1. Abstract

Il progetto si concentra sulla progettazione di una base di dati per il Centro Benessere Loto Bianco, una struttura dedicata alla cura e al benessere della persona. Il centro offre trattamenti estetici e terapeutici personalizzati, rivolti sia al viso sia al corpo.

Una delle peculiarità della struttura è la presenza di un'esclusiva area termale, accessibile unicamente ai clienti abbonati. Tale area viene gestita e mantenuta da personale specializzato che si occupa delle regolari manutenzioni.

Tutti i clienti del centro, siano essi abbonati o meno, hanno la possibilità di prenotare trattamenti specifici. I trattamenti vengono realizzati utilizzando prodotti professionali, forniti da aziende esterne selezionate, la cui gestione è tracciata all'interno del sistema informativo.

Il personale del centro si suddivide in tre principali categorie operative:

- Gli specialisti dei trattamenti, che eseguono le prestazioni richieste;
- Gli addetti all'area termale, responsabili della manutenzione e del corretto funzionamento degli spazi dedicati;
- Il personale di segreteria, incaricato di ordinare i prodotti dai fornitori e rilasciare le fatture ai clienti.

La base di dati è progettata per supportare la gestione di tutte queste attività, assicurando un'organizzazione efficiente delle informazioni riguardanti i clienti, trattamenti, dipendenti, fornitori e operazioni amministrative.

2. Analisi dei Requisiti

Il database registrerà informazioni relative a clienti, dipendenti, aziende fornitrici, trattamenti offerti e fatture emesse.

Clienti e Abbonati

Un cliente che si reca al centro benessere è caratterizzato da:

- Codice Fiscale (identificatore univoco)
- Nome
- Cognome
- Numero di telefono
- Indirizzo (composto da città, via e CAP)

I clienti possono sottoscrivere un abbonamento per accedere all'area termale esclusiva. Un cliente abbonato è caratterizzato inoltre da:

- Numero carta (identificativo dell'abbonamento)
- Data di iscrizione

Trattamenti

Il cliente può prenotare diversi tipi di trattamenti, che sono caratterizzati da:

- Codice (identificatore univoco)
- Prezzo
- Data
- Ora

I trattamenti si suddividono in due categorie:

- Pulizia viso, caratterizzata dalla tipologia di pelle
- Massaggio, caratterizzato dalla zona del corpo trattata e dalla durata (in minuti)

Personale e Dipendenti

Nel centro benessere lavorano dipendenti caratterizzati da:

- Codice Fiscale (identificatore univoco)
- Nome
- Cognome
- Telefono
- Stipendio

I dipendenti si suddividono in tre categorie:

1. Specialisti dei trattamenti, che hanno un titolo professionale
2. Segretari, che gestiscono le prenotazioni e le fatturazioni
3. Addetti all'area termale, che hanno turni di lavoro specifici e si occupano della manutenzione delle aree

Prodotti e Fornitori

Per effettuare i trattamenti, gli specialisti utilizzano prodotti specifici caratterizzati da:

- Marca
- Nome
- Ingredienti
- Data di scadenza

I prodotti vengono forniti da aziende selezionate dal centro, caratterizzate da:

- Partita IVA (identificatore univoco)
- Nome
- Sede

Area Termale

L'area termale è caratterizzata da:

- ID area (identificatore univoco)
- Orario di apertura
- Orario di chiusura

Fatture

Le fatture vengono emesse dai segretari ai clienti che usufruiscono dei servizi e sono caratterizzate da:

- Numero (identificatore univoco)
- Costo del trattamento
- Modalità di pagamento

3. Progettazione Concettuale

3.1 Schema Entità-Relazione (E-R)

Lo schema E-R si compone delle seguenti entità e relazioni:

Entità principali:

