Tutorial 05 : Optimization

Exercice 01:

1- Queries and optimization:

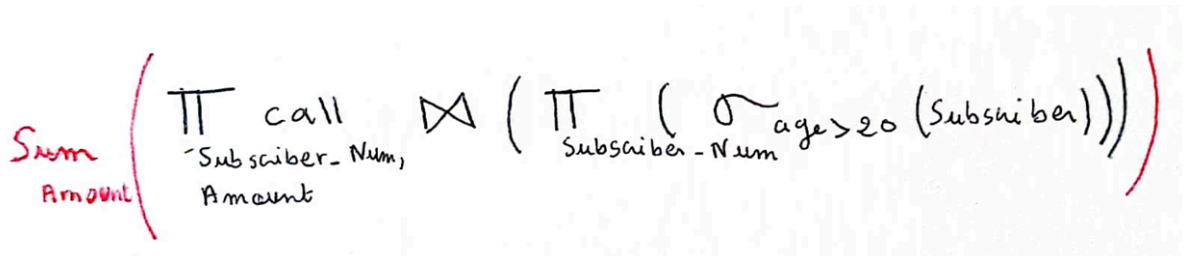
a- The logical model of the data warehouse (ROLAP):

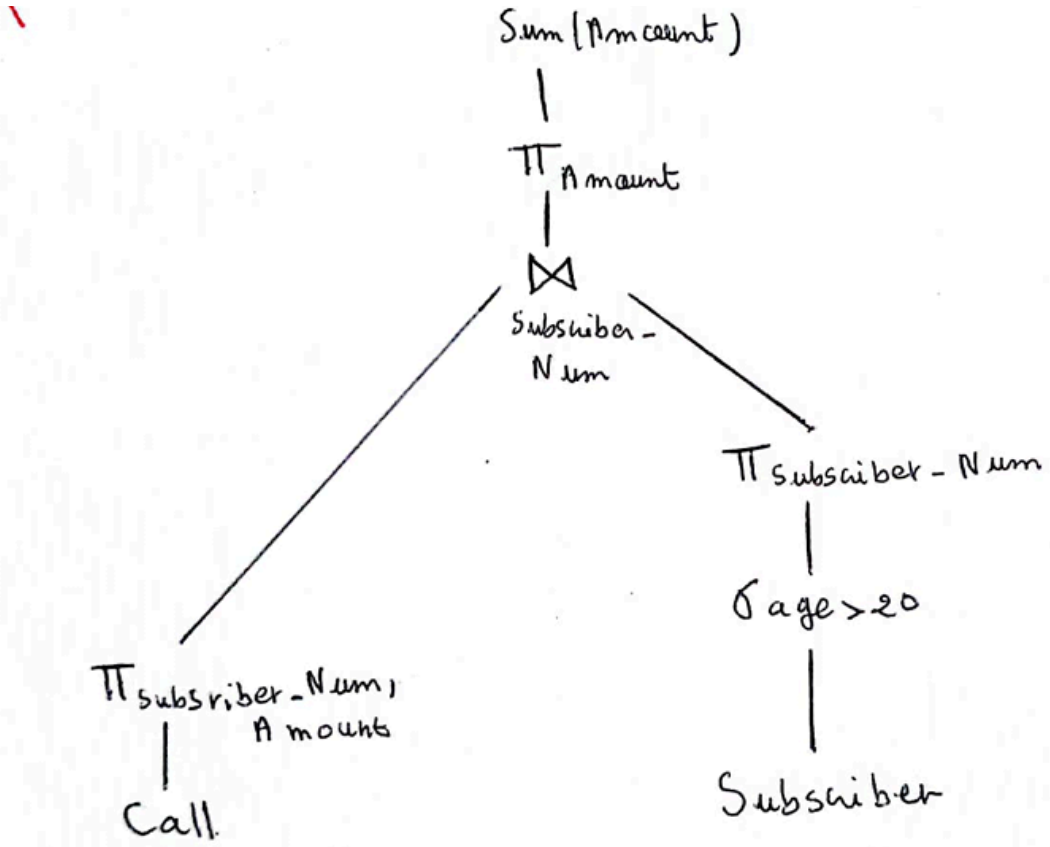
Subscriber (Subscriber_Num, Age, Address, Profession, Category)
Relay (IdRelay, Coordinate-GPS, City, Department)
Time (IdTime, Hour, Day, Month, Year)
Promotion (IdPromo, Name, Type, EndDate)
Call (Subscriber_Num*, IdRelay*, IdTime*, IdPromo*, Duration, Amount)

b- Considering the following queries:

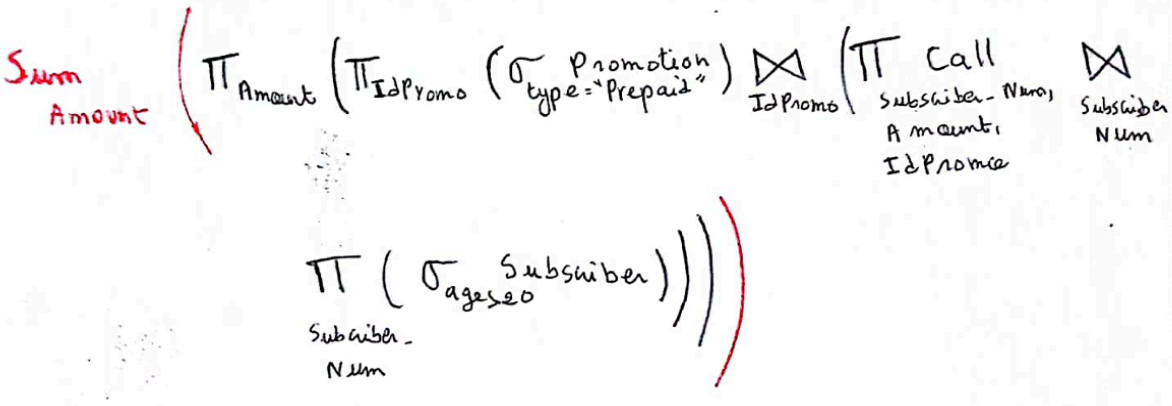
1- The algebraic tree of the query:

