.            Leadership'
17.    )

```

Screenshot query:

```

-- 2
-- Using subquery, show a list of trainers who are responsible for the "Project Management dan Leadership" course.
SELECT
    *
FROM
    Teacher
WHERE
    Teacher_Id IN (
        SELECT
            t.Teacher_Id
        FROM
            Teacher t JOIN
            Course c ON
            t.Teacher_Id = c.Teacher
        WHERE
            Course_Name = 'Project Management dan Leadership'
    )

```

3. Diasumsikan yang dimaksud oleh active participants and courses **during January 2022** adalah OnGoingCourse yang statusnya masih active **pada saat January 2022** sehingga Start\_date dan End\_date nya tidak harus = January 2022. **Contohnya**, course A start datenya Desember 2021 dan end datenya Februari 2022, maka pada January 2022 course tersebut masih active. Saya mengasumsikan courses seperti itu juga termasuk ke dalam list active participants and courses during January 2022.

Query:

```

1. -- 3
2. -- Create a view to show active participants and courses during
   January 2022.
3. CREATE VIEW [Active Participants and Courses During January
   2022]

```

```

4. AS
5. SELECT
6.      c.*, p.*
7. FROM
8.      OnGoingCourse ogc JOIN
9.      Course c ON
10.         ogc.Course_Id = c.Course_Id JOIN
11.         Participant p ON
12.         p.Participant_Id = ogc.Participant_Id
13. WHERE
14.     [Start_date] <= '2022-01-31'
15. AND
16.     [End_date] >= '2022-01-01'
17. AND
18.     [Status] = 'Active'

```

Screenshot query:

```

-- 3
-- Create a view to show active participants and courses during January 2022.
CREATE VIEW [Active Participants and Courses During January 2022]
AS
SELECT
    c.*, p.*
FROM
    OnGoingCourse ogc JOIN
    Course c ON
    ogc.Course_Id = c.Course_Id JOIN
    Participant p ON
    p.Participant_Id = ogc.Participant_Id
WHERE
    [Start_date] <= '2022-01-31'
    AND
    [End_date] >= '2022-01-01'
    AND
    [Status] = 'Active'

```

**Namun**, jika yang dimaksud active during January 2022 adalah hanya courses yang Start\_date dan End\_date nya berada dalam January 2022, maka yang perlu diubah hanyalah condition pada WHERE, yaitu seperti berikut.

```

WHERE
    [Start_date] >= '2022-01-01'
    AND
    [End_date] <= '2022-01-31'
    AND
    [Status] = 'Active'
-

```

#### 4. Query:

```

1. -- 4
2. -- Create a trigger to update the "participant" table, with a
   success message: "Data has been recorded".
3. CREATE TRIGGER TR_UPDATE_tblParticipant
4. ON Participant
5. FOR UPDATE
6. AS
7. SET NOCOUNT ON
8. BEGIN
9.     DECLARE @success_msg VARCHAR(30) = 'Data has been recorded'
10.
11.     IF UPDATE (Participant_Id)
12.         PRINT @success_msg
13.     ELSE IF UPDATE (PFirst_Name)
14.         PRINT @success_msg
15.     ELSE IF UPDATE (PLast_Name)
16.         PRINT @success_msg
17.     ELSE IF UPDATE (Phone_Number)
18.         PRINT @success_msg
19.     ELSE IF UPDATE (Participant_email)
20.         PRINT @success_msg
21.     ELSE IF UPDATE (Client_Id)
22.         PRINT @success_msg
23. END

```

Screenshot query:

```
-- 4
-- Create a trigger to update the "participant" table, with a success message: "Data has been recorded".
CREATE TRIGGER TR_UPDATE_tblParticipant
ON Participant
FOR UPDATE
AS
SET NOCOUNT ON
BEGIN
    DECLARE @success_msg VARCHAR(30) = 'Data has been recorded'


