UNIVERSITÀ
DI SIENA
1240

Basi di Dati

Appunti delle Esercitazioni

Corso di Laurea in
INGEGNERIA INFORMATICA E DELL'INFORMAZIONE

Indice

Capitolo 1

Esercitazione

1.1 I dipendenti

Progettare il modello relazionale dei dipendenti di un'azienda, memorizzando

- per ogni dipendente: nome, ruolo, stipendio, sede e città di residenza
- per ogni sede: il nome della sede e la città dove si trova
- per ogni città: il nome e il numero di abitanti

Il modello relazionale risulta

- tabella **Dipendenti**

codiceDip	**nomeDip**	**ruolo**	**sede**	**cittàResidenza**	**stipendio**

- tabella **Sedi**

codiceSede	**nomeSede**	**città**

- tabella **Città**

codiceCittà	**nomeCittà**	**numeroAbitanti**

e lo schema logico risulta

- **Dipendenti(codiceDip, nomeDip, ruolo, sede, cittàResidenza, stipendio)**
- **Sedi(codiceSede, nomeSede, città)**
- **Città(codiceCittà, nomeCittà, numeroAbitanti)**

con i seguenti vincoli di integrità referenziale

- **Dipendenti.sede → Sedi.codiceSede**
- **Dipendenti.cittàResidenza → Città.codiceCittà**
- **Sedi.città → Città.codiceCittà**

Scrivere in algebra relazionale con theta-join e calcolo sulle tuple, le interrogazioni:

1. determinare le città in cui lavorano e il nome di tutti i dipendenti che guadagnano più di 1700
   - algebra relazionale con theta-join

$$\pi_{\text{nomeDip, nomeCittà}}(\sigma_{\text{stipendio}>1700}(\text{Dipendenti}) \bowtie_{\text{sede = codiceSede}} (\text{Sedi}) \bowtie_{\text{città = codiceCittà}} (\text{Città}))$$

   - calcolo sulle tuple

$$\{\text{Città: c(nomeCittà), Nome: d.(nomeDip)} \mid \mid \text{d(Dipendenti), s(Sedi), c(Città)} \mid \text{d.Sede} = \text{s.codiceSede} \wedge \text{s.città} = \text{c.codiceCittà} \wedge \text{d.stipendio} > 1700\}$$

2. determinare in nome dei dipendenti che non sono residenti a Siena
   - algebra relazionale con theta-join

$$\text{DipendentiDiSiena} = \pi_{\text{nomeDip}}(\text{Dipendenti} \bowtie_{\text{cittàResidenza = codiceCittà}} (\sigma_{\text{nomeCittà}\neq\text{'Siena'}}(\text{Città})))$$

   - calcolo sulle tuple

$$\{\text{Nome: d.(nomeDip)} \mid \text{d(Dipendenti), c(Città)} \mid \text{d.cittàResidenza} = \text{c.codiceCittà} \wedge \text{c.nomeCittà} \neq \text{'Siena'}$$

3. determinare il nome delle sedi che non hanno dipendenti residenti a Siena
   - algebra relazionale con theta-join

$$\text{SediConDipSiena} = \pi_{\text{codiceSede, nomeSede}}(\text{Sedi} \bowtie_{\text{codiceSede = sede}})(Dipendenti \bowtie_{\text{cittàResidenza = codiceCittà}} (\sigma_{\text{nomeCittà = 'Siena'}}(Citt)))$$
$$\pi_{\text{nomeSede}}(\pi_{\text{codiceSede, nomeSede}}(Sedi) - SediConDipSiena)$$

   - calcolo sulle tuple

$$\{\text{Nome: s.(nomeSede)} \mid \text{s(Sedi)} \mid \neg\ \exists\ \text{c(Città)}(\exists\ \text{d(Dipendenti)}) \text{ d.sede} = \text{s.codiceSede} \ \wedge\ \text{d.cittàResidenza} = \text{c.codiceCittà} \wedge \text{c.nomeCittà} = \text{'Siena'})\}$$

4. determinare il nome delle sedi che hanno almeno due dipendenti

   - algebra relazionale con theta-join

$$\text{Dipendenti1} = \rho_{\text{codiceDip1}\leftarrow\text{codiceDip,nomeDip1}\leftarrow\text{nomeDip,ruolo1}\leftarrow\text{ruolo,}}$$

$$_{\text{cittàResidenza1}\leftarrow\text{cittàResidenza,stipendio1}\leftarrow\ \text{stipendio}}(\text{Dipendenti})$$

$$\text{Dipendenti2} = \rho_{\text{codiceDip2}\leftarrow\text{codiceDip,nomeDip2}\leftarrow\text{nomeDip,ruolo2}\leftarrow\text{ruolo,}}$$

$$_{\text{cittàResidenza2}\leftarrow\text{cittàResidenza,stipendio2}\leftarrow\ \text{stipendio}}(\text{Dipendenti})$$

$$\text{CodiceSediConAlmenoDueDip :}$$

$$\rho_{\text{sede}\leftarrow\text{sede1}}(\text{Dipendenti} \bowtie_{\text{codiceDip1}\neq\text{codiceDip2}\ \wedge\ \text{sede1=sede2}} \text{Dipendenti2}))$$

$$\pi_{\text{nomeSede}}(\text{Sedi} \bowtie_{\text{sede = codiceSede}} \text{CodiceSediConAlmenoDueDip})$$

