Progetto modelli dei dati e DBMS di nuova generazione

Flavio Forenza

Matricola: 938367

1 Esercizio (Object-Relational)

Di seguito viene riportata una progettazione concettuale del dominio proposto, in particolare, nella figura 1, viene mostrato uno schema UML che rappresenta la progettazione effettuata.

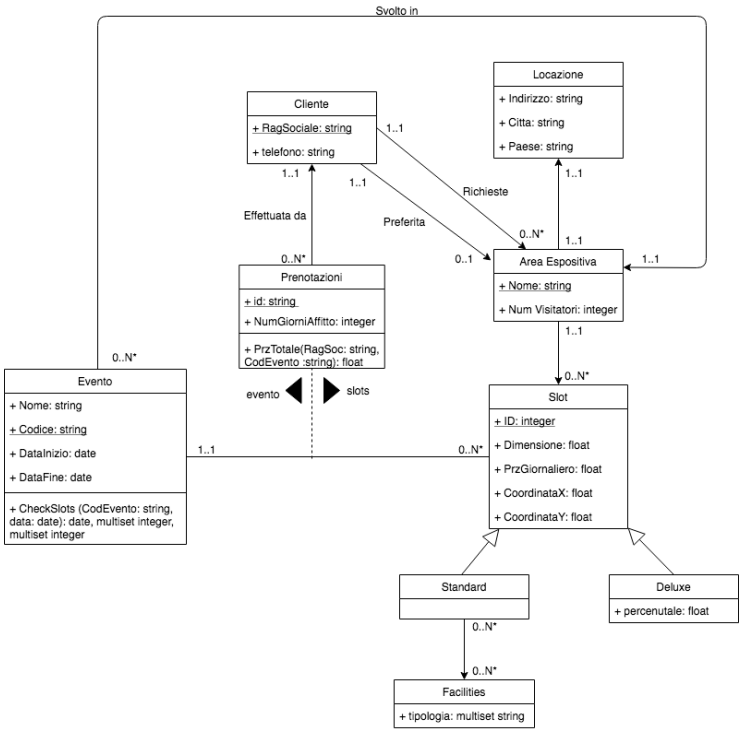

Figure 1: Schema UML

E' bene specificare che quando si effettuerà il passaggio al modello logico, le chiavi primarie sono tutte quelle che sono state sottolineate nello schema UML.

1.1 Versi

Di seguito viene riportata la motivazione che ha portato a scegliere tali versi di associazione fra le classi raffigurate:

- **Area Espositiva - Slot**: Un'area espositiva può contenere diversi slot e ogni slot può appartenere solamente a quell'area espositiva;
- **Area Espositiva - Locazione**: Un'area espositiva è situata in una ed una sola locazione, quest'ultima formata da informazioni quali l'indirizzo, la città e il paese;
- **Cliente - Area Espositiva**: Un cliente stabilisce due relazioni con l'area espositiva. La prima relazione riguarda tutte le differenti aree espositive che che ha richiesto in passato, mentre la seconda riguarda l'area espositiva preferita, preferenza che può esistere o meno;
- **Evento - Area Espositiva**: un evento si può tenere in una specifica area espositiva, mentre in un'area espositiva si possono tenere più eventi, a patto che questi si svolgano in periodi diversi in quanto, a parer mio, non si potrebbe conoscere a priori il numero dei clienti che potrebbero affittare gli slots. Pertanto un evento, quando viene creato, mette a disposizione tutti gli slots di un'area espositiva;
- **Prenotazioni - Cliente**: Una prenotazione può essere effettuata da un cliente, mentre un cliente può aver effettuato più prenotazioni in passato. L'associazione scelta è utile in quanto si potrebbero ricercare le prenotazioni utilizzando le specifiche del cliente, come la sua ragione sociale, oppure l'id della prenotazione posseduta dal cliente;
- **Prenotazioni - (Evento - Slot)**:Una prenotazione, effettuata da un cliente, è composta da due fattori principali: il primo è l'evento, a cui il cliente parteciperà, e il secondo riguardano gli slots che egli ha intenzione di affittare;
- **Standard - Facilities**: Uno slot di tipologia standard può aver associato una lista di tipologie di facilities. Non è necessario, in quanto inutile, creare un'associazione delle facilities con gli slots di tipo deluxe in quanto questi li contengono tutti.

1.2 Vincoli

Ci sono alcuni aspetti che devono essere verificati dal database ma che non sono direttamente resi espliciti dal diagramma UML. In particolare, i vincoli da rispettare sono:

1. In un'area espositiva possono svolgersi al massimo un evento per volta, in base al periodo di quest'ultimi. Quindi un evento, per potersi svolgere, deve ricadere in un periodo in cui non ci siano ulteriori eventi, in quanto ci potrebbero essere conflitti per quanto riguarda la disponibilità degli slots;

2. Non ci possono essere due aree espositive in cui si verifica lo stesso evento in quanto un evento è legato ad una sola area espositiva e ai suoi slots;

3. Il numero dei giorni di affitto, per ogni prenotazione, è al massimo uguale al numero dei giorni in cui si svolge un determinato evento. Questo vincolo è motivato anche dal fatto che non ci sarebbe alcun senso prenotare uno o più slot in una data in cui un evento è già finito o non è nemmeno iniziato;

4. In una prenotazione, gli slots che sono stati prenotati, devono tutti appartenere alla stessa area espositiva in cui si svolge l'evento a cui il cliente vuole partecipare;

5. Non possono esistere due o più prenotazioni che affittino lo stesso slot, contemporaneamente, per un singolo evento. Se uno slot è già stato prenotato questo è occupato;

6. Un cliente può effettuare più di una prenotazione, per lo stesso evento, col vincolo che gli slots differiscano in ogni prenotazione;

7. Ogni slot prenotato resta tale fino alla fine dell'evento, non permettendo a nessun altro cliente, di ri-prenotarlo durante un evento. Ciò è valido anche nel caso in cui il primo cliente non né usufruisca più. Nelle specifiche del progetto non è richiesto di inserire una data in ogni prenotazione. Tale operazione, se fosse concessa, avrebbe migliorato la gestione di ogni slot, in quanto si poteva associare una data di prenotazione ad uno o più slot. Pertanto, a parer mio, la mancanza di tale implementazione aggrava sulla gestione delle prenotazioni;

