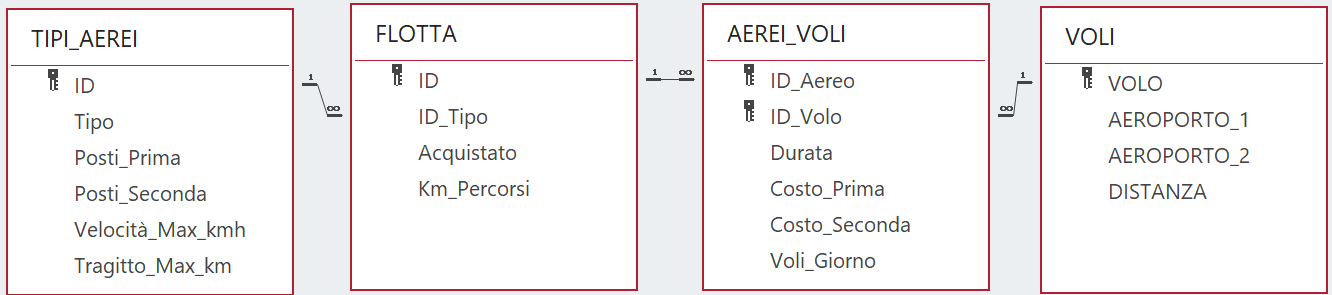
**GESTIONE DELL’INFORMAZIONE AZIENDALE**

**PRIMO APPELLO SESSIONE AUTUNNALE**

**TESTO E SOLUZIONE**

**Schema ER utilizzato nelle domande e negli esercizi**

****

**Fig. 1** Diagramma Entità Relazioni

In figura 1 è riportato il diagramma Entità Relazioni (ER) di un DBR relativo ad un Sistema Informativo per la gestione di una compagnia aerea; in particolare:

* la tabella TIPI\_AEREI definisce le differenti tipologie di aerei (es. Boeing 747, Airbus 320, ecc.),
* la tabella FLOTTA definisce gli aerei che formano la flotta della compagnia,
* la tabella VOLI definisce tutte le tratte coperte dalla compagnia. Si tratta di tutte le coppie di aeroporti (Aeroporto\_1 e Aeroporto\_2) che sono collegate da un volo diretto. Si noti che tale tabella definisce il grafo della rete i cui nodi sono gli aeroporti e i cui archi sono i voli.
* la tabella AEREI\_VOLI è una tabella ponte che definisce quali aerei sono assegnati a quali tratte. Uno stesso aereo, infatti, potrebbe essere utilizzato per coprire più tratte, così come una stessa tratta potrebbe essere coperta da più aerei. In questa tabella viene anche specificato, mediante il campo Voli\_Giorno, il numero di volte che un aereo percorre una certa tratta in andata e ritorno.
* Le chiavi primarie FLOTTA.ID e VOLO.ID sono alfanumerici (es. A1 sta per Aereo\_1 e L1 sta per Linea\_1)

**Domanda 1.**

Si spieghi cosa s’intende per fatto di un Data Warehouse. A tal fine si usi come esempio un fatto di manutenzione contraddistinto da almeno 2 dimensioni (entrambe con gerarchia) ed almeno 3 metriche. Si concluda disegnandone il relativo diagramma DFM.

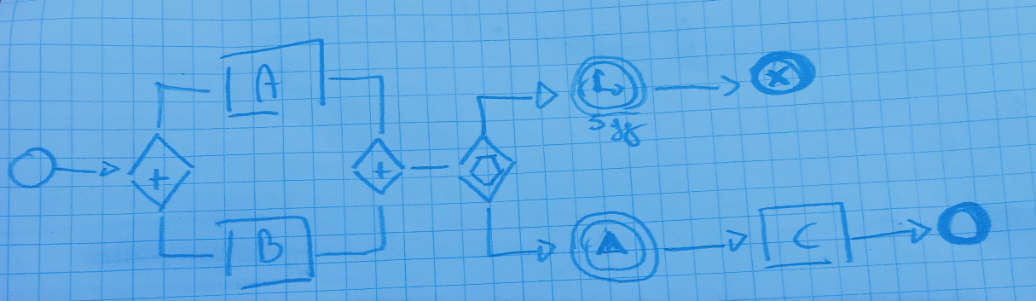
**SOLUZIONE**

Per la soluzione si vedano le dispense.

