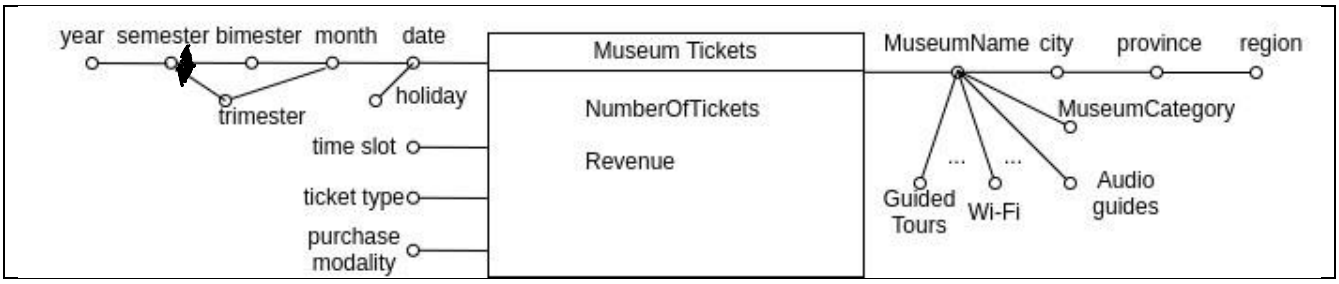
Data science and database technology
Homework #1 - A.Y. 2022/2023

Draft solution

1-Data warehouse – Conceptual schema



1-Data warehouse – Logical schema

Museum (**Id_museum**, museum_name, city, province, region, museum_category, guided_tour, wifi,.., audioguides)
TimeDim (**id_time,** date, month, bimester, trimester, semester, year, holiday)
TimeSlot (**Id_timeslot,** timeslot)

(a) Solution with Junk dimension

TicketInfo (**Id_ticket_info,** ticket_type, purchase_modality)
MuseumTickets (**Id_museum**, **id_time**, **d_timeslot**, **Id_ticket_info,** NumberOfTickets, Revenue)

(b) Solution with Push down

MuseumTickets(**Id_museum**, **id_time, d_timeslot, ticket_type**, **purchase_modality,** NumberOfTickets, Revenue)

The FOLLOWING query solutions are written considering the push down solution

2. Queries.

(a) Separately for each ticket type and for each month (of the ticket validity), analyze: the average daily revenue, the cumulative revenue from the beginning of the year, the percentage of tickets related to the considered ticket type over the total number of tickets of the month.

```
SELECT ticket_type, month, year sum(revenue)/count(distinct date),
sum(sum(revenue)) over (partition by ticket_type, year
                        order by month
                        rows unbounded preceding),
100*sum(numtickets)/sum(sum(numtickets)) over (partition by month)
FROM museums_tickets mt, timedim t
WHERE mt.id_time = t.id_time
GROUP BY ticket_type, month, year;
```

(b) Considering the ticket of 2021. Separately for each museum and ticket type analyze: the average revenue for a ticket, the percentage of revenue over the total revenue for the corresponding museum category and ticket type, assign a rank to the museum, for each ticket type, according to the total number of tickets in decreasing order.

```
SELECT  museum_name, museum_category, ticket_type,
sum(revenue)/sum(numtickets),
100*sum(revenue)/sum(sum(revenue)) over (partition by ticket_type,
                                                    museum_category),
rank() over (partition by ticket_type
             order by sum(num_tickets) desc )
FROM museums_tickets mt, timedim t, museums m
WHERE mt.id_time = t.id_time and mt.id_museum = m.id_museum and year=2021
GROUP BY museum_name, museum_category, ticket_type;
```

3. Create and update a materialized view with **CREATE MATERIALIZED VIEW** and **CREATE MATERIALIZED VIEW LOG** in ORACLE

3.1. Analysis of the target queries

Consider the following frequent queries of interest:

1) Separately for each ticket type and for each month analyze the average daily revenue.

```
aggregated values SUM(Revenue), COUNT(DISTINCT Date)
GROUP BY ticket_type, month
```