8. Il prezzo totale dev'essere calcolato sulla singola prenotazione (o più di una) effettuata dallo stesso cliente in questione, per lo stesso evento;

9. Se per vari motivi uno slot, già prenotato, non venisse reso disponibile al cliente, il prezzo dello stesso sarà decurtato dal prezzo totale;

10. La percentuale di uno slot deluxe deve essere sempre positiva e maggiore di zero, in quanto essa è utile per la maggiorazione del prezzo di affitto del singolo slot;

11. L'inserimento di tutti i dati dev'essere corretto, specialmente quando si inseriscono informazioni quali la data di inizio e la data di fine di un evento dove quest'ultima dev'essere cronologicamente successiva (in termini di giorno, mese ed anno), o al massimo uguale, rispetto alla data di inizio.

12. E' importante specificare un numero positivo per le seguenti informazioni: Numero dei visitatori in un'area espositiva, Dimensione di uno slot, Numero dei giorni in affitto in ogni prenotazione;

13. Per poter identificare la posizione di uno slot, entrambe le sue coordinate devono essere inserite. I valori che possono assumere possono essere sia positivi che negativi;

14. Due slots, appartenenti alla stessa area espositiva, non possono avere delle coordinate uguali, in quanto devono essere distanti o affiancati;

15. Gli eventi si differenziano solo per il proprio codice, pertanto ci potrebbero essere eventi con lo stesso nome che potrebbero essere svolti in aree espositive diverse, con date uguali, o diverse. Questa informazione è utile per l'utilizzo della funzione CheckSlots (spiegata nei prossimi paragrafi), situata nella classe Evento nello schema UML, la quale va a ricercare la disponibilità degli slots appartenenti ad un evento, andando a inserire in input parametri quali il codice dell'evento e la data in cui esso si è tenuto.

16. Un evento, che abbia lo stesso nome uguale ad un altro, può ripetersi in un'area espositiva con il vincolo che abbia un identificativo diverso;

17. L'area preferita di un cliente esiste se e solo se fra le prenotazioni compare almeno uno slot che appartiene a tale area;

18. Le aree richieste in passato dal cliente esistono se e solo se esistono fra le prenotazioni degli slots appartenenti a tali aree espositive.

L'implementazione dei vincoli citati sono visibili all'interno dello script SQL ("Vincoli in SQL.sql") in allegato nella cartella del progetto.

1.3 Dal modello concettuale al modello logico

Di seguito viene mostrata una tabella contenente il passaggio dal modello concettuale ad oggetti, al modello logico relazionale ad oggetti. Si vuole sottolineare l'importanza della classe Prenotazione, questa considerata una *Entry-Point* per la formulazione delle interrogazioni.

CLASSE	RELAZIONE	TIPO	RELAZIONE TIPATA	ATTRIBUTI
LOCAZIONE		t_locazione		Indirizzo: string Citta: string Paese: string
SLOT		t_slots	Slots	ID: integer Dimensione: float PrzGiornaliero: float CoordinataX: float CoordinataY: float
FACILITIES		t_facilities		Tipologia: multiset string
STANDARD		t_standard under t_slots		Faciltities: multiset t_facilities
DELUXE		t_deluxe under t_slots		Percentuale: float
AREA ESPOSITIVA		t_area	Area	Nome: string Locazione: t_locazione NumVisitatori: integer Slots: multiset ref t_slots
EVENTO		t_evento	Evento	Nome: string Codice: string DataInizio: date DataFine: date AreaEspositiva: ref t_area
CLIENTE		t_cliente	Cliente	RagSociale: string Telefono: string AreaPreferita: ref t_area AreeRichieste: multiset ref t_area
PRENOTAZIONE		t_prenotazione	Prenotazione	id: string NGiorniAffitto: integerEvento: ref t_evento SlotsPrenotati: multiset ref t_slots

Figure 2: Mappatura dal modello concettuale ad oggetti al modello logico relazionale ad oggetti

1.4 Metodi

Da come si può vedere nello schema UML, nelle classi Evento e Prenotazione compaiono dei metodi, utili a svolgere alcuni punti richiesti nelle specifiche di progetto. In particolare, dato che i metodi si suddividono in funzioni e procedure, in questo caso sono state scelte delle funzioni che restituiscono dei valori, oltre che a stampare a video dei messaggi. Di seguito, vengono specificati i loro compiti e le loro implementazioni in Oracle utilizzando il linguaggio *PL/SQL*:
CheckSlots: La seguente funzione è presente nella classe Evento, ed è utile per poter restituire lo stato di ogni singolo slot, nello specifico di restituire informazioni utili a confermare o meno la disponibilità di ogni singolo slot, in una data specifica in cui si verifica un evento. La sua implementazione avviene direttamente nella definizione del tipo *t-evento*, come nell'immagine seguente:

```
create or replace type t_evento as object(
    nome varchar(30),
    codice varchar(30),
    DataInizio date,
    DataFine date,
    AreaEspositiva ref t_area,
    member function CheckSlots (CodEvento string, data date) return set_check_slots
)
```

Figure 3: Implementazione metodo CheckSlots

In particolare, si nota che la *member function* accetta in input due tipi di parametri quali il codice dell'evento (*CodEvento*) e la data (*Data*) in cui si svolge tale evento. I due parametri sono utili per rispondere alla specifica richiesta dal progetto, dove compare la seguente frase: *"La Firra s.r.l. ha sempre bisogno di sapere per ogni evento quali sono gli slot disponibili e quali sono quelli prenotati in un determinato giorno"*. Quando si effettua un'interrogazione che va a richiamare tale funzione, oltre all'evento, chi inserirà i dati sarà interessato soprattutto alla data in cui esso si svolge. La data inserita sarà confrontata con le date in cui si tiene l'evento specificato, pertanto verranno restituiti a video tutti gli slots che sono disponibili o occupati. Ritornando alla parte implementativa, all'interno di tale metodo vi è una chiamata a due metodi racchiusi all'interno di un ***Package***.

```
create or replace package p_check_slot as
    function get_slots_liberi(CodEvento string, data date) RETURN set_id;
    function get_slots_occupati(CodEvento string, data date) RETURN set_id;
end check_slot;
```

