INFORMATICA GENERALE

Andrea Pollini

Il tema d'esame proposto quest'anno consiste di quattro parti ben distinte. Il sistema descritto nella traccia è un classico sistema gestionale di una azienda che possiede filiali, definite nella traccia "Punti di Accettazione (PDA)", nei quali vengono eseguite delle operazioni, in questo caso la riparazione di apparecchi televisivi e telefonici. Vista la struttura naturalmente decentrata dell'architettura aziendale, ipotizziamo che sia già implementata una comunicazione di rete tra la sede centrale e le sedi periferiche. Supponiamo inoltre che tale comunicazione sia di tipo criptato così da garantire la sicurezza dei dati in transito sulla rete di comunicazione. Per quel che riguarda il controllo di accesso, si può ricorrere a sistemi di autenticazione esterni all'applicazione, quali ad esempio LDAP.
Fatte queste ipotesi, tra le varie architetture possibili per il sistema informativo che andiamo a descrivere, scegliamo una struttura di tipo *client-server*, ovvero il client effettua una richiesta al server che esegue l'operazione richiesta se il client possiede i diritti per effettuarla, e ritorna al client il risultato dell'esecuzione. L'applicazione potrebbe quindi essere una *web application*, che utilizza le tecnologie standard di internet per mostrare ai client, che saranno dei semplici *web browser,* la *gui* tramite la quale operare sui dati.

SCHEMA CONCETTUALE

La realtà che andiamo a descrivere è quella di un'azienda composta da più punti di accettazione, detti PDA. Visto che ogni PDA è univocamente determinato all'interno della nostra architettura e che in particolare nella query 5 viene richiesto di calcolare un aggregato per ogni PDA, questo fa propendere per considerare il PDA stesso un'entità del nostro schema concettuale. Essendo ogni PDA un negozio fisico, esso avrà una denominazione ed un indirizzo.
Altra entità è quella del *Ticket*, ovvero dell'intervento di riparazione. Ogni ticket contiene alcune informazioni, come il codice del PDA dove è stato creato, il nominativo del cliente, una mail o telefono del cliente, la data di creazione del ticket, modello, marca e tipologia dell'apparecchio, tipo di intervento richiesto, difetto riscontrato. Inoltre sarà presente un campo che descrive lo stato dell'intervento. Dalla traccia sembra a prima vista che siano sufficienti tre stati: APERTO, IN_RIPARAZIONE e CHIUSO. Tuttavia lo stato di CHIUSO rappresenta sia lo stato di chiuso e riparato che di chiuso e non riparato. Visto però che nella query 4 vanno indicate solo riparazioni andate a buon fine, dividiamo lo stato di CHIUSO in due; per cui gli stati saranno i seguenti

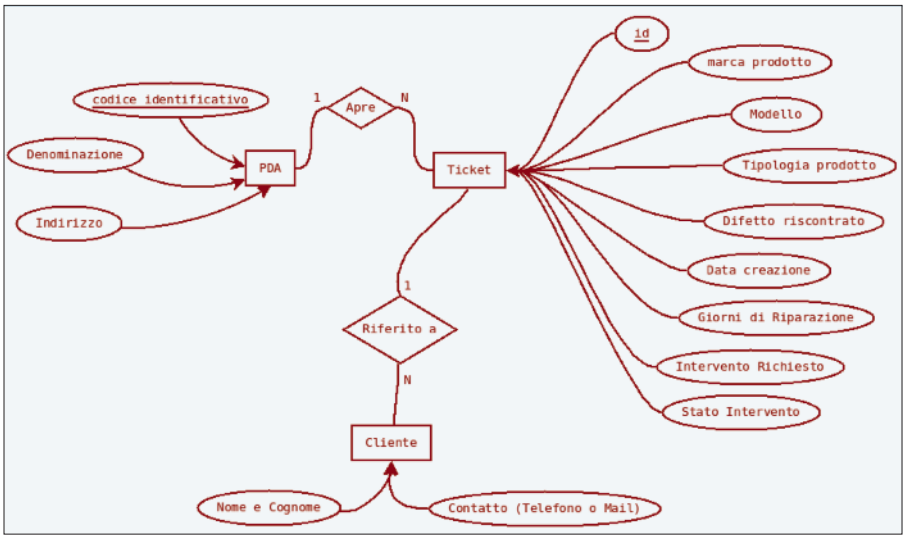
LA TRACCIA MINISTERIALE

INFORMATICA GENERALE,
APPLICAZIONI TECNICO-SCIENTIFICHE
INFORMATICA
(Testo valevole per i corsi di ordinamento e per i corsi sperimentali del Progetto "SIRIO")

