**Università degli Studi di Verona**

DIPARTIMENTO DI INFORMATICA
Corso di Laurea in Informatica

Basi di Dati

# Elaborato G33
# Sistema informativo per cartelle cliniche di una divisione ospedaliera

Candidati:
**Enrico Giordano**
**Matricola VR359169**

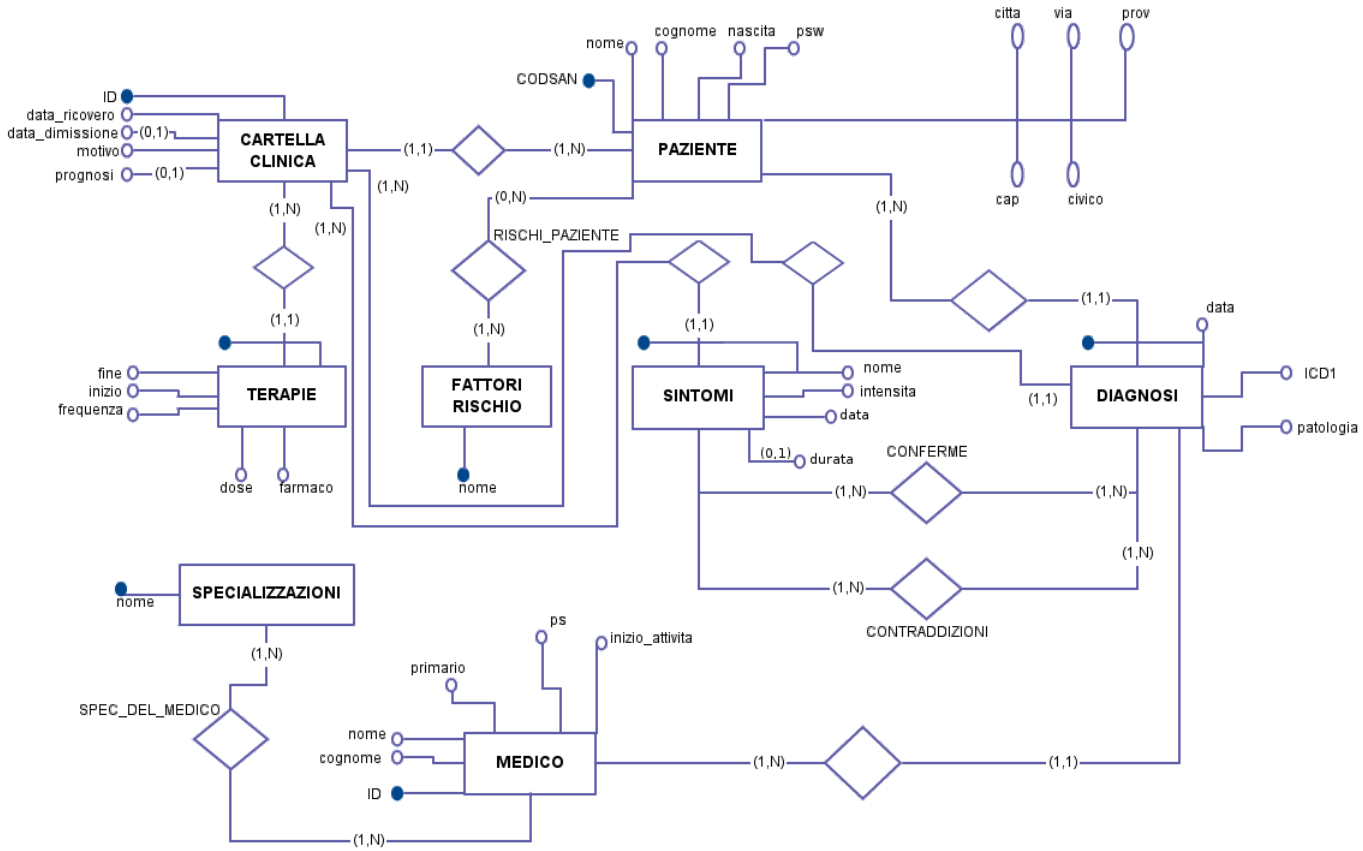
**Cristian Pinna**
**Matricola VR361121**