    IF UPDATE(Participant_Id)
        PRINT @success_msg
    ELSE IF UPDATE(PFirst_Name)
        PRINT @success_msg
    ELSE IF UPDATE(PLast_Name)
        PRINT @success_msg
    ELSE IF UPDATE(Phone_Number)
        PRINT @success_msg
    ELSE IF UPDATE(Participant_email)
        PRINT @success_msg
    ELSE IF UPDATE(Client_Id)
        PRINT @success_msg
END
```

5. Terdapat beberapa masalah pada database yang disebabkan oleh berbagai concurrency issues seperti lost update, uncommitted dependency, dan inconsistent analysis.

*Lost update* terjadi ketika ada lebih dari satu *concurrent transaction* yang melakukan *update* pada data yang sama sehingga salah satu *update* yang dilakukan hilang karena di-*overwrite* oleh *transaction* lainnya. Berikut adalah contoh *lost update*.

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T2	Read PROD_QOH	35
3	T1	PROD_QOH = 35 + 100	
4	T2	PROD_QOH = 35 - 30	
5	T1	Write PROD_QOH ( <b>Lost update</b> )	135
6	T2	Write PROD_QOH	5

*Uncommitted dependency* terjadi ketika misalnya terdapat dua *concurrent transaction* T1 dan T2. Namun, T1 ternyata dilakukan *rollback*, tetapi T2 sudah terlanjur mengakses data yang ingin di-*rollback* oleh T1 sehingga menyebabkan kesalahan pada data. Berikut adalah contoh *uncommitted dependency*.

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	z135
4	T2	Read PROD_QOH (Read uncommitted data) 	135
5	T2	PROD_QOH = 135 - 30	
6	T1	**** ROLLBACK ****	35
7	T2	Write PROD_QOH	105

*Inconsistent analysis* terjadi ketika sebuah *transaction* mengakses data sebelum dan sesudah *transaction* lain mengubah data tersebut. Berikut adalah contoh *inconsistent analysis*.

TRANSACTION T1		TRANSACTION T2	
SELECT	SUM(PROD_QOH)	UPDATE	PRODUCT
FROM	PRODUCT	SET	PROD_QOH = PROD_QOH + 10
		WHERE	PROD_CODE = '1546-QQ2'
		UPDATE	PRODUCT
		SET	PROD_QOH = PROD_QOH - 10
		WHERE	PROD_CODE = '1558-QW1'
		COMMIT;	

TIME	TRANSACTION	ACTION	VALUE	TOTAL
1	T1	Read PROD_QOH for PROD_CODE = '11QER/31'	8	8
2	T1	Read PROD_QOH for PROD_CODE = '13-Q2/P2'	32	40
3	T2	Read PROD_QOH for PROD_CODE = '1546-QQ2'	15	
4	T2	PROD_QOH = 15 + 10		
5	T2	Write PROD_QOH for PROD_CODE = '1546-QQ2'	25	
6	T1	Read PROD_QOH for PROD_CODE = '1546-QQ2'	25	(After) 65
7	T1	Read PROD_QOH for PROD_CODE = '1558-QW1'	23	(Before) 88
8	T2	Read PROD_QOH for PROD_CODE = '1558-QW1'	23	
9	T2	PROD_QOH = 23 - 10		
10	T2	Write PROD_QOH for PROD_CODE = '1558-QW1'	13	
11	T2	***** COMMIT *****		
12	T1	Read PROD_QOH for PROD_CODE = '2232-QTY'	8	96
13	T1	Read PROD_QOH for PROD_CODE = '2232-QWE'	6	102