**Domanda 2.**

Si spieghi la differenza tra un gateway parallelo ed un gateway ad eventi (nella notazione BPMN 2.0). A tal fine si faccia riferimento ad un processo composto di tre attività, rispettivamente A, B e C. In particolare, A e B vengono svolte in parallelo e, se entro 3 giorni dal loro completamento arriva un segnale di prosecuzione (emesso da un altro processo), si attiva l’attività C. Il processo termina in maniera standard non appena si conclude l’attività C, mentre termina rilasciando un segnale d’errore, se il segnale di prosecuzione non arriva entro 3 giorni dal completamento di A e di B.

**SOLUZIONE**



**Esercizio 1.**

* Si scriva una query SQL che restituisce il numero di km percorsi ogni giorno da ciascun aereo della flotta. Si spieghi, inoltre, come andrebbe modificata la query per limitatamente la ricerca ai soli aerei che coprono almeno una tratta superiore a 2000 km.
* Si supponga che per andare dall’aeroporto **A** all’aeroporto **H** sia necessario effettuare 2 scali, prendendo i seguenti voli: L1 (da **A** a **B**), L2 (da **B** a **G**) e L3 (da **G** a **H**). Si scriva una query SQL che restituisce il numero complessivo di km e il numero complessivo di ore di volo corrispondenti a questo viaggio.
* Si aggiungano (allo schema di figura 1) tutte le tabelle ritenute necessarie a registrare l’acquisto di biglietti. Si tenga presente che un biglietto può comprendere più voli, come nel caso precedente, in cui, per andare da **A** a **H** è necessario effettuare 2 scali. Si mostri, inoltre, come verrebbe registrato l’acquisto di un biglietto da **A** a **H**.

**SOLUZIONE**

SELECT FLOTTA.ID, SUM(Distanza\*2\*Voli\_Giorno) AS Km\_Giorno

FROM VOLI INNER JOIN (TIPI\_AEREI INNER JOIN (FLOTTA INNER JOIN AEREI\_VOLI ON FLOTTA.ID = AEREI\_VOLI.ID\_Aereo) ON TIPI\_AEREI.ID = FLOTTA.ID\_Tipo) ON VOLI.VOLO = AEREI\_VOLI.ID\_Volo

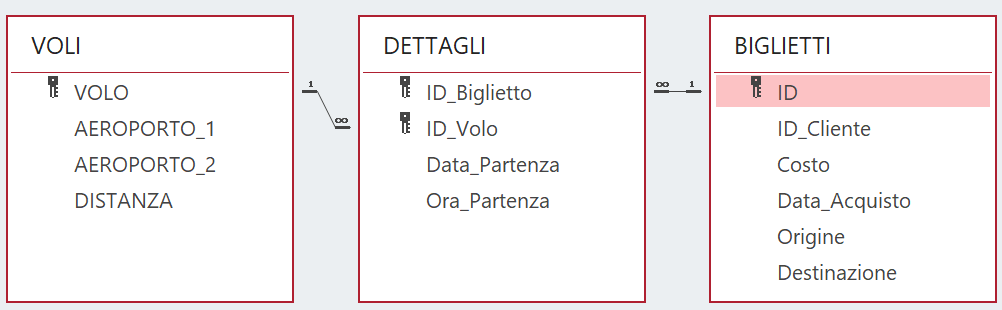
GROUP BY FLOTTA.ID

HAVING Max(Distanza) > 2000

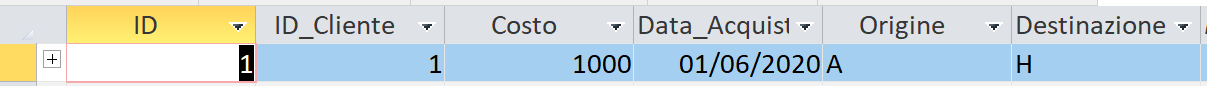
SELECT Sum(VOLI.Distanza) AS Distanza\_Tot, Sum(AEREI\_VOLI.Durata) AS Durata\_Tot

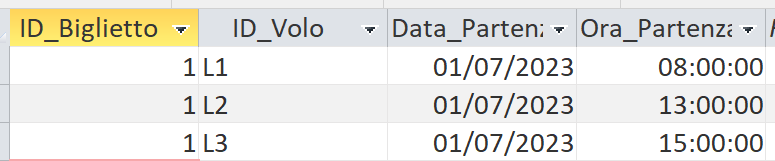
FROM VOLI INNER JOIN AEREI\_VOLI ON VOLI.VOLO = AEREI\_VOLI.ID\_Volo

WHERE VOLO In ('L1','L2','L3')



Ad esempio, un biglietto da A ad H verrebbe così registrato, rispettivamente nella tabella BIGLIETTI e nella tabella DETTAGLI.