2) Separately for each ticket type and for each month analyze the cumulative revenue from the beginning of the year.

```
aggregated values SUM(Revenue)
GROUP BY ticket_type, month, year
```

3) Separately for each ticket type and for each month analyze the total number of tickets, the total revenue and the average revenue.

```
aggregated values SUM(Revenue), SUM(NumberOfTickets)
GROUP BY ticket_type, month
```

4) Separately for each ticket type and for each month analyze the total number of tickets, the total revenue and the average revenue for year 2021.

```
As in (3), but including condition Year = 2021
```

5) Analyze the percentage of tickets related to each ticket type and month over the total number of tickets of the month.

```
aggregated values SUM(NumberOfTickets)
GROUP BY ticket_type, month
```

3.2. Definition of materialized view VM1

Option (1)

- GROUP BY ticket_type, month, (+ year)
- This solution that does not allow computing query 1 because it is not possible to keep the aggregated vale COUNT(DISTINCT Data)

Option (2)

- GROUP BY ticket_type, date (+month +year)
- This solution allows computing all queries, but the cardinality of the materialized view table increases by about 30 times

Option (1) is selected

```
create materialized view VM1
build immediate refresh FAST ON COMMIT
--enable query rewrite
as
SELECT Month, Year, ticket_type , SUM(NumberOfTickets) as NumTickets,
SUM(Revenue) as TotRevenue
FROM museums_tickets mt, timedim t
WHERE mt.id_time = t.id_time
GROUP BY Month, Year, ticket_type;
```

3.3. Materialized view identifiers: ticket_type, month

3.4 Materialized view logs

```
CREATE MATERIALIZED VIEW LOG ON museums_tickets
WITH SEQUENCE, ROWID
(id_time, ticket_type, NumberOfTickets, Revenue)
INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW LOG ON TIMEDIM
WITH SEQUENCE, ROWID
(id_time, Month, Year)
INCLUDING NEW VALUES;
```

4. Trigger

```
CREATE TABLE VM1 (

DateMonth DATE CHECK (DateMonth IS NOT NULL),

DateYear INTEGER CHECK (DateYear IS NOT NULL),

Ticket_Type VARCHAR(20) CHECK (phoneRate IS NOT NULL),

TOT_NumberOfTickets INTEGER,

TOT_Revenue INTEGER);



INSERT INTO VM1 (DateMonth, DateYear, Ticket_Type,
TOT_NumberOfTickets, TOT_Revenue)

(SELECT DateMonth, DateYear, Ticket_Type,
SUM(NumberOfTickets), SUM(Revenue)

FROM FACTS F, TIMEDIM T

WHERE F.ID_time = T.ID_time

GROUP BY DateMonth, DateYear, Ticket_Type);
```

```
create TRIGGER TriggerForViewVM1
AFTER INSERT ON museums_tickets
FOR EACH ROW
DECLARE
N NUMBER;
VAR_DateMonth DATE;
VAR_DateYear NUMBER;

BEGIN
--- Read values “Month” and “Year” needed to update VM1
SELECT DateMonth, DateYear INTO VAR_DateMonth, VAR_DateYear
FROM TIMEDIM
WHERE ID_time = :NEW.ID_time;

--- Check if the record exists in VM1
SELECT Count(*) INTO N
FROM VM1
WHERE DateMonth=Var_DateMonth AND Ticket_Type=:NEW.ticket_type;

IF (N > 0) THEN
--the record exists in VM1
UPDATE VM1
SET TOT_NumberOfTickets=TOT_NumberOfTickets+:NEW.NumberOfTickets,
    TOT_Revenue = TOT_Revenue +:NEW.Revenue,
WHERE DateMonth= Var_DateMonth AND
      Ticket_Type=:NEW.ticket_type;

ELSE
--the record does not exist in VM1

INSERT into VM1 (DateMonth, DateYear, Ticket_Type,
                                Tot_NumberOfTickets, Tot_Revenue)
VALUES(Var_DateMonth, Var_DateYear, :NEW.ticket_type,
:NEW.NumberOfTickets, :NEW.Revenue);
```

```
END IF;
END;
```