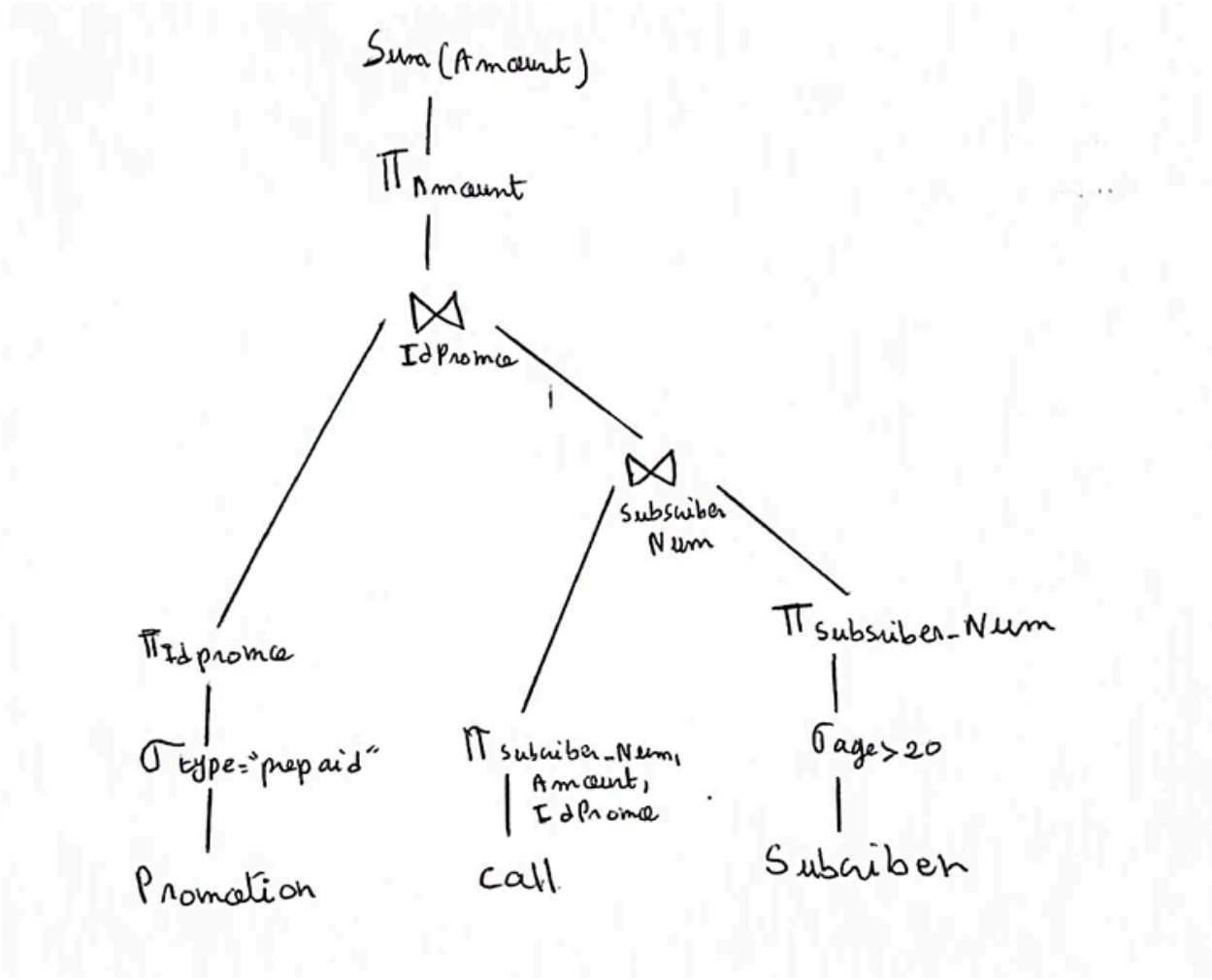
Q1 : The total amount of calls made by subscribers aged over 20 years.