****

****

**Esercizio 2. (4.5 punti)**

Si crei la funzione Descrivi\_Aereo(), che riceve in input l’ID di un aeroplano, e restituisce in output una stringa che ne indica la tipologia e il numero di km percorsi. Ad esempio, *Descrivi\_Aereo(“A1”)* -> Boeing 747, km totali 999876. Tale funzione deve essere basata su DFunction e, pertanto, non deve fare uso di Recordset.

**SOLUZIONE**

Public Function Descrivi\_Aereo(Id As String) As String

Dim ID\_Tipo As Integer

Dim Tipo As String

Dim Km As Long

ID\_Tipo = DLookup("ID\_Tipo", "FLOTTA", "ID = '" & Id & "'")

Tipo = DLookup("Tipo", "TIPI\_AEREI", "ID = " & ID\_Tipo)

Km = DLookup("Km\_Percorsi", "FLOTTA", "ID = '" & Id & "'")

Descrivi\_Aereo = "Tipo: " & Tipo & " Km Percorsi: " & Km

End Function

**Esercizio 3.**

Si scriva la funzione Nodi\_Contigui() che, ricevendo in input l’ID (o il nome) di un aeroporto di partenza, restituisce un vettore contenente l’ID di tutti gli aeroporti che sono ad esso connessi mediante un volo diretto.

***Suggerimento:*** si crei una query che restituisce tutti i voli (con origine e destinazione) che connettono l’aeroporto di partenza ad un altro aeroporto e si usi tale query per aprire un recordset. Si usi tale recordset per popolare il vettore contenente gli aeroporti contigui.

**SOLUZIONE**

Public Function Nodi\_Contigui(Aeroporto As String) As Variant

Dim SQL As String

Dim Nodi() As Variant

Dim Nodo As String

Dim Rcs As Recordset2

Dim N As Integer, I As Integer

SQL = "SELECT AEROPORTO\_1, AEROPORTO\_2 FROM VOLI "

SQL = SQL & "WHERE AEROPORTO\_1 = '" & Aeroporto & "' OR AEROPORTO\_2 = '" & Aeroporto & "'"

Set Rcs = CurrentDb.OpenRecordset(SQL)

Rcs.MoveLast

N = Rcs.RecordCount

ReDim Nodi(1 To N)

For I = 1 To N

Nodo = Rcs.Fields(0)

If Nodo = Aeroporto Then Nodo = Rcs.Fields(1)

Nodi(I) = Nodo

Rcs.MovePrevious

Next I

Nodi\_Contigui = Nodi

End Function

**Esercizio 4.**

Dopo aver disegnato il grafo corrispondente alla tabella VOLI (Tabella. 1), si spieghi a cosa servono le due funzioni seguenti e se ne dettagli il flusso logico di funzionamento. A tal fine, si consideri la chiamata *Trova\_Percorsi(“A”, “C”)*, e se ne indichi il risultato.

**Tab. 1.** La tabella VOLI

|  |  |  |  |
| --- | --- | --- | --- |
| **VOLO** | **AEROPORTO\_1** | **AEROPORTO\_2** | **DISTANZA** |
| L1 | A | B | 5000 |
| L2 | B | C | 3000 |
| L3 | B | D | 7000 |
| L4 | A | C | 2000 |

***Suggerimenti.***

* *Percors*o è la stringa che definirà tutti i nodi che definiscono un percorso tra un’origine ed una destinazione (es. da KXYZ definisce il percorso da K a Z passando per X e per Y).
* *Percorsi* è la stringa che definirà tutti i percorsi che connettono due punti (i percorsi, infatti possono essere più d’uno). I percorsi saranno separati da un opportuno carattere di separazione.
* La procedura *Trova()* opera in maniera ricorsiva, e crea i percorsi percorrendo il grafo a ritroso, a partire dal nodo di arrivo.

Public Function Trova\_Percorsi(Origine As String, Destinazione As String) As String

Dim Percorsi As String

Percorsi = “” 'Inizialmente vuoto, andrà costruito step by step

'Trova() è funzione ricorsiva che aggiorna la variabile Percorsi, si noti che Percorso = Destinazione, dato che

'il percorso viene creato a partire dalla destinazione

Call Trova(Origine, Destinazione, Destinazione, Percorsi)

Trova\_Percorsi = Mid(Percorsi, 2)

End Function

'Procedura ricorsiva usata dalla funzione pubblica Trova\_Percorsi()

Private Sub Trova(Origine As String, Destinazione As String, Percorso As String, Percorsi As String)

Dim Contigui As Variant, Nodo As Variant

Dim Nuovo\_Percorso As String, Nd As String

Contigui = Nodi\_Contigui(Destinazione) 'I nodi contigui ottenuti con la funzione dell’esercizio 3

For Each Nodo In Contigui

Nd = CStr(Nodo)