Yang terjadi pada contoh diatas adalah T2 ingin mengubah nilai PROD\_QOH dari 2 buah PROD\_CODE. Namun, ketika T1 ingin menghitung sum PROD\_QOH, T1 mengambil nilai PROD\_QOH untuk kedua PROD\_CODE tersebut sebelum dan sesudah perubahan dilakukan sehingga SUM yang dihitung juga salah nilainya.

Pada kasus yang diberikan oleh soal, *concurrency issue* yang terjadi adalah ***inconsistent analysis***. *Inconsistent analysis* ini terjadi pada t3-t4 dan t5-t6.

	T1	T2	T3	Balx	Baly	Sum
t3	Begin_transaction	Read(balx)	Read(balx)	150	50	0
t4	Read(baly)	Sum=Sum+balx	Balx=Balx+50	150	50	150

Pada t3-t4, dapat dilihat bahwa T2 ingin menghitung nilai *sum* dengan menggunakan nilai *Balx*. Namun, di saat yang sama, T3 ingin menambahkan 50 pada nilai *Balx*. Yang terjadi disini adalah T2 menggunakan nilai *Balx* sebelum nilai *Balx* di-update oleh T3 untuk menghitung nilai *sum* yang baru. Maka, nilai *sum* yang dihitung T2 adalah  $sum = 0 + 150$ . Padahal, seharusnya T2 menggunakan nilai *Balx* setelah di-update oleh T3, yaitu  $Balx = 150 + 50 = 200$ .

	T1	T2	T3	Balx	Baly	Sum
t5	Baly=Baly-20	Read(baly)	Write Balx	200	50	150

Akibatnya, pada t5 bisa kita lihat bahwa *sum* sekarang memiliki nilai yang salah, yaitu 150 (seharusnya 200). Hal tersebut terjadi karena T3 baru bisa menuliskan nilai baru *Balx* pada t5, sesudah T2 membaca nilai *Balx* yang lama.

	T1	T2	T3	Balx	Baly	Sum
t5	Baly=Baly-20	Read(baly)	Write Balx	200	50	150
t6	Write(baly)	Sum=sum+baly	Commit	200	30	200

Kemudian, pada t5-t6 terjadi lagi *inconsistent analysis*. Yang terjadi disini sangat mirip dengan yang terjadi pada t3-t4. Pada t5, T2 membaca nilai *Baly* untuk digunakan dalam menghitung nilai *sum* yang baru. Namun, disaat yang bersamaan T1 juga ingin mengubah nilai *Baly*. Yang terjadi adalah T2 akan menggunakan nilai lama *Baly* (50) dalam perhitungan *sum* sebelum T1 dapat menuliskan nilai baru *Baly* setelah diubah, yaitu 30. Maka, pada t6, terjadi kesalahan nilai *sum*. Seharusnya, nilai *sum* adalah 180 (Asumsikan nilai *sum* sebelumnya, yaitu 150, adalah benar) yang didapat dari  $150+30$ .



*Inconsistency analysis* yang terjadi pada kasus ini dapat kita hindari dengan menulis ulang semua *transaction* menggunakan *2PL preventive technique*. Berikut adalah hasilnya.

	T1	T2	T3	Balx	Baly	Sum
t1		Begin_transaction		150	50	-
t2		Sum=0	Begin_transaction	150	50	0
t3	Begin_transaction		Write_lock(Balx)	150	50	0
t4	Write_lock(Baly)	Read_lock(Balx)	Read(Balx)	150	50	0
t5	Read(Baly)	WAIT	Balx = Balx + 50	150	50	0