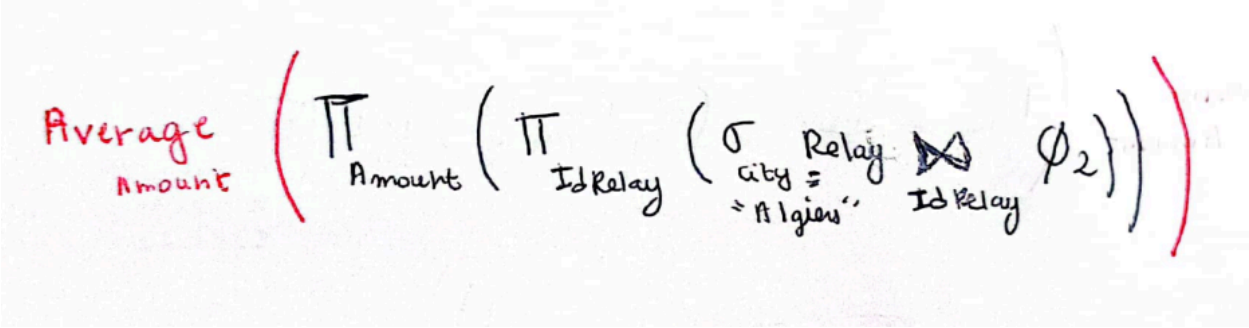


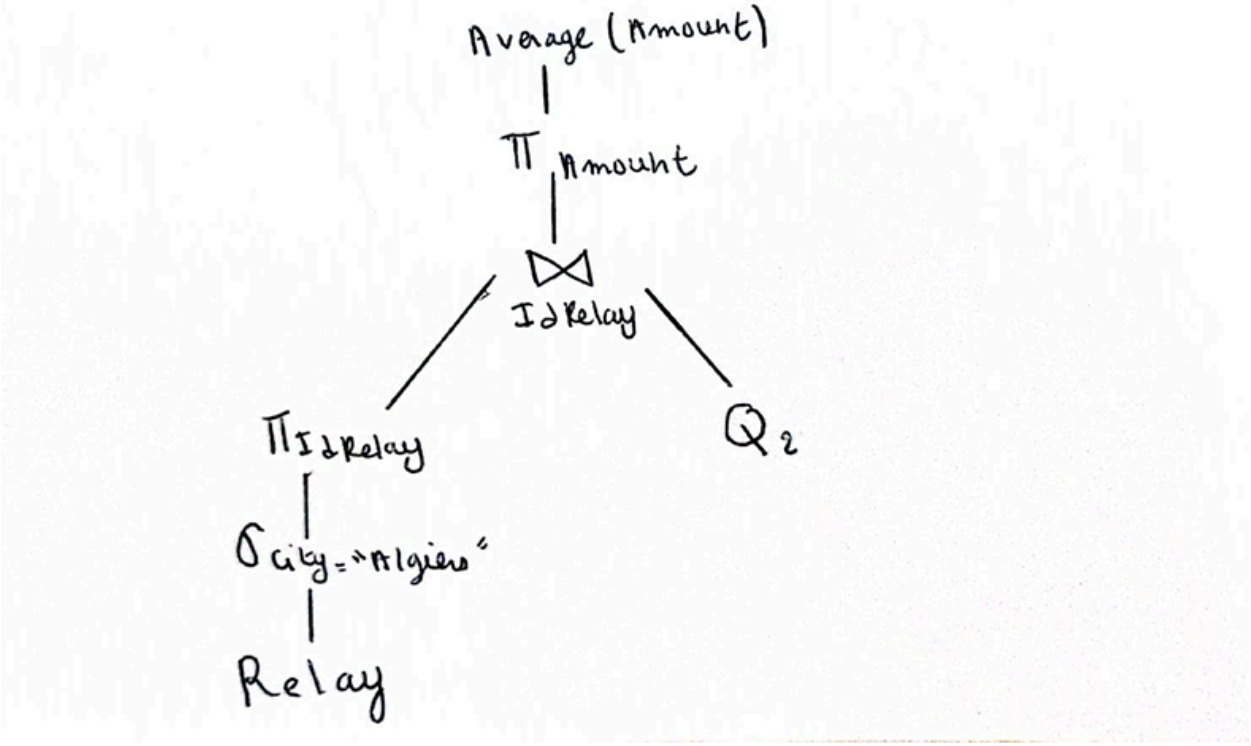
Q2 : The total amount of calls made by subscribers aged over 20 for type promotions of type "Prepaid".