**Anno Accademico 2013-2014**

# Indice

Parte I

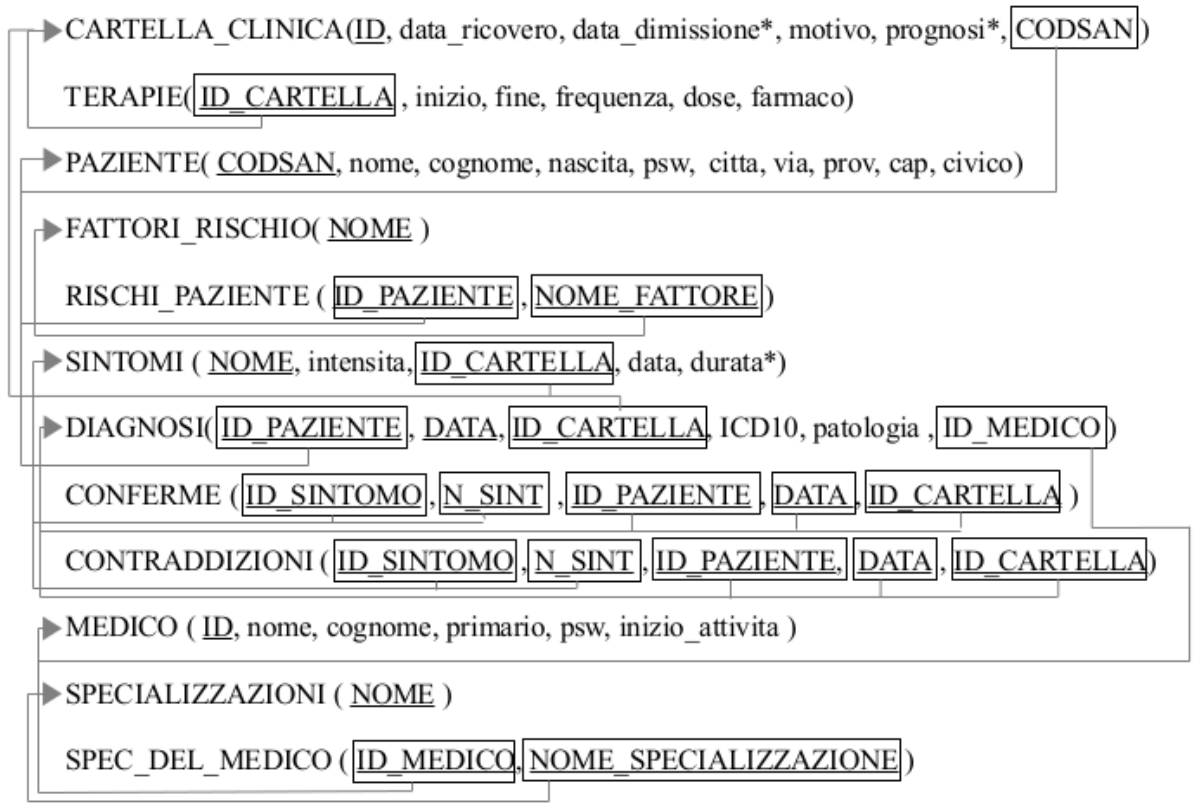
# Progettazione Concettuale



Elenco delle relazioni:

1. Relazione TERAPIE - CARTELLA CLINICA:
   - Cardinalità (1,1), una TERAPIA è associata univocamente ad una CARTELLA CLINICA
   - Cardinalità (1,N), ad una CARTELLA CLINICA può corrispondere più TERAPIE
2. Relazione CARTELLA CLINICA - PAZIENTE:
   - Cardinalità (1,1), una CARTELLA CLINICA è associata univocamente ad un PAZIENTE
   - Cardinalità (1,N), ad un PAZIENTE può corrispondere più CARTELLE CLINICHE
3. Relazione CARTELLA CLINICA - SINTOMI:
   - Cardinalità (1,N), ad una CARTELLA CLINICA può corrispondere uno o più SINTOMI
   - Cardinalità (1,1), un SINTOMO è associato univocamente ad una CARTELLA CLINICA
4. Relazione CARTELLA CLINICA - DIAGNOSI:
   - Cardinalità (1,N), ad una CARTELLA CLINICA può corrispondere una o più DIAGNOSI
   - Cardinalità (1,1), una DIAGNOSI è associata univocamente ad una CARTELLA CLINICA
5. Relazione PAZIENTE - FATTORI RISCHIO:
   - Cardinalità (0,N), ad un PAZIENTE può corrispondere nessuno o più FATTORI RISCHIO
   - Cardinalità (1,N), ad un FATTORE RISCHIO può corrispondere uno o più PAZIENTI
6. Relazione PAZIENTE - DIAGNOSI:
   - Cardinalità (1,N), ad un PAZIENTE può corrispondere una o più DIAGNOSI
   - Cardinalità (1,1), una DIAGNOSI è associata univocamente ad un PAZIENTE
7. Relazione SINTOMI - DIAGNOSI:
   - Cardinalità (1,N), ad un SINTOMO può corrispondere una o più DIAGNOSI
   - Cardinalità (1,N), ad una DIAGNOSI può corrispondere uno o più SINTOMI
8. Relazione DIAGNOSI - MEDICO:
   - Cardinalità (1,1), una DIAGNOSI è associata univocamente ad un MEDICO
   - Cardinalità (1,N), ad un MEDICO può corrispondere una o più DIAGNOSI
9. Relazione MEDICO - SPECIALIZZAZIONI:
   - Cardinalità (1,N), ad un MEDICO può corrispondere una o più SPECIALIZZAZIONI
   - Cardinalità (1,N), ad una SPECIALIZZAZIONE può corrispondere uno o più MEDICI

## Parte II
# Schema Logico




Questo schema logico rappresenta una visione globale sull'elenco dei vari attributi, sulle chiavi primarie (attributi sottolineati) e sulle relazione tra di essi (i riquadri attorno al nome dell'attributo e la relativa freccia che punta alla relazione).

È stato tradotto nel file "database.sql" e popolato tramite il file "popola.sql". Quest'ultimo file esegue degli script che contengono molti insert per ogni tabella. Per generare questi script, sono stati creati dei programmi in grado di generare file .sql con un numero considerevole di insert in base al design del database, in modo da poter testare su grandi numeri il sito (e il comportamento del database).

Parte III

# Page Schema

**page-schema Homepage unique (**

informazioni: link (info, *InfoPage.jsp);
personale_medico: link (personale, *PersonalePage.jsp);
patologie: link (patologie, *PatologiePage.jsp);
login: link (login, *Login.html);

**);**

**page-schema InfoPage unique (**

primario: String;
informazioni: text;
foto: list_of(data[]);

**);**

**DB to page-schema InfoPage (**

primario:
```
select *
from medico as m
where m.primario = 'si';
```

**);**

**page-schema LoginPage unique (**

login_paziente: form(
login: text;
pw: password;
invia: submit();
);

login_medico: form(
login: text;
pw: password;
invia: submit();
);

**);**

**DB to page-schema LoginPage (**

login_cliente:

```
if(     select *
        from paziente as p
        where p.codsan = ?codsan?
        and p.psw = ?psw?
)
then *PazientePage else *LoginPage
end;
```

login_medico:

```
if(     select m.*
        from medico as m
        where m.id = ?id?
        and m.psw = ?psw?
)
then *DiagnosiPage else *LoginPage
end;
```

**);**

**page-schema PazientePage (**

```
dati: (
        codice_sanitario: string;
        nome: string;
        cognome: string;
        data_nascita: string;
        via: string;
        civico: string;
        );
cartelle_cliniche: list of(
        link(cartella clinica, *CartellaPage);
        );

fattori_rischio: string;
elenco_medici: list of(
        nome: string;
        cognome: string;
);
```

**);**

**DB to page-schema PazientePage (**

dati:
```
select p.*
from paziente as p
where p.codsan = ?codsan?
```

cartelle_cliniche:
```
select c.*
from cartella_clinica as c, paziente as p
where c.codsan = ?codsan?
and c.codsan = ?codsan?
```

);

fattori_rischio:
```
select r.*
from paziente as p, rischi_paziente as r
where p.codsan = (:codsan)
and p.codsan = r.id_paziente
```

elenco_medici:
```
select distinct m.
from paziente as p, medico as m, diagnosi as d
where p.codsan = ?codsan)?
and p.codsan = d.id_paziente
and m.id = d.id_medico
```

**);**

**page-schema CartellaPage (**

dati: (
- ID: string;
- dataRic: string;
- motivo: string;
- prognosi: string;

);

diagnosi: list of (
- medico: (
  - nome_medico: string;
  - cognome_medico: string;
- );
- data: date;
- patologia: string;
- icd10: string;
- conferme: list of (
  - sintomi: string;
  - );
- contraddizioni: list of (
  - sintomi: string;
  - );