Una Società di riparazione di apparecchi telefonici e televisivi desidera commissionare ad una software house la realizzazione di un sistema informativo per la gestione delle richieste di riparazione (apertura del TICKET di riparazione) degli articoli trattati da parte dei negozi specializzati (rivendite o riparatori, denominati PDA, Punti Di Accettazione).
Il processo di gestione prevede la memorizzazione di tutte le informazioni contenute nel modulo cartaceo utilizzato abitualmente dai PDA: codice identificativo dell'intervento (numero progressivo generato in automatico e non modificabile), codice del PDA, nominativo e recapito telefonico e/o e-mail del cliente, data di invio della richiesta di intervento, tipologia, marca e modello dell'apparecchio telefonico/televisivo, difetto o anomalia segnalata, ecc...
L'accesso al nuovo sistema informativo, verrà regolamentato mediante l'inserimento di username e password, in modo da garantire un accesso sicuro. Gli utenti potranno usufruire di alcune funzionalità: compilazione di un nuovo modulo di prenotazione on line, visualizzazione dello stato dei ticket (APERTO: il riparatore ha preso in carico la richiesta; IN_RIPARAZIONE: l'articolo è in riparazione; CHIUSO: l'articolo, riparato o non riparato, è pronto per la consegna al PDA), visualizzazione del tempo residuo stimato per la conclusione dell'intervento stesso.
Il candidato, formulate le opportune ipotesi aggiuntive, realizzi:
• Una analisi della realtà di riferimento, completa dello schema funzionale dell'architettura proposta.
• Uno schema concettuale ed uno schema logico del data base.
• La definizione delle relazioni e le seguenti interrogazioni espresse in linguaggio SQL:
1. visualizzare l'elenco, in ordine cronologico, di tutti gli interventi di riparazione di telefoni cellulari del tipo "sostituzione display" effettuati nell'arco di un mese;
2. visualizzare i dati dei clienti ai quali, dopo trenta giorni, non è stato ancora riparato l'articolo consegnato;
3. data una marca ed un modello, visualizzare la durata media degli interventi di riparazione per i prodotti di tale marca e modello;
4. calcolare e visualizzare quanti interventi di riparazione, andati a buon fine, sono stati effettuati, suddivisi per marca, nell'arco di un anno;
5. calcolare e visualizzare quanti TICKET sono stati compilati da ciascun PDA e la durata media di lavorazione dei TICKET;
6. dato il codice identificativo di un intervento, visualizzarne lo stato.
• La codifica, in un linguaggio di programmazione per il web, di un segmento significativo del progetto realizzato.

• APERTO
• IN_LAVORAZIONE
• CHIUSO_RIPARATO
• CHIUSO_NON_RIPARATO.

Il campo giorni di riparazione, nel caso di un ticket in lavorazione, contiene una stima del tempo di riparazione.
Le relazioni che fanno parte del diagramma concettuale del sistema informativo che stiamo modellizzando sono la relazione *Apre* che descrive l'operazione di apertura di un ticket che è sempre opera di un PDA e la relazione *Riferito a* che rappresenta la relazione esistente tra un *Ticket* e il *Cliente* che possiede l'apparecchio in riparazione cui fanno riferimento i dati contenuti nel *Ticket* stesso.



Schema logico del Database

Nella creazione del modello logico partiamo da quanto ottenuto dal modello concettuale, ma introduciamo ulteriori informazioni quali ad esempio il tipo di dato (numerico, stringa) relativo ad ogni attributo ed eventuali vincoli aggiuntivi.
La prima relazione che andiamo ad analizzare è *PDA*. Questa relazione verrà tradotta nello schema logico con la tabella *pda*. Il codice identificativo presente nello schema concettuale sarà denominata *id*, come per convenzione si è soliti fare nella progettazione degli schemi logici di un'applicazione.
Indirizzo e denominazione del PDA saranno campi stringa, identificati dal tipo dati SQL VARCHAR con una dimensione ragionevolmente grande. La definizione della tabella *pda* sarà la seguente:

TABELLA	***pda***	
Campo	**Tipo Dato**	**Vincoli**
id	SERIAL	NOT NULL PRIMARY KEY
denominazione	VARCHAR(120)	NOT NULL
indirizzo	VARCHAR(120)	NOT NULL

Il campo *id* ha SERIAL come tipo dati. Tale tipologia di dato non è altro che un intero autoincrementante, che rafforza il vincolo PRIMARY KEY. Tale vincolo impone che i valori del campo *id* siano tutti distinti, essendo la chiave primaria del record, ovvero l'informazione che consente di identificare il record in modo univoco.