- **Cliente**: rappresenta chi usufruisce dei servizi del centro
- **Abbonato**: sottotipo di Cliente che ha accesso all'area termale
- **Trattamento**: rappresenta un servizio offerto dal centro
- **Massaggio e Pulizia_viso**: sottotipi di Trattamento
- **Dipendente**: rappresenta il personale del centro
- **Specialista, Segretario, Addetto_at**: sottotipi di Dipendente
- **Prodotto**: rappresenta i prodotti utilizzati nei trattamenti
- **Azienda**: rappresenta i fornitori dei prodotti
- **Area_termale**: rappresenta le zone termali del centro
- **Fattura**: rappresenta le ricevute di pagamento

Relazioni principali:

- **Prenotazione**: collega Cliente e Trattamento
- **Associato**: collega Cliente e Fattura

- **Rilascio**: collega Fattura e Segretario
- **Pagamento**: collega Fattura e Trattamento
- **Utilizzo**: collega Trattamento e Prodotto
- **Ordinazione**: collega Prodotto e Segretario
- **Fornitura**: collega Prodotto e Azienda
- **Esecuzione**: collega Trattamento e Specialista
- **Accesso**: collega Abbonato e Area_termale
- **Manutenzione**: collega Area_termale e Addetto_at

Generalizzazioni:

- **Cliente** ⇐ **Abbonato** (generalizzazione parziale)
- **Trattamento** ⇐ **Massaggio, Pulizia_viso** (generalizzazione totale ed esclusiva)
- **Dipendente** ⇐ **Specialista, Segretario, Addetto_at** (generalizzazione totale ed esclusiva)

3.2 Valutazione formale dello schema E-R

Lo schema E-R proposto presenta una struttura logica e coerente che rispecchia adeguatamente i requisiti del centro benessere:

1. **Correttezza delle entità**: Le entità identificate coprono tutti gli aspetti rilevanti del dominio applicativo.
2. **Correttezza delle relazioni**: Le relazioni tra le entità sono adeguatamente definite e rispecchiano i processi operativi del centro benessere.
3. **Correttezza delle cardinalità**: Le cardinalità delle relazioni appaiono appropriate per il contesto descritto.
4. **Correttezza delle generalizzazioni**: Le generalizzazioni sono appropriate; in particolare, la generalizzazione parziale di Cliente in Abbonato è corretta poiché non tutti i clienti sono abbonati.
5. **Identificatori**: Gli identificatori scelti per ciascuna entità sono appropriati e garantiscono l'unicità delle istanze.

Tuttavia, si evidenziano alcune criticità minori:

- La relazione tra Fattura e Trattamento sembra avere una cardinalità non completamente chiara
- Non è chiaro se un cliente possa prenotare più volte lo stesso trattamento
- La relazione tra Specialista e Trattamento presenta cardinalità (1,1) che potrebbero essere troppo restrittive

4. Progettazione Logica

4.1 Analisi delle Ridondanze

L'attributo "totale" dell'entità "Fattura" rappresenta una ridondanza, in quanto potrebbe essere calcolato come somma dei prezzi dei trattamenti associati alla fattura stessa.

Analizziamo se conviene mantenere questa ridondanza considerando le operazioni più frequenti:

Operazione 1: Inserimento di un nuovo trattamento (1000 alla settimana) **Operazione 2**: Visualizzazione del totale di una fattura (100 alla settimana)

Volume dei dati:

- Fatture: 20.000 record
- Trattamenti: circa 100.000 record

Con ridondanza: Per l'operazione 1:

- 1 accesso in scrittura a Trattamento
- 1 accesso in lettura a Fattura
- 1 accesso in scrittura a Fattura Totale: (100 × 1) + (100 × 1) + (100 × 2) = 400 accessi giornalieri

Per l'operazione 2:

- 1 accesso in lettura a Fattura Totale: 10 × 1 = 10 accessi giornalieri

Costo settimanale totale: 400 × 7 + 10 × 7 = 2.870 accessi

Senza ridondanza: Per l'operazione 1:

- 1 accesso in scrittura a Trattamento Totale: 100 × 2 = 200 accessi giornalieri

Per l'operazione 2:

- Si assume che in media ci siano 5 trattamenti per fattura
- 5 accessi in lettura a Trattamento Totale: 10 × 5 = 50 accessi giornalieri

Costo settimanale totale: 200 × 7 + 50 × 7 = 1.750 accessi

Conclusione: L'eliminazione della ridondanza comporta una riduzione significativa degli accessi (2.870 vs 1.750). Pertanto, conviene rimuovere l'attributo "totale" dalla tabella Fattura e calcolarlo dinamicamente quando necessario.

4.2 Eliminazione delle Generalizzazioni

4.2.1 Generalizzazione Cliente-Abbonato

La generalizzazione parziale Cliente-Abbonato viene risolta introducendo una relazione "e_abb" tra Cliente e Abbonato, dove l'identificatore di Abbonato è la relazione con Cliente. Gli attributi specifici di Abbonato (numero_carta, data_iscrizione) vengono mantenuti nella sua entità.

4.2.2 Generalizzazione Trattamento

La generalizzazione totale ed esclusiva di Trattamento in Massaggio e Pulizia_viso viene risolta accorpando i figli nel padre e aggiungendo un attributo "tipo" per distinguere le due tipologie. Gli attributi specifici (tipologia per Pulizia_viso, zona e durata per Massaggio) diventano attributi opzionali di Trattamento.

4.2.3 Generalizzazione Dipendente

La generalizzazione totale ed esclusiva di Dipendente viene risolta introducendo le relazioni "e_spec", "e_seg" e "e_add" tra Dipendente e le entità figlie Specialista, Segretario e Addetto_at. Gli attributi specifici di ciascuna entità figlia vengono mantenuti nelle rispettive entità.

4.3 Schema Relazionale Finale

```
cliente(cf, nome, cognome, telefono, citta, via, cap)
abbonato(cliente, numero_carta, data_iscrizione)
    abbonato.cliente → cliente.cf

trattamento(codice, prezzo, data, ora, tipo, tipologia_pelle, zona, durata,
specialista, fattura)
    trattamento.specialista → dipendente.cf
    trattamento.fattura → fattura.numero

prodotto(codice, marca, nome, ingredienti, data_scadenza, segretario,
azienda)
    prodotto.segretario → dipendente.cf
    prodotto.azienda → azienda.piva

utilizzo(trattamento, prodotto)
    utilizzo.trattamento → trattamento.codice
    utilizzo.prodotto → prodotto.codice

azienda(piva, nome, sede)

dipendente(cf, nome, cognome, telefono, stipendio)

specialista(dipendente, titolo)
```

```
    specialista.dipendente → dipendente.cf

segretario(dipendente)
    segretario.dipendente → dipendente.cf

addetto_at(dipendente, turno, area)
    addetto_at.dipendente → dipendente.cf
    addetto_at.area → area_termale.id

area_termale(id, orario_apertura, orario_chiusura)

manutenzione(addetto, area_termale, data)
    manutenzione.addetto → addetto_at.dipendente
    manutenzione.area_termale → area_termale.id

accesso(abbonato, area_termale)
    accesso.abbonato → abbonato.cliente
    accesso.area_termale → area_termale.id

fattura(numero, mod_pagamento, cliente, segretario)
    fattura.cliente → cliente.cf
    fattura.segretario → segretario.dipendente
```

5. Query e Indici

5.1 Query Rilevanti

Query 1: Clienti che hanno ricevuto più di un trattamento con specialisti diversi nello stesso giorno

```sql
SELECT c.nome, c.cognome, t.data, COUNT(DISTINCT t.specialista) AS
specialisti_coinvolti
FROM cliente c
JOIN trattamento t ON c.cf = t.cliente
GROUP BY c.cf, c.nome, c.cognome, t.data
HAVING COUNT(DISTINCT t.specialista) > 1;
```