   - calcolo sulle tuple

$$\{\text{Nome: s(nomeSede)} \mid \text{s(Sedi), d1(Dipendenti), d2(Dipendenti)} \mid$$

$$\text{d1.sede} = \text{s.codiceSede} \wedge \text{d2.sede} = \text{s.codiceSede} \wedge \neg(\text{d1.cod} = \text{d2.cod})\}$$

5. determinare il nome delle sedi che hanno esattamente un dipendente

   - algebra relazionale con theta-join

$$\text{SediSenzaDipendenti} = \pi_{\text{codiceSede}}(\text{Sedi}) - \rho_{\text{codiceSede}\leftarrow\text{sede}}(\pi_{\text{sede}}(\text{Dipendenti}))$$

$$\text{Dipendenti1} = \rho_{\text{codiceDip1}\leftarrow\text{codiceDip,nomeDip1}\leftarrow\text{nomeDip,ruolo1}\leftarrow\text{ruolo,}}$$

$$_{\text{cittàResidenza1}\leftarrow\text{cittàResidenza,stipendio1}\leftarrow\ \text{stipendio}}(\text{Dipendenti})$$

$$\text{Dipendenti2} = \rho_{\text{codiceDip2}\leftarrow\text{codiceDip,nomeDip2}\leftarrow\text{nomeDip,ruolo2}\leftarrow\text{ruolo,}}$$

$$_{\text{cittàResidenza2}\leftarrow\text{cittàResidenza,stipendio2}\leftarrow\ \text{stipendio}}(\text{Dipendenti})$$

$$\text{CodiceSediConAlmenoDueDip :}$$

$$\rho_{\text{sede}\leftarrow\text{sede1}}(\text{Dipendenti} \bowtie_{\text{codiceDip1}\neq\text{codiceDip2}\ \wedge\ \text{sede1=sede2}} \text{Dipendenti2}))$$

$$\pi_{\text{nomeSede}}(\text{Sedi} \bowtie_{\text{sede = codiceSede}}$$

$$((\text{Sedi - SediSenzaDipendenti}) - \text{CodiceSediConAlmenoDueDip})$$

   - calcolo sulle tuple

$$\{\text{Nome: s.(nomeSede)} \mid \text{s(Sedi)} \mid$$

$$\exists\ \text{d1(Dipendenti) d1.sede} = \text{s.codiceSede} \wedge$$

$$\neg(\exists\ \text{d2(Dipendenti) d2.sede} = \text{s.codiceSede} \wedge\ \neg(\text{d1.cod} = \text{d2.cod}))\}$$

1.2 La videoteca

Progettare il modello relazionale di una videoteca, memorizzando

- per ogni film: titolo, registra, anno di produzione e costo del noleggio
- per ogni attore o registra: nome, cognome, sesso, data di nascita, nazionalità
- per gli attori si memorizzi anche l'insieme dei film a cui ha partecipato indicando anche il personaggio interpretato

Il modello relazionale risulta

- tabella **Film**

codiceFilm	**titolo**	**regista**	**anno**	**costoNoleggio**

- tabella **Artisti**

codiceArtista	**cognome**	**nome**	**sesso**	**dataDiNascita**	**nazionalità**

- tabella **Interpretazioni**

codiceFilm	**codiceAttore**	**personaggio**

e lo schema logico risulta

- **Film(codiceFilm, titolo, regista, anno, costoNoleggio)**
- **Artisti(codiceArtista, cognome, nome, sesso, dataDiNascita, nazionalità)**
- **Interpretazioni(codiceFilm, codiceAttore, personaggio)**

con i seguenti vincoli di integrità referenziale

- **Film.regista → Artisti.codiceArtista**
- **Interpretazioni.codiceFilm → Film.codiceFilm**
- **Interpretazioni.codiceAttore → Artisti.codiceArtista**

Scrivere in algebra relazionale e calcolo sulle tuple, le seguenti interrogazioni:

1. determinare i titoli e anno dei film diretti da Steven Spielberg
   - algebra relazionale con join naturale

$$\pi_{\text{titolo, anno}}(\rho_{\text{regista}\leftarrow\text{codiceArtista}}(\sigma_{\text{cognome = 'Spielberg'}\wedge\text{nome = 'Steven'}}(\text{Artisti})) \bowtie \text{Film})$$

2. determinare i titoli dei film interpretati da Henry Fonda

   - algebra relazionale con join naturale

   $$\pi_{\text{titolo}}(\text{Interpretazioni} \bowtie \text{Film} \bowtie \rho_{\text{codiceAttore} \leftarrow \text{codiceArtista}}(\sigma_{\text{cognome = 'Fonda'} \wedge \text{nome = 'Henry'}}(\text{Artisti})))$$

