Homework 1

Data Science and Database Technology

Data analysts of the National Association of Italian Museums are interested in analyzing the average revenue per ticket.

In particular, they would like the analyses to address the following features.

A museum has a unique name, and it is located in a specific city. The province and region are also stored. The same city can host different museums. Each museum belongs to a specific category (e.g., “Art”, “Historic sites”, “Natural history”).

A museum may have some additional services available for its public. The systems record which services are available for each museum. Examples of additional services are “guided tours”, “audio guides”, “wardrobe”, “café”, “Wi-Fi”. The number of additional services is 10 and their complete list is known.

The tickets sold by each museum are recorded. There are 3 different types of tickets: “Full Revenue”, “Reduced-student” (for students from 14 to 24 years old) and “Reduced-junior” (for young people less than 14 years old).

The systems also store how the ticket is purchased. A ticket can be purchased in three modalities: online, in authorized ticket offices, or directly at the entrance of the museum.

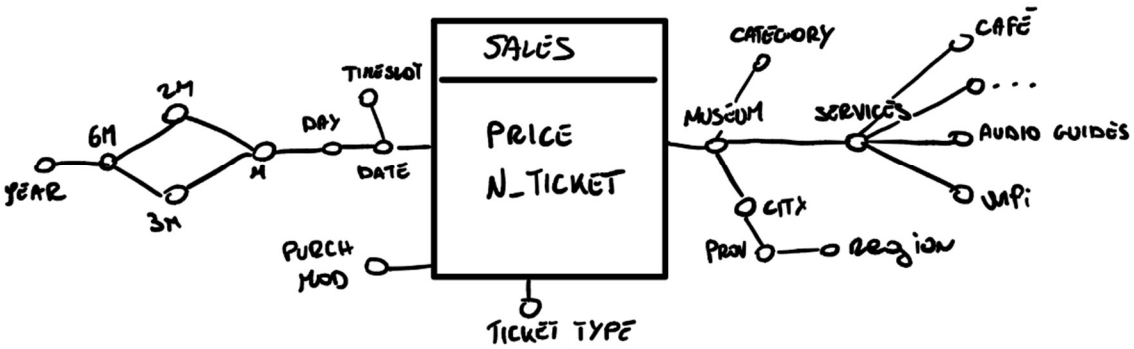
The analyses must be carried out considering the date, month, bimester, trimester, semester, year, if the date is a working day or a holiday, and time slot of the ticket validity date. The time slot is stored in 3 ranges of 4-hour blocks (08:00-12:00, 12:01-16:00, 16:0120:00).

The company is interested in the statistics on the average revenue per ticket. The analysis must be carried on based on:

- museum name, museum category, city, province, region
- museum services
- Ticket type (full-Revenue, reduced-student, reduced-junior)
- Purchase modality (online, in authorized ticket offices, or at the museum entrance)
- Ticket validity date, month, bimester, trimester, semester, working day, and time slot

Homework tasks

1. Design the data warehouse to address the specifications and to efficiently answer to all the provided frequent queries. Draw the conceptual schema of the data warehouse and the logical schema (fact and dimension tables).




```
SALES(TIME_ID, MUSEUM_ID, PRICE, N_TICKET, PURCH_MODE, TICKET_TYPE)
TIMEDIM(TIME_ID, DATE, TIMESLOT, DAY, DATEMONTH, 2M, 3M, 6M, DATEYEAR)
MUSEUM(MUSEUM_ID, CATEGORY, CITY, PROV, REGION, WIFI, CAFÈ, AUDIO_GUIDES, ...)
```

2. Write the following frequent queries using the extended SQL language.

    2.1. 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 S.TICKET_TYPE, T.DATEMONTH,
    SUM(S.PRICE) / COUNT(DISTINCT T.DATE),

    SUM(SUM(S.PRICE)) OVER (PARTITION BY T.DATEYEAR),

    100 * SUM(S.N_TICKET) /
    SUM(SUM(S.N_TICKET)) OVER (PARTITION BY T.DATEMONTH)
FROM SALES S, TIMEDIM T
WHERE S.TIME_ID = T.TIME_ID
GROUP BY S.TICKET_TYPE, T.DATEYEAR, T.DATEMONTH;
```

2.2. 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, assign a rank to the museum, for each ticket type, according to the total number of tickets in decreasing order.

```
SELECT M.MUSEUM_ID, S.TICKET_TYPE,
    SUM(S.PRICE) / SUM(S.N_TICKET),

    100 * SUM(S.PRICE) /
    SUM(SUM(S.PRICE)) OVER (PARTITION BY M.CATEGORY),

    RANK () OVER (ORDER BY SUM(S.N_TICKET) DESC)
FROM SALES S, TIMEDIM T, MUSEUM M
WHERE S.TIME_ID = T.TIME_ID AND T.DATEYEAR = 2021
    AND S.MUSEUM_ID = M.MUSEUM_ID
GROUP BY M.MUSEUM_ID, S.TICKET_TYPE, M.CATEGORY;
```

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

Consider the following frequent queries of interest:

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

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

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

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.

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

3.1. Define a materialized view with CREATE MATERIALIZED VIEW useful to reduce the response time of the reported frequent queries.