t6	Baly = Baly – 20	WAIT	Write(Balx)	200	50	0
t7	Write(Baly)	WAIT	Commit/unlock(Balx)	200	30	0
t8	Commit/unlock(Baly)	Read(Balx)		200	30	0
t9		Sum = Sum + Balx		200	30	200
t10		Read_lock(Baly)		200	30	200
t11		Sum = Sum + Baly		200	30	230
t12		Commit/unlock(Balx, Baly)		200	30	230

Final value of balx = 200

Final value of baly = 30

Final value of sum = 230

6. Berikut adalah penjelasan mengenai teknik *striping* dan *mirroring* pada RAID.

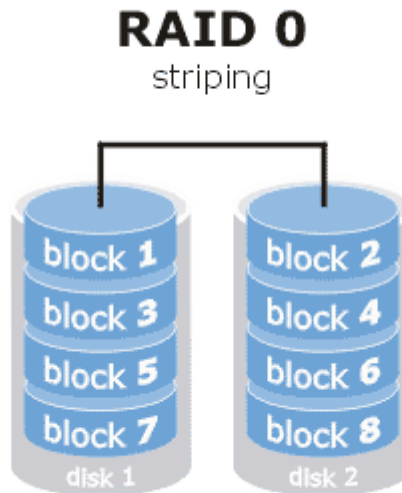
- **Striping**

*Striping* merupakan proses membagi sebuah *body of data* menjadi beberapa *block* dan menyebarkan blok-blok data tersebut pada lebih dari satu *storage devices*, seperti *harddisk* atau SSD dalam grup RAID.

Karena metode *striping* menyebarkan data-data pada lebih dari satu *drive*, konten pada sebuah file juga dapat diakses oleh lebih dari satu *disk* sehingga operasi I/O juga dapat dilakukan dengan lebih cepat.

Namun, metode *striping* ini tidak memiliki *parity* sehingga tidak ada *redundancy* atau *fault tolerance*. Oleh karena itu, jika ada sebuah *drive* yang gagal, semua data pada *drive* tersebut akan hilang.

Berikut adalah ilustrasi metode *striping*.



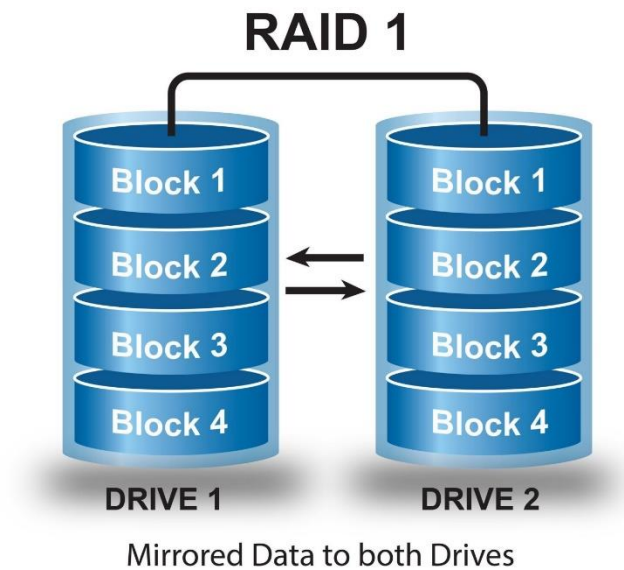
- **Mirroring**

Metode *mirroring* merupakan metode yang dibuat lebih baik dari *striping*. Metode ini tidak menggunakan *striping* sama sekali. Cara metode ini bekerja adalah dengan mencerminkan (*mirrors*) *disks* dalam *sets of two* sehingga untuk setiap *file* terdapat dua salinan.

Jika ada kegagalan yang terjadi, *controller* akan menggunakan *disk* kedua untuk menyediakan datanya sehingga ketersediaan data bisa terjamin. Saat *disk* kedua menggantikan *disk* pertama yang gagal, *controller* juga menyalin lagi data dari *disk* kedua yang tidak gagal tersebut.

Pada metode ini, semua data direkam secara bersamaan pada kedua *disk*. Metode ini memberikan ketersediaan data yang lebih tinggi dari *striping* tetapi memerlukan *disk space* 2 kali lipat sehingga dapat meningkatkan *cost*.

Berikut adalah ilustrasi metode *mirroring*.



Jika harus memilih antara *striping* (RAID 0) dan *mirroring* (RAID 1), menurut saya tipe RAID yang paling cocok digunakan untuk database GoTraining adalah ***mirroring* (RAID 1)**. Hal ini dikarenakan *mirroring* dapat lebih menjamin ketersediaan data dibanding *striping*. Database GoTraining menyimpan seluruh data yang berkaitan dengan *courses*, *trainers*, dan *participants* sehingga GoTraining tidak bisa mengambil risiko kehilangan seluruh data tersebut dengan menggunakan metode *striping*. Dengan metode *mirroring*, GoTraining mempunyai *backup* data jika terjadi sesuatu yang tidak diinginkan, terutama karena GoTraining mempunyai cabang di berbagai kota dengan satu pusat database saja sehingga ketersediaan data sangat penting.