   - calcolo su tuple

   $$\{\text{f.(titolo)} \mid \text{f(Film), i(Interpretazioni), a(Artisti)} \mid \text{f.codiceFilm} = \text{i.codiceFilm} \wedge \text{i.codiceAttore} = \text{a.codiceArtista} \wedge \text{a.cognome} = \text{'Fonda'} \wedge \text{a.nome} = \text{'Henry'}\}$$

   oppure

   $$\{\text{f.(titolo)} \mid \text{f(Film)} \mid \exists\ \text{i(Interpretazioni)}, \exists\ \text{a(Artisti)} \ \text{f.codiceFilm} = \text{i.codiceFilm} \wedge \text{i.codiceAttore} = \text{a.codiceArtista} \wedge \text{a.cognome} = \text{'Fonda'} \wedge \text{a.nome} = \text{'Henry'}\}$$

3. determinare i titoli dei film per i quali il regista sia stato anche interprete

   - algebra relazionale con join naturale

   $$\pi_{\text{titolo}}(\sigma_{\text{regista = codiceAttore}}(\text{Interpretazioni} \bowtie \text{Film}))$$

   - calcolo su tuple

   $$\{\text{f.(titolo)} \mid \text{f(Film), i(Interpretazioni)} \mid \text{f.codiceFilm} = \text{i.codiceFilm} \wedge \text{i.codiceAttore} = \text{f.regista}\}$$

   oppure

   $$\{\ \text{f.(titolo)} \mid \text{f(Film)} \mid \exists\ \text{i(Interpretazioni)} \ \text{f.codiceFilm} = \text{i.codiceFilm} \wedge \text{i.codiceAttore} = \text{f.regista}\}$$

4. determinare i titoli dei film in cui gli attori noti sono tutti dello stesso sesso

   - algebra relazionale con join naturale: non si possono confrontare valori fra tuple diverse in algebra relazionale e si devono usare gli operatori insiemistici (sono tutti i film meno quelli che contengono sia attori maschili che femminili)

   $$\text{Interpreti} = \rho_{\text{codiceAttore = codiceArtista}}(\text{Artisti}) \bowtie \text{Interpretazioni} \bowtie \text{Film}$$

   $$\pi_{\text{titolo}}(\text{Film}) - (\pi_{\text{titolo}}(\sigma_{\text{sesso = 'M'}}(\text{Interpreti})) \cap \sigma_{\text{titolo}}(\sigma_{\text{sesso = 'F'}}(\text{Interpreti})))$$

   - calcolo su tuple

   $$\{\text{f.(titolo)} \mid \text{f(Film)} \mid \neg\ (\exists\ \text{i1(Interpretazioni)}\ \exists\ \text{a1(Artisti)}\ \exists\ \text{i2(Interpretazioni)}\ \exists\text{a2(Artisti)}) \ \text{f.codiceFilm} = \text{i1.codiceFilm} \wedge \text{i1.codiceAttore} = \text{a1.codiceArtista} \wedge \text{f.codiceFilm} = \text{i2.codiceFilm} \wedge \text{i2.codiceAttore} = \text{a2.codiceArtista} \wedge \text{a2.sesso} <> \text{a1.sesso}\}$$

5. determinare il nome e il cognome dell'artista più giovane

   - algebra relazione con join naturale

$$\text{Artisti1} = \rho_{\text{codiceArtista1, dataNascita1} \leftarrow \text{codiceArtista, dataNascita}}$$
$$(\pi_{\text{codiceArtista, dataNascita}}(\text{Artisti}))$$
$$\text{Artisti2} = \rho_{\text{codiceArtista2, dataNascita2} \leftarrow \text{codiceArtista, dataNascita}}$$
$$(\pi_{\text{codiceArtista, dataNascita}}(\text{Artisti}))$$
$$\text{NonIlPiùGiovane} = \rho_{\text{codiceArtista} \leftarrow \text{codiceArtista1}}(\pi_{\text{codiceArtista1}}($$
$$\sigma_{\text{dataNascita1} < \text{DataNascita2}}(\text{Artisti1} \bowtie \text{Artisti2})))$$
$$\pi_{\text{nome, cognome}}(\text{Artisti} \bowtie (\pi_{\text{codiceArtista}}(\text{Artisti}) - \text{NonIlPiùGiovane}))$$

1.3 Commissioni parlamentari

Lo schema logico risulta

- **Deputati(<u>codice</u>, nome, commissione, provincia, collegio)**
- **Commissioni(<u>numero</u>, nome, presidente)**
- **Collegi(<u>provincia</u>, <u>numero</u>, nome)**
- **Province(<u>sigla</u>, nome, regione)**
- **Regioni(<u>codice</u>, nome)**

con i seguenti vincoli di integrità referenziale

- **Deputati.commissione → Commissione.numero**
- **Deputati.provincia → Province.sigla**
- **Deputati.(provincia, collegio) → Collegi.(provincia, numero)**
- **Collegi.provinvia → Province.sigla**
- **Province.regione → Regioni.codice**

