5. La progettazione logica

5.4 ristrutturazione dello schema logico

1. introduzione alla progettazione logica
2. ristrutturazione dello schema ER
3. traduzione diretta nel modello relazionale
4. **ristrutturazione dello schema logico**

Cosa sappiamo dopo la traduzione

- Abbiamo rispettato la modularizzazione concettuale
- Si possono presentare potenziali problemi di efficienza rispetto allo spazio
  - valori nulli (solo quelli dovuti ad attributi opzionali)
  - ci possono essere due relazioni R_1 e R_2 con chiavi K_1 e K_2 tali che valga sia $R_1[K_1] \subseteq R_2[K_2]$ sia $R_2[K_2] \subseteq R_1[K_1]$
  - ridondanze lasciate
- Si possono presentare potenziali problemi di efficienza rispetto al tempo di esecuzione delle query
  - ridondanze lasciate
  - numero e struttura delle relazioni

Ci devono essere dei buoni motivi legati all'**efficienza** per cambiare le scelte fatte

Modello di costo per lo schema logico

- Una relazione occupa un certo numero di pagine di memoria secondaria
- Il numero di accessi alle pagine della memoria secondaria domina l'elaborazione in memoria centrale
- Un accesso a memoria secondaria avviene ad una pagina intera
- Il numero complessivo di pagine accedute dipende in generale:
  - dal tipo di operazione
  - dal numero di tuple delle relazioni coinvolte
  - dal numero di tuple per pagina di memoria secondaria (determina il numero di pagine delle relazioni coinvolte)

Modello di costo: esempi

Per una relazione **R** denotiamo con

- $N_P(R)$ il numero di pagine in memoria secondaria occupate da **R**
- $N_{TP}(R)$ il numero di tuple per ogni pagina di **R**

Costo delle operazioni:

- la selezione su una relazione **R** ha costo pari a $N_P(R)$
- la proiezione su una relazione **R** ha costo pari a $N_P(R)$
- il join di **R** con **Q** si basa su un doppio ciclo
  - se non ci sono indici, allora il costo è $N_P(R) \times N_P(Q)$:
    per ogni tupla **r** di **R**
    per ogni tupla **q** di **Q**
    se **r** e **q** sono in join, metti la tupla nel risultato
  - se **Q** ha un indice molto selettivo sull'attributo di join, allora il costo è $N_P(R) + N_P(R) \times N_{TP}(R)$

Criteri generali per individuare potenziali problemi

- relazione con
  - tante tuple → la relazione occupa molte pagine
  - tanti attributi → una pagina contiene poche tuple
- attributi con tanti valori nulli
  → spreco di spazio → una pagina contiene poche tuple
- la proiezione è costosa (quando una pagina contiene poche tuple)
- il join è costoso (quasi sempre)
- la verifica di vincoli è costosa

Conclusione: per una relazione **R** rilevante (con tante tuple) occorre tentare di tenere basso $N_P(R)$

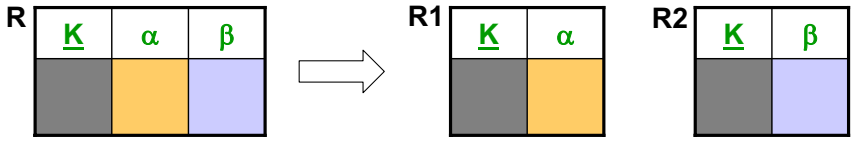
Ristrutturazioni dello schema logico

- **Decomposizione**
  - *Verticale* (sempre sulla chiave)
    - per facilitare l'accesso (con selezioni e proiezioni)
    - per normalizzazione (ignoreremo questo aspetto)
  - *Orizzontale*
    - per facilitare l'accesso (con selezioni)
  - *Mista*
    - per evitare valori nulli
- **Accorpamento**
  - per facilitare l'accesso (evita join)
  - per eliminare relazioni inutili

Le relazioni dello schema originario possono essere ricostruite attraverso la definizione di opportune viste

Nota: le ristrutturazioni si applicano in presenza di determinati attributi che formano una chiave di relazione. Sulle slide indicheremo tali attributi con **K**, intendendo che **K** è una chiave della relazione corrispondente

Decomposizione verticale per facilitare l'accesso



Vincoli dello schema ristrutturato:

- foreign key: R1[K] ⊆ R2[K]
  foreign key: R2[K] ⊆ R1[K]
- vincoli di inclusione da e verso **R** si definiscono su **R1** (o **R2**)
- tutti gli altri vincoli che coinvolgono **R** vanno riformulati