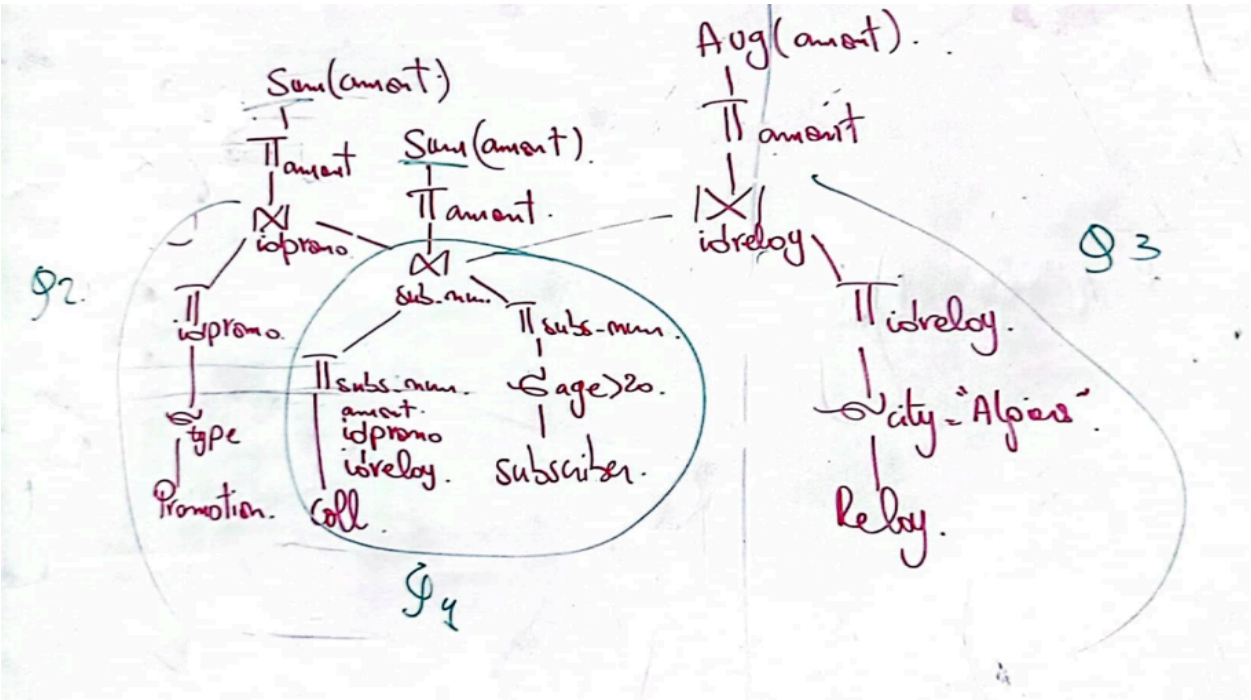


Q3 : The average amount of calls made by subscribers aged over 20 passing through relays located in the city of Algiers.