Scrivere in algebra relazionale e calcolo sulle tuple, le seguenti interrogazioni:

1. determinare il nome e cognome dei presidenti di commissioni cui partecipa almeno un deputato eletto in una provincia della Sicilia

   - algebra relazionale con theta join

     (a) le province della Sicilia

$$\text{ProvinceSicilia} = \pi_{\text{sigla}}(\text{Province} \bowtie_{\text{regione = codice}}$$
$$\pi_{\text{codice}}(\sigma_{\text{nome='Sicilia'}}(\text{Regioni})))$$

(b) i deputati della Sicilia

$$\text{DeputatiSicilia} = \text{Deputati} \bowtie_{provincia=sigla} \text{ProvinceSicilia}$$

(c) i presidenti delle commissioni con deputati siciliani

$$\text{PresidentiSicilia} = \pi_{presidente}(\text{Commissioni} \bowtie_{\text{numero=commissione}} \pi_{\text{commissione}}(\text{DeputatiSicilia}))$$

(d) nome e cognome dei presidenti delle commissioni

$$\pi_{\text{nome, cognome}}(\text{Deputati} \bowtie_{\text{presidente=codice}} \text{PresidentiSicilia})$$

quindi complessivamente

$$\pi_{\text{nome, cognome}}(\text{Deputati} \bowtie_{\text{presidente=codice}} \pi_{\text{presidente}}(\text{Commissioni} \bowtie_{\text{numero=commissione}} \pi_{\text{commissione}}(\text{Deputati} \bowtie_{\text{provincia=sigla}} \pi_{\text{sigla}}(\text{Province} \bowtie_{\text{regione=codice}} \pi_{\text{codice}}(\sigma_{\text{nome='Sicilia'}}(\text{Regioni})))))$$

- calcolo su tuple

$$\{\text{p.(nome, cognome)} \mid \text{p(Deputati)} \mid \exists\ \text{c(Commissioni)}\ (\text{p.codice=c.presidente} \wedge \exists\ \text{d(Deputati)}\ (\text{d.commissione=c.numero} \wedge \exists\ \text{p(Province)}\ (\text{d.provincia=p.sigla} \wedge \exists\ \text{r(Regioni)}\ (\text{p.regione=r.codice} \wedge \text{r.nome} = \text{'Sicilia'}))))\}$$

2. determinare il nome e il cognome dei deputati della commissione Bilancio

- algebra relazionale con theta join

$$\pi_{\text{nome, cognome}}(\text{Deputati} \bowtie_{\text{commissione=numero}} \pi_{\text{numero}}(\sigma_{\text{nome='Bilancio'}}(\text{Commissioni})))$$

- calcolo su tuple

$$\{\text{d.(nome, cognome)} \mid \text{d(Deputati), d(Commissioni)} \mid \text{d.commissione} = \text{c.numero} \wedge \text{c.nome} = \text{'Bilancio'}$$

3. determinare le regioni in cui vi sia un solo collegio, indicando nome e cognome del deputato ivi eletto

- algebra relazionale con join naturale: non si possono confrontare tuple diverse di una stessa relazione quindi occorre utilizzare il join per creare una tabella con coppie di regioni e trovare quelle che hanno almeno due collegi diversi (le regioni con un solo collegio saranno quelle che non hanno almeno due collegi)

(a) determinare la tabella dei collegi con la regione

$$\text{CollegiRegioni} = \pi_{\text{provincia, numero, regione}}(\text{Collegi} \bowtie$$

$$\rho_{\text{provincia, nomeProv}\leftarrow\text{sigla, nome}}(\text{Province}))$$

(b) si trovano le coppie di collegi nella stessa regione

$$\text{CoppieCollegi} = \rho_{\text{pr1, nr1}\leftarrow\text{provincia, numero}}(\text{CollegiRegioni}) \bowtie$$

$$\rho_{\text{pr2, nr2}\leftarrow\text{provincia, numero}}(\text{CollegiRegioni})$$

(c) si selezionano le regioni con coppie di collegi distinti

$$\text{RegioniMul} = \pi_{\text{regione}}(\sigma_{\neg((\text{pr1}=\text{pr2})\wedge(nr1=nr2))}(\text{CoppieCollegi}))$$

(d) si trovano le regioni con un solo collegio

$$\text{RegioniUni} = \text{Regioni} - \rho_{\text{codice}\leftarrow\text{regione}}(\text{RegioniMul}) \bowtie \text{Regioni}$$

(e) si trovano i deputati eletti in tali regioni

$$\pi_{\text{nome, cognome, nomeRegione}}(\rho_{\text{provEle}\leftarrow\text{provincia}}(\text{Deputati})$$

$$\bowtie_{\text{provEle}=\text{provincia}\wedge\text{collegio}=\text{numero}} \text{CollegioRegioni}$$

$$\bowtie_{\text{regione}=\text{codice}} \rho_{\text{nomeRegione}\leftarrow\text{nome}}(\text{RegioniUni}))$$