Figure 4: Definizione del package

I metodi interessati sono: *getslotsiberi(CodEvento string, data date) e getslotsoccupati(CodEvento string, data date)*. Da come si evince dal nome, un metodo ha il compito di restituire tutti quei slot che risultano essere liberi in una certa data, mentre il secondo restituirà tutti quei slot che sono occupati in un certa data. Lo sviluppo delle due funzioni avviene all'interno del *body* del package, come mostrato nella seguente figura:

```
create or replace package body p_check_slot as
    FUNCTION get_slots_liberi(CodEvento string, data date)
    RETURN set_id
    IS slots_liberi set_id;
    BEGIN
        select value(sl).id bulk collect into slots_liberi
        from evento p1, table(p1.areaespositiva.slots) sl
        where (p1.codice = CodEvento and value(sl).id not in(
        select value(sl1).id
        from prenotazione p, table(p.slots) sl1))
        and (data between p1.datainizio and p1.datafine);
        RETURN slots_liberi;
    end get_slots_liberi;
    FUNCTION get_slots_occupati(CodEvento string, data date)
    RETURN set_id
    IS slots_occupati set_id;
    BEGIN
        select value(sl).id bulk collect into slots_occupati
        from evento p1, table(p1.areaespositiva.slots) sl
        where (p1.codice = CodEvento and value(sl).id in(
        select value(sl1).id
        from prenotazione p, table(p.slots) sl1))
        and (data between p1.datainizio and p1.datafine);
        RETURN slots_occupati;
    end get_slots_occupati;
end check_slot;
```

Figure 5: Body del package

Dopo aver creato il body, e dopo aver popolato il suo corpo, si deve sviluppare la funzione principale (CheckSlots) dichiarata nel tipo t-evento, affinché richiamasse il package e le funzioni implementate in quest'ultimo.

```
create or replace type body t_evento
as
    member function CheckSlots(CodEvento string, data date) RETURN set_check_slots
is
    numberID set_check_slots;
begin
    DBMS_OUTPUT.put_line ('Informazioni sugli slot relativi all evento: '|| CodEvento || ' in data: ' ||data);
    for el in(
    select p_check_slot.get_slots_liberi(CodEvento,data) as lib,
    p_check_slot.get_slots_occupati(CodEvento,data) as occ
    from dual)
    Loop
    if el.occ is not null
    then
    DBMS_OUTPUT.put_line ('Slots occupati:' );
        for i in 1..el.occ.count()
        Loop
            DBMS_OUTPUT.put_line (el.occ(i));
        end loop;
    else
    DBMS_OUTPUT.put_line ('Tutti gli slots sono liberi.');
    end if;
    if el.lib is not null
    then
    DBMS_OUTPUT.put_line ('Slots liberi:' );
        for i in 1..el.lib.count()
        Loop
            DBMS_OUTPUT.put_line (el.lib(i));
        end loop;
    else
    DBMS_OUTPUT.put_line ('Tutti gli slots sono occupati.');
    end if;
    end Loop;

    select check_slots(data, pp.lib, pp.occ) bulk collect into numberID
    from(
    select P_check_slot.get_slots_liberi(CodEvento,data) as lib, p_check_slot.get_slots_occupati(CodEvento,data) as occ
    from dual
    )pp;
    return numberID;
    end CheckSlots;
end;
```

Figure 6: Funzione CheckSlots

E' doveroso esplicitare che il tipo di ritorno di questa funzione (*set-check-slots*) a sua volta è una tabella del tipo check-slot il quale restituisce la data e le informazioni circa gli slot liberi e occupati.

```
create or replace type check_slots is object(
    Data date,
    liberi set_id,
    occupati set_id
);
```

Figure 7: Tipo check-slots

Un esempio di output a video che restituirà tale funzione è il seguente:

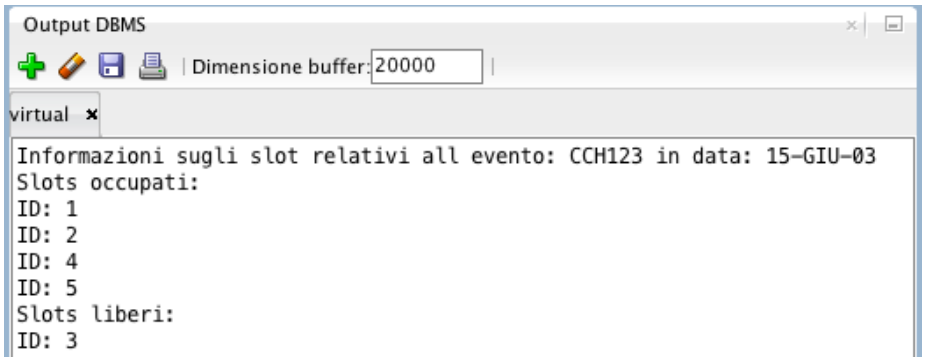

Figure 8: Output DBMS funzione CheckSlots

Ovviamente, se nell'interrogazione venissero inseriti dati sbagliati, non verrà restituito nulla in output. Un esempio di interrogazione è la seguente:

```
SET SERVEROUTPUT ON;
select value(llv).data as Data, value(SlotLiberi), value(SlotOccupati)
from evento e, table(e.CheckSlots('CCHi123', to_date('2003/06/15', 'yy/mm/dd'))) llv,
    table(value(llv).liberi) SlotLiberi, table(value(llv).occupati) SlotOccupati;
```

Figure 9: Esempio query per la ricerca degli slots

A questo punto passiamo alla descrizione del secondo metodo. **PrezTotale**: tale metodo è presente nella classe "*Prenotazioni*" e da come si può intuire, è utile per poter calcolare il prezzo totale che un cliente è tenuto a pagare. La signature di tale funzione è stata direttamente dichiarata all'interno della definizione del tipo t-prenotazione:

```
create or replace type t_prenotazione as object(
    Cliente ref t_cliente,
    NGiorniAffitto number(5,2),
    Evento ref t_evento,
    Slots set_slots,
    member function PrezTot (rsCliente string, codEvento string) return number
)
```