Nuovo\_Percorso = Nd & Percorso

If Nd = Origine Then

Percorsi = Percorsi & "\" & Nuovo\_Percorso 'Percorso trovato

Else 'Lancia la ricorsione se il nuovo nodo non è già presente

If InStr(1, Percorso, Nd) = 0 Then Call Trova\_Per(Origine, Nd, Nuovo\_Percorso, Percorsi)

End If

Next Nodo

End Sub

**SOLUZIONE**

La funzione Trova\_Percorsi(), sfrutta la procedura ricorsiva Trova() per generare una stringa che contiene tutti i possibili percorsi che connettono due nodi (aeroporti). Ciascun percorso è dato dalla sequenza dei nodi che connettono l’origine alla destinazione e i percorsi sono separati tra loro mediante il simbolo “\”. Si noti, infatti, che potrebbero esistere più percorsi (siano essi diretti o con scali) per connettere due nodi. Per cui, dal grafo corrispondente alla Tabella 1 (qui non riportato), è immediato capire che la funzione Trova\_Percorsi(“A”, “C”) restituirà la stringa AB\ABC. Esiste, infatti un volo diretto e un collegamento che richiede uno scalo intermedio.

Per spiegare questo fatto consideriamo la chiamata d’esempio Trova\_Percorsi(“A”, “C”), che al suo interno lancia Trova() con i seguenti argomenti: *Origine = A*, *Destinazione = C*, *Percorso = C*, *Percorsi = “”* (istruzione Call Trova(Origine, Destinazione, Destinazione, Percorsi)).

Inizialmente, quindi, il percorso contiene solo il nodo di destinazione, il nodo C. A questo punto, per ripercorrere a ritroso il grafo, vengono cercati tutti i nodi adiacenti a C (istruzione Contigui = Nodi\_Contigui(Destinazione)), che in questo caso sono A e B. Dato che A è il nodo di origine abbiamo già trovato una soluzione. AC rappresenta, infatti, la connessione diretta tra i due nodi. Tale percorso viene aggiunta alla stringa Percorsi che diventa \AC (istruzione If Nd = Origine Then Percorsi = Percorsi & "\" & Nuovo\_Percorso).

Non abbiamo però finito, dato che anche passando per B potrebbe essere possibile arrivare al nodo C. Viene quindi considerato il percorso BC rilanciando ricorsivamente Trova con i seguenti argomenti: Origine = A, Destinazione = B, Percorso = BC, Percorsi = \AC (istruzione If InStr(1, Percorso, Nd) = 0 Then Call Trova\_Per(Origine, Nd, Nuovo\_Percorso, Percorsi), si noti che InStr(1, Percorso, Nd) serve a verificare che il nuovo nodo non sia già presente nel percorso, questo creerebbe un ciclo cosa non ammessa che termina la ricorsione - condizione di escape -).

Si cercano le destinazioni contigue a B che sono A e B. Dato che A è il nodo di origine abbiamo trovato un altro collegamento, ossia ABC. La stringa percorsi diventa \AC\ABC. Non abbiamo ancora finito perché anche il percorso DBC potrebbe condurre ad A. Per cui lanciamo ancora la ricorsione. I nodi adiacenti a D sono soltanto B; tuttavia, dato che B è già incluso nel percorso DBC aggiungerlo non avrebbe senso, si otterrebbe un ciclo. Ecco che la ricorsione si arresta, non ha senso andare oltre.

La soluzione finale è allora la stringa AC\ABC che contiene i due soli percorsi che connettono A e C. Si noti che il primo carattere di separazione è stato eliminato tramite l’istruzione Trova\_Percorsi = Mid(Percorsi, 2), con cui termina la funzione Trova\_Percorsi().

**Esercizio Alternativo**

Si scriva una query parametrica, operante sul DBR di figura 1, che restituisce tutti i voli diretti che partono da un certo aeroporto (parametro della query). In particolare, la query deve generare una tabella con i seguenti campi: aeroporto di origine, aeroporto di destinazione, distanza, durata ID e Tipo dell’aereo che copre quella tratta.

**SOLUZIONE**

SELECT FLOTTA.ID, TIPI\_AEREI.Tipo, AEROPORTO\_1, AEROPORTO\_2, Distanza, Durata

FROM VOLI INNER JOIN ((TIPI\_AEREI INNER JOIN FLOTTA ON TIPI\_AEREI.ID = FLOTTA.ID\_Tipo) INNER JOIN AEREI\_VOLI ON FLOTTA.ID = AEREI\_VOLI.ID\_Aereo) ON VOLI.VOLO = AEREI\_VOLI.ID\_Volo

WHERE AEROPORTO\_1 = [Inserire origine];