- calcolo su tuple

$$\{\text{d.(nome, cognome), r.(nome)} \mid \text{d(Deputati), p(Province), r(Regioni)} \mid$$

$$\text{d.provincia=p.sigla} \wedge \text{r.codice=p.regione} \wedge$$

$$\neg(\exists\ \text{c2(Collegi)}\ \exists\ \text{p2(Province)}$$

$$(\text{c2.provincia=p2.sigla} \wedge \text{p2.regione=p.regione} \wedge$$

$$\neg((\text{c2.provincia=d.provincia}) \wedge (\text{c2.numero=d.collegio}))))\}$$

4. determinare i collegi in cui è stato eletto almeno un deputato che ha un omonimo nella stessa regione

- algebra relazionale con join naturale: non si possono confrontare tuple diverse di una stessa relazione quindi occorre utilizzare il join per creare una tabella con coppie di deputati e trovare quelle che hanno i due nomi propri uguali e la stessa regione

(a) si associa la regione al collegio di elezione di ogni deputato

$$\text{DeputatiRegioni} = \pi_{\text{codice, nome, cognome, provincia, collegio, regione}}($$

$$\text{Deputati} \bowtie \rho_{\text{provincia, nomeProv}\leftarrow\text{sigla, nome}}(\text{Province}))$$

(b) si considerano le coppie di deputati eletti in collegi della stessa regione

$$\text{CoppieDeputati} =$$

$$\rho_{\text{cod1,cog1,nom1,prov1,coll1,reg1}\leftarrow\text{codice,cognome,nome,provincia,collegio,regione}}($$

$$\text{DeputatiRegioni}) \bowtie$$

$$\rho_{\text{cod2,cog2,nom2,prov2,coll2,reg2}\leftarrow\text{codice,cognome,nome,provincia,collegio,regione}}($$

$$\text{DeputatiRegioni})$$

(c) si selezionano le coppie di deputati diversi con lo stesso nome proprio e si estrae il collegio

$$\rho_{\text{provincia, collegio}\leftarrow\text{prov1,coll1}}(\pi_{\text{prov1,coll1}}($$

$$\sigma_{\text{nom1=nom2}\wedge\text{reg1=reg2}\wedge\neg(\text{cod1=cod2})}(\text{CoppieDeputati})))$$

- calcolo su tuple

$$\{\text{d.(provincia, collegio)} \mid \text{d(Deputati), p(Province)} \mid$$

$$\text{d.provincia=p.sigla}\wedge$$

$$(\exists\ \text{d2(Deputati)}\ \exists\ p\text{2(Province)}$$

$$(\text{d2.provincia=p2.sigla} \wedge \text{p2.regione=p.regione}\wedge$$

$$\neg(\text{d.codice=d2.codice}) \wedge (\text{d.nome=d2.nome})))\}$$

Capitolo 2

SQL DDL

2.1 La biblioteca

Dato il seguente schema logico

- **Libri(titolo, nomeAutore, cognomeAutore, dataDiNascitaAutore, dataPubblicazionePrimaVersione, dataPubblicazioneQuestaVersione, sede, ISBN, prezzo, numeroPagine)**
- **Autori(nome, cognome, dataDiNascita, LuogoDiNascita)**
- **Sedi(id, indirizzo)**

e i seguenti vincoli di integrità referenziale

- **Libri.nomeAutore, Libri.cognomeAutore, Libri.dataDiNascitaAutore → Autori.nome, Autori.cognome, Autori.dataDiNascita**
- **Libri.sede → Sedi.id**

si scriva il codice SQL che crea le tabelle corrispondenti. Inoltre, considerare che

- il valore di **dataPubblicazionePrimaVersione** deve essere uguale o precedente a **dataPubblicazioneQuestaVersione**
- il codice **ISBN** è lungo 13 caratteri
- il valore di default per il **prezzo** e **numeroPagine** è 0
- il **prezzo** e **numeroPagine** non può essere negativo

code 2.1: biblioteca.sql

```sql
CREATE TABLE Sedi (
  id SERIAL NOT NULL PRIMARY KEY,
  indirizzo VARCHAR(20)
);

CREATE TABLE Autori (
  nome VARCHAR(20) NOT NULL,
```

```
  cognome VARCHAR(20) NOT NULL,
  dataDiNascita DATE NOT NULL,
  luogoDiNascita VARCHAR(20),
  PRIMARY KEY(nome, cognome, dataDiNascita)
);

CREATE TABLE Libri (
  titolo VARCHAR(20),
  nomeAutore VARCHAR(20),
  cognomeAutore VARCHAR(20),
  dataDiNascitaAutore DATE,
  dataPubblicazionePrimaVersione DATE,
  dataPubblicazioneQuestaVersione DATE CHECK
    (dataPubblicazionePrimaVersione <=
      dataPubblicazioneQuestaVersione),
  sede INTEGER REFERENCES Sedi(id),
  ISBN CHAR(13) NOT NULL PRIMARY KEY,
  prezzo DECIMAL(5,2) DEFAULT 0 CHECK (prezzo >= 0),
  numeroPagine INTEGER DEFAULT 0 CHECK (prezzo >= 0),

  FOREIGN KEY(nomeAutore, cognomeAutore, dataDiNascitaAutore)
  REFERENCES Autori(nome, cognome, dataDiNascita)
);
```