Query 2: Specialisti che hanno eseguito trattamenti utilizzando almeno 3 prodotti diversi in un solo appuntamento

```sql
SELECT d.nome, d.cognome, t.codice, COUNT(u.prodotto) AS num_prodotti
FROM dipendente d
JOIN specialista s ON d.cf = s.dipendente
```

```
JOIN trattamento t ON s.dipendente = t.specialista
JOIN utilizzo u ON t.codice = u.trattamento
GROUP BY d.cf, d.nome, d.cognome, t.codice
HAVING COUNT(u.prodotto) >= 3;
```

Query 3: Fatture con importi superiori alla media dei trattamenti dello stesso tipo

```
WITH media_tipi AS (
    SELECT tipo, AVG(prezzo) AS prezzo_medio
    FROM trattamento
    GROUP BY tipo
)
SELECT f.numero, c.nome, c.cognome, SUM(t.prezzo) AS totale, mt.prezzo_medio
FROM fattura f
JOIN cliente c ON f.cliente = c.cf
JOIN trattamento t ON f.numero = t.fattura
JOIN media_tipi mt ON t.tipo = mt.tipo
GROUP BY f.numero, c.nome, c.cognome, mt.prezzo_medio
HAVING SUM(t.prezzo) > mt.prezzo_medio;
```

Query 4: Prodotti utilizzati più frequentemente nei trattamenti di pulizia viso

```
SELECT p.nome, p.marca, COUNT(*) AS utilizzi
FROM prodotto p
JOIN utilizzo u ON p.codice = u.prodotto
JOIN trattamento t ON u.trattamento = t.codice
WHERE t.tipo = 'pulizia_viso'
GROUP BY p.codice, p.nome, p.marca
ORDER BY utilizzi DESC
LIMIT 10;
```

Query 5: Per ogni specialista, il giorno in cui ha effettuato il maggior numero di trattamenti

```
WITH trattamenti_per_giorno AS (
    SELECT
        d.cf,
        d.nome,
        d.cognome,
        t.data,
        COUNT(*) AS num_trattamenti,
        RANK() OVER(PARTITION BY d.cf ORDER BY COUNT(*) DESC) AS rank
    FROM dipendente d
```

```
    JOIN specialista s ON d.cf = s.dipendente
    JOIN trattamento t ON s.dipendente = t.specialista
    GROUP BY d.cf, d.nome, d.cognome, t.data
)
SELECT cf, nome, cognome, data, num_trattamenti
FROM trattamenti_per_giorno
WHERE rank = 1
ORDER BY num_trattamenti DESC;
```

5.2 Indici

Indice 1: Per ottimizzare le query sui trattamenti per data

```
CREATE INDEX idx_trattamento_data ON trattamento(data);
```

Questo indice migliora le prestazioni delle query che cercano trattamenti in date specifiche, come la Query 1.

Indice 2: Per ottimizzare le query su prodotti utilizzati nei trattamenti

```
CREATE INDEX idx_utilizzo_prodotto_trattamento ON utilizzo(prodotto,
trattamento);
```

Questo indice migliora le prestazioni delle query che analizzano quali prodotti sono stati utilizzati in quali trattamenti, come la Query 2 e 4.

Indice 3: Per ottimizzare le query sugli specialisti

```
CREATE INDEX idx_trattamento_specialista ON trattamento(specialista, data);
```

Questo indice migliora le prestazioni delle query che analizzano i trattamenti eseguiti da uno specialista in date specifiche, come la Query 5.