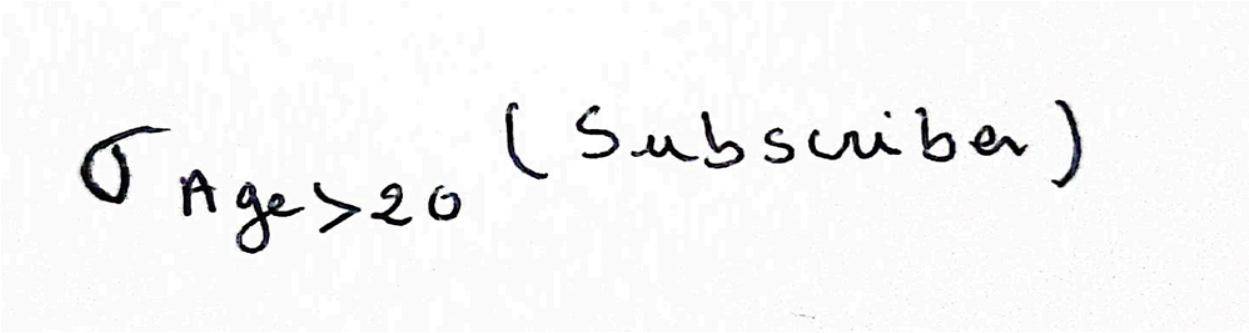


2. The global algebraic tree grouping the three queries at same time:

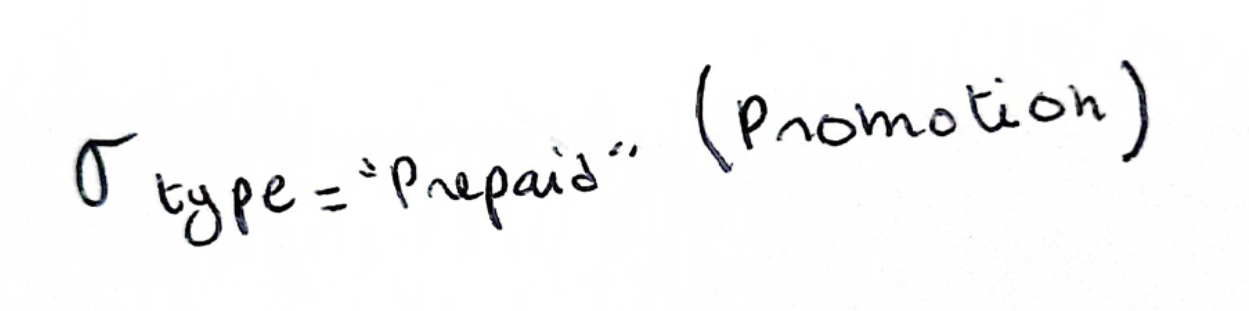


3. The algebraic expressions of all candidates materialized views to optimize these queries:

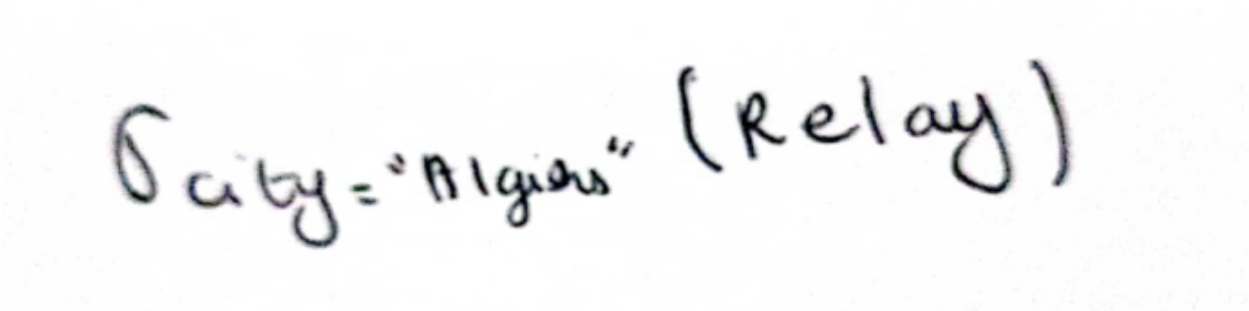
Materialized view 01:



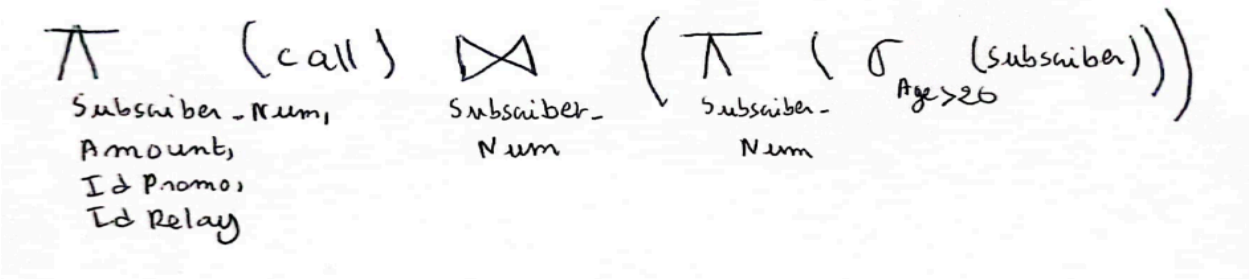
Materialized view 02:



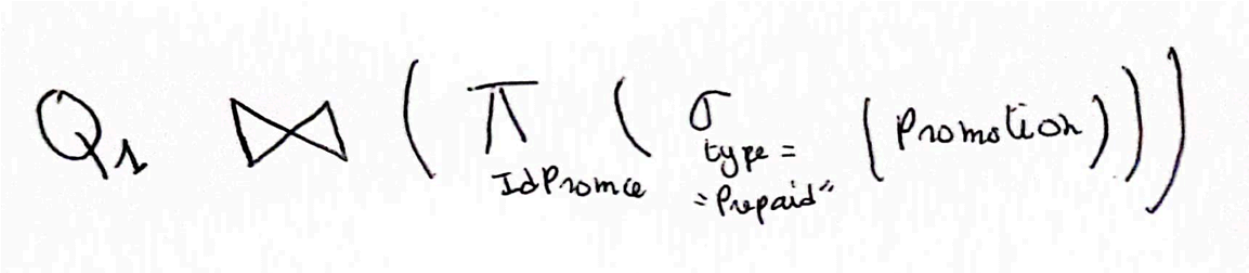
Materialized view 03:



Materialized view 04:



Materialized view 05:



4. The most interesting materialized view:

The most interesting materialized view is the fourth one, because it is the most used one.