- Si applica quando gli accessi ad **R** avvengono prevalentemente in modo separato sugli attributi α rispetto agli attributi β
- $N_{TP}(R1)$ e $N_{TP}(R2)$ sono alti rispetto a $N_{TP}(R)$ e quindi $N_P(R1)$ e $N_P(R2)$ sono bassi rispetto a $N_P(R)$
- La relazione **R** può essere ricostruita attraverso una vista che calcola il join tra **R1** ed **R2** su **K**

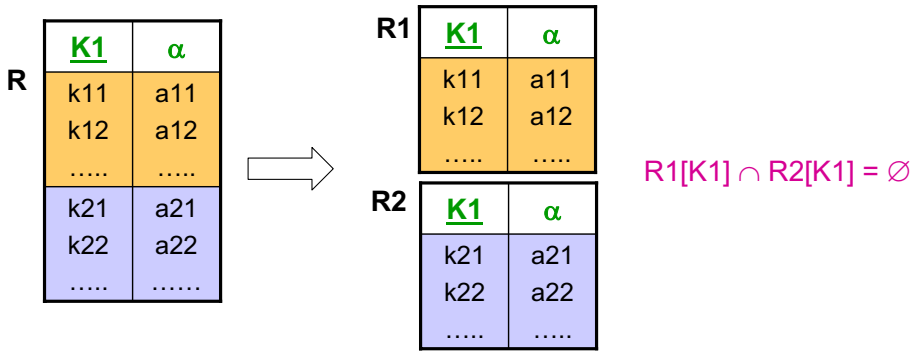
Decomposizione verticale: esempio

Immobile(Codice, Particella, Zona, Indirizzo, Città, Mq, NumVani, Valore)
foreign key: Immobile[Città] ⊆ Città [Cod]
Agenzia(CodiceAg, Immobile)
foreign key: Agenzia[Immobile] ⊆ Immobile[Codice]

Supponiamo che ai dati catastali degli immobili (particella, zona, indirizzo, città) si acceda prevalentemente in modo separato rispetto ai dati commerciali (metri quadri, numero di vani, valore). Applichiamo quindi la decomposizione verticale, ed otteniamo:

ImmobileCatasto(Codice, Particella, Zona, Indirizzo, Città)
foreign key: ImmobileCatasto[Città] ⊆ Città [Cod]
foreign key: ImmobileCatasto[Codice] ⊆ ImmobileComm [Codice]
ImmobileComm(Codice, Mq, NumVani, Valore)
foreign key: ImmobileComm[Codice] ⊆ ImmobileCatasto[Codice]
Agenzia(CodiceAg, Immobile)
foreign key: Agenzia[Immobile] ⊆ ImmobileCatasto[Codice]

Decomposizione orizzontale per facilitare l'accesso



$R1[K1] \cap R2[K1] = \varnothing$

Ulteriori vincoli dello schema ristrutturato:

- vincoli di inclusione da **R** diventano vincoli di inclusione da **R1** e da **R2**
- vincoli di inclusione a **R** diventano vincoli di inclusione a **R1** $\cup$ **R2**
- tutti gli altri vincoli che coinvolgono **R** vanno riformulati

- Si applica quando gli accessi alle tuple di **R** di una delle due "fasce" avvengono separatamente dagli accessi alle tuple dell'altra fascia
- $N_P(R1)$ e $N_P(R2)$ sono bassi rispetto a $N_P(R)$
- **R** può essere ricostruita attraverso una vista che calcola l'unione di **R1** ed **R2**

Decomposizione orizzontale: esempio

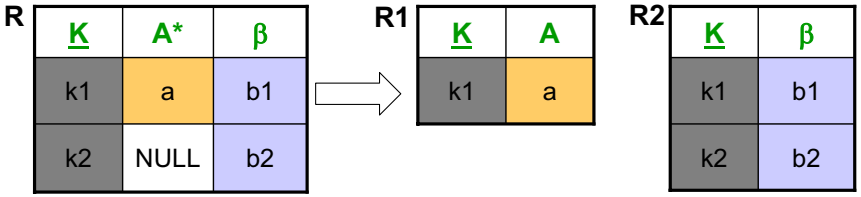
Telefonata(Codice, OrarioInizio, OrarioFine, UtenzaInvio, UtenzaDestinazione)
foreign key: Telefonata[UtenzaInvio] $\subseteq$ Utenza[Cod]
foreign key: Telefonata[UtenzaDestinazione] $\subseteq$ Utenza[Cod]
Centrale(Codice, Telefonata)
foreign key: Centrale[Telefonata] $\subseteq$ Telefonata[Codice]