```
CREATE MATERIALIZED VIEW
BUILD IMMEDIATE
REFRESH FAST ON DEMAND
ENABLE QUERY REWRITE
AS (
    SELECT S.TICKET_TYPE, T.DATEMONTH, T.DATEYEAR
        SUM(S.PRICE),
        SUM(S.N_TICKET)
    FROM SALES S, TIMEDIM T
    WHERE S.TIME_ID = T.TIME_ID
    GROUP BY S.TICKET_TYPE, T.DATEMONTH, T.DATEYEAR
);
```

3.2. Define the materialized view logs with CREATE MATERIALIZED VIEW LOG for each table where you deem it necessary. For which tables is it useful to keep track of logs? Identify *all* and *only* the necessary tables. Furthermore, for each table identify *all* and *only* the attributes for which it is necessary to keep track of the variations.

```
CREATE MATERIALIZED VIEW LOG ON SALES
WITH SEQUENCE, ROWID
(TIME_ID, TICKET_TYPE, PRICE, N_TICKET)
INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW LOG ON TIMEDIM
WITH SEQUENCE, ROWID
(TIME_ID, DATEMONTH, DATEYEAR)
INCLUDING NEW VALUES;
```

3.3. Specify which operations (e.g. INSERT on a specific table) cause an update of the defined MATERIALIZED VIEW

A materialized view log is a table that records modification to the master table associated with the materialized view. The monitored operations are INSERT, UPDATE, DELETE. In this case one of these operations on tables SALES and TIMEDIM, and specified attributes. This is needed to update the materialized view with the fast option, without re-creating it from scratch at every single modification.

4. Update and management of views via Trigger

Assuming that the CREATE MATERIALIZED VIEW command is not available, create the materialized views defined in the previous exercise and define the update procedure starting from changes on the fact table created by means of a trigger.

4.1 Create the structure of the materialized view with CREATE TABLE VM1 (…)

```
CREATE TABLE VM1 (
    TicketType VARCHAR(20),
    DateMonth VARCHAR(20),
    DateYear INTEGER,
    TotPrice INTEGER CHECK (TotPrice IS NOT NULL AND TotPrice > 0),
    TotTickets INTEGER CHECK (TotTickets IS NOT NULL AND TotTickets > 0),
    PRIMARY KEY (TicketType, DateMonth, DateYear)
);
```

4.2 Specify an example of statement to populate the VM1 table with the necessary records using the statement

```
INSERT INTO VM1 (TicketType, DateMonth, DateYear, TotPrice, TotTickets)
(
    SELECT S.TICKET_TYPE, T.DATEMONTH, T.DATEYEAR
        SUM(S.PRICE),
        SUM(S.N_TICKET)
    FROM SALES S, TIMEDIM T
    WHERE S.TIME_ID = T.TIME_ID
    GROUP BY S.TICKET_TYPE, T.DATEMONTH, T.DATEYEAR
)
```

4.3 Write the triggers necessary to propagate the changes (insertion of a new record) made in the FACTS table to the materialized view VM1.

```
CREATE TRIGGER InsertNewSale
AFTER INSERT ON SALES
FOR EACH ROW
DECLARE
N number;
TicketTypeVar VARCHAR(20);
DateMonthVar VARCHAR(20);
DateYearVar INTEGER;

BEGIN

SELECT TICKET_TYPE INTO TicketTypeVar
FROM SALES
WHERE TIME_ID = :NEW.TIME_ID AND MUSEUM_ID = :NEW.MUSEUM_ID;

SELECT DATEMONTH, DATEYEAR INTO DateMonthVar, DateYearVar
FROM TIMEDIM
WHERE TIME_ID = :NEW.TIME_ID;

SELECT COUNT(*) INTO N
FROM VM1
WHERE TicketType = TicketTypeVar AND DateMonth = DateMonthVar
   AND DateYear = DateYearVar;

IF(N > 0) THEN
--update the record
    UPDATE VM1
    SET TotPrice = TotPrice + :NEW.PRICE,
        TotTickets = TotTickets + :NEW.N_TICKET
    WHERE TicketType = TicketTypeVar
    AND DateMonth = DateMonthVar
    AND DateYear = DateYearVar;

ELSE
--insert a new record
    INSERT INTO VM1(TicketType, DateMonth, DateYear, TotPrice, TotTickets)
    VALUES (TicketTypeVar, DateMonthVar, DateYearVar, :NEW.PRICE,
:NEW.N_TICKET);

END IF;
END;
```

4.4 Specify which operations (e.g. INSERT) trigger the trigger created in 4.3. *The operation is specified when creating the trigger, in this case it is an INSERT operation.*