Davide Cortellucci
MAT: 260321

Informatica Applicata
2016/2017

Basi di Dati

Relazione progetto

Indice generale

ANALISI DEI REQUISITI

Requisiti espressi in linguaggio naturale

Si vuole implementare una piattaforma online per lo sviluppo di contatti professionali.

Lo scopo di tale piattaforma è di consentire agli utenti registrati di mantenere una lista di persone conosciute e ritenute affidabili in ambito lavorativo.

Ciascun utente costituisce un nodo, le persone della lista vengono definite "connessioni" dell'utente all'interno della piattaforma. La rete di contatti a disposizione dell'utente è costituita da tutte le connessioni dell'utente ("connessioni di primo grado"), da tutte le connessioni delle sue connessioni ("connessioni di secondo grado") e da tutte le connessioni delle connessioni di secondo grado ("connessioni di terzo grado").

Gli account utente possono essere di due tipi:

1. account gratuito (utente cercatore di lavoro);
2. account a pagamento (utente datore di lavoro).

Ad ogni utente è riservato uno spazio, denominato "curriculum", atto a contenere le sue informazioni (dati anagrafici, studi, professioni ricoperte, ecc.) e il curriculum stesso, dove può elencare i riconoscimenti ed i premi ottenuti.

Gli utenti hanno accesso ad una sezione "Offerte di lavoro", che contiene le diverse richieste professionali.

Il servizio si distingue inoltre per la presenza di gruppi, formati da utenti che hanno qualcosa in comune, come un particolare percorso di carriera lavorativa, interessi di business simili, una specifica provenienza geografica o altro.

Attraverso la piattaforma è possibile:

- essere presentati a qualcuno che si desidera conoscere attraverso un contatto mutuo e affidabile;
- trovare offerte di lavoro, persone, opportunità di business con il supporto di qualcuno presente all'interno della propria lista di contatti o del proprio network;
- pubblicare offerte e ricercare potenziali candidati (solo per i possessori di un account a pagamento);
- consultare tali offerte nella sezione Offerte di lavoro (possessori di un account gratuito);
- evidenziare nel profilo personale le proprie abilità professionali e le competenze lavorative, oltre alle classiche informazioni presenti in un curriculum;

- segnalare utenti con i quali si è collaborato – collega, datore di lavoro o cliente – al fine di riconoscerne il talento, la preparazione, la professionalità.

Glossario dei termini

Termine	Descrizione	Collegamenti
Utente	Persona che costituisce il database e vi interagisce. Può essere un datore di lavoro o un lavoratore semplice.	Account, Contatti, Connessioni, Curriculum, Offerte, Gruppi
Account	Tipo di account dell'utente. Ne esistono due tipi, con diversi permessi: account a pagamento, account gratuito.	Utente, Offerte
Contatti	Lista di contatti conosciuti e ritenuti affidabili dall'utente.	Utente, Connessioni
Connessioni	Persone della lista di contatti dell'utente. Connessioni di primo, secondo e terzo grado costituiscono la rete di contatti di ciascun utente.	Utente, Contatti
Curriculum	Profilo dell'utente. Questi può modificarlo inserendo i propri dati ed esperienze/competenze lavorative.	Utente, Gruppi
Offerte	Lista di offerte di lavoro, consultabile dall'utente. Solo utenti con account a pagamento possono modificarla.	Utente, Account
Gruppi	Insieme di utenti con situazioni in comune.	Utente, Curriculum

Lista delle operazioni

- Aggiornamenti del database

1. Inserimento nuovo utente
2. Eliminazione utente
3. Modifica informazioni personali
4. Segnalare collaborazione con utente, aggiungendolo così alla lista contatti
5. Pubblicare offerte di lavoro (solo account a pagamento)
6. Rimuovere offerte di lavoro (solo account a pagamento)

- Interrogazioni al database

1. Elencare nome e cognome di tutti gli utenti
2. Elencare nome, cognome ed occupazione degli utenti
3. Elencare id di ogni contatto con nome e cognome dell'utente che indica
4. Stampare la lista contatti di un utente, dato il suo id
5. Elencare di quali utenti è connessione di secondo grado un utente
6. Elencare i nomi dei gruppi con una determinata affinità
7. Elencare i partecipanti ad un gruppo specifico
8. Consultare le offerte di lavoro con almeno un posto disponibile filtrando in base alla figura professionale richiesta
9. Stampare nome, cognome e data di nascita degli utenti nati dopo l'anno dato
10. Autenticazione utente

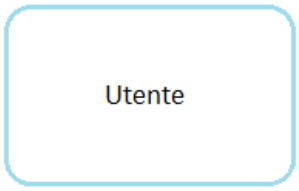
PROGETTAZIONE CONCETTUALE

Rappresentazione concettuale dei dati

Ora è il momento di rappresentare i concetti a disposizione secondo le regole del modello E-R (Entity-Relationship), che è il modello concettuale scelto in questo caso.
Prima di iniziare la costruzione dello schema E-R

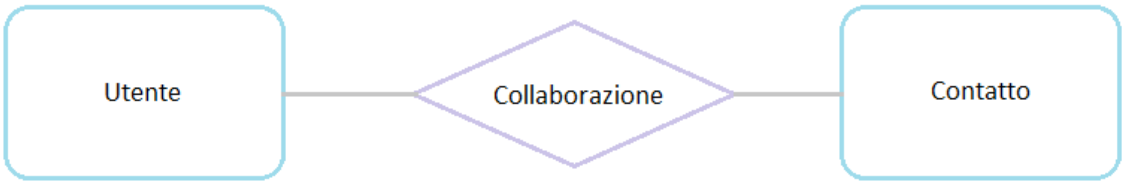
Entità

Per definizione, un'entità è un costrutto che rappresenta una classe di oggetti che hanno proprietà comuni ed esistenza autonoma all'interno dell'applicazione. Una *occorrenza* di entità è una sua istanza. Ad esempio l'entità "Lavoro" potrebbe avere come istanza "Dottore" o "Professore".
L'entità verrà rappresentata con una figura simile ad un rettangolo.



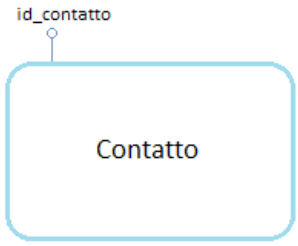
Associazioni

Un'associazione (o Relazione, dall'inglese "relationship") rappresenta un legame logico, significativo per l'applicazione d'interesse, tra due o più entità. E' possibile avere associazioni ricorsive, cioè tra un'entità e se stessa, o stabilire i ruoli delle entità coinvolte nell'associazione stessa, avendo così una associazione asimmetrica.
Graficamente corrisponde ad un rombo, collegato alle entità con delle linee.



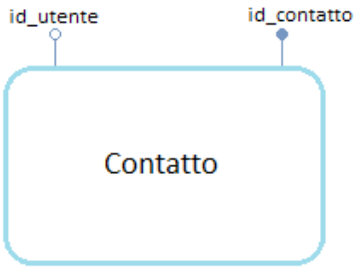
Attributi

Si definisce *attributo* un concetto, relativo alla base di dati, che ha una struttura semplice e non possiede proprietà rilevanti associate. Gli attributi sono legati a costrutti più "grandi" per descriverne le proprietà; nello specifico, uno o più attributi possono essere associati a *Entità* o ad *Associazioni.*

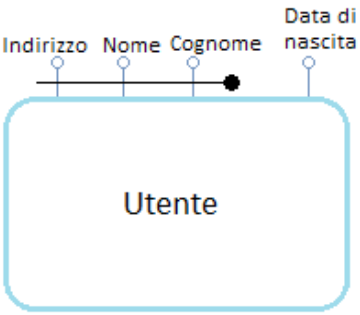


Identificatori

Vengono specificati per ciascuna entità; sono attributi che permettono di identificare univocamente le occorrenze di un'entità. Uno o più attributi possono essere identificatori, e possono coinvolgere solo l'entità a cui fanno riferimento (identificatori interni) o anche concetti esterni (identificatori esterni). Ogni entità deve averne almeno uno.
Nella figura sotto, "id_contatto" è un identificatore (interno).



La figura sotto mostra un identificatore con più di un attributo (Indirizzo, Nome e Cognome).

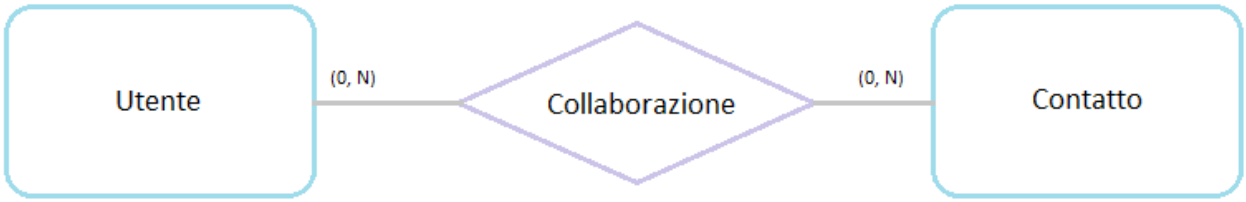


Cardinalità

Posso essere riferite ad associazioni e descrivono il numero minimo e massimo di occorrenze di associazione a cui un'occorrenza di entità può partecipare. Solitamente sono necessari solo tre numeri per rappresentare i diversi numeri di occorrenze:

- 0 e 1 per la cardinalità minima; nello specifico, 0 indica la partecipazione *opzionale* dell'entità all'associazione, mentre 1 ne denota la partecipazione *obbligatoria*;
- N per la cardinalità massima, cioè *molti*.