Supponiamo che alle telefonate si acceda per fasce orarie (giorno, sera, notte) determinate sul tempo di inizio. Applichiamo quindi la decomposizione orizzontale (opportuni vincoli detteranno le regole per i valori corretti del campo OrarioInizio delle tabelle e realizzeranno quindi i vincoli di disgiuntezza), ed otteniamo:

TelefonataGiorno(Codice, OrarioInizio, OrarioFine, UtenzaInvio, UtenzaDestinazione)
foreign key: TelefonataGiorno[UtenzaInvio] $\subseteq$ Utenza[Cod]
foreign key: TelefonataGiorno[UtenzaDestinazione] $\subseteq$ Utenza[Cod]
TelefonataSera(Codice, OrarioInizio, OrarioFine, UtenzaInvio, UtenzaDestinazione)
foreign key: TelefonataSera[UtenzaInvio] $\subseteq$ Utenza[Cod]
foreign key: TelefonataSera[UtenzaDestinazione] $\subseteq$ Utenza[Cod]
TelefonataNotte(Codice, OrarioInizio, OrarioFine, UtenzaInvio, UtenzaDestinazione)
foreign key: TelefonataNotte[UtenzaInvio] $\subseteq$ Utenza[Cod]
foreign key: TelefonataNotte[UtenzaDestinazione] $\subseteq$ Utenza[Cod]
Centrale(Codice, Telefonata)

Vincolo esterno:
Centrale[Telefonata] $\subseteq$ TelefonataGiorno[Codice] $\cup$ TelefonataSera[Codice] $\cup$ TelefonataNotte[Codice]

Decomposizione mista per evitare valori nulli



A è l'attributo su cui si vogliono evitare valori nulli, e β denota un insieme di attributi.

Vincoli dello schema ristrutturato:

- foreign key: R1[K] $\subseteq$ R2[K]
- vincoli di inclusione da e verso **R** diventano inclusioni da e verso **R2**
- tutti gli altri vincoli che coinvolgono **R** vanno riformulati

- Si applica quando la relazione ha tanti valori nulli in A
- $N_{TP}(R1)$, $N_{TP}(R2)$ sono alti rispetto a $N_{TP}(R)$ e quindi $N_P(R1)$, $N_P(R2)$ sono bassi rispetto a e $N_P(R)$
- **R** può essere ricostruita attraverso una vista che calcola il join esterno tra **R1** ed **R2** su **K**.

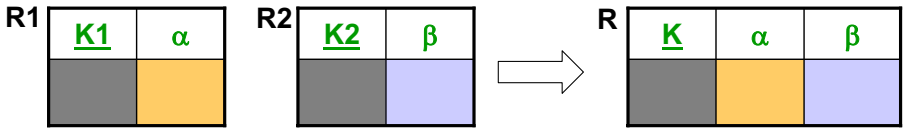
Decomposizione mista: esempio

Persona(CodFiscale, CittàNascita, NumTel*, DataMatrimonio*)
foreign key: Persona[CittàNascita] $\subseteq$ Città[Cod]
Città(Cod, Sindaco)
foreign key: Città[Sindaco] $\subseteq$ Persona[CodFiscale]

Supponiamo di non volere valori nulli. Applichiamo quindi due volte in cascata la decomposizione mista, ed otteniamo:

Persona(CodFiscale, CittàNascita)
foreign key: Persona[CittàNascita] $\subseteq$ Città[Cod]
PersonaConTelefono(CodFiscale, NumTel)
foreign key: PersonaConTelefono[CodFiscale] $\subseteq$ Persona[CodFiscale]
PersonaSposata(CodFiscale, DataMatrimonio)
foreign key: PersonaSposata[CodFiscale] $\subseteq$ Persona[CodFiscale]
Città(Cod, Sindaco)
foreign key: Città[Sindaco] $\subseteq$ Persona[CodFiscale]

Accorpamento per facilitare l'accesso



foreign key: R1[K1] ⊆ R2[K2]

foreign key: R2[K2] ⊆ R1[K1]

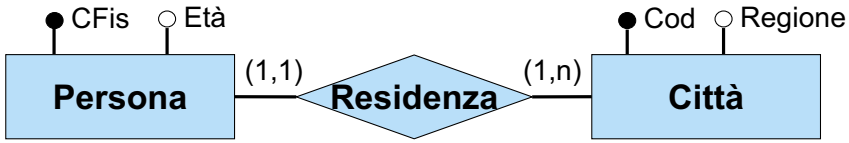
Si noti che **K1** e **K2** sono chiavi, anche non primarie. Si noti anche che se β manca, **R** coincide con **R1**, e l'effetto è quello di eliminare **R2**.