2.2 Studenti e Corsi

Dato il seguente schema logico

- **Studenti(<u>matricola</u>, <u>università</u>, nome, cognome, dataDiNascita, codiceFiscale)**
- **Corsi(<u>id</u>, nome, docente, numeroStudentiPrevisto)**
- **Esami(<u>studente</u>, <u>università</u>, <u>corso</u>, data, voto, lode)**

e i seguenti vincoli di integrità referenziale

- **Esami.studente, Esami.università → Studenti.matricola, Studenti.università**
- **Esami.corso → Corsi.id**

si scriva il codice SQL che crea le tabelle corrispondenti. Inoltre, considerare che

- il voto deve essere compreso fra 0 e 30 e la lode può essere presa solo con 30
- il codice fiscale è unico per ciascun studente ed è lungo 16 caratteri
- nessun attributo può essere nullo eccetto la data di nascita dello studente
- il valore di default per il numero di studenti previsto in **Corsi** è 20

code 2.2: studenti.sql

```sql
CREATE TABLE Studenti (
  matricola VARCHAR(20) NOT NULL,
  università VARCHAR(20) NOT NULL,
  nome VARCHAR(20) NOT NULL,
  cognome VARCHAR(20) NOT NULL,
  dataDiNascita DATE
  codiceFiscale CHAR(16) NOT NULL UNIQUE,

  PRIMARY KEY(matricola, università)
);

CREATE TABLE Corsi (
  id SERIAL NOT NULL PRIMARY KEY,
  nome VARCHAR(20) NOT NULL,
  docente VARCHAR(20) NOT NULL,
  numeroStudentiPrevisto INTEGER DEFAULT 20
);

CREATE TABLE Esami (
  studente VARCHAR(20) NOT NULL
  università VARCHAR(20) NOT NULL,
  corso INT REFERENCES Corsi(id) NOT NULL,
  data DATE NOT NULL,
  voto INTEGER NOT NULL,
  lode BIT NOT NULL CHECK(lode = 0 OR Voto = 30),

  PRIMARY KEY(studente, università, corso),

  FOREIGN KEY(studente, università)
    REFERENCES Studenti(matricola, università)
);
```

Capitolo 3

SQL DML

3.1 Le fatture

Dato il seguente schema logico

- **Clienti(codice, nome, indirizzo, partitaIVA)**
- **Prodotti(codice, nome, descrizione, prezzo)**
- **Fatture(codice, cliente, data)**
- **RigheFattura(codice, fattura, prodotto, quantità, prezzo)**

e i seguenti vincoli di integrità referenziale

- **RigheFattura.fattura → Fatture.codice**
- **RigheFattura.prodotto → Prodotti.codice**
- **Fatture.cliente → Clienti.codice**

si scriva il codice SQL che crea le tabelle corrispondenti. Il codice SQL risulta

code 3.1: fatture.sql

```sql
CREATE TABLE Clienti (
  codice SERIAL NOT NULL PRIMARY KEY,
  nome VARCHAR(20),
  indirizzo VARCHAR(40),
  partitaIVA CHAR(11) UNIQUE
);

CREATE TABLE Prodotti (
  codice SERIAL NOT NULL PRIMARY KEY,
  nome VARCHAR(20),
  descrizione VARCHAR(256),
  prezzo DECIMAL(8,2)
);

CREATE TABLE Fatture (
  codice SERIAL NOT NULL PRIMARY KEY,
```

```
  cliente INTEGER NOT NULL REFERENCES Clienti(codice),
  data DATE
);

CREATE TABLE RigheFattura (
  codice SERIAL NOT NULL PRIMARY KEY,
  fattura INTEGER NOT NULL REFERENCES Fatture(codice),
  prodotto INTEGER NOT NULL REFERENCES Prodotti(codice),
  quantità INTEGER
  prezzo DECIMAL(8,2)
);
```

3.1.1 Interrogazione SQL

Date le tabelle precedenti, si scriva il codice SQL che implementi le seguenti interrogazioni.