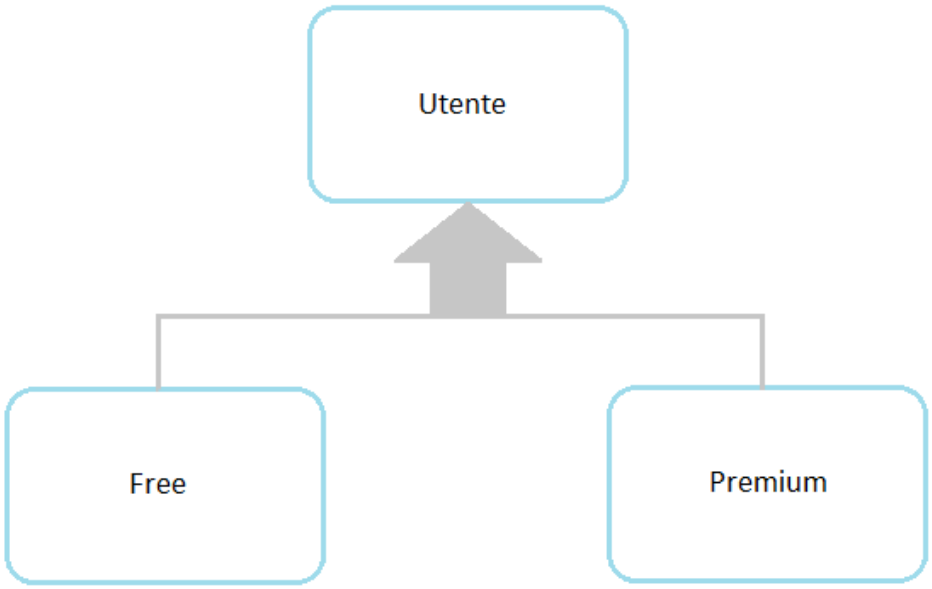
In base a ciò, possono esistere relazioni *uno ad uno*, *uno a molti* e *molti a molti*.



Dalla figura si evince che la cardinalità viene rappresentata tramite una coppia di numeri posti vicino alle linee di collegamento, in uscita dalla relativa associazione.
Le cardinalità possono essere riferite anche ad attributi.

Generalizzazioni

Per "generalizzazione" s'intende un legame logico tra un'entità padre (in questo caso Utente) e delle entità figlie (Free e Premium), le quali sono casi particolari dell'entità padre e ne ereditano gli attributi (ereditarietà).



Di seguito la categorizzazione dei concetti, trattati per Entità, appartenenti alla base di dati:

1) Utente

Entità principale del database; rappresenta l'utente stesso e le sue interazioni con la piattaforma. Esso ingloba il concetto di *account*, per questo lo si distingue in due tipologie, di cui è generalizzazione: utente *free* (account gratuito) e utente *premium* (account a pagamento).

Attributi di Utente:

- id_utente numero intero che identifica univocamente l'utente;
- nome nome dell'utente;
- cognome cognome dell'utente;
- sesso sesso dell'utente;
- email indirizzo e-mail con cui l'utente si è registrato;
- password password associata dall'utente al suo account;
- data_nascita data di nascita dell'utente;
- provenienza luogo di nascita dell'utente;
- occupazione denota l'occupazione dell'utente;
- luogo_lavoro città in cui l'utente lavora attualmente;
- curriculum area di testo contenente il curriculum dell'utente.

Associazioni di Utente:

- può avere uno o molti contatti;
- può consultare molte offerte di lavoro;
- può aggiungere una o più offerte di lavoro (se è premium);
- può avere uno o molti utenti appartenenti a connessioni di secondo e terzo grado;
- può appartenere ad uno o molti gruppi.

2) Contatto

Le occorrenze di questa entità costituiscono la lista contatti di ciascun utente, cioè rappresenta gli i contatti con cui l'utente ha collaborato e/o ritiene affidabile.

Attributi di Contatto:

- id_contatto intero che identifica univocamente ciascuna occorrenza di Contatto;
- utente attributo che specifica il contatto in questione.

Associazioni di Contatto:

- può essere contatto di uno o molti utenti.

3) Nodo

Questa entità costituisce le connessioni di secondo e terzo grado dell'utente, vale a dire le persone nella lista contatti del proprio contatto (connessioni di secondo grado) e le persone nella lista contatti delle persone nella lista contatti del proprio contatto (connessioni di terzo grado).
Per rappresentare questi tipi diversi di connessioni, sono necessarie due associazioni.

Attributi di Nodo:

- id_nodo intero che identifica univocamente ciascuna occorrenza di Nodo;
- utente attributo che rappresenta l'utente che si ha come connessione;
- contatto attributo che specifica con quale contatto si è in connessione.

Associazioni di Nodo:

- può essere connessione di secondo grado di uno o molti utenti;
- può essere connessione di terzo grado di uno o molti utenti; in particolare questa associazione ha un attributo "connessione" che fa riferimento all'utente della connessione di secondo grado.

4) Gruppo

Entità che serve a rappresentare l'esistenza dei gruppi, cioè un insieme di utenti che hanno dei dati in comune (lavoro, luogo di lavoro, nascita, ecc.).

Attributi di Gruppo:

- id_gruppo numero intero che identifica univocamente un gruppo;
- affinita attributo che specifica il parametro che accomuna gli utenti del gruppo;
- nome_gruppo nome del gruppo, dipende da attinenza.

Associazioni di Gruppo:

- può appartenervi uno o più utenti.

5) Offerta

Le occorrenze di Offerta formano la lista di offerte di lavoro presente nella piattaforma.

Attributi di Offerta:

- id_offerta numero intero che identifica univocamente l'offerta;
- emittente identifica l'utente premium che ha creato l'offerta di lavoro;
- occupazione indica la tipologia di impiego offerto;
- num_posti numero di posti di lavoro disponibili;
- luogo luogo di lavoro;
- durata durata del contratto di lavoro.

Associazioni di Offerta:

- è consultabile da più utenti ed un utente può consultare più offerte;

- un utente premium può pubblicare una o più offerte di lavoro;
- può essere pubblicata da un solo utente premium.

6) Città

Entità introdotta per permettere di avere una lista di città predefinite, in modo da alleggerire il peso del database ed evitare incongruenze tra i nomi.

Attributi di Città:

- id_citta numero intero che identifica univocamente una città;
- nome_citta stringa contenente il nome della città.

Associazioni di Città:

- possono esservi nati più utenti;
- possono attualmente lavorarci più utenti.

7) Lavoro

Similmente a Città, l'entità Lavoro serve ad avere una lista di lavori predefiniti.
Attributi di Lavoro:

- id_lavoro numero intero che identifica univocamente un lavoro;
- nome_lavoro stringa contenente il nome del lavoro.

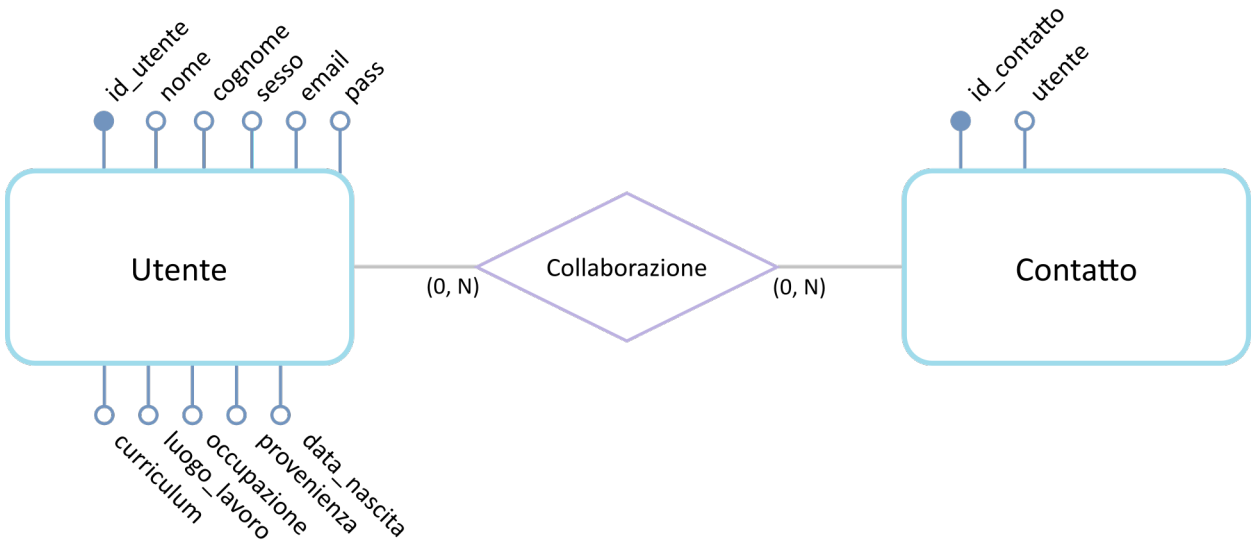
Associazioni di lavoro:

- può essere l'impiego di più utenti

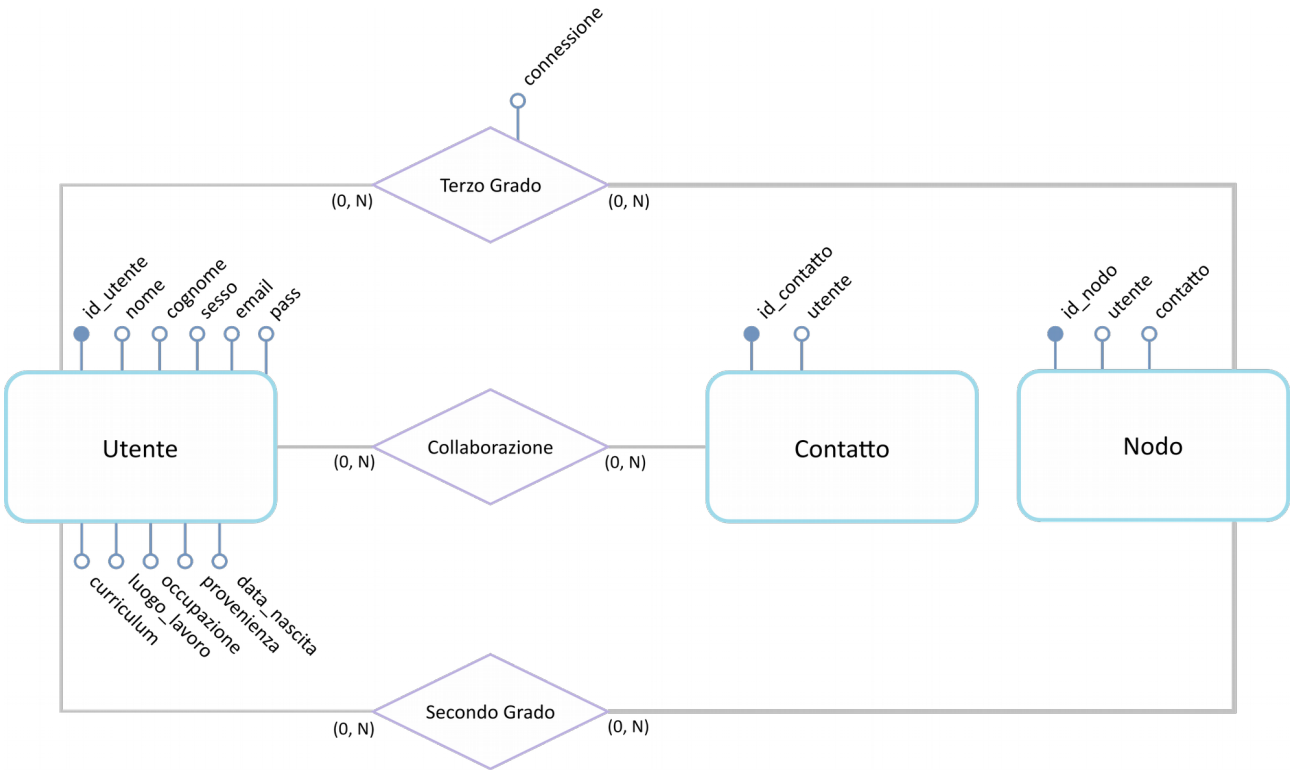
Strategia di progetto e sviluppo schema E-R

Come strategia di progetto è stata scelta la *inside-out*, che può essere intesa come un caso particolare della strategia *bottom-up*.
A partire dalle specifiche, si individuano i concetti più importanti e da questi si procede espandendosi "a macchia d'olio" fino a coprire interamente le specifiche.

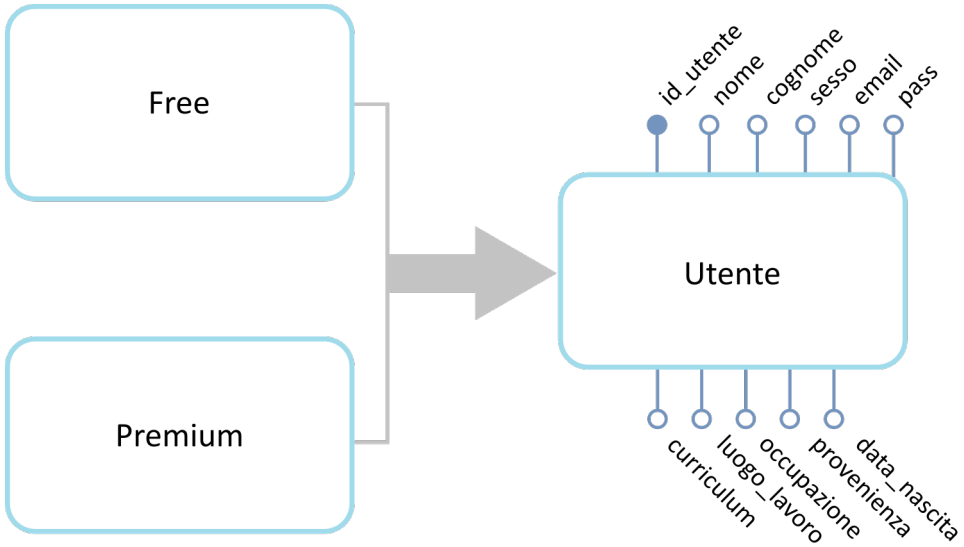
Innanzitutto si considera la relazione tra Utente e Contatto, che ci dice che ciascun utente ha collaborato con uno o più contatti, e che ciascun contatto (cioè ogni utente nella lista contatti di un altro utente) può aver collaborato con uno o più utenti, e quindi può essere nella lista contatti di uno o più utenti:



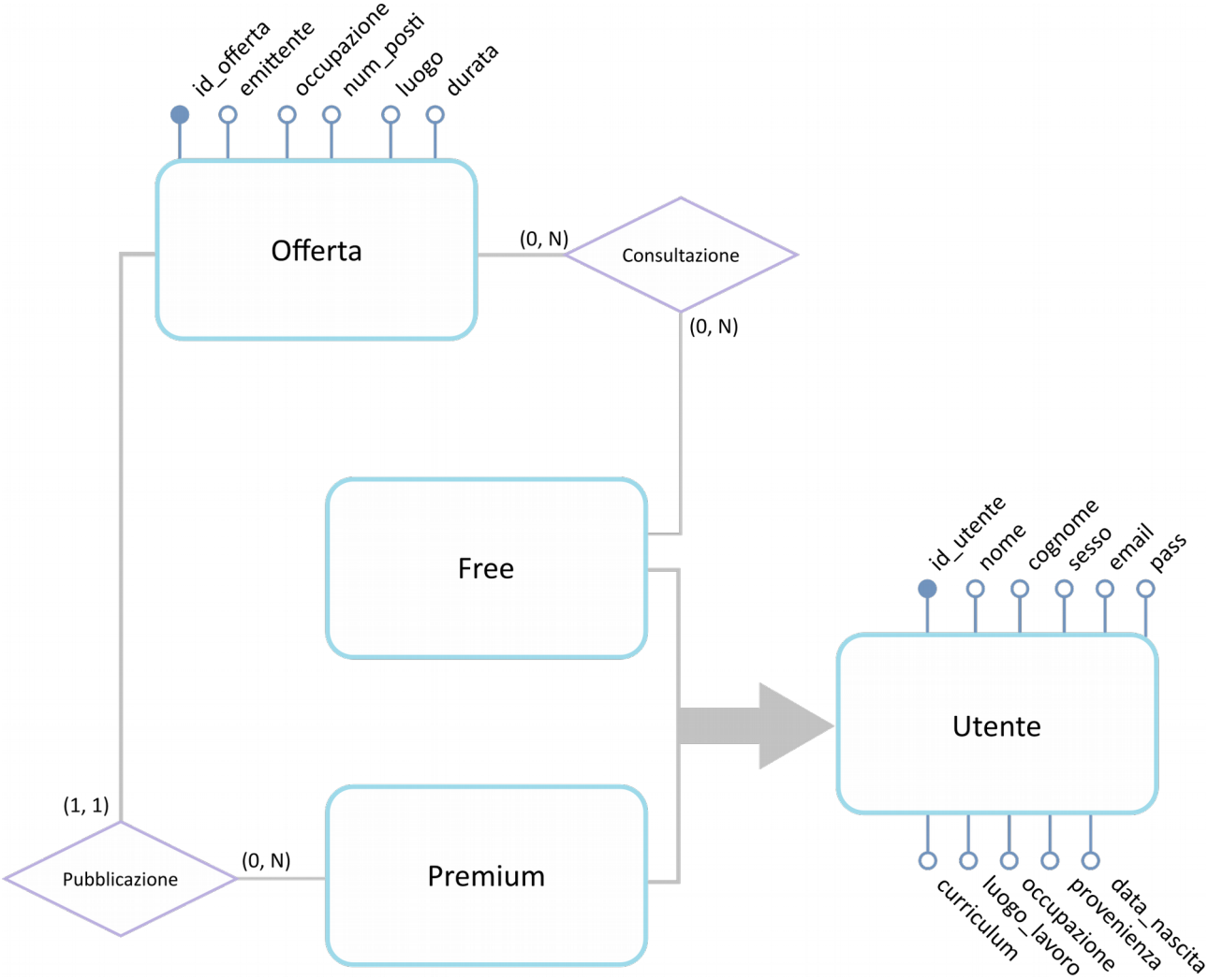
Successivamente si analizza la relazione tra l'utente e le sue connessioni mediante due associazioni tra Utente e Nodo, le quali specificano il tipo di connessione (secondo o terzo grado). Si noti che l'associazione Terzo Grado ha un attributo che denota il nodo utente appartenente alla connessione di secondo grado:



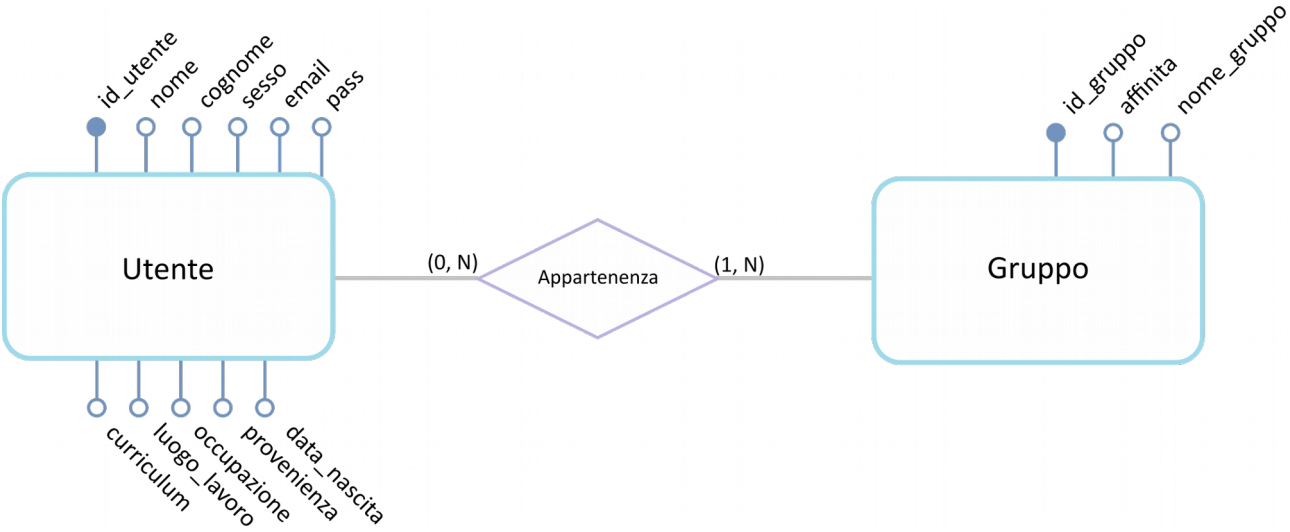
Quindi si passa alla relazione fra Utente e il concetto di *account*, che si è deciso di rappresentare mediante l'utilizzo di una generalizzazione; nello specifico, Utente diventa generalizzazione delle entità Free (account gratuito) e Premium (account a pagamento):