Vincoli dello schema ristrutturato:

- tutti i vincoli che coinvolgono **R1** o **R2** vanno riformulati su **R**

- Si applica per facilitare gli accessi a **R1** e **R2** quando questi avvengono prevalentemente insieme e richiedono di calcolare il join tra **R1** e **R2** con **K1=K2**; in altre parole si applica per evitare il join tra **R1** e **R2**
- Le relazioni **R1** e **R2** possono essere ricostruite attraverso due viste che calcolano rispettivamente le proiezioni di **R** su **(K, α)** e su **(K, β)**
- Quando non c'è β, serve ad eliminare una relazione inutile (cioè **R2**)

Accorpamento: esempio 1

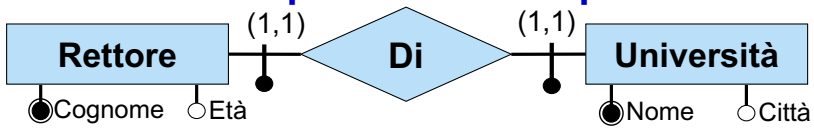


Persona(CFis, Età)
  foreign key: Persona[CFis] ⊆ Residenza[Persona]
Residenza(Persona, Città)
  foreign key: Residenza[Persona] ⊆ Persona[CFis]
  foreign key: Residenza[Città] ⊆ Città[Cod]
Città(Cod, Regione)
  inclusione: Città[Cod] ⊆ Residenza[Città]

Supponiamo che quando si accede alla persone si acceda spesso anche alla sua città di residenza. Applichiamo quindi l'accorpamento, ed otteniamo:

Persona(CFis, Età, Città)
  foreign key: Persona[Città] ⊆ Città[Cod]
Città(Cod, Regione)
  inclusione: Città[Cod] ⊆ Persona[Città]

Accorpamento: esempio 2



Rettore(Cognome, Età)
  foreign key: Rettore[Cognome] ⊆ Di[Rettore]
Di(Rettore, Università)
  foreign key: Di[Rettore] ⊆ Rettore[Cognome]
  foreign key: Di[Università] ⊆ Università[Nome]
  chiave: Rettore
Università(Nome, Città)
  foreign key: Università[Nome] ⊆ Di[Università]

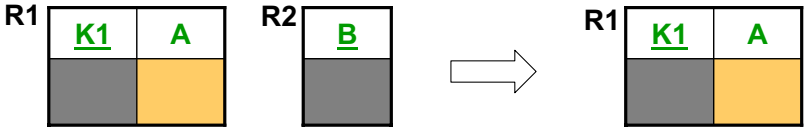
Supponiamo che quando si accede ad una università si acceda anche al suo rettore. Applichiamo l'accorpamento di Università e Di, ed otteniamo

Rettore(Cognome, Età)
  foreign key: Rettore[Cognome] ⊆ Università[Rettore]
Università(Nome, Città, Rettore)
  foreign key: Università[Rettore] ⊆ Rettore[Cognome]

Supponiamo che quando si accede ad un rettore si acceda anche ai dati relativi alla sua università. Applichiamo un ulteriore accorpamento ed otteniamo

UniversitàRettore(Nome, Città, CognomeRettore, EtàRettore)
  chiave: CognomeRettore

Accorpamento per eliminare relazioni inutili



foreign key: R1[A] ⊆ R2[B]

inclusione: R2[B] ⊆ R1[A]

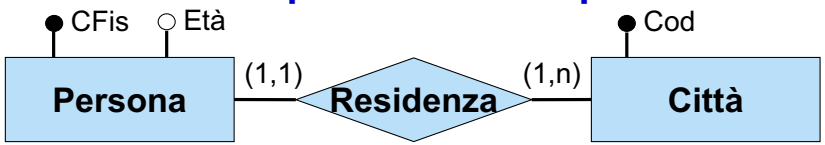
Si noti che **B** è chiave, ma **A** non lo è.