3.1.1.1 Numero 1

Ricercare tutti i prodotti acquistati da Pippo, mostrando: la data della fattura, il nome del prodotto e il prezzo come da fattura

```
SELECT Fatture.data, Prodotti.nome, RigheFattura.prezzo
FROM Clienti, Fatture, Prodotti, RigheFattura
WHERE Fatture.codice = RigheFattura.fattura AND
Prodotti.codice = RigheFatture.prodotto AND
Clienti.codice = Fatture.cliente AND
Clienti.nome = 'Pippo'
```

3.1.1.2 Numero 2

Ricercare tutti i prodotti con le rispettive fatture, mostrando: il nome del prodotto, la data della fattura e il prezzo a cui è stato venduto avendo cura di visualizzare anche i prodotti che non sono stati mai venduti

```
SELECT Prodotti.nome, Fatture.data, RigheFattura.prezzo
FROM Prodotti LEFT JOIN RigheFattura
ON (Prodotti.codice = RigheFattura.prodotto)
LEFT JOIN Fatture ON (Fatture.codice = RigheFattura.fattura)
```

Il secondo `JOIN` deve essere un `LEFT JOIN` per fare in modo di ottenere tutti i prodotti che non hanno una fattura, se usassimo `INNER JOIN` allora la riga dei prodotti che non hanno fatture verrebbero eliminate e quindi la tabella finale non risulta corretta.

Capitolo 4

Database relazionale in SQL

4.1 Creazione del database

Creare il seguente database relazionale:

- **Candidato(codice, cognome, nome, viaResidenza, capResidenza, cittaResidenza, dataNascita, luogoNascita)**
- **Telefono(numero, candidato)**
- **TitoloStudio(codice, descrizione)**
- **Istruzione(candidato, titoloStudio, data, voto, istituto)**
- **Attivita(codice, mansione)**
- **Esperienza(candidato, attivita, dataInizio, dataFine, azienda)**

rispettando i seguenti vincoli di integrità referenziale:

- **Telefono.candidato → Candidato.codice**
- **Istruzione.candidato → Candidato.codice**
- **Istruzione.titoloStudio → TitoloStudio.codice**
- **Esperienza.candidato → Candidato.codice**
- **Esperienza.attivita → Attivita.codice**

Il codice SQL risulta

code 4.1: candidati_create.sql

```sql
CREATE TABLE Candidato (
  codice SERIAL NOT NULL PRIMARY KEY,
  cognome VARCHAR(50) NOT NULL,
  nome VARCHAR(50) NOT NULL,
  viaResidenza VARCHAR(100),
  capResidenza CHAR(5),
  cittaResidenza VARCHAR(30),
```

```
  dataNascita DATE NOT NULL,
  luogoNascita VARCHAR(30)
);

CREATE TABLE Telefono (
  numero VARCHAR(15) NOT NULL PRIMARY KEY,
  candidato INTEGER REFERENCES Candidato(codice)
);

CREATE TABLE TitoloStudio (
  codice SERIAL NOT NULL PRIMARY KEY,
  descrizione VARCHAR(50) NOT NULL
);

CREATE TABLE Istruzione (
  candidato INTEGER REFERENCES Candidato(codice),
  titoloStudio INTEGER,
  data DATE,
  voto INTEGER,
  istituto VARCHAR(100),
  FOREIGN KEY (titoloStudio) REFERENCES TitoloStudio(codice),
  PRIMARY KEY(candidato, titoloStudio)
);

CREATE TABLE Attivita (
  codice SERIAL NOT NULL PRIMARY KEY,
  mansione VARCHAR(100) NOT NULL
);

CREATE TABLE Esperienza (
  candidato INTEGER REFERENCES Candidato(codice),
  attivita INTEGER REFERENCES Attivita(codice),
  dataInizio DATE NOT NULL,
  dataFine DATE,
  azienda VARCHAR(100),
  PRIMARY KEY(candidato, attivita)
);
```

4.2 Popolamento del database

code 4.2: candidati_insert.sql

```
INSERT INTO Candidato(cognome, nome,
viaResidenza, capResidenza, cittaResidenza, dataNascita,
   luogoNascita)
VALUES ('Rossi', 'Mario', 'Via dei Rossi', '53100', 'Siena', '
   1995-01-01', 'Siena'),
('Rossi', 'Luigi', 'Via dei Rossi', '53100', 'Siena', '2000-01-01'
   , 'Siena'),
```

```
('Bianchi', 'Andrea', 'Via Roma', '53100', 'Siena', '2001-01-01',
   'Siena'),
('Neri', 'Luca', 'Via dei Rozzi', '53100', 'Siena', '2002-01-01',
   'Siena');

INSERT INTO Telefono(numero, candidato)
VALUES ('123456789', 1), ('987654321', 2);

INSERT INTO TitoloStudio(descrizione)
VALUES ('diploma'), ('laurea triennale'), ('laurea magistrale');
```

4.3 Interrogazione sul database

1. Visualizzare, per ogni candidato, il cognome, il nome, la data di nascita ed il luogo di nascita

   ```
   SELECT cognome, nome, dataNascita, luogoNascita FROM Candidato
   ```

2. Visualizzare, per ogni candidato, il cognome, il nome, ed il numero di telefono

   ```
   SELECT Candidato.cognome, Candidato.nome, Telefono.numero
   FROM Candidato, Telefono
   WHERE Candidato.codice = Telefono.candidato
   ```