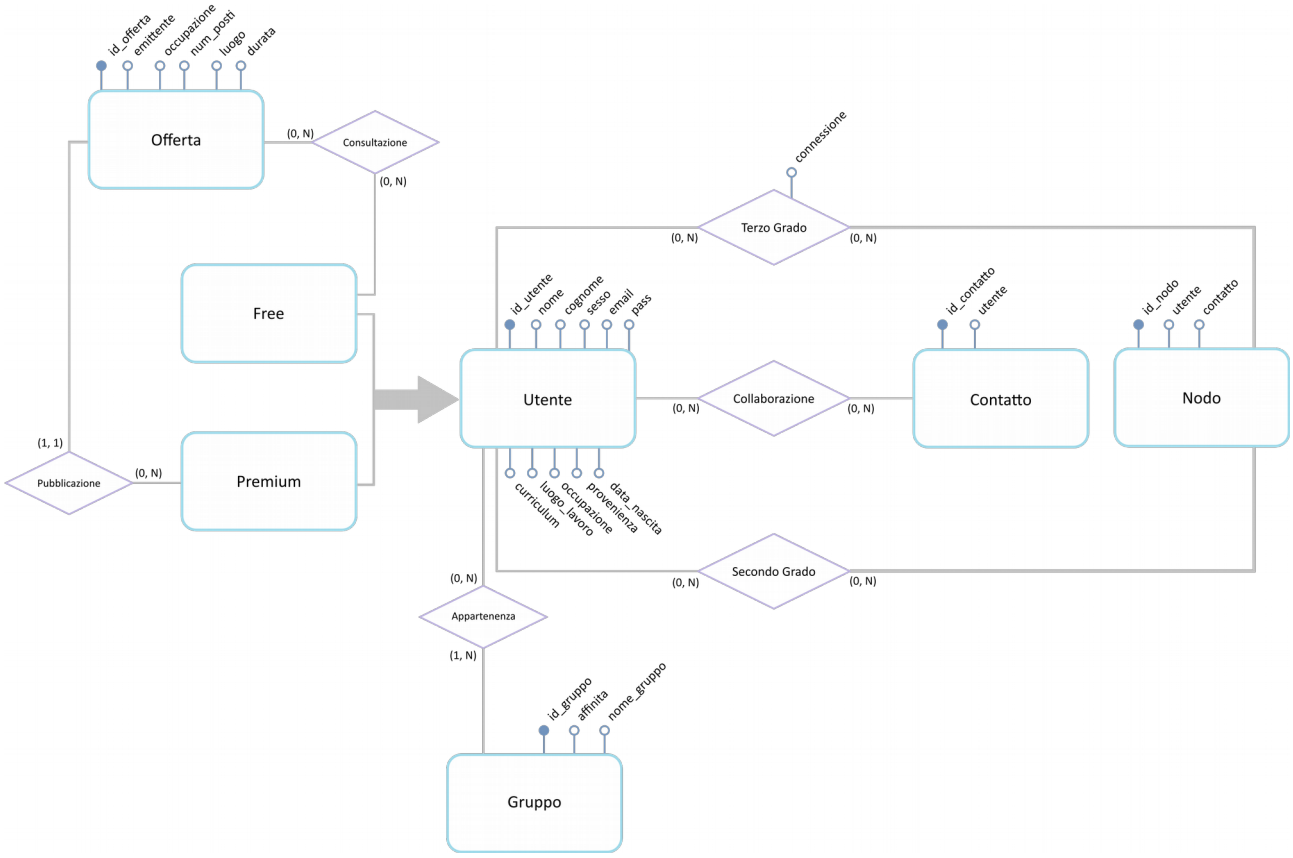
In seguito si distinguono due associazioni con l'entità Offerta, dal momento che solo utenti premium possono pubblicare offerte di lavoro:



Infine si schematizza la relazione tra Utente e Gruppi tramite una associazione tra le due entità:



Riassumendo tutto in un unico schema:


id_offerta
emittente
occupazione
num_posti
luogo
durata
Offerta
(0, N)
Consultazione
(0, N)
Free
(1, 1)
Pubblicazione
(0, N)
Premium
connessione
Terzo Grado
(0, N)
(0, N)
id_utente
nome
cognome
sesso
email
pass
Utente
Collaborazione
(0, N)
(0, N)
id_contatto
utente
Contatto
id_nodo
utente
contatto
Nodo
curriculum
luogo_lavoro
occupazione
provenienza
data_nascita
Secondo Grado
(0, N)
(0, N)
(0, N)
Appartenenza
(1, N)
id_gruppo
affinita
nome_gruppo
Gruppo

PROGETTAZIONE LOGICA

La progettazione logica è quella fase che permette di tradurre lo schema concettuale della base di dati in uno schema logico, cioè uno schema che non astrae dall'implementazione della base di dati. Essa si divide in due fasi: *ristrutturazione* dello schema Entità-Relazione e *traduzione* dal modello concettuale a quello logico.
I dati in ingresso di una progettazione logica sono lo schema E-R, il *carico applicativo* previsto (dimensione dei dati e caratteristiche delle operazioni) ed il modello logico scelto; alla fine della fase di ristrutturazione si ottiene uno schema E-R ristrutturato, che non è più uno schema concettuale in quanto tiene conto degli aspetti realizzativi. Tale schema è l'input della fase di traduzione, che produce lo schema logico finale, cioè l'output della progettazione logica. Durante il secondo stadio è possibile effettuare operazioni di verifica della qualità.
Il modello logico che verrà utilizzato è il modello relazionale, descritto in seguito, con tutte le implicazioni che ne conseguono.

Ristrutturazione

La ristrutturazione comprende i seguenti passi, da effettuare in sequenza:

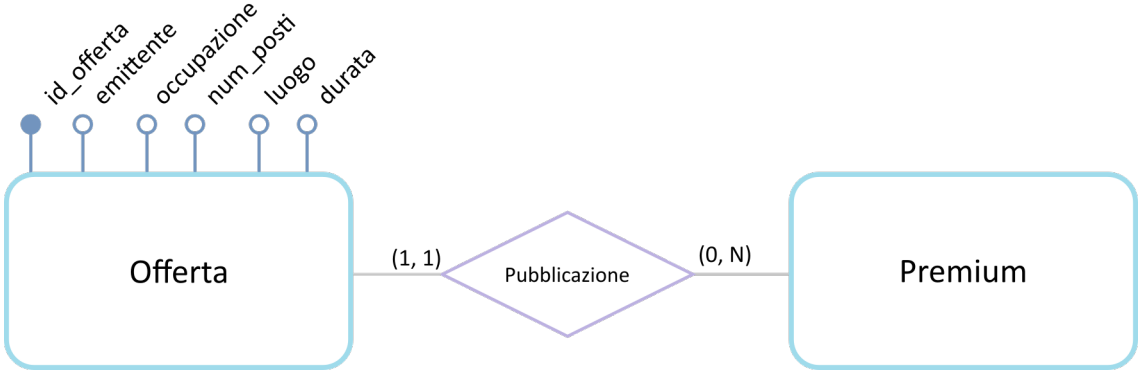
- Ottimizzazione;
- Semplificazione.

Ottimizzazione

Durante questo passaggio, viene mostrata l'analisi di una particolare scelta di progetto per valutarne l'impatto sulle prestazioni della base di dati. Per valutare ciò si fa uso di tre tabelle:

- *Tabella dei volumi:* in essa vengono riportati i concetti dello schema E-R (entità ed associazioni) con il volume previsto, cioè il numero delle loro occorrenze.
- *Tabella delle operazioni:* qui si elencano, per ogni operazione prevista sulla base di dati, la frequenza prevista ed il tipo (I – interattiva, B – batch).
- *Tabella degli accessi:* è riferita ad una operazione ed ad essa associa le entità, il numero ed il tipo di accessi (lettura o scrittura) necessari per svolgere tale operazione. La distinzione tra lettura e scrittura è fondamentale, poiché generalmente le operazioni di scrittura sono molto più onerose di quelle in lettura.

In questo caso, viene presa in considerazione l'associazione Pubblicazione tra le entità Premium ed Offerta.



In particolare, ci si focalizza sulla presenza dell'attributo *num_posti* di Offerta, il quale indica il numero di posti disponibili per tale offerta di lavoro. La presenza dell'attributo è una scelta di progetto, in quanto esso può essere considerato una ridondanza, cioè "un dato che è derivabile da altri dati".
Più specificamente, supponendo uno schema alternativo che prevede un posto di lavoro per ogni offerta pubblicata, *num_posti* potrebbe essere omesso e calcolato semplicemente contando il numero di offerte di lavoro uguali pubblicate da uno stesso utente.
Dal momento che l'eliminazione delle ridondanze non porta necessariamente dei benefici, si valutano i costi di accesso sui quali operare la scelta ottimale.

L'ottimizzazione effettuata di seguito ha lo scopo di valutare le ripercussioni in termini di costo computazionale e di utilizzo della memoria dei due modi differenti di strutturare lo schema E-R, cioè con e senza ridondanza.

La tavola dei volumi ingloba questi tre concetti:

Concetto	Tipo	Volume
Utente	E	1000
Offerta	E	100
Pubblicazione	R	100

Il volume è indicativo, ci si vuole focalizzare maggiormente sul rapporto tra i concetti; notare che il volume di Offerta e Pubblicazione coincide, poiché la cardinalità dice che un'offerta viene pubblicata da un solo utente.

Per quanto riguarda le operazioni, se ne esaminano due:

1. pubblicare un'offerta di lavoro (**operazione 1**);
2. cercare il numero di posti di lavoro disponibili (**operazione 2**).