Vincoli dello schema ristrutturato:

- tutti i vincoli che coinvolgono **R2** vanno riformulati su **R1**

- Si applica per eliminare la relazione **R2** che è inutile, visto che tutti i valori nell'unico suo attributo si ritrovano nell'attributo **A** della relazione **R1**
- La relazione **R2** può essere ricostruita attraverso la proiezione della relazione **R1** sull'attributo **A**

Accorpamento: esempio 3



Persona(CFis, Età)
  foreign key: Persona[CFis] ⊆ Residenza[Persona]
Residenza(Persona, Città)
  foreign key: Residenza[Persona] ⊆ Persona[CFis]
  foreign key: Residenza[Città] ⊆ Città[Cod]
Città(Cod)
  inclusione: Città[Cod] ⊆ Residenza[Città]

Supponiamo che quando si accede alla persone si acceda spesso anche alla sua città di residenza. Applichiamo quindi l'accorpamento, ed otteniamo:

Persona(CFis, Età, Città)
  foreign key: Persona[Città] ⊆ Città[Cod]
Città(Cod)
  inclusione: Città[Cod] ⊆ Persona[Città]

Applichiamo quindi di nuovo l'accorpamento per eliminare la relazione inutile (Città), ed otteniamo:

Persona(CFis, Età, Città)

Traduzione in SQL

Ogni relazione viene definita tramite la "create table", nel modo ovvio. In particolare, i vincoli vengono tradotti secondo queste indicazioni generali:

- attributi obbligatori: `not null`
- chiave primaria: `primary key`
- chiave: `unique`
- foreign key: `foreign key`
- interdipendenza valori nulli:

```
check ((A is null and B is null) or
       (A is not null and B is not null))
```

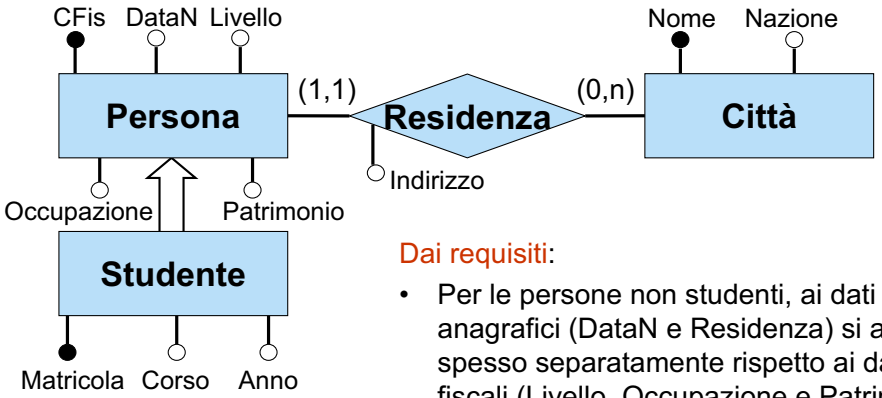
- inclusione: `check (A in (select B from R))`
- disgiunzione: `check (A not in (select B from R))`
- cardinalità (i,j):

```
check ((i ≤ (select count(*) from R where ...) and
       (j ≥ (select count(*) from R where ...))
```

- vincoli esterni: `assertion` o controllati a livello di applicazione

Esercizio 7: effettuare la progettazione logica

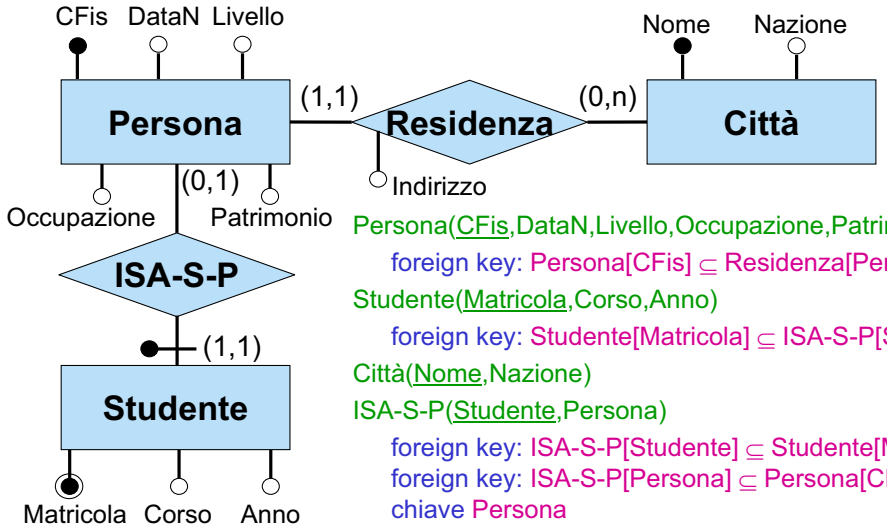
Schema concettuale:



Dai requisiti:

- Per le persone non studenti, ai dati anagrafici (DataN e Residenza) si accede spesso separatamente rispetto ai dati fiscali (Livello, Occupazione e Patrimonio)
- Agli studenti si accede prevalentemente per matricola, e si accede spesso ai dati universitari (Corso, Anno) insieme alla data di nascita e ai dati fiscali (Livello, Occupazione e Patrimonio)