Figure 10: Dichiarazione funzione PrezTot

Tale funzione richiede di specificare la ragione sociale del cliente e il codice dell'evento a cui esso ha partecipato. Il calcolo del prezzo totale tiene conto di diversi aspetti. Il prezzo finale sarà composto dalla somma dei singoli prezzi di ogni slot, a cui va sommata la percentuale degli slot deluxe (se prenotati), il tutto moltiplicato per il numero di giorni della prenotazione. L'implementazione di tale funzione, a differenza della prima, non richiede la creazione di alcun package, ma è stato sviluppato soltanto il suo corpo:

```
create or replace type body t_prenotazione
as
    member function PrezTot (rsCliente string, codEvento string) return number
is
    totale number(7,2) :=0;
    percentuale number;
    numGiorni number :=0;
begin
    DBMS_OUTPUT.put_line ('--- Prenotazioni del cliente '||rsCliente || ' ---');
    for slot in (
    select deref(value(sl)) as ss, p.ngiorniaffitto as giorni
    from prenotazione p, table(p.slots) sl
    where p.cliente.ragsociale = rsCliente
    and p.evento.codice = codEvento
    )
    LOOP
        if slot.ss is of(t_deluxe)
        then
            select TREAT(value(s) as t_deluxe).percentuale into percentuale
            from slots s
            where s.id = slot.ss.id;
            DBMS_OUTPUT.put_line ('Slot ID: '||slot.ss.id||' --> Tipo Deluxe €' ||slot.ss.pzgiornaliero||'+'||percentuale||'%');
            totale := totale + slot.ss.pzgiornaliero;
            totale := totale + (slot.ss.pzgiornaliero/100)*percentuale;
        else
            DBMS_OUTPUT.put_line ('Slot ID: '||slot.ss.id||' --> Tipo Standard €'||slot.ss.pzgiornaliero);
            totale := totale + slot.ss.pzgiornaliero;
        end if;
        numGiorni := slot.giorni;
    end loop;
    totale := totale * numGiorni;
    DBMS_OUTPUT.put_line ('Numero giorni affitto: ' || numGiorni);
    DBMS_OUTPUT.put_line ('Prezzo totale: €'||totale);
    return totale;
    end PrezTot;
end;
```

Figure 11: Body funzione PrezTotale

La funzione è richiamabile eseguendo una semplice query come la seguente:

```
SET SERVEROUTPUT ON;
select p.PrezTot('qwe11', 'CCH123') as totale
from prenotazione p
where rownum = 1
```

Figure 12: Esempio query per il calcolo del prezzo totale

Un esempio di output a video è il seguente:

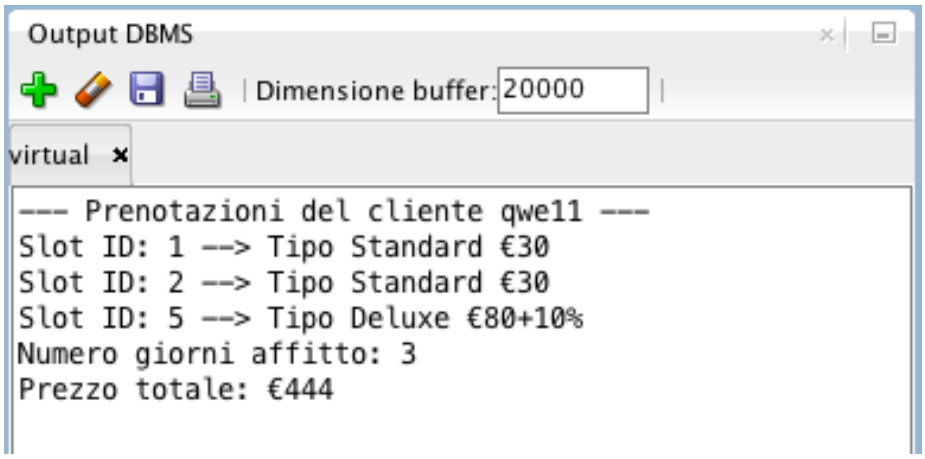

Figure 13: Output DBMS funzione PrezTot

Anche qui, come per l'altra funzione, se i dati inseriti non fossero corretti, allora verrà restituito il valore 0.

2 Esercizio (XML)

Per la creazione dello schema XML, affinché tutti i punti richiesti venissero rispettati, sono stati utilizzati alcuni concetti validi nella creazione di tali strutture, ovvero:

- **Unique**: l'elemento unique è stato utile per poter definire l'univocità dell'attributo ***id*** appartenente a ciascuno slot, in quanto era richiesto che non ci possono essere due o più slots con gli stessi id;
- **Keyref**: l'elemento keyref è risultato utile in quanto permette di riferirsi al valore di un elemento. In questo caso è risultato utile per far combaciare gli idetnificativi degli slots presenti nel cammino, con gli identificativi degli slots presenti nel documento;
- **Elemento Globale**: all'interno dello schema vi è la presenza di un singolo elemento globale (*Visita*). Tale dichiarazione è utile per creare un unico figlio per l'intero schema;
- **Tipi**: il sistema dei tipi (complexType e simpleType) è stato utilizzato per la costruzione di alcuni elementi presenti nell'elemento globale.