TABELLA	***clienti***	
Campo	**Tipo Dato**	**Vincoli**
id	SERIAL	NOT NULL PRIMARY KEY
nominativo	VARCHAR(120)	NOT NULL
contatto	VARCHAR(120)	NOT NULL

L'entità *Cliente* viene mappata nello schema logico del database nella tabella *clienti* (*v. sopra*).
Da notare come il campo contatto possa contenere o la mail o il numero di telefono indistintamente ma essendo una di queste due informazioni un campo obbligatorio, andremo ad imporre un vincolo di tipo NOT NULL per rafforzare questa richiesta.
L'ultima entità da inserire nel modello logico del database è quella che fa riferimento all'entità *Ticket* e che chiameremo *ticket*. Non utilizziamo il nome maiuscolo perchè questo nella realizzazione del codice SQL potrebbe creare problemi su alcuni tipi di database.
La definizione della tabella ticket, seguendo le informazioni che abbiamo definito nello schema concettuale, risulta essere:

TABELLA	***ticket***	
Campo	**Tipo Dato**	**Vincoli**
id	SERIAL	NOT NULL PRIMARY KEY
pda_id	INTEGER	NOT NULL REFERENCES (pda.id)
cliente_id	INTEGER	NOT NULL REFERENCES (clienti.id)
marca	VARCHAR(120)	NOT NULL
modello	VARCHAR(120)	NOT NULL
tipologia	VARCHAR(120)	NOT NULL
difetto	VARCHAR(120)	NOT NULL
intervento	VARCHAR(120)	NOT NULL
data_creazione	DATE	DEFAULT TODAY()
giorni_riparazione	INTEGER	NOT NULL DEFAULT 0
stato	VARCHAR(120)	CHECK IN('APERTO','IN_LAVORAZIONE','CHIUSO_RIPARATO','CHIUSO_NON_RIPARATO')

I campi *pda_id* e *client_id* rappresentano i riferimenti alle tabella *pda* e *clienti* secondo quanto definito dalle relazioni presenti nello schema concettuale. I vincoli di tipo REFERENCES rafforzano il fatto che ogni ticket debba far riferimento a pda e clienti esistenti nel database.
Da notare che il campo *data_creazione* ha come tipo dati DATE, e tale data è posta di default alla data attuale di creazione del ticket.
Nel campo stato poi viene fatto un controllo con la clausola CHECK IN che il campo contenga un tipo di stato tra quelli definiti nello schema concettuale.

Interrogazioni SQL

Prima query

La prima interrogazione richiesta è abbastanza complessa. Dobbiamo selezionare «l'elenco, in ordine cronologico, di tutti gli interventi di riparazione di telefoni cellulari del tipo "sostituzione display" effettuati nell'arco di un mese». Ogni database mette a disposizione delle funzioni per calcolare la differenza tra due date. Negli esempi riportati utilizziamo le funzioni di MySQL (*http://www.mysql.org*), database liberamente utilizzabile dagli studenti. Useremo la funzione DATEDIFF che ritorna la differenza in gironi tra due date. Una data sarà quella memorizzata nella base dati, contenente la data di apertura del ticket. La seconda sarà quella scelta dall'utente per calcolare l'inizio del mese di interesse (30 giorni) sommata al numero di giorni necessari alla riparazione, per cui fa fede la data di chiusura del ticket, che possiamo recuperare con questa operazione.
La query finale sarà quindi la seguente:

```
SELECT  *
FROM ticket
WHERE
        DATEDIFF(data_creazione + INTERVAL giorni_riparazione DAY,<data-selezionata>) < 30     AND
        tipologia='cellulare'
        AND intervento='sostituzione display'
        AND stato = 'CHIUSO_RIPARATO'
ORDER BY data_creazione ASC
```

Seconda query

Dobbiamo quindi recuperare le informazioni relative al cliente partendo dal ticket, utilizzando la chiave esterna client_id. Quindi imponiamo che lo stato sia o 'APERTO' o 'IN_LAVORAZIONE' e la differenza tra la creazione del ticket e la data attuale sia maggiore di 30. Aggiungiamo una clausola DISTINCT per evitare doppioni di nominativi cliente nel caso in cui ci siano più ticket aperti riferiti allo stesso cliente.

```
SELECT DISTINCT clienti.nominativo,clienti.contatto FROM clienti, ticket
WHERE
   ticket.cliente_id = clienti.id
        AND
        DATEDIFF(data_creazione,NOW()) > 30
        AND  (stato ='APERTO' OR stato='IN_LAVORAZIONE')
```

Assumiamo che l'espressione 'dopo 30 giorni' indichi il fatto che il ticket è aperto ed è più vecchio di 30 giorni. Intendo invece questa espressione come 'ticket che sono stati aperti per più di 30 giorni' dovremmo selezionare oltre a quelli indicati anche i ticket chiusi che hanno valori maggiori di 30 per quel che riguarda il campo giorni_riparazione.