6. Bozza del Software C per l'Accesso al Database

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <libpq-fe.h>

// Definizione delle costanti di connessione
#define HOST "localhost"
#define PORT "5432"
#define DB "centro_benessere"
```

```c
#define USER "postgres"
#define PASS "password"

// Prototipi delle funzioni
void printMenu();
void executeQuery(PGconn *conn, PGresult **res, const char *query);
void printTable(PGresult *res);
void getInputString(const char *prompt, char *input, int size);
void getInputDate(const char *prompt, char *date);

int main() {
    // Connessione al database
    char conninfo[256];
    sprintf(conninfo, "host=%s port=%s dbname=%s user=%s password=%s",
            HOST, PORT, DB, USER, PASS);

    PGconn *conn = PQconnectdb(conninfo);

    if (PQstatus(conn) != CONNECTION_OK) {
        fprintf(stderr, "Errore di connessione: %s\n",
PQerrorMessage(conn));
        PQfinish(conn);
        exit(1);
    }

    printf("Connessione al database stabilita con successo.\n");

    int choice = 0;
    PGresult *res = NULL;
    char query[1024];
    char input[100];
    char date[11];

    do {
        printMenu();
        printf("Inserisci la tua scelta: ");
        scanf("%d", &choice);
        getchar(); // Consuma il newline

        switch (choice) {
            case 1:
                // Clienti con più trattamenti nello stesso giorno
                strcpy(query, "SELECT c.nome, c.cognome, t.data,
COUNT(DISTINCT t.specialista) AS specialisti_coinvolti "
                              "FROM cliente c "
                              "JOIN trattamento t ON c.cf = t.cliente "
                              "GROUP BY c.cf, c.nome, c.cognome, t.data "
                              "HAVING COUNT(DISTINCT t.specialista) > 1;");
                executeQuery(conn, &res, query);
                printTable(res);
```

```
                PQclear(res);
                break;

            case 2:
                // Specialisti che hanno eseguito trattamenti con almeno 3
prodotti
                strcpy(query, "SELECT d.nome, d.cognome, t.codice,
COUNT(u.prodotto) AS num_prodotti "
                              "FROM dipendente d "
                              "JOIN specialista s ON d.cf = s.dipendente "
                              "JOIN trattamento t ON s.dipendente =
t.specialista "
                              "JOIN utilizzo u ON t.codice = u.trattamento "
                              "GROUP BY d.cf, d.nome, d.cognome, t.codice "
                              "HAVING COUNT(u.prodotto) >= 3;");
                executeQuery(conn, &res, query);
                printTable(res);
                PQclear(res);
                break;

            case 3:
                // Prodotti più utilizzati in una data specifica
                getInputDate("Inserisci la data (YYYY-MM-DD): ", date);
                sprintf(query, "SELECT p.nome, p.marca, COUNT(*) AS utilizzi
"
                               "FROM prodotto p "
                               "JOIN utilizzo u ON p.codice = u.prodotto "
                               "JOIN trattamento t ON u.trattamento =
t.codice "
                               "WHERE t.data = '%s' "
                               "GROUP BY p.codice, p.nome, p.marca "
                               "ORDER BY utilizzi DESC "
                               "LIMIT 10;", date);
                executeQuery(conn, &res, query);
                printTable(res);
                PQclear(res);
                break;

            case 4:
                // Fatturato giornaliero
                getInputDate("Inserisci la data (YYYY-MM-DD): ", date);
                sprintf(query, "SELECT t.data, SUM(t.prezzo) AS
fatturato_totale "
                               "FROM trattamento t "
                               "WHERE t.data = '%s' "
                               "GROUP BY t.data;", date);
                executeQuery(conn, &res, query);
                printTable(res);
                PQclear(res);
                break;
```