Operazione	Tipo	Frequenza
Operazione 1	I	10 volte al giorno
Operazione 2	I	50 volte al giorno

A questo punto si considerano individualmente le due scelte e si formano le rispettive tabelle degli accessi (una per operazione).

IN PRESENZA DI RIDONDANZA

Operazione 1			
Concetto	Costrutto	Accessi	Tipo
Utente	E	1	L
Offerta	E	2	S
Pubblicazione	R	1	S

Nel dettaglio, pubblicare un'offerta di lavoro comporta un accesso in lettura su Utente per sapere chi pubblica, un accesso in scrittura su Pubblicazione per memorizzare una nuova coppia utente-offerta, e due accessi in scrittura su Offerta, uno per memorizzare una nuova offerta, l'altro per scrivere un valore su *num_posti*.

Operazione 2			
Concetto	Costrutto	Accessi	Tipo
Offerta	E	1	L

Cercare il numero di posti di lavoro disponibili comporta in questo caso il solo accesso in lettura all'attributo *num_posti* di Offerta.

Ora si può calcolare il costo delle operazioni estraendo i dati dalle tabelle; in particolare sono utili la frequenza giornaliera ed il numero di accessi delle operazioni. Per simulare la maggiore onerosità delle operazioni di scrittura, nel calcolo seguente il costo di accesso ad esse sarà il doppio rispetto alle operazioni di lettura.

Il costo della prima operazione consiste in:

- 1 accesso in lettura su Utente, 10 volte al giorno (1*10 = 10);
- 1 accesso in scrittura su Pubblicazione, 10 volte (1*10*2 = 20);
- 2 accessi in scrittura su Offerta, 10 volte (2*10*2 = 40).

Sommando il tutto, mediamente l'operazione 1 richiede 70 operazioni al giorno.

Il costo della seconda operazione è facilmente calcolabile: 1 accesso in lettura * 50 volte al giorno, quindi 50 accessi.

Ricapitolando, in presenza di ridondanza le due operazioni richiedono 120 accessi giornalieri.
C'è inoltre da considerare la maggiore quantità di memoria richiesta per memorizzare l'attributo: supponendo l'uso di 4 byte per ogni attributo (sufficienti per memorizzare un intero), moltiplicando per il volume di Offerta si ottengono 400 byte, una richiesta esigua di memoria aggiuntiva.

IN ASSENZA DI RIDONDANZA

Operazione 1			
Concetto	Costrutto	Accessi	Tipo
Utente	E	1	L
Offerta	E	1	S
Pubblicazione	R	1	S

La tabella degli accessi è simile a quella in presenza di ridondanza, con la differenza che l'assenza dell'attributo comporta un accesso in meno in scrittura su Offerta (non bisogna scrivere sull'attributo).

Operazione 2			
Concetto	Costrutto	Accessi	Tipo
Offerta	E	1	L
Pubblicazione	R	10	L

Cercare il numero di posti di lavoro disponibili comporta anche dieci accessi in lettura a Pubblicazione, dove tale numero è stato stimato dividendo il volume di Utenti con il volume di Pubblicazione, cioè facendo una media (ogni offerta dello stesso tipo è stata pubblicata mediamente 10 volte dallo stesso utente: 1000/100 = 10).

Stavolta la prima operazione comporta:

- 1 accesso in lettura su Utente, 10 volte al giorno (1*10 = 10);
- 1 accesso in scrittura su Offerta, 10 volte (1*10*2 = 20);
- 1 accesso in scrittura su Pubblicazione, 10 volte (1*10*2 = 20).

Quindi 50 accessi al giorno complessivi.

Per quanto riguarda la seconda operazione, in questo caso servono:

- 1 accesso in lettura su Offerta, 50 volte al giorno (1*50 = 50);
- 10 accessi in lettura su Pubblicazione, 50 volte (10*50 = 500).

Cioè 550 accessi giornalieri.

Quindi si può dire che, in assenza di ridondanza, le due operazioni richiedono mediamente 600 accessi al giorno, contro i 120 in presenza di ridondanza.
Si evince che il primo approccio è molto più conveniente, poiché, a fronte di un piccolo incremento della memoria richiesta, il vantaggio prestazionale è notevole.

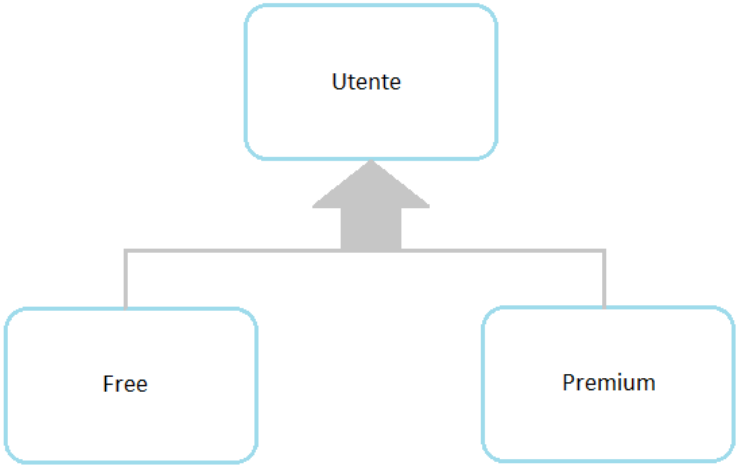
Semplificazione

Questa fase prevede le seguenti operazioni sullo schema E-R:

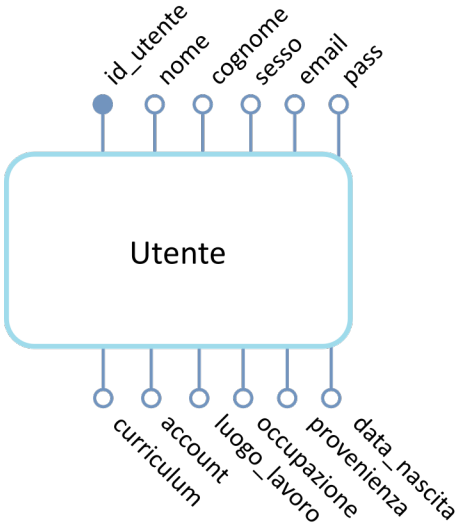
- Eliminazione delle generalizzazioni;
- Eliminazione di attributi composti e di identificatori esterni;
- Scelta degli identificatori principali.

Eliminazione delle generalizzazioni

Necessaria poiché i sistemi tradizionali per la gestione di basi di dati (incluso il modello relazionale) non permettono di rappresentare direttamente una generalizzazione.
In questo caso, l'unico episodio di generalizzazione presente è Utente, entità-genitore di Free e Premium:



Questa generalizzazione è *totale* (gli utenti premium e free costituiscono tutti gli utenti della base di dati) ed *esclusiva* (non esistono utenti che sono premium e free allo stesso tempo).
Si è ritenuto opportuno accorpare le entità-figlie nell'entità-genitore, poiché questa scelta garantisce un numero minore di accessi (l'entità Utente è coinvolta in molte associazioni) ed è attuabile semplicemente aggiungendo un attributo all'entità-genitore che identifichi le occorrenze di tale entità.



Eliminazione di attributi composti e di identificatori esterni

Dal momento che nello schema E-R finale non ci sono attributi composti o identificatori esterni, questa fase di ristrutturazione non viene effettuata.

Scelta degli identificatori principali

Nella selezione degli identificatori si è tenuto conto dei seguenti criteri:

- ➔ attributi che possono avere valori nulli sono da scartare;
- ➔ si preferiscono identificatori composti da un solo attributo;
- ➔ si favoriscono identificatori interni rispetto a quelli esterni;
- ➔ è preferibile un identificatore usato da molte operazioni per accedere alle occorrenze dell'entità ad esso associata.

Per tutte le entità, come evidenziato dallo schema finale, si utilizzeranno numeri interi auto-incrementanti come identificatori.

Traduzione

Il modello relazionale si basa sul concetto di *relazione* e *tabella.* Per quanto riguarda la prima, si tratta della relazione matematica su n insiemi (*domini* della relazione), la quale è un *sottoinsieme* del prodotto cartesiano dei domini stessi; invece, il termine *tabella* si riferisce al concetto intuitivo di tabella.

Le cosiddette *tuple* della relazione costituiscono le righe della tabella e, dal momento che una relazione matematica è un insieme, le tuple di una relazione sono distinte e non è definito alcun ordinamento tra di esse. Gli elementi di una tupla, inoltre, sono ordinati.
Nel modello relazionale le intestazioni delle colonne delle tabelle si dicono *attributi*, e sono i nomi ad esse associati, pertanto diversi l'uno dall'altro.

Nel caso di valori sconosciuti, inesistenti o senza informazione, si estende il concetto di relazione prevedendo che il dominio di un valore di una tupla possa assumere un *valore nullo*, cioè senza informazione.
Devono essere posti delle moderazioni sui valori nulli per non rischiare di avere, ad esempio, tuple senza informazione.

Il concetto di *vincolo di integrità* definisce delle proprietà che devono essere rispettate dalle istanze delle relazioni affinché rappresentino informazioni corrette ai fini dell'applicazione. Può essere visto come un *predicato*, che associa un valore di vero o falso ad ogni istanza.
Un *vincolo di integrità referenziale* si definisce fra gli attributi X di una relazione R1 ed una relazione R2 ed è "soddisfatto" se i valori di X sono anche valori della *chiave* dell'istanza di R2.

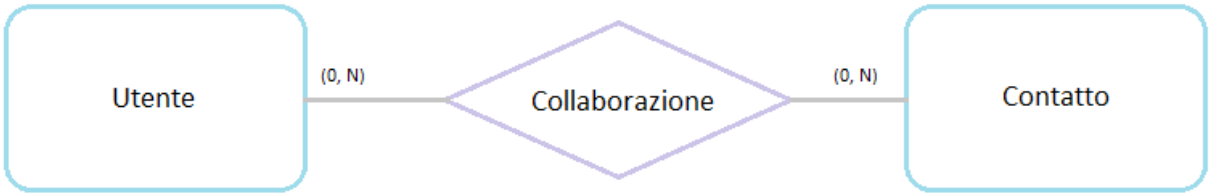
Una *superchiave* è un insieme di uno o più attributi che identifica univocamente le tuple di una relazione. Una superchiave K è *chiave* di una relazione se non esiste un'altra superchiave K' contenuta in K come sottoinsieme proprio.
L'esigenza di porre dei limiti al numero di chiavi che possono assumere valori nulli porta a definire, con il nome di *chiave primaria*, una chiave che non può avere valori nulli.