Di seguito viene riportata l'intera struttura dello schema XML (allegato nel progetto come "SchemaXML-Fiere"):

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <xsd:complexType name="tSlot">
        <xsd:sequence>
                <xsd:element name="Nome" type="xsd:string"/>
                <xsd:element name="Tipo">
                        <xsd:simpleType>
                            <xsd:restriction base="xsd:string">
                                <xsd:enumeration value="standard"/>
                                <xsd:enumeration value="deluxe"/>
                            </xsd:restriction>
                        </xsd:simpleType>
                </xsd:element>
                <xsd:element name="TipoProdottiEsposti" type="xsd:string" maxOccurs="unbounded"/>
                <xsd:element name="NumPassaggi" type="xsd:integer"/>
                <xsd:element name="NomeAzineda" type="xsd:string"/>
                <xsd:element name="Voto" minOccurs="0" maxOccurs="1">
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                            <xsd:enumeration value="MiPiace"/>
                            <xsd:enumeration value="NonMiPiace"/>
                            <xsd:enumeration value="Indifferente"/>
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:element>
        </xsd:sequence>
            <xsd:attribute name="id" type="xsd:string" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="tPercorso">
        <xsd:sequence>
            <xsd:element name="Slot" maxOccurs="unbounded">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name = "OraEntrata" type="xsd:time"/>
                        <xsd:element name = "OraUscita" type="xsd:time"/>
                        <xsd:element name = "TempoTrascorso" type="xsd:time"/>
                    </xsd:sequence>
                    <xsd:attribute name="ref" type="xsd:string" use="required"/>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="Visita">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Slots">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="Slot" type="tSlot"
                            minOccurs="0" maxOccurs="unbounded"/>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="Percorso" type="tPercorso" minOccurs="0"/>
            </xsd:sequence>
            <xsd:attribute name="Id" type="xsd:string" use="required"/>
            <xsd:attribute name="Evento" type="xsd:string" use="required"/>
            <xsd:attribute name="Giorno" type="xsd:date" use="required"/>
            <xsd:attribute name="Entrata" type="xsd:time" use="required"/>
        </xsd:complexType>
        <xsd:unique name="unique-SlotId">
                    <xsd:selector xpath="Slots/Slot"/>
                    <xsd:field xpath="@id"/>
        </xsd:unique>
        <xsd:keyref name="aKeyRef" refer="unique-SlotId">
            <xsd:selector xpath="Percorso/Slot" />
            <xsd:field xpath="@ref"/>
        </xsd:keyref>
    </xsd:element>

</xsd:schema>
```

Figure 14: XML Schema

Ovviamente, quando si va a popolare l'intero file XML, ci sono alcuni **vincoli** da rispettare. Per esempio, oltre alla validità del tipo da inserire, il numero dei passaggi dev'essere un numero sempre positivo in quanto o al massimo uguale a

zero. Un altro campo che richiede attenzione è il campo "Tempo Trascorso", che dev'essere dato dalla differenza tra l'ora di entrata e l'ora di uscita. Un esempio di documento XML corretto, derivante dallo schema costruito, è il seguente:

```xml
<?xml version="1.0" encoding="utf-8"?>
<Visita xsi:noNamespaceSchemaLocation="schema.xsd" Id="1234" Evento="Misto"
        Giorno="2001-10-05" Entrata="12:28:30.68"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <!--Esempio di documento XML valido per lo schema definito-->

   <Slots>
    <Slot id="sl123">
      <Nome>Tech</Nome>
      <Tipo>standard</Tipo>
      <TipoProdottiEsposti>Telefonia</TipoProdottiEsposti>
      <TipoProdottiEsposti>TV</TipoProdottiEsposti>
      <TipoProdottiEsposti>Audio</TipoProdottiEsposti>
      <NumPassaggi>46</NumPassaggi>
      <NomeAzineda>Sony</NomeAzineda>
      <Voto>MiPiace</Voto>
    </Slot>
    <Slot id="sl124">
      <Nome>Bonta</Nome>
      <Tipo>deluxe</Tipo>
      <TipoProdottiEsposti>Alimentari</TipoProdottiEsposti>
      <NumPassaggi>38</NumPassaggi>
      <NomeAzineda>Esselunga</NomeAzineda>
      <Voto>MiPiace</Voto>
    </Slot>
    <Slot id="sl125">
      <Nome>Auto</Nome>
      <Tipo>standard</Tipo>
      <TipoProdottiEsposti>Autoveicoli</TipoProdottiEsposti>
      <NumPassaggi>149</NumPassaggi>
      <NomeAzineda>Ferrari</NomeAzineda>
      <Voto>Indifferente</Voto>
    </Slot>
  </Slots>
  <Percorso>
    <Slot ref="sl123">
      <OraEntrata>11:10:00.00</OraEntrata>
      <OraUscita>13:20:00.00</OraUscita>
      <TempoTrascorso>02:10:00.00</TempoTrascorso>
    </Slot>
    <Slot ref="sl124">
      <OraEntrata>16:59:00.00</OraEntrata>
      <OraUscita>17:13:00.00</OraUscita>
      <TempoTrascorso>00:14:00.00</TempoTrascorso>
    </Slot>
  </Percorso>
</Visita>
```

Figure 15: XML valido

E' giusto premette che all'interno del percorso, uno o più slots possono comparire più volte in quanto il visitatore potrebbe passarci o visitarli più volte. Il voto, per quanto un visitatore non sia tenuto obbligatoriamente a darlo, potrebbe risultare o meno. Per quanto riguarda le informazioni quali l'id, l'evento, il giorno e l'entrata, esse sono rese tutte come attributi della Visita.

3 Esercizio

Esercizio 3.a Per poter creare un documento XML che riporti gli slots disponibili per un dato evento, viene fatto uso del linguaggio XML/SQL. La query, utile a produrre quanto detto, va a verificare la presenza o meno di uno slot, in cui viene svolto uno specifico evento, all'interno della tabella "Prenotazione" (discussa nell'esercizio 1). La prima query è funzionante per tutti gli eventi, ovvero andrà a ricercare tutti gli slots liberi per ogni evento:

```
select xmlelement("SlotsLiberi",
xmlagg(
        xmlelement("Evento",
        xmlattributes(p1.codice as "Codice", p1.nome as "Nome"),
        xmlagg(
            xmlelement("Slot",
                xmlattributes(value(sl).id as "ID"),
                xmlforest(
                        value(sl).dimensione as "Dimensione",
                        value(sl).pzGiornaliero as "PrzGiornaliero",
                        value(sl).CoordinataX as "CoordinataX",
                        value(sl).CoordinataY as "CoordinataY"
                    )
                )
            )
        )
    )
)result
from evento p1, table(p1.areaespositiva.slots) sl
where value(sl) not in(
select value(sl1)
from prenotazione p, table(p.slots) sl1
where p.evento.codice = p1.codice)
group by p1.codice, p1.nome;
```

Figure 16: Query per gli slots liberi per ogni evento

Di seguito viene mostrato un esempio di quanto prodotto dalla query precedente:

```
<SlotsLiberi>
    <Evento  Codice="cY" Nome="Y">
        <Slot  ID="13">
            <Dimensione>50</Dimensione>
            <PrzGiornaliero>80</PrzGiornaliero>
            <CoordinataX>15</CoordinataX>
            <CoordinataY>15</CoordinataY>
        </Slot>
        <Slot  ID="19">
            <Dimensione>40</Dimensione>
            <PrzGiornaliero>80</PrzGiornaliero>
            <CoordinataX>15</CoordinataX>
            <CoordinataY>15</CoordinataY>
        </Slot>
        <Slot  ID="18">
            <Dimensione>40</Dimensione>
            <PrzGiornaliero>80</PrzGiornaliero>
            <CoordinataX>15</CoordinataX>
            <CoordinataY>15</CoordinataY>
        </Slot>
        <Slot  ID="17">
            <Dimensione>50</Dimensione>
            <PrzGiornaliero>80</PrzGiornaliero>
            <CoordinataX>15</CoordinataX>
            <CoordinataY>15</CoordinataY>
        </Slot>
    </Evento>
    <Evento  Codice="6789" Nome="Ibiza">
        <Slot  ID="10">
            <Dimensione>50</Dimensione>
            <PrzGiornaliero>80</PrzGiornaliero>
            <CoordinataX>15</CoordinataX>
            <CoordinataY>15</CoordinataY>
        </Slot>
        <Slot  ID="12">
            <Dimensione>20</Dimensione>
            <PrzGiornaliero>30</PrzGiornaliero>
            <CoordinataX>13</CoordinataX>
            <CoordinataY>13</CoordinataY>
        </Slot>
    </Evento>
</SlotsLiberi>
```

Figure 17: Esempio di Slots liberi per tutti gli eventi

Di seguito, viene mostrata invece la query che si occupa della ricerca degli slots liberi di un singolo evento specificato. Come esempio è stato scelto di restituire tutti gli slots liberi per l'evento avente codice "6789". Si ricorda che è meglio specificare il codice che non il nome, in quanto ci potrebbero essere eventi di nome uguale, ma che si verificano in aree diverse:

```
select xmlelement("SlotsLiberi",
xmlagg(
        xmlelement("Evento",
        xmlattributes(p1.codice as "Codice", p1.nome as "Nome"),
        xmlagg(
            xmlelement("Slot",
                xmlattributes(value(sl).id as "ID"),
                xmlforest(
                        value(sl).dimensione as "Dimensione",
                        value(sl).pzGiornaliero as "PrzGiornaliero",
                        value(sl).CoordinataX as "CoordinataX",
                        value(sl).CoordinataY as "CoordinataY"
                    )
                )
            )
        )
    )
)result
from evento p1, table(p1.areaespositiva.slots) sl
where p1.codice = '6789' and value(sl) not in(
select value(sl1)
from prenotazione p, table(p.slots) sl1
where p.evento.codice = '6789')
group by p1.codice, p1.nome;
```

Figure 18: Query per gli slots liberi dell'evento con codice "6789"

Se invece di inserire il codice dell'evento, inserissimo il suo nome, se tale evento è già ripetuto in altre aree, la query restituirà tutti quei slots che sono disponibili in aree espositive diverse. Di seguito, il documento l'XML contenente gli slots liberi per tale evento:

```
<SlotsLiberi>
    <Evento  Codice="6789" Nome="Ibiza">
        <Slot  ID="10">
            <Dimensione>50</Dimensione>
            <PrzGiornaliero>80</PrzGiornaliero>
            <CoordinataX>15</CoordinataX>
            <CoordinataY>15</CoordinataY>
        </Slot>
        <Slot  ID="12">
            <Dimensione>20</Dimensione>
            <PrzGiornaliero>30</PrzGiornaliero>
            <CoordinataX>13</CoordinataX>
            <CoordinataY>13</CoordinataY>
        </Slot>
    </Evento>
</SlotsLiberi>
```

Figure 19: Esempio di Slots liberi per un singolo evento

Esercizio 3.b Per determinare l'evento dell'anno, ovvero l'evento che ha portato un maggior incasso possibile, si è tenuto conto di tutte le prenotazioni che sono state fatte per tale evento. La query prodotta andrà quindi a calcolare l'incasso di ogni evento, e stamperà a video l'evento che ha ottenuto il maggior

incasso. Per determinare il guadagno finale, ottenuto da ogni evento, si sono prese in considerazioni informazioni quali il prezzo degli slots, con aggiunta della percentuale se di tipo deluxe, e del numero di giorni che un cliente ha usufruito degli slots. La strutturazione della query è composta da ulteriori sotto query, una responsabile di restituire la somma di tutti gli slots deluxe, con l'aggiunta della percentuale, e l'altra di restituire la somma per tutti gli slots di tipo standard. Le somme restituite tengono conto del numero di giorni (effettuandone una moltiplicazione). Per effettuare la somma finale, proveniente dalle due sotto query, è stato necessario effettuare tre join fra le seguenti tabelle: "Prenotazione, Tdeluxe, TStandard". Le join sonot risultate utile per la group by, in quanto questa ha raggruppato, in base al codice dell'evento, tutte le somme restituite per tale evento. Sarà la funzione "sum", dichiarata nella prima select, che assieme alla group by, effettuerà la somma finale. La query si conclude con il controllo da parte della funzione "having" la quale prende la somma, di ogni evento, e la confronta con il massimo, per determinarne appunto l'evento che con incasso maggiore di tutti. Ovviamente, se ci fossero eventi con un incasso maggiore uguale, allora verranno restituiti tutti. Di seguito la query:

```
select p.evento.codice, sum(distinct(Tdeluxe.prezzo+Tstandard.prezzo)) as tot
from prenotazione p join (
    select p1.evento.codice as codD, sum(((value(slot).pzgiornaliero) +
    (value(slot).pzgiornaliero/100)*(TREAT(deref(value(slot)) as t_deluxe).percentuale))
    *p1.ngiorniaffitto) as prezzo
    from prenotazione p1, table(p1.slots) slot
    where deref(value(slot)) is of type (t_deluxe)
    group by p1.evento.codice
)Tdeluxe on (p.evento.codice = Tdeluxe.codD) join
(
    select p3.evento.codice as codS,
    sum((value(slot).pzgiornaliero)*p3.ngiorniaffitto) as prezzo
    from prenotazione p3, table(p3.slots) slot
    where deref(value(slot)) is of type (t_standard)
    group by p3.evento.codice
)Tstandard on (Tdeluxe.codD = Tstandard.codS)
group by p.evento.codice
having  sum(distinct(Tdeluxe.prezzo+Tstandard.prezzo))>= ALL(
select max(Tdeluxe.prezzo+Tstandard.prezzo)
from prenotazione p join (
    select p1.evento.codice as codD,
    sum(((value(slot).pzgiornaliero) +
    (value(slot).pzgiornaliero/100)*(TREAT(deref(value(slot)) as t_deluxe).percentuale))
    *p1.ngiorniaffitto) as prezzo
    from prenotazione p1, table(p1.slots) slot
    where deref(value(slot)) is of type (t_deluxe)
    group by p1.evento.codice
)Tdeluxe on (p.evento.codice = Tdeluxe.codD) join
(
    select p3.evento.codice as codS,
    sum((value(slot).pzgiornaliero)*p3.ngiorniaffitto) as prezzo
    from prenotazione p3, table(p3.slots) slot
    where deref(value(slot)) is of type (t_standard)
    group by p3.evento.codice
)Tstandard on (Tdeluxe.codD = Tstandard.codS)
group by p.evento.codice
)
```

Figure 20: Query 3b. Evento con incasso massimo

La query precedente alla having, restituisce il seguente risultato:

	EVENTO.CODICE	TOT
1	12345	530
2	CCH123	708
3	6789	472
4	909090	708

Figure 21: Incasso totale di tutti gli eventi

Mentre, dopo il confronto effettuato dalla funzione having, il risultato finale è il seguente:

	EVENTO.CODICE	TOT
1	CCH123	708
2	909090	708

Figure 22: Risultato finale. Evento con incasso maggiore

Esercizio 3.c La risoluzione di tale punto richiede l'utilizzo del linguaggio SQL-XML in quanto si vanno ad effettuare operazioni su una tabella contenente un campo XMLTYPE. La tabella in questione riporta le seguenti informazioni per ogni visitatore:

- Nome
- Cognome
- Email
- Visite

Il campo "Visite" conterrà tutti i documenti XML prodotti dal braccialetto (richiesto nell'esercizio 2) , dove compaiono gli slots visitati dal singolo visitatore, con il relativo percorso. Per ridurre il numero di duplicazione delle informazioni, e quindi ridurre la ridondanza dei dati, si è pensato di rendere il campo "Email" come chiave primaria, in quanto ogni Email è univoca e appartiene ad un solo individuo. Successivamente, dato che un visitatore può aver partecipato a più eventi, il campo Visite può contenere uno o più documenti XML. La struttura della tabella creata sarà la seguente:

```
create or replace type set_visite as table of XMLTYPE;

create table Visitatori(
    nome varchar(30),
    cognome varchar(30),
    email varchar(50),
    visite set_visite,
    primary key(email)
)
nested table visite store as tab_visite;
```

Figure 23: Struttura tabella Visitatori

La query richiesta, per come strutturata dal sottoscritto, va a prendere gli slots di ogni evento, specificandone il codice di quest'ultimo. Lo scopo finale è quello di restituire a video lo slot che ha ricevuto un numero di visite più alto ricavato dalla somma del numero di passaggi di ogni visitatore. Per come strutturata la tabella, **l'elaborazione di ogni documento XML può avvenire solamente in maniera sequenziale**, pertanto sarà necessario sviluppare una vista, o un'altra tabella, per poter inserire al suo interno tutti i risultati restituiti dalla query. Quest'ultima va a creare n tuple uguale al numero di slots presente nel singolo evento. Una volta aver restituito tutti gli slots, la query andrà ad effettuare la somma del numero di visite e del numero di rating impostati come "Mi Piace". Il risultato è inserito in una vista dalla quale solamente da qui si potrà estrapolare lo slot con il numero di visite massimo. La query in questione è la seguente:

```
create or replace view Final as(
    select xmltab."Evento" as evento, xmltab."Slot" as slot,
    sum(xmltab."Passaggi") as pass,
    sum(xmltab."Rating") as rat
    from visitatori v, table(v.visite) vs, evento even, XMLTABLE(
    '
        let $ev := /Visita[@Evento=$NomeEvento]
        let $slots := distinct-values($ev/Slots/Slot/@id)
        let $master :=(
            <SingoloEvento>
                {
                    for $slot in $slots
                        return
                        <SingoloSlot>
                            <Evento>{$ev/@Evento}</Evento>
                            <Id>{$slot}</Id>
                            <NumPassaggi>{$ev//Slot[@id=$slot]/NumPassaggi}</NumPassaggi>
                            <Rating>{count($ev/Slots/Slot[@id =$slot]/Voto="MiPiace")}</Rating>
                        </SingoloSlot>
                }
            </SingoloEvento>
        )