Namun, jika memungkinkan, akan lebih baik jika GoTraining dapat menggunakan tipe RAID yang menggabungkan *striping* dan *mirroring* seperti RAID 0+1 atau RAID 1+0. Dengan RAID 0+1 atau RAID 1+0, kelebihan utama dari *striping* (kecepatannya) dan *mirroring* (*safety*-nya) digabung sehingga metodenya lebih baik dan berguna untuk GoTraining karena *safety*-nya sangat penting dan datanya bisa sangat banyak sehingga kecepatan juga menjadi hal yang dapat menguntungkan GoTraining.

7. Terdapat beberapa tipe fragmentation database, diantaranya adalah *horizontal fragmentation*, *vertical fragmentation*, dan *mixed fragmentation*.

*Horizontal fragmentation* dilakukan dengan membagi table secara horizontal dan mengelompokkannya berdasarkan *value* dari satu atau lebih kolom. Contohnya pada tabel

di soal, *horizontal fragmentation* dapat dilakukan dengan mengelompokkan *tuples* berdasarkan *Client\_Id* nya.

*Vertical fragmentation* dilakukan dengan membagi table secara vertikal (satu atau lebih kolom dikelompokkan). Dalam setiap fragmen, harus terdapat minimal sebuah kolom yang menjadi *primary key* pada fragmen table tersebut.

*Mixed fragmentation* dilakukan dengan menggabungkan kedua teknik *fragmentation* diatas, yaitu horizontal dan vertikal. *Mixed fragmentation* dapat dilakukan dengan dua cara. Yang pertama, buat terlebih dahulu *horizontal fragmentation* nya, kemudian buat *vertical fragmentation* dari hasil *horizontal fragmentation* tersebut. Yang kedua, buat terlebih dahulu *vertical fragmentation* nya, kemudian buat *horizontal fragmentation* dari hasil *vertical fragmentation* tersebut.

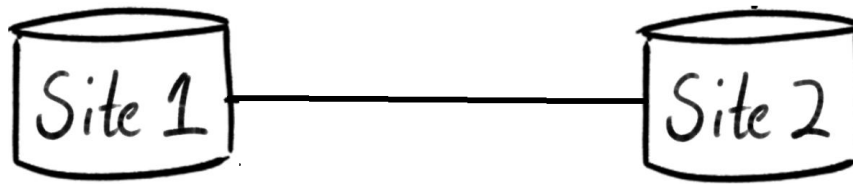
Menurut saya, tipe *fragmentation* yang paling cocok untuk table Participant pada database GoTraining adalah ***mixed fragmentation***. Di kasus-kasus dunia nyata, *mixed fragmentation* lebih sering digunakan karena implementasi salah satu tipe fragmentasi saja tidak cukup. Hal ini disebabkan banyaknya aplikasi atau query yang memerlukan fragmentasi lebih kompleks dari horizontal saja atau vertikal saja.