5. Give the SQL query that creates this view:

```
CREATE MATERIALIZED VIEW V4 AS
SELECT * FROM Subscriber S, Call C
WHERE S.Subscriber_Num = C.Subscriber_Num
AND S.Age > 20;
```

6. Re-writing the queries Q1, Q2 and Q3 regarding this view:

Q1:

```
SELECT SUM(AMOUNT)
FROM V4;
```

Q2:

```
SELECT SUM(AMOUNT)
FROM V4 v, Promotion p
WHERE v.IdPromo = p.IdPromo
AND p.type = “Prepaid”;
```

Q3:

```
SELECT AVG(AMOUNT)
FROM V4 v, Relay r
WHERE v.IdRelay = r.IdRelay
AND r.City = “Algiers”;
```

2- Fragmentation:

1. Proposing a fragmentation of the table “Subscriber” to optimize the query Q1 along with the algebraic expressions.

We do a horizontal fragmentation on table Subscriber.

Fragment 01: Subscriber_less_20 = σ age <= 20 (Subscriber)

Fragment 02: Subscriber_over_20 = σ age > 20 (Subscriber)

2. Proposing fragmentation of the table “Calls”. Give the algebraic expressions.

We do fragmentation by reference.

Fragment 01: call_less_20 = Call $\triangleright\triangleleft$ Subscriber_less_20
Fragment 02: call_over_20 = Call $\triangleright\triangleleft$ Subscriber_over_20

3. Giving the scripts of fragmentation for the tables “Subscriber” and “Call”.

Subscriber:

```
CREATE TABLE Subscriber_Partitioned (
Subscriber_Num VARCHAR(20) PRIMARY KEY,
Age NUMBER (2),
Address VARCHAR (20),
Profession VARCHAR (20),
Category VARCHAR (20)
)
PARTITION BY RANGE (Age) (
PARTITION Subscriber_less_20 VALUES LESS THAN (21),
PARTITION Subscriber_over_20 VALUES LESS THAN (MAXVALUE) )
);
```

Call:

```
CREATE TABLE Call_Partitioned (
Duration Number,
Amount Number,
Subscriber_Num VARCHAR(20),
IdRelay VARCHAR(20),
IdPromo VARCHAR(20),
IdTime VARCHAR(20),
CONSTRAINT Sub_Num FOREIGN KEY ( Subscriber-Num) REFERENCES
Subscriber_Partitioned ( Subscriber-Num),
CONSTRAINT id-rel FOREIGN KEY ( IdRelay) REFERENCES Relay (
Subscriber-Num),
CONSTRAINT idpro FOREIGN KEY ( IdPromo) REFERENCES Promotion (
IdPromo),
CONSTRAINT idt FOREIGN KEY ( IdTime) REFERENCES Time ( IdTime)
```

```
)
PARTITION BY REFERENCE (Sub_Num)
);
```

4. Rewriting the queries Q1 and Q3 on the fragmented schema.

Q1 : SELECT SUM (Amount) FROM Call_Partitioned;
Q3 : SELECT AVG (Amount) FROM Call_Partitioned C, Relay R
ON C.IdRelay = R.IdRelay WHERE City=”Algiers”;

2- Bitmap Index:

1. Proposing a join bitmap index to optimize the query Q2.

We will create an index on table Promotion , on attribute Type, on the value “Prepaid”.

```
CREATE OR REPLACE BITMAP INDEX Call_Promotion ON
Call (Promotion.type)
WHERE Call.IdPromo = Promotion.IdPromo;
```

2. Calculating the size of this index in KB.

We have a danse index here.

We have :

| Call | = 100 000 000 records.
Index Size = Card (Table) * Card (Attribute) / 8 * 1024.
Index Size = Card (Call) * Card (Type) / 8 * 1024.
Index Size = 100000000 * 2 / 8 * 1024.
Index Size = 100000000 * 2 / 8 * 1024 = 23.84 Kb.

3. Using the index to answer query Q2.

The system will load the index , it retrieves the Row Ids (Rid) from table “Call” where the type value = ”Prepaid” then it returns it to the OS.
It gets back the tuples of the “Call” table, then it performs a join with table “Subscriber” where age is over 20.