```
            case 5:
                // Informazioni su un cliente specifico
                getInputString("Inserisci il codice fiscale del cliente: ",
input, 17);
                sprintf(query, "SELECT c.nome, c.cognome, c.telefono, "
                               "(SELECT COUNT(*) FROM trattamento t WHERE
t.cliente = c.cf) AS num_trattamenti, "
                               "CASE WHEN a.cliente IS NOT NULL THEN 'Sì'
ELSE 'No' END AS abbonato "
                               "FROM cliente c "
                               "LEFT JOIN abbonato a ON c.cf = a.cliente "
                               "WHERE c.cf = '%s';", input);
                executeQuery(conn, &res, query);
                printTable(res);
                PQclear(res);
                break;

            case 0:
                printf("Uscita dal programma...\n");
                break;

            default:
                printf("Scelta non valida. Riprova.\n");
        }

        printf("\nPremi INVIO per continuare...");
        getchar();

    } while (choice != 0);

    // Chiusura della connessione
    PQfinish(conn);
    return 0;
}

void printMenu() {
    printf("\n===== SISTEMA GESTIONE CENTRO BENESSERE =====\n");
    printf("1. Visualizza clienti con più trattamenti nello stesso
giorno\n");
    printf("2. Visualizza specialisti con almeno 3 prodotti utilizzati\n");
    printf("3. Visualizza prodotti più utilizzati in una data specifica\n");
    printf("4. Visualizza fatturato giornaliero\n");
    printf("5. Visualizza informazioni su un cliente\n");
    printf("0. Esci\n");
}

void executeQuery(PGconn *conn, PGresult **res, const char *query) {
    *res = PQexec(conn, query);
```

```c
    if (PQresultStatus(*res) != PGRES_TUPLES_OK && PQresultStatus(*res) !=
PGRES_COMMAND_OK) {
        fprintf(stderr, "Errore nell'esecuzione della query: %s\n",
PQerrorMessage(conn));
        PQclear(*res);
        PQfinish(conn);
        exit(1);
    }
}

void printTable(PGresult *res) {
    int rows = PQntuples(res);
    int cols = PQnfields(res);

    if (rows == 0) {
        printf("Nessun risultato trovato.\n");
        return;
    }

    // Stampa intestazioni colonne
    for (int i = 0; i < cols; i++) {
        printf("%-20s", PQfname(res, i));
    }
    printf("\n");

    // Stampa separatore
    for (int i = 0; i < cols; i++) {
        printf("--------------------");
    }
    printf("\n");

    // Stampa righe
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%-20s", PQgetvalue(res, i, j));
        }
        printf("\n");
    }

    printf("\nTotale risultati: %d\n", rows);
}

void getInputString(const char *prompt, char *input, int size) {
    printf("%s", prompt);
    fgets(input, size, stdin);

    // Rimuovi il newline finale
    input[strcspn(input, "\n")] = '\0';
}
```

```c
void getInputDate(const char *prompt, char *date) {
    printf("%s", prompt);
    fgets(date, 11, stdin);

    // Rimuovi il newline finale
    date[strcspn(date, "\n")] = '\0';
}
```

7. Conclusioni

Il progetto di basi di dati per il Centro Benessere Loto Bianco è stato sviluppato seguendo una metodologia rigorosa che ha portato alla definizione di uno schema concettuale coerente e di una struttura logica ottimizzata.

L'analisi delle ridondanze ha permesso di identificare e rimuovere dati superflui che avrebbero appesantito il sistema, privilegiando l'efficienza nelle operazioni più frequenti.

Lo schema relazionale finale rispecchia fedelmente i requisiti del dominio applicativo e garantisce:

- La corretta gestione dei clienti e dei loro abbonamenti
- L'amministrazione efficiente dei trattamenti offerti
- Il monitoraggio preciso dei prodotti utilizzati e dei loro fornitori
- La gestione del personale specializzato e delle relative competenze
- Il tracciamento delle operazioni di manutenzione dell'area termale
- La gestione accurata della fatturazione

Le query proposte coprono le principali esigenze informative del centro benessere, mentre gli indici creati ottimizzano le prestazioni del database per le operazioni più frequenti.

Il software di accesso al database fornisce un'interfaccia funzionale per le operazioni quotidiane, facilitando l'interazione con i dati da parte del personale del centro.