Terza query

Dovremo selezionare il campo giorni_riparazione tra i ticket chiusi per una marca ed un modello, e poi aggregare questo dato utilizzando la funzione SQL AVG(). La query risultante sarà'CHIUSO_RIPARATO' or stato =

```
SELECT
        AVG(giorni_riparazione)
FROM
        ticket
WHERE
        marca=<marca>
        AND
        modello=<modello>
        AND
        ( stato = 'CHIUSO_RIPARATO' or stato = 'CHIUSO_NON_RIPARATO')
```

Quarta query

La funzione YEAR() ritorna l'anno del campo passato come parametro.

```
SELECT
        COUNT(*),marca
FROM
        ticket
WHERE
        stato = 'CHIUSO_RIPARATO'
        AND
        YEAR(data_creazione) = <anno>
GROUP BY marca
```

Quinta query

Selezioniamo i ticket chiusi da ogni PDA, poi aggiungiamo le funzioni di aggregazione COUNT() e AVG() come già visto per ottenere le informazioni che servono:

```
SELECT
        pda.denominazione, COUNT(ticket.giorni_riparazione), AVG(ticket.giorni_riparazione)
FROM pda,  ticket
WHERE
        ticket.pda_id = pda.id
        AND
        stato = 'CHIUSO_RIPARATO'
GROUP BY pda.denominazione
```

Sesta query
Questa query è una semplicissima SELECT fatta sul campo stato del ticket che ha come id il valore scelto. Per cui otteniamo la query

```
SELECT stato
FROM ticket
WHERE ticket.id = <id>
```

Implementazione di una porzione di progetto
Per implementare il nostro progetto con un linguaggio per il web, adottiamo la tecnica di progettazione MVC, Model, View e Controller, che ben rispecchia quanto già descritto per la struttura generale dell'applicazione. Le funzioni che fanno parte del *model* si occupano di interagire con la nostra base dati e ritornano i record al *controller*, che poi li passa alla *view* corrispondente, che produce la pagina web visualizzata dall'utente. Nel controller vengono analizzati i parametri passati in input dall'utente per le query parametrizzate ed è memorizzata la logica di funzionamento dell'operazione richiesta, la *business logic*.
Ad esempio la funzione del *model* che gestisce l'ultima query avrà un parametro selezionato dall'utente, che sarà quello relativo al numero del ticket di cui visualizzare a video lo stato.
La funzione avrà questa struttura:

```
def stato_ticket(id_ticket):
        if not id_ticket: #se id_ticket non esiste
                raise Error,'Numero ticket scorretto'
        else:
                stato = query( "SELECT stato FROM ticket
                                WHERE ticket.id = %d" ,id_ticket)
                return stato # ritorna lo stato
```

la *view* corrispondente è la seguente:

```
def view_stato(ticket_info):
        print  "<html><body>Lo stato del ticket %d è %s</body></html>"  %
(ticket_info['id'],ticket_info['stato'])
```

Il parametro *ticket_info* è un array associativo contente relativamente alla chiave '*id'* l'identificativo del ticket di cui visualizzare lo stato e nella chiave '*stato*' l'informazione relativa allo stato recuperata dal *model*.
Il *controller* relativo effettua come prima operazione un controllo sulle credenziali utente, per verificare che sia un operatore autenticato. Per fare questo, controlla la presenza di token di autenticazione in una funzione *controlla_login()*. Nel caso non sia presente un token valido, reindirizza alla pagina di login, dove l'utente inserisce le proprie credenziali di accesso. La password sul server sarà mantenuta criptata attraverso un meccanismo crittografico di hashing forte come SHA1.

```
def controller_visualizza_stato(ticket_id):
        if not controlla_login():
                redirect_to('login')
        stato =  stato_ticket(ticket_id)

        ticket_info = {}
        ticket_info['id'] = ticket_id
        tcket_indo['stato'] = stato
        view_stato(ticket_info)
```

Sebbene a prima vista questo modo di scomporre il problema possa sembrare complesso, in un progetto reale questa separazione tra le diverse componenti dell'applicazione risulta essenziale per uno sviluppo corretto e facilmente testabile ed espandibile della stessa.

In prospettiva

Il quesito di quest'anno consente allo studente di risolvere il problema assegnato in modo agevole, senza particolari intoppi. La parte più complessa è la prima query assegnata, di difficoltà superiore alla media delle richieste degli anni precedenti, anche per il fatto che non è formulata in modo troppo chiaro, lasciando aperte alcune possibilità interpretative. La suddivisione della traccia, simile a quella già assegnata altri anni, segue quello che è il percorso logico che porta dall'analisi del problema alla formulazione di una possibile soluzione. Interessante l'accenno a considerazioni di sicurezza, che inducono lo studente a considerare l'autenticazione come componente dell'architettura dell'applicazione.

Andrea Pollini
Università Cattolica del Sacro Cuore, Brescia