Esercizio 7: ristrutturazione e traduzione diretta



Persona(CFis,DataN,Livello,Occupazione,Patrimonio)
  foreign key: Persona[CFis] ⊆ Residenza[Persona]
Studente(Matricola,Corso,Anno)
  foreign key: Studente[Matricola] ⊆ ISA-S-P[Stud]
Città(Nome,Nazione)
ISA-S-P(Studente,Persona)
  foreign key: ISA-S-P[Studente] ⊆ Studente[Matricola]
  foreign key: ISA-S-P[Persona] ⊆ Persona[CFis]
  chiave Persona
Residenza(Persona,Indirizzo,Città)
  foreign key: Residenza[Persona] ⊆ Persona[CFis]
  foreign key: Residenza[Città] ⊆ Città[Nome]

Esercizio 7: ristrutturazioni dello schema logico

Per le persone non studenti valgono criteri di accesso diversi rispetto alle persone che sono studenti, anche relativamente ai dati relativi alla residenza:

- **decomposizione orizzontale** di Persona in PersonaNonStudente e PersonaStudente
- **decomposizione orizzontale** di Residenza in ResidenzaNonStudente e ResidenzaStudente

PersonaNonStudente(CFis,DataN,Livello,Occupazione,Patrimonio)
  foreign key: PersonaNonStudente[CFis] ⊆ ResidenzaNonStudente[Persona]
PersonaStudente(CFis,DataN,Livello,Occupazione,Patrimonio)
  foreign key: PersonaStudente[CFis] ⊆ ISA-S-P[Persona]
  foreign key: PersonaStudente[CFis] ⊆ ResidenzaStudente[Studente]
Studente(Matricola,Corso,Anno)
  foreign key: Studente[Matricola] ⊆ ISA-S-P[Studente]
Città(Nome,Nazione)
ISA-S-P(Studente,Persona)
  foreign key: ISA-S-P[Studente] ⊆ Studente[Matricola]
  foreign key: ISA-S-P[Persona] ⊆ PersonaStudente[CFis] chiave: Persona
ResidenzaNonStudente(Persona,Indirizzo,Città)
  foreign key: Residenza[Persona] ⊆ PersonaNonStudente[CFis]
  foreign key: Residenza[Città] ⊆ Città[Nome]
ResidenzaStudente(Studente,Indirizzo,Città)
  foreign key: ResidenzaStudente[Studente] ⊆ PersonaStudente[CFis]
  foreign key: Residenza[Città] ⊆ Città[Nome]
Vincolo: PersonaNonStudente[CFis] ∩ PersonaStudente[Cfis] = ∅

Esercizio 7: ristrutturazioni dello schema logico

Per le persone non studenti, ai dati anagrafici (DataN e Residenza) si accede spesso separatamente rispetto ai dati fiscali (Livello, Occupazione e Patrimonio):

- **decomposizione verticale** di PersonaNonStudente in PersonaDatiAnagrafe e PersonaDatiFisco

PersonaDatiAnagrafe(CFis,DataN)
  foreign key: PersonaDatiAnagrafe[CFis] ⊆ PersonaDatiFisco[CFis]
  foreign key: PersonaDatiAnagrafe[CFis] ⊆ ResidenzaNonStudente[Persona]
PersonaDatiFisco(CFis,Livello,Occupazione,Patrimonio)
  foreign key: PersonaDatiFisco[CFis] ⊆ PersonaDatiAnagrafe[CFis]
PersonaStudente(CFis,DataN,Livello,Occupazione,Patrimonio)
  foreign key: PersonaStudente[CFis] ⊆ ISA-S-P[Persona]
  foreign key: PersonaStudente[CFis] ⊆ ResidenzaStudente[Studente]
Studente(Matricola,Corso,Anno)
  foreign key: Studente[Matricola] ⊆ ISA-S-P[Studente]
Città(Nome,Nazione)
ISA-S-P(Studente,Persona)
  foreign key: ISA-S-P[Studente] ⊆ Studente[Matricola]
  foreign key: ISA-S-P[Persona] ⊆ PersonaStudente[CFis] chiave: Persona
ResidenzaNonStudente(Persona,Indirizzo,Città)
  foreign key: Residenza[Persona] ⊆ PersonaDatiAnagrafe[CFis]
  foreign key: Residenza[Città] ⊆ Città[Nome]
ResidenzaStudente(Studente,Indirizzo,Città)
  foreign key: ResidenzaStudente[Studente] ⊆ PersonaStudente[CFis]
  foreign key: Residenza[Città] ⊆ Città[Nome]
Vincolo: PersonaDatiAnagrafe[CFis] ∩ PersonaStudente[Cfis] = ∅