Le definizioni riportate sopra saranno utili durante la traduzione nel modello relazionale.

Associazioni molti a molti

- Ad ogni entità corrisponde una relazione con lo stesso nome (cioè una tabella), avente come attributi gli stessi attributi dell'entità ed il suo identificatore come chiave.
- Le associazioni si traducono con una relazione dallo stesso nome, avente per attributi gli eventuali attributi dell'associazione e gli identificatori delle entità coinvolte.

Ad esempio, l'associazione Collaborazione tra Utente e Contatto



diventa:

UTENTE(id_utente, nome, cognome, sesso, …, account)
CONTATTO(id_contatto, utente)
COLLABORAZIONE(id_utente, id_contatto)

e contiene due vincoli di integrità referenziale tra gli attributi id_utente e id_contatto di COLLABORAZIONE e gli omonimi attributi delle entità UTENTE e CONTATTO.

Associazioni uno a molti

Si consideri come esempio l'associazione Pubblicazione tra Utente e Offerta.
Si possono seguire le stesse regole enunciate per le associazioni molti a molti, con una differenza:

UTENTE(id_utente, nome, cognome, sesso, …, account)
OFFERTA(id_offerta, emittente, occupazione, …, durata)
PUBBLICAZIONE(id_offerta, id_utente)

cioè nella relazione PUBBLICAZIONE la chiave è formata solo dall'identificatore di OFFERTA, poiché la cardinalità dell'associazione rivela che un'offerta è pubblicata da un solo utente; ma dal momento che esiste una corrispondenza biunivoca tra le occorrenze di OFFERTA e PUBBLICAZIONE, è possibile fonderle in un'unica relazione:

UTENTE(id_utente, nome, cognome, sesso, …, account)
OFFERTA(id_offerta, emittente, occupazione, …, durata)

In questo caso sono state fuse insieme OFFERTA e PUBBLICAZIONE in OFFERTA.

Modello relazionale

Alla luce di quanto detto, si può ora procedere alla traduzione dallo schema E-R ristrutturato al modello relazionale:

- innanzitutto l'associazione tra Utente e Contatto:

  UTENTE(id_utente, nome, cognome, … , provenienza, occupazione, luogo_lavoro, account)
  CONTATTO(id_contatto, utente)
  COLLABORAZIONE(id_utente, id_contatto)

  l'attributo utente in CONTATTO referenzia id_utente di UTENTE (vincolo di integrità referenziale); gli attributi *provenienza*, *luogo_lavoro* ed *occupazione* referenziano rispettivamente id_citta di CITTA' ed id_lavoro di LAVORO.

- associazione tra Utente e Gruppo:

  UTENTE(id_utente, nome, cognome, sesso, …, account)
  GRUPPO(id_gruppo, affinita, nome_gruppo)
  APPARTENENZA(id_utente, id_gruppo)

  l'attributo *affinita* referenzia id_lavoro di LAVORO.

- associazione Pubblicazione tra Utente ed Offerta:

  UTENTE(id_utente, nome, cognome, sesso, …, account)
  OFFERTA(id_offerta, emittente, occupazione, num_posti, luogo, durata)

  l'attributo emittente in OFFERTA referenzia id_utente, mentre occupazione referenzia id_lavoro; infine, luogo referenzia id_citta.

- associazione Consultazione tra Utente ed Offerta:

  UTENTE(id_utente, nome, cognome, sesso, …, account)
  OFFERTA(id_offerta, emittente, occupazione, …, durata)
  CONSULTAZIONE(id_utente, id_offerta)

- associazione Secondo fra Utente e Nodo:

  UTENTE(id_utente, nome, cognome, sesso, …, account)
  NODO(id_nodo, contatto, utente)
  SECONDO(id_utente, id_nodo)

  con gli attributi utente e contatto di NODO referenzianti rispettivamente id_utente ed id_contatto.

- associazione Terzo fra Utente e Nodo:
  UTENTE(id_utente, nome, cognome, sesso, …, account)
  NODO(id_nodo, contatto, utente)
  TERZO(id_utente, id_nodo, connessione)

  con gli attributi utente e contatto di NODO referenzianti rispettivamente id_utente ed id_contatto, mentre connessione referenzia id_utente.

- un utente può essere nato/lavorare in una ed una sola città, quindi:
  UTENTE(id_utente, nome, cognome, sesso, …, account)
  CITTA(id_citta, nome_citta)

- un utente può avere uno ed un solo lavoro:
  UTENTE(id_utente, nome, cognome, sesso, …, account)
  LAVORO(id_lavoro, nome_lavoro).

Normalizzazione

La teoria della normalizzazione è un test di qualità studiato per il modello relazionale.
Essa prevede l'analisi di alcune proprietà, dette *forme normali*, che certificano la qualità di una relazione. Infatti quando una relazione non soddisfa una forma normale, essa si presta a ridondanza ed a comportamenti indesiderabili durante le operazioni di aggiornamento.
Il termine *normalizzazione* sta ad indicare il processo di trasformazione che permette di modificare relazioni non normalizzate in relazioni che rispettano i criteri delle forme normali.

Prima Forma Normale (1NF)

Una relazione è in prima forma normale se:

- tutte le tuple della relazione hanno lo stesso numero di attributi;
- ciascun attributo è definito su un dominio con valori atomici;
- tutti i valori di un attributo appartengono allo stesso dominio;
- esiste un insieme di attributi che identifica in modo univoco ogni tupla della relazione;
- l'ordine delle righe è irrilevante.

Questo database rispetta le proprietà sopra elencate, alcune delle quali sono implicate nella definizione di *relazione*.

Seconda Forma Normale (2NF)

Una relazione è in seconda forma normale quando:

- è in prima forma normale;
- per ogni relazione tutti gli attributi non-chiave dipendono funzionalmente dall'intera chiave composta (ovvero la relazione non ha attributi che dipendono funzionalmente da una parte della chiave).

Viene rispettato anche questo criterio, in quanto tutte le relazioni sono in prima forma normale, inoltre tutte le relazioni hanno un identificatore autoincrementante; pertanto ogni attributo dipende sempre dall'intera chiave, che non è composta.

Terza Forma Normale (3NF)

Una relazione è in terza forma normale se:

- è in seconda forma normale;
- tutti gli attributi non-chiave dipendono dalla chiave soltanto, ossia non esistono attributi che dipendono da altri attributi non-chiave.

Anche in questo caso, le due proprietà vengono rispettate dalle relazioni del database.

Forma normale di Boyce e Codd (BCNF)

Per studiare la BCNF è necessario il concetto di *dipendenza funzionale*:

- ➔ la *dipendenza funzionale* è un tipo di vincolo di integrità fra due sottoinsiemi di attributi non vuoti X ed Y di una relazione r su uno schema R(Z); si dice che c'è una dipendenza funzionale tra X ed Y (indicata con X → Y) se per ogni coppia di tuple t_1 e t_2 aventi gli stessi valori su X, allora esse hanno valori uguali anche su Y.
- ➔ Una dipendenza funzionale X → Y si dice *banale* se Y compare tra gli attributi di X.

Utente	Provenienza	Gruppo
Grassi	Teramo	Abruzzo
Rossi	Pesaro	Marche
Grassi	Teramo	Infermieri
Cecchini	Torino	Ingegneri

In questo esempio c'è dipendenza funzionale tra l'attributo Utente e Provenienza, perché tali attributi assumono gli stessi valori nelle tuple (Grassi, Teramo, Abruzzo) e (Grassi, Teramo, Infermieri).

Dunque una relazione r si dice in forma normale di Boyce e Codd se:

- è in prima forma normale;
- per ogni dipendenza funzionale non banale X → Y (cioè Y non è contenuto in X), X è una superchiave per r.

Essendo tutte le relazioni di questo database in terza forma normale ed avendo tutte una sola chiave, si può dire che tale database rispetta la forma normale di Boyce e Codd.

IMPLEMENTAZIONE

Ora è possibile passare alla traduzione dello schema relazionale in linguaggio SQL. Di seguito si traduce ciascuna relazione dello schema relazionale in una tabella.

- CITTA(id_citta, nome_citta)

```
CREATE TABLE `citta` (
  `id_citta` smallint(20) NOT NULL AUTO_INCREMENT,
  `nome_citta` varchar(30) NOT NULL,
  PRIMARY KEY (`id_citta`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- LAVORO(id_lavoro, nome_lavoro)

```
CREATE TABLE `lavoro` (
  `id_lavoro` smallint(20) NOT NULL AUTO_INCREMENT,
  `nome_lavoro` varchar(30) NOT NULL,
  PRIMARY KEY (`id_lavoro`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- UTENTE(id_utente, nome, cognome, sesso, email, pass, data_nascita, provenienza, occupazione, luogo_lavoro, curriculum, account)

```
CREATE TABLE `utente` (
  `id_utente` smallint(20) NOT NULL AUTO_INCREMENT,
  `nome` varchar(30) NOT NULL,
  `cognome` varchar(30) NOT NULL,
  `sesso` enum('m','f') NOT NULL,
  `email` varchar(30) NOT NULL,
  `pass` varchar(30) NOT NULL,
  `data_nascita` date NOT NULL,
  `provenienza` smallint(20) DEFAULT NULL,
  `occupazione` smallint(20) NOT NULL,
  `luogo_lavoro` smallint(20) DEFAULT NULL,
  `curriculum` varchar(300) DEFAULT NULL,
  `account` enum('free','premium'),
  PRIMARY KEY (`id_utente`),
  KEY `citta_nascita` (`provenienza`),
  KEY `lavoro` (`occupazione`),
  KEY `citta_lavoro` (`luogo_lavoro`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE `utente`
  ADD CONSTRAINT `utentecitta_fk_1` FOREIGN KEY (`provenienza`) REFERENCES
`citta` (`id_citta`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `utentelavoro_fk` FOREIGN KEY (`occupazione`) REFERENCES
`lavoro` (`id_lavoro`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `utentecitta_fk_2` FOREIGN KEY (`luogo_lavoro`) REFERENCES
`citta` (`id_citta`) ON DELETE CASCADE ON UPDATE CASCADE;
```