);

terapie: list of (
farmaco: string;
dose: float;
posologia: string;
inizio cura: date;
fine cura: date;

);
);

**DB to page-schema CartellaPage (**

dati:

```
select c.*
from cartella_clinica as c, paziente as p
where c.id = ?id?
```

terapie:

```
select t.*
from cartella_clinica as c, terapie as t
where c.id = ?id?
and c.id = t.id_cartella
```

diagnosi:

```
select d.*, m.*
from cartella_clinica as c, diagnosi as d,
paziente as p, medico as m
where c.id = ?id?
and p.codsan = d.id_paziente
and p.codsan = c.codsan
and d.id_medico = m.id
```

terapie:

```
select t.*
from cartella_clinica as c, terapie as t
where c.id = ?id?
and c.id = t.id_cartella
```

**);**

**page-schema PatologiePage (**

patologie: text;
**);**

**DB to page-schema PatologiePage (**

patologie:

```
select d.*
from diagnosi as d
```

**);**

**page-schema PersonalePage (**

dati: text;
numero pazienti: int;
**);**

**DB to page-schema PersonalePage (**

dati:

```
select d.*
from diagnosi as d
```

numero pazienti:

```
select count(*)
from diagnosi
where patologia = (:patologia)
and icd10 = (:icd10)
group by icd10, patologia
```

**);**

**page-schema DiagnosiPage (**

new_diagnosi: form (
paziente: string;
cartella clinica: string;
data: date;
ICD10: string;
sintomi: list of (string);
tipologia: list of (string);
);
**);**

**DB to page-schema DiagnosiPage (**

new_diagnosi: *INSERT INTO DIAGNOSI ;*
new_sintomi: *INSERT INTO SINTOMI;*
new_conferme: *INSERT INTO CONFERME ;*
new_contraddizioni: *INSERT INTO CONTRADDIZIONI;*
**);**

# Parte IV
# Pagine Web

## 1 Homepage HTML

## 2 Info JSP

## 3 Login HTML

## 4 PazientePage JSP

## 5 CartellaPage JSP

## 6 PersonalePage JSP

## 7 PatologiePage JSP

## 8 DiagnosiPage JSP Javascript Ajax

# Parte V
# Strategie progettuali e considerazioni personali

Per rendere più realistico il database, sono stati aggiunti dei vincoli sugli attributi:

- la cartella clinica deve avere una data di ricovero maggiore o uguale della data di nascita del paziente;
- la cartella clinica deve avere una data di ricovero minore della data di dimissioni;
- le terapie devono essere fatte in un arco di tempo compreso tra la data di ricovero e dimissione della cartella clinica corrispondente;
- le terapie devono avere una data di inizio minore o uguale della data di fine;
- le diagnosi devono essere fatte in un arco di tempo compreso tra la data di ricovero e dimissione della cartella clinica corrispondente.

Considerazioni personali e strategie adottate durante lo sviluppo del progetto:

- È stato resa la data dimissione (attributo della cartella clinica) come attributo opzionale in quanto si è pensato che possono esserci delle cartelle cliniche di pazienti ancora ricoverati in reparto (nonostante le specifiche non esplicitassero questo fatto);
- Realizzazione del DB in modo tale da poter ottenere più relazioni possibili con la cartella clinica;

- Utilizzo del metodo Hibernate durante la realizzazione del progetto in modo tale da poter semplificare le query, tenendo presente che esse restituivano tanti valori nidificati a cui ci si poteva raggiungere tramite superchiavi;
- Durante la creazione della pagina relativa alle diagnosi (DiagnosiPage) il campo delle cartelle cliniche viene popolato tramite uno script ajax-json-jquery a seconda del paziente selezionato, in modo tale da evitare l'inserimento manuale di una cartella clinica potenzialmente errata;
- Per la realizzazione generale della pagina web che gestisce l'intero progetto ci siamo sentiti di renderla più gradevole graficalmente inserendo uno stile di impaginazione html in formato css;
- ECLIPSE pls!