Esercizio 7: ristrutturazioni dello schema logico

Per le persone non studenti, ai dati anagrafici (DataN e Residenza) si accede spesso separatamente rispetto ai dati fiscali (Livello, Occupazione e Patrimonio):

- **accorpamento** tra PersonaDatiAnagrafe e ResidenzaNonStudente

PersonaDatiAnagrafe(CFis,DataN,Indirizzo,Città)
  foreign key: PersonaDatiAnagrafe[CFis] ⊆ PersonaDatiFisco[CFis]
  foreign key: PersonaDatiAnagrafe[Città] ⊆ Città[Nome]
PersonaDatiFisco(CFis,Livello,Occupazione,Patrimonio)
  foreign key: PersonaDatiFisco[CFis] ⊆ PersonaDatiAnagrafe[CFis]
PersonaStudente(CFis,DataN,Livello,Occupazione,Patrimonio)
  foreign key: PersonaStudente[CFis] ⊆ ISA-S-P[Persona]
  foreign key: PersonaStudente[CFis] ⊆ ResidenzaStudente[Studente]
Studente(Matricola,Corso,Anno)
  foreign key: Studente[Matricola] ⊆ ISA-S-P[Studente]
Città(Nome,Nazione)
ISA-S-P(Studente,Persona)
  foreign key: ISA-S-P[Studente] ⊆ Studente[Matricola]
  foreign key: ISA-S-P[Persona] ⊆ PersonaStudente[CFis] chiave: Persona
ResidenzaStudente(Studente,Indirizzo,Città)
  foreign key: ResidenzaStudente[Studente] ⊆ PersonaStudente[CFis]
  foreign key: Residenza[Città] ⊆ Città[Nome]
Vincolo: PersonaDatiAnagrafe[CFis] ∩ PersonaStudente[Cfis] = ∅

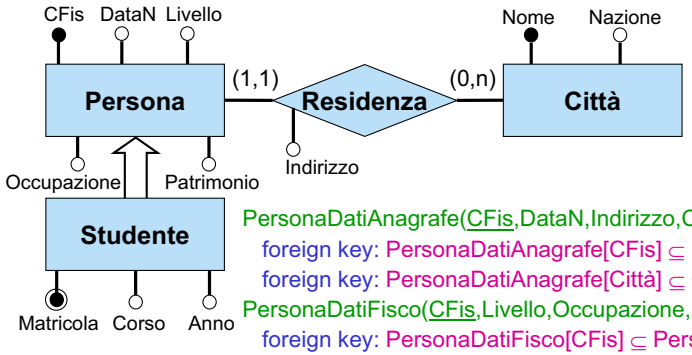
Esercizio 7: ristrutturazioni dello schema logico

Agli studenti si accede prevalentemente per matricola, e si accede spesso ai dati universitari (Corso, Anno) insieme alla data di nascita e ai dati fiscali (Livello, Occupazione e Patrimonio):

- **accorpamento** tra Studente, ISA-S-P e PersonaStudente

PersonaDatiAnagrafe(CFis,DataN,Indirizzo,Città)
  foreign key: PersonaDatiAnagrafe[CFis] ⊆ PersonaDatiFisco[CFis]
  foreign key: PersonaDatiAnagrafe[Città] ⊆ Città[Nome]
PersonaDatiFisco(CFis,Livello,Occupazione,Patrimonio)
  foreign key: PersonaDatiFisco[CFis] ⊆ PersonaDatiAnagrafe[CFis]
Studente(Matricola,Corso,Anno,CFis,DataN,Livello,Occupazione,Patrimonio)
  foreign key: Studente[CFis] ⊆ ResidenzaStudente[Studente]
  chiave: CFis
Città(Nome,Nazione)
ResidenzaStudente(Studente,Indirizzo,Città)
  foreign key: ResidenzaStudente[Studente] ⊆ Studente[CFis]
  foreign key: Residenza[Città] ⊆ Città[Nome]
Vincolo: PersonaDatiAnagrafe[CFis] ∩ Studente[Cfis] = ∅

Esercizio 7: schema logico e confronto col concettuale



PersonaDatiAnagrafe(CFis,DataN,Indirizzo,Città)
  foreign key: PersonaDatiAnagrafe[CFis] ⊆ PersonaDatiFisco[CFis]
  foreign key: PersonaDatiAnagrafe[Città] ⊆ Città[Nome]