- CONTATTO(id_contatto, utente)

```
CREATE TABLE `contatto` (
  `id_contatto` smallint(20) NOT NULL AUTO_INCREMENT,
  `utente` smallint(20) NOT NULL,
  PRIMARY KEY (`id_contatto`),
  KEY `utente` (`utente`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE `contatto`
  ADD CONSTRAINT `utentecontatto_fk` FOREIGN KEY (`utente`) REFERENCES
`utente` (`id_utente`) ON DELETE CASCADE ON UPDATE CASCADE;
```

- COLLABORAZIONE(id_utente, id_contatto)

```
CREATE TABLE `collaborazione` (
  `id_utente` smallint(20) DEFAULT NULL,
  `id_contatto` smallint(20) DEFAULT NULL,
  KEY `id_utente` (`id_utente`),
  KEY `id_contatto` (`id_contatto`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE `collaborazione`
  ADD CONSTRAINT `collaborazioneutente_fk` FOREIGN KEY (`id_utente`)
REFERENCES `utente` (`id_utente`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `collaborazionecontatto_fk` FOREIGN KEY (`id_contatto`)
REFERENCES `contatto` (`id_contatto`) ON DELETE CASCADE ON UPDATE CASCADE;
```

- OFFERTA(id_offerta, emittente, occupazione, num_posti, luogo, durata)

```
CREATE TABLE `offerta` (
  `id_offerta` smallint(50) NOT NULL AUTO_INCREMENT,
  `emittente` smallint(20) NOT NULL,
  `occupazione` smallint(20) NOT NULL,
  `num_posti` smallint(6) NOT NULL,
  `luogo` smallint(20) NOT NULL,
  `durata` date NOT NULL,
  PRIMARY KEY (`id_offerta`),
  KEY `utente` (`emittente`),
  KEY `lavoro` (`occupazione`),
  KEY `citta` (`luogo`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE `offerta`
  ADD CONSTRAINT `offertautente_fk` FOREIGN KEY (`emittente`) REFERENCES
`utente` (`id_utente`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `offertalavoro_fk` FOREIGN KEY (`occupazione`) REFERENCES
`lavoro` (`id_lavoro`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `offertacitta_fk` FOREIGN KEY (`luogo`) REFERENCES `citta`
(`id_citta`) ON DELETE CASCADE ON UPDATE CASCADE;
```

- CONSULTAZIONE(id_utente, id_offerta)

```
CREATE TABLE `consultazione` (
  `id_utente` smallint(20) DEFAULT NULL,
  `id_offerta` smallint(20) DEFAULT NULL,
  KEY `id_utente` (`id_utente`),
  KEY `id_offerta` (`id_offerta`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE `consultazione`
  ADD CONSTRAINT `consultazioneutente_fk` FOREIGN KEY (`id_utente`)
REFERENCES `utente` (`id_utente`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `consultazioneofferta_fk` FOREIGN KEY (`id_offerta`)
REFERENCES `offerta` (`id_offerta`) ON DELETE CASCADE ON UPDATE CASCADE;
```

- GRUPPO(id_gruppo, affinita, nome)

```
CREATE TABLE `gruppo` (
  `id_gruppo` smallint(10) NOT NULL AUTO_INCREMENT,
  `affinita` smallint(20) NOT NULL,
  `nome_gruppo` varchar(30) NOT NULL,
  PRIMARY KEY (`id_gruppo`),
  KEY `citta_lavoro` (`affinita`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE `gruppo`
  ADD CONSTRAINT `cittagruppo_fk` FOREIGN KEY (`affinita`) REFERENCES
`lavoro` (`id_lavoro`) ON DELETE CASCADE ON UPDATE CASCADE;
```

- APPARTENENZA(id_utente, id_gruppo)

```
CREATE TABLE `appartenenza` (
  `id_utente` smallint(20) DEFAULT NULL,
  `id_gruppo` smallint(20) DEFAULT NULL,
  KEY `id_utente` (`id_utente`),
  KEY `id_gruppo` (`id_gruppo`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE `appartenenza`
  ADD CONSTRAINT `appartenenzautente_fk` FOREIGN KEY (`id_utente`) REFERENCES
`utente` (`id_utente`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `appartenenzagruppo_fk` FOREIGN KEY (`id_gruppo`) REFERENCES
`gruppo` (`id_gruppo`) ON DELETE CASCADE ON UPDATE CASCADE;
```

- NODO(id_nodo, contatto, utente)

```
CREATE TABLE `nodo` (
  `id_nodo` smallint(20) NOT NULL AUTO_INCREMENT,
  `contatto` smallint(20) NOT NULL,
  `utente` smallint(20) NOT NULL,
  PRIMARY KEY (`id_nodo`),
  KEY `contatto` (`contatto`),
  KEY `utente` (`utente`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE `nodo`
  ADD CONSTRAINT `nodocontatto_fk` FOREIGN KEY (`contatto`) REFERENCES
`contatto` (`id_contatto`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `nodoutente_fk` FOREIGN KEY (`utente`) REFERENCES `utente`
(`id_utente`) ON DELETE CASCADE ON UPDATE CASCADE;
```

- SECONDO(id_utente, id_nodo)

```
CREATE TABLE `secondo` (
  `id_utente` smallint(20) DEFAULT NULL,
  `id_nodo` smallint(20) DEFAULT NULL,
  KEY `id_utente` (`id_utente`),
  KEY `id_nodo` (`id_nodo`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE `secondo`
  ADD CONSTRAINT `secondoutente_fk` FOREIGN KEY (`id_utente`) REFERENCES
`utente` (`id_utente`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `secondonodo_fk` FOREIGN KEY (`id_nodo`) REFERENCES `nodo`
(`id_nodo`) ON DELETE CASCADE ON UPDATE CASCADE;
```

- TERZO(id_utente, id_nodo, connessione)

```
CREATE TABLE `terzo` (
  `id_utente` smallint(20) DEFAULT NULL,
  `id_nodo` smallint(20) DEFAULT NULL,
  `connessione` smallint(20) DEFAULT NULL,
  KEY `id_utente` (`id_utente`),
  KEY `id_nodo` (`id_nodo`),
  KEY `connessione` (`connessione`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE `terzo`
  ADD CONSTRAINT `terzoutente_fk` FOREIGN KEY (`id_utente`) REFERENCES
`utente` (`id_utente`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `terzonodo_fk` FOREIGN KEY (`id_nodo`) REFERENCES `nodo`
(`id_nodo`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `connessione_fk` FOREIGN KEY (`connessione`) REFERENCES
`utente` (`id_utente`) ON DELETE CASCADE ON UPDATE CASCADE;
```

Operazioni di aggiornamento

- Inserire un nuovo utente:

```
INSERT INTO utente VALUES
(`id_utente`, `nome`, `cognome`, `sesso`, `email`, `pass`, `data_nascita`,
`provenienza`, `occupazione`, `luogo_lavoro`, `curriculum`, `account`)
```

- Eliminare un utente:

```
DELETE FROM utente
WHERE id_utente = x
```

- Modificare informazioni personali:

```
UPDATE utente
SET curriculum = ‘nuovo testo’, luogo_lavoro = x
WHERE id_utente = y
```

- Aggiungere utente a lista contatti:

```
INSERT INTO collaborazione (id_utente, id_contatto) VALUES
(x, y)
```

- Pubblicare un’offerta di lavoro:

```
INSERT INTO offerta (id_offerta, emittente, occupazione, num_posti, luogo,
durata) VALUES
(`id_offerta`,`emittente`,`occupazione`,`num_posti`,`luogo`,`durata`)
```

- Rimuovere un’offerta di lavoro:

```
DELETE FROM offerta
WHERE id_offerta = x
```

Operazioni di interrogazione

- Elencare nome e cognome di tutti gli utenti:

```
SELECT nome, cognome
FROM utente
```

- Elencare nome, cognome ed occupazione degli utenti:

```
SELECT u.nome, cognome, l.nome_lavoro
FROM utente u, lavoro l
WHERE u.occupazione = l.id_lavoro
```

- Elencare id di ogni contatto con nome e cognome dell'utente che indica:

```
SELECT con.id_contatto, u.nome, cognome
FROM contatto con JOIN utente u
ON con.utente = u.id_utente
```

- Stampare la lista contatti di un utente, dato il suo id:

```
SELECT u.nome, cognome, c.id_contatto
FROM utente u JOIN collaborazione c
WHERE u.id_utente = x AND c.id_utente = u.id_utente
```

- Elencare di quali utenti è connessione di secondo grado un utente (x):

```
SELECT CONCAT(utente.nome, ' ', cognome) AS connessioni
FROM utente
WHERE utente.id_utente IN (SELECT id_utente
                           FROM secondo
                           WHERE secondo.id_nodo IN (SELECT id_nodo
                                                     FROM nodo
                                                     WHERE utente = x))
```

- Elencare i nomi dei gruppi con una determinata affinità:

```
SELECT nome_gruppo
FROM gruppo
WHERE affinita = x
```

- Elencare i partecipanti ad un gruppo specifico:

```
SELECT CONCAT(utente.nome, ' ', cognome) AS elementi
FROM utente
WHERE id_utente IN (SELECT id_utente
                    FROM appartenenza
                    WHERE appartenenza.id_gruppo IN
                              (SELECT id_gruppo
                               FROM gruppo
                               WHERE nome_gruppo = 'Gruppo Sviluppatori'))
```

- Consultare le offerte di lavoro con almeno un posto disponibile filtrando in base alla figura professionale richiesta:

```
SELECT occupazione, num_posti
FROM offerta
WHERE occupazione = x AND num_posti >= 1
```

- Stampare nome, cognome e data di nascita degli utenti nati dopo l'anno x:

```
SELECT nome, cognome, data_nascita
FROM utente
WHERE data_nascita >= 'x-01-01'
```

- Autenticazione utente:

```
SELECT id_utente
FROM utente
WHERE email = 'd.cortellucci@campus.uniurb.it'
      AND pass = 'progettobdd'
```

PROGETTAZIONE FISICA

La progettazione fisica è l'ultima fase dell'implementazione di una base di dati. Essa studia l'installazione dei dati su dispositivi di memoria di massa (come ad esempio gli hard disk, o dischi), in particolare analizza in che modo vengono salvati i dati e dove.