3. Visualizzare, per ogni candidato, il cognome, il nome, ed il numero di telefono, includendo nell'elenco anche i candidati di cui non si conosce il numero di telefono

   ```
   SELECT Candidato.cognome, Candidato.nome, Telefono.numero
   FROM Candidato LEFT JOIN Telefono
   ON (Candidato.codice = Telefono.candidato)
   ```

4. Visualizzare, per ogni candidato, i titoli di studio conseguiti, includendo la data, il voto e il luogo in cui i titoli sono stati conseguiti

   ```
   SELECT Candidato.cognome, Candidato.nome,
   TitoloStudio.descrizione, Istruzione.data,
   Istruzione.voto, Istruzione.istituto
   FROM Candidato, Istruzione, TitoloStudio
   WHERE Candidato.codice = Istruzione.candidato AND
   TitoloStudio.codice = Istruzione.titoloStudio
   ```

5. Visualizzare l'elenco dei candidati che hanno conseguito la laurea magistrale

   ```
   SELECT Candidato.cognome, Candidato.nome
   FROM Candidato, Istruzione, TitoloStudio
   WHERE Candidato.codice = Istruzione.candidato AND
   TitoloStudio.codice = Istruzione.titoloStudio
   AND TitoloStudio.descrizione = 'laurea magistrale'
   ```

6. Visualizzare i lavori svolti attualmente dai candidati

```
SELECT Candidato.cognome, Candidato.nome, mansione
FROM Candidato, Esperienza, Attivita
WHERE Candidato.codice = Esperienza.candidato AND
Esperienza.attivita = Attivita.codice AND datafine >= NOW()
```

7. Visualizzare l'elenco dei candidati che hanno conseguito la laurea magistrale con un voto maggiore di 100

```
SELECT Candidato.cognome, Candidato.nome
FROM Candidato, Istruzione, TitoloStudio
WHERE Candidato.codice = Istruzione.candidato AND
TitoloStudio.codice = Istruzione.titoloStudio
AND TitoloStudio.descrizione = 'laurea magistrale' AND voto > 100
```

8. Visualizzare il voto medio dei candidati

```
SELECT AVG(voto)
FROM Istruzione
```

9. Visualizzare il numero di lavori effettuati da ogni candidato

```
SELECT Candidato.cognome, Candidato.nome, COUNT(*)
FROM Candidato, Esperienza
WHERE Candidato.codice = Esperienza.candidato
GROUP BY Candidato.cognome, Candidato.nome
```

10. Visualizzare i candidati che hanno svolto più di un lavoro durante la loro carriera lavorativa

```
SELECT Candidato.cognome, Candidato.nome, COUNT(*)
FROM Candidato, Esperienza
WHERE Candidato.codice = Esperienza.candidato
GROUP BY Candidato.cognome, Candidato.nome
HAVING COUNT(*) > 1
```

11. Visualizzare nome e cognome dei candidati che non hanno precedenti esperienze lavorative (utilizzare le query nidificate)

```
SELECT Candidato.nome, Candidato.cognome
FROM Candidato WHERE codice <> ALL (SELECT candidato
FROM Esperienza WHERE dataFine IS NOT NULL GROUP BY candidato)
```

12. Visualizzare nome e cognome dei candidati che hanno almeno due titoli di studio

```
SELECT Candidato.nome, Candidato.cognome
FROM Candidato
WHERE codice = ANY (SELECT candidato
FROM Istruzione
GROUP BY candidato
HAVING COUNT(*) >= 2)
```

13. Visualizzare un elenco di tutti i nomi e cognomi dei candidati che hanno almeno un'esperienza lavorativa (N.B. Se il candidato Mario Rossi ha un'esperienza lavorativa, nella tabella risultato dovrà apparire una riga con il valore Mario ed una riga con il valore Rossi - Suggerimento: utilizzare l'operatore union)

```
SELECT Candidato.nome
FROM Candidato
WHERE codice = ANY (SELECT candidato
FROM Esperienza GROUP BY candidato)
UNION
SELECT Candidato.cognome
FROM Candidato
WHERE codice = ANY (SELECT candidato
FROM Esperienza GROUP BY candidato)
```

14. Creare una vista contenente codice, nome e cognome dei candidati che hanno avuto un'esperienza lavorativa conclusa

```
CREATE VIEW CandidatiEsperienza (codice, cognome, nome)
AS SELECT codice, cognome, nome
FROM Candidato WHERE codice = ANY (SELECT candidato
FROM Esperienza WHERE dataFine IS NOT NULL
GROUP BY candidato)
```

15. Ripetere l'interrogazione 11 utilizzando la vista creata al punto 14

```
SELECT VIEW CandidatiEsperienza (codice, cognome, nome) AS
SELECT nome, cognome
FROM Candidato WHERE codice <> ALL (SELECT candidato
FROM Esperienza WHERE dataFine IS NOT NULL GROUP BY candidato)
```