        let $multiSlot := $master//SingoloSlot

        return
        $multiSlot
    '
        passing value(vs), even.nome as "NomeEvento"
        columns
        "Evento" varchar(50) path '//SingoloSlot/Evento/@Evento',
        "Slot" varchar(50) path '//SingoloSlot/Id',
        "Passaggi" number path '//SingoloSlot/NumPassaggi',
        "Rating" number path '//SingoloSlot/Rating'
    )xmltab
    group by xmltab."Evento", xmltab."Slot"
);
```

Figure 24: Query 3c. Slot preferito

Si vuole far notare che il nome degli eventi sono stati passati direttamente dalla tabella "*Eventi*" sviluppata nell'esercizio 1. Nel caso si è interessati allo slot preferito del singolo evento, allora si può specificarlo direttamente senza passare tutti i nomi degli eventi. All'intero della view creata, ci saranno campi quali:

- Evento
- Slot: rappresentante lo slots maggiormente preferito dell'evento,
- Pass: numero di passaggi (o visite) totali dato dalla somma di tutti i passaggi in ogni evento per tale slot;
- Rat: rappresenta il numero dei ratings (voti) dati dai visitatori a tale slot (verrà considerato solo il voto "Mi Piace").

Un esempio do output a video è il seguente:

	EVENTO	SLOT	PASS	RAT
1	Elettronica	sl125	45	2
2	House	sl11	57	2
3	Elettronica	sl123	33	2
4	House	sl12	12	2
5	Elettronica	sl126	230	1
6	House	sl13	28	2
7	House	sl14	46	2
8	Elettronica	sl124	30	2

Figure 25: Contenuto della view

Effettuando una query sulla view precedentemente creata, possiamo andare ad estrarre il singolo slot, per ogni evento, che ha ottenuto un numero di passaggi maggiore, utilizzando la seguente query:

```
select evento, slot, pass, rat
from final v1
where pass in (
  select max(pass)
  from final v2
  where  v1.evento = v2.evento
)
```

Figure 26: Query per la ricerca degli slots preferiti per ogni evento

L'output generato dalla query è il seguente:

	EVENTO	SLOT	PASS	RAT
1	House	sl11	57	2
2	Elettronica	sl126	230	1

Figure 27: Query per la ricerca degli slots preferiti per ogni evento

Tuttavia, se il database fosse strutturato in maniera diversa, ovvero avesse avuto un unico documento XML all'interno del quale erano riportati tutte le informazioni di ogni visitatore, tale organizzazione avrebbe permesso di creare una

query SQL-XML unica, in grado di calcolare direttamente la somma di ogni visita e di ogni voto ricevuto. Ovviamente questa soluzione è molto opinabile ma rimane la più efficacie per poter gestire le informazioni in questo tipo di linguaggio in quanto si ricorda che ogni tupla viene gestita in maniera del tutto sequenziale, pertanto il contenuto di ogni documento XML viene processato uno alla volta, in base al numero di tuple, ognuna appartenente ad un visitatore.
Esercizio 3.d Per poter svolgere la richiesta, dobbiamo assicurarci che vengano prima creati un nuovo cliente 'X' e un nuovo evento "Y":

```
insert into cliente values('X', '3234567890', null, set_area());

insert into evento values('Y', 'cY', to_date('2003/07/15', 'yy/mm/dd'),
                        to_date('2003/07/18', 'yy/mm/dd'), null);
```

Figure 28: Creazione di un cliente e di un evento

Successivamente creiamo una nuova area e gli assegniamo diversi slots (di cui almeno 3 di tipo deluxe).

```
insert into area values('Test', null, 50, set_slots());

insert into table(select slots from area where nome = 'Test')
select ref(s) from slots s where s.id IN (13,14,15,16,17,18,19);
```

Figure 29: Creazione di un'area con assegnazione di diversi slots

Successivamente, assegniamo all'evento Y l'area precedentemente creata. La seguente query va ad assegnare la prima area che ha 3 slots deluxe liberi (condizione verificata tramite un conteggio), e quindi non prenotati, aventi metratura più alta fra quelli disponibili in tale area. In questo caso si suppone che l'unica area presente nel database sia quella precedentemente creata (Y).

```
update evento
set areaespositiva = (select ref(a) from area a where a.nome =
(--va a selezionare l'area che ha gli slots con maggiore metratura
    select area.Narea
    from(
    --restituisce la prima area che ha almeno 3 slots dalla dimensione massima
        select a1.nome as Narea, max(value(sl).dimensione) as dim, count(*)
        from area a1, table(a1.slots) sl
        where value(sl).id not in (
        select value(slots).id
        from prenotazione p, table(p.slots) slots
        )
        and deref(value(sl)) is of type(t_deluxe)
        group by a1.nome, value(sl).dimensione
        having count(*)>=3
    )area
    where rownum = 1
    group by area.Narea)
)
where nome = 'Y'
```

Figure 30: Assegnamento di un'area ad un evento, con relativo controllo sugli slots

Creiamo una prenotazione associandogli il cliente X, un numero di giorni pari a 2, un evento Y, e un set di slot non ancora specificato (verrà specificato successivamente).

```
insert into prenotazione
select ref(c), 2, ref(e), set_slots()
from cliente c, evento e
where c.ragSociale = 'X' and e.codice = 'cY';
```

Figure 31: Query per la creazione della prenotazione + assegnazione del cliente e dell'evento

A questo punto, andiamo ad assegnare alla prenotazione precedentemente creata, i 3 slots deluxe, con metratura più alta, come richiesto, con il seguente inserimento:

```
insert into table(select p.slots
from prenotazione p where p.evento.codice = 'cY')
select ref(s) from slots s, (
        select value(sl).id as slotArea, value(sl).dimensione
        from evento e, table(e.areaespositiva.slots) sl
        where value(sl) not in (
        select value(slots)
        from prenotazione p, table(p.slots) slots
        )
        and deref(value(sl)) is of type(t_deluxe)
        and e.areaespositiva.nome = 'Test'
        --solo i primi 3 slots
        and rownum<=3
        group by value(sl)
        having max(value(sl).dimensione)>= ALL(
            select value(sl).dimensione
            from evento e, table(e.areaespositiva.slots) sl
            where value(sl) not in (
            select value(slots)
            from prenotazione p, table(p.slots) slots
            )
            and deref(value(sl)) is of type(t_deluxe)
            and e.areaespositiva.nome = 'Test'
            )
         )ev
where s.id = ev.slotArea
```

Figure 32: Assegnamento dei 3 deluxe slots alla prenotazione

Infine, come richiesto, l'area preferita del cliente 'X' diventa l'area precedentemente creata. In questo caso l'area è nominata con il nome di "Test", pertanto effettuiamo una update del cliente X:

```
update cliente
set areapreferita=(select ref(a) from area a where a.nome = 'Test')
where ragsociale = 'X';
```

Figure 33: Assegnamento dell'area Test all'area preferita del cliente X