Verranno analizzate le prestazioni valutando il numero di operazioni di I/O necessarie per eseguire una determinata interrogazione/aggiornamento. Si prendono in considerazione tre tipi di organizzazione dei dati.

Organizzazione indicizzata

Questo modo prevede la presenza di indici, cioè piccoli blocchi di memoria contenenti gli indirizzi dei blocchi ove risiedono i record del file a cui appartengono. L'indicizzazione dei record permette di fare riferimento all'*i*-esimo record tramite il corrispondente indice *i*.
L'efficienza delle interrogazioni rappresenta il principale vantaggio di questo metodo; di contro, il costo degli aggiornamenti è alto, poiché è necessario aggiornare anche gli indici ad ogni operazione.

Si consideri la tabella *utente*:

```
CREATE TABLE `utente` (
  `id_utente` smallint(3) NOT NULL AUTO_INCREMENT,
  `nome` varchar(30) NOT NULL,
  `cognome` varchar(30) NOT NULL,
  `sesso` enum('m','f') NOT NULL,
  `email` varchar(30) NOT NULL,
  `pass` varchar(30) NOT NULL,
  `data_nascita` date NOT NULL,
  `provenienza` smallint(3) DEFAULT NULL,
  `occupazione` smallint(3) NOT NULL,
  `luogo_lavoro` smallint(3) DEFAULT NULL,
  `curriculum` varchar(300) DEFAULT NULL,
  `account` enum('free','premium') NOT NULL
);
```

Numero di byte necessari per memorizzare un'istanza della tabella *utente*, supponendo le seguenti richieste di memoria:

- ➔ smallint 2 byte * 4 = 8,
- ➔ enum 1 byte * 2 = 2,
- ➔ varchar(30) 30 byte * 3 = 90,
- ➔ varchar(300) 300 byte * 1 = 300,
- ➔ date 10 byte * 1 = 10.

Il totale è di 410 byte, si vuole ora calcolare il numero di pagine necessarie per memorizzare tutte le possibili istanze della tabella *utente* (NP); ponendo la dimensione di ogni pagina di 1kB e considerando 1000 istanze (NT):
NP_{utente}=NT * 410B / 1kB = 1000 * 410B/ 1kB = 400,39 → 401 pagine .

Sulla base di questo valore si valuta il numero di accessi per il caso medio(NP+1/2) ed il caso pessimo (è necessario scorrere tutti gli indici per trovare il record corrispondente, oppure non esiste). Non si tiene in considerazione il caso ottimo, poiché inutile ai fini dell'analisi.
Gli indici fanno riferimento ad un attributo di un record (la chiave di ricerca), per cui, volendo stimare il costo degli accessi, bisogna tener conto di due casi:

1) Il file è **ordinato** rispetto all'attributo scelto, si fa distinzione tra:
- esistenza dell'attributo: $\log_2(NP) = 8{,}64 \rightarrow 9$ accessi;
- non esistenza dell'attributo: $\log_2(NP) + 1 = 10$ accessi.

2) Il file è **disordinato** rispetto all'attributo scelto, quindi si ha:
- caso medio: (NP + 1 / 2) = 201 accessi;
- caso pessimo: NP = 401 accessi.

Si trae un vantaggio notevole se il file è già ordinato, altrimenti bisognerebbe ordinarlo, utilizzando risorse.

Organizzazione sequenziale

La struttura sequenziale è una struttura molto semplice: i record vengono memorizzati sul disco esattamente nell'ordine in cui si presentano. Essa si dice infatti disordinata, o *heap*.
Sebbene molto efficiente per le operazioni di inserimento, il contro maggiore è dato dalle scarse prestazioni della ricerca, poiché essa richiede una ricerca sequenziale lungo tutti i blocchi del file.

Si riprendano i valori calcolati precedentemente per gli indici, anche in questo caso si considerano due possibilità:

1) File **ordinato**:
- si fa una semplice media matematica, quindi (NP + 1) / 2 = 201 accessi.

2) File **disordinato**:
- caso medio: (NP + 1 / 2) = 201 accessi;
- caso pessimo: NP = 401 accessi.

Si noti che il caso disordinato comporta le stesse prestazioni del caso disordinato nell'organizzazione ad indici.

Organizzazione ad albero (B+-tree)

Per organizzazione ad albero si intende una struttura che comprende nodi, foglie e collegamenti tra di essi; in particolare si fa riferimento ad una particolare struttura, la *b+-tree*, una struttura ad albero

bilanciato in cui le chiavi sono contenute solo nelle foglie ed i nodi primari contengono dei puntatori atti a descrivere il percorso da seguire per arrivare ad una chiave particolare.

Si vuole analizzare un albero con indice clustered nella ricerca, ipotizzando i seguenti valori:

- l'utilizzo *u* delle foglie pari al 70%;
- il grado *g* dell'albero uguale a 3;
- NT = 1000, come già detto;
- *len(k)* lunghezza della chiave;
- *len(p)* lunghezza dei puntatori ai record;
- *len(k) + len(p)* in questo caso è uguale allo spazio di occupazione della tabella utente (410B);
- *D* dimensione delle foglie (cioè la dimensione delle pagine del disco), in questo calcolo 1kB;
- NL numero di foglie, con $NL = [NT * (len(k) + len(p)] / (D * u)$;
- *h* altezza minima dell'albero, con $h = log_{2g+1} (NL) + 1$;
- NP = 401, numero di pagine occupato dalla tabella *utente*;
- Ca_{id} costo di accesso di un indice clustered su un albero B+.

Dichiarati tutti i dati necessari, si comincia calcolando il numero di foglie NL:

$NL = [NT * (len(k) + len(p)] / (D * u) = [1000 * 410B] / (1kB * 0.70) = 2242,18 \rightarrow 2243$

poi si passa all'altezza minima dell'albero:

$h = log_{2g+1} (NL) + 1 = log_{2g+1} (2243) + 1 = 5,56 + 1 = 6,56 \rightarrow 7$

infine si calcola il costo di accesso all'indice:

$Ca_{id\_utente} = h -1 + [EK/NK * NL] + [EK/NK * NP]$

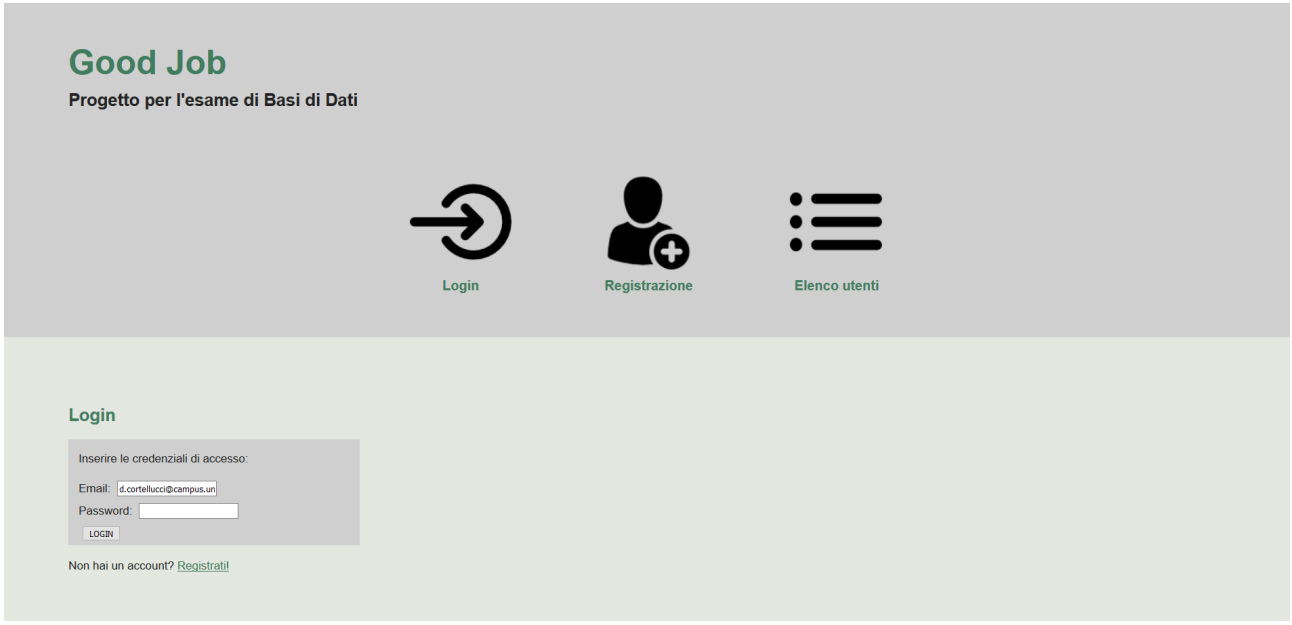
dove EK rappresenta le chiavi distinte che si vogliono reperire (in questo caso 1), NK il numero di chiavi totali (1 * 1000 istanze = 1000), dunque:

$Ca_{id\_utente} = 7 – 1 + [(1/1000 * 2243] + [1/1000 * 401] = 6 + 2,24 + 0,40 = 8,64 \rightarrow 9$

che è uguale nel caso di indicizzazione con file ordinato, con la differenza che non c'è necessità che il file sia già ordinato (caso particolare).

INTERFACCIA

Per il progetto è stata sviluppata anche una interfaccia web con l'uso dei linguaggi HTML, CSS e PHP.



Tale interfaccia permette di eseguire tre operazioni sul database:

1. Login (autenticazione)
2. Elenco utenti (visualizzare tutti gli utenti presenti nel database)
3. Registrazione (inserimento nuovo utente)