Selain itu, pada table Participant, kolom-kolom yang ada tidak perlu digabungkan menjadi 1 tabel penuh. Misalnya, untuk mengetahui PhoneNumber dari salah satu partisipan, kita tidak memerlukan data Client\_Id. Maka, kita bisa melakukan *vertical fragmentation* untuk mendapatkan fragmentasi-fragmentasi tabel dimana kita dapat mengakses data/kolom yang diperlukan dengan lebih efisien serta lebih aman (Misalnya jika memerlukan data PhoneNumber, database tidak perlu mencari dan menampilkan Client\_Id).

Setelah *vertical fragmentation* dilakukan, kita melakukan *horizontal fragmentation* dari hasil *vertical fragmentation* tersebut. Fragmentasi horizontal ini dapat dilakukan dengan mengelompokkan semua tuple berdasarkan *value* dari Client\_Id nya sehingga jika kita ingin mengakses salah satu Participant, kita tidak perlu mengakses keseluruhan table. Hal ini akan sangat berguna jika table Participant sudah memiliki data/tuple yang jumlahnya sangat banyak. Misalnya terdapat 100 Client dan setiap Client memiliki 1000 Participant yang berbeda, tanpa *horizontal fragmentation*, kita harus mengakses seluruh tabel ( $100 * 1000$  tuple) untuk mengakses salah satu tuple saja.

Berikut adalah hasil dari *mixed fragmentation* nya.

- Buat terlebih dahulu *vertical fragmentation* nya



Client Id	Participant Id
C12	PP152
C12	PP514
C11	PP432
C10	PP651

Participant Id	Pfirst Name	Plast Name	PhoneNumber	Participant_email
PP152	David	Ford	++6286161717	David.Ford@mail.com
PP514	Matthew	Lee	+62856171611	Matthew.Lee@mail.com
PP432	Maria	White	+62815625416	Maria.White@mail.com
PP651	Susan	Susan	+62816161719	Susan.susan@mail.com

Hasil fragmentasi ada 2, yaitu fragmentasi Client\_Id dengan Participant\_Id (Untuk mengetahui setiap client memiliki participant siapa saja, dan mengetahui setiap participant berada di client mana) dan fragmentasi data-data Participant (Untuk mengetahui data personal Participant seperti nama dan nomor HP nya, tidak perlu menggunakan Client\_Id).

- Kemudian buat *horizontal fragmentation* nya

#### **SITE 1.1**

Client Id	Participant Id
C12	PP152
C12	PP514

#### **SITE 1.2**

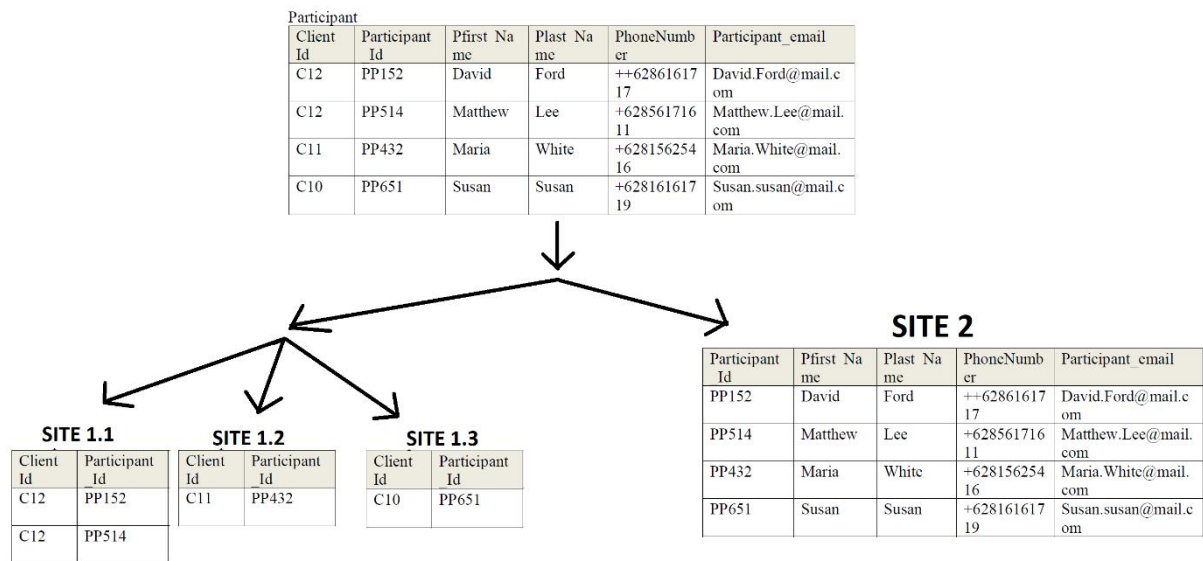
Client Id	Participant Id
C11	PP432

#### **SITE 1.3**

Client Id	Participant Id
C10	PP651

Dari fragmentasi Site 2 pada *vertical fragmentation*, dihasilkan 3 fragmentasi baru berdasarkan Client\_Id. (Site 1.1 Client\_Id = C12, Site 1.2 Client\_Id = C11, Site 1.3 Client\_Id = C10)

Hasil dari *mixed fragmentation* setelah menggabungkan *vertical* dan *horizontal fragmentation*:



8. Link video (Google Drive):  
[https://drive.google.com/file/d/1rnz7sEulNtvRwTzc\\_0tO3A\\_vgX7H1BNB/view?usp=sharing](https://drive.google.com/file/d/1rnz7sEulNtvRwTzc_0tO3A_vgX7H1BNB/view?usp=sharing)