# Parte VI
# Tecnologie aggiuntive utilizzate

## 9 Hibernate

Hibernate è un sistema o piattaforma middleware che offre un'interfaccia tra programmatore e database in modo da semplificare e gestire al meglio in maniera trasparente il database da interrogare o aggiornare. Questo sistema, tramite classi Java che rappresentano le entità del database, permette di avere un mapping tra variabili delle classi e attributi delle entità; in questo modo si può avere accesso agli attributi del database lavorando con i metodi get e set sulle variabili interessate.

Il mapping è reso possibile tramite file xml che chiarificano al sistema come mappare sia gli attributi delle entità sia le relazioni con la rispettiva cardinalità.

La progettazione si è incentrata sulla corretta scrittura dei file xml e delle relative classi; si è cercato inoltre di esplicitare quali fossero gli identificatori raggruppandoli in sottoclassi(qualora ce ne fossero più di uno), in modo da rendere ancora più chiara la programmazione: ogni classe avrà una dichiarazione sia di attributi sia di identificatori (quindi la creazione di una sottoclasse).

Oltre agli attributi e agli identificatori, si è deciso di associare la classe referenziata tramite attributo esterno alla classe interessata; in questo modo si può avere accesso a tutti i campi della classe referenziata applicando i metodi get e set alla sottoclasse, presente quindi nella classe interessata come variabile instanziata.

Come esempio pratico, la classe CartellaClinica ha i seguenti attributi:

```
private String id;
private Paziente paziente;
private Date dataRicovero;
private Date dataDimissione;
private String motivo;
private String prognosi;
private Set terapies = new HashSet(0);
private Set diagnosis = new HashSet(0);
private Set sintomis = new HashSet(0);
```

Come si può notare, la referenza all'entità Paziente (tramite attributo "codsan" nel modello relazionale) è resa tramite la dichiarazione della variabile paziente; quindi tramite una singola interrogazione di una specifica cartella clinica si possono sapere anche i campi del rispettivo paziente associato ad essa.

Come metodo di verifica della correttezza della progettazione per il sistema Hibernate, è stato utilizzato un plugin di Eclipse chiamato JBoss Tools, che contiene diverse features per Hibernate e permette di verificare la correttezza sia dei file xml sia delle classi Java a partire dalla descrizione del database.

Bisogna notare inoltre che le relazioni molti-a-molti hanno prodotto una referenza tradotta in Java con un HashSet, in modo da poter aver accesso ad ogni valore semplicemente di questo tipo di entità applicando i metodi set e get a questi HashSet. Questo si è rilevato di notevole importanza in quanto, per sapere la quantità di elementi di un certo tipo associati ad una entità, è bastato vedere la grandezza di questi HashSet utilizzando il metodo size, senza quindi fare un'ulteriore query. È stato necessario però forzare il sistema, in alcuni casi, ad avere un comportamento non "lazy", quindi impostando a false tale attributo nel file xml, in modo da caricare in memoria anche le istanze referenziate tramite le relazioni e quindi avere libero accesso ad esse.

# 10 Ajax JSON e JQuery

Poiché si è voluta rendere facile la scelta dei pazienti e delle relative cartelle cliniche nella DiagnosiPage (per rendere sicuro e affidabile un possibile insert da parte del medico nel database), è stato deciso di utilizzare un'insieme di tecnologie che potessero rendere dinamica la scelta da parte del medico. Sono state utilizzate tre tecnogie di sviluppo web chiamate Ajax, JSON e JQuery.

Ajax è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive. Consiste nel creare una servlet Java che riceve parametri da parte di una pagina web (nel nostro caso una JSP); questa servlet elabora i dati e li invia alla pagina chiamante, in modo che possa ricevere dinamicamente dati e quindi modificare il suo aspetto o comportamento.

Si è voluta utilizzare questa tecnologia per rendere dinamica la selezione in un campo del form di tipo select; poiché però questi campi devono essere "popolati" con dati provenienti dal database, è stato necessario introdurre la tecnologia JQuery e JSON.

JQuery consiste in un insieme di librerie per semplificare la programmazione web, nel nostro caso è stato utilizzato internamente ad Ajax per eseguire una query sul database e mappare in un LinkedHashMap la risposta. Per inviare poi questa risposta alla pagina web, è stato utilizzato JSON, una teconogia che consente lo scambio di dati tra architetture client-server; in questo modo si passa il risultato del calcolo della servlet alla pagina in maniera trasparente senza usare metodi doGet e doPost.

Tutto questo è visibile nella servlet ActionServlet e nell'intestazione della pagina DiagnosiPage.