PersonaDatiFisco(CFis,Livello,Occupazione,Patrimonio)
  foreign key: PersonaDatiFisco[CFis] ⊆ PersonaDatiAnagrafe[CFis]

Studente(Matricola,Corso,Anno,CFis,DataN,Livello,Occupazione,Patrimonio)
  foreign key: Studente[CFis] ⊆ ResidenzaStudente[Studente]
  chiave: CFis

Città(Nome,Nazione)

ResidenzaStudente(Studente,Indirizzo,Città)
  foreign key: ResidenzaStudente[Studente] ⊆ Studente[CFis]
  foreign key: Residenza[Città] ⊆ Città[Nome]

Vincolo: PersonaDatiAnagrafe[CFis] ∩ Studente[Cfis] = ∅

Si lascia come esercizio la definizione delle viste per ricostruire le relazioni dello schema logico originario

Esercizio 8: ristrutturare il seguente schema ...

Volo(Codice, Comp, Durata)
  foreign key: Volo[Comp] ⊆ Compagnia[Nome]
  foreign key: Volo[Codice,Comp] ⊆ ArrPart[Codice,Comp]

ArrPart(Codice, Comp, Arrivo, Partenza)
  foreign key: ArrPart[Arrivo] ⊆ Aeroporto[Codice]
  foreign key: ArrPart[Partenza] ⊆ Aeroporto[Codice]
  foreign key: ArrPart[Codice,Comp] ⊆ Volo[Codice,Comp]
  chiave: Comp, Arrivo, Partenza

VoloCharter(Codice, Comp, TipoAereo)
  foreign key: VoloCharter[Codice,Comp] ⊆ Volo[Codice,Comp]

Aeroporto(Codice, Nome)
  foreign key: Aeroporto[Codice] ⊆ LuogoAeroporto[Aeroporto]

LuogoAeroporto(Aeroporto, NomeCittà, NazCittà)
  foreign key: LuogoAeroporto[Aeroporto] ⊆ Aeroporto[Codice]
  foreign key: LuogoAeroporto[NomeCittà,NazCittà] ⊆ Città[Nome,Nazione]

Città(Nome, Nazione, NumAbitanti)

Compagnia(Nome, AnnoFond)
  foreign key: Compagnia[Nome] ⊆ Sede[Comp]

Esercizio 8: ristrutturare il seguente schema...(parte 2)

Sede(Comp)
  foreign key: Sede[Comp] ⊆ Compagnia[Nome]
  foreign key: Sede[Comp] ⊆ CittàSede[Comp]
  inclusione: Sede[Comp] ⊆ TelefonoComp[Comp]

CittàSede(Comp, NomeCittà, NazCittà)
  foreign key: CittàSede[Comp] ⊆ Sede[Comp]
  foreign key: CittàSede[NomeCittà,NazCittà] ⊆ Città[Nome,Nazione]

TelefonoComp(Numero, Comp)
  foreign key: Telefono[Comp] ⊆ Sede[Comp]
  foreign key: TelefonoComp[Numero] ⊆ Telefono[Numero]

Telefono(Numero)
  foreign key: Telefono[Numero] ⊆ TelefonoComp[Numero]

Tappa(CodVoloCharter, Comp, Aeroporto, Ordine)
  foreign key: Tappa[CodVoloCharter,Comp] ⊆ VoloCharter[Codice,Comp]
  foreign key: Tappa[Aeroporto] ⊆ Aeroporto[Codice]

Vincolo esterno: vincolo su Ordine in Tappa

... tenendo conto delle seguenti specifiche

- Si accede spesso all'insieme dei voli charter separatamente dagli altri voli, sia per quanto riguarda la durata, sia per quanto riguarda gli aeroporti di arrivo e partenza
- Quando si accede ad un volo (charter o no) si vuole spesso conoscere tutte le proprietà di tale volo (durata, tipo aereo se volo charter, aeroporto di arrivo e aeroporto di partenza)
- Quando si accede ad una compagnia aerea si accede anche ai dati relativi alla sua sede

Esercizio 8: soluzione – ristrutturazioni

- Si accede spesso all'insieme dei voli charter separatamente dagli altri voli, sia per quanto riguarda la durata, sia per quanto riguarda gli aeroporti di arrivo e partenza:
  - **decomposizione orizzontale** di Volo in
    - VoloNonCharter
    - DatiVoloCharter
  - **decomposizione orizzontale** di ArrPart in
    - ArrPartVoloNonCharter
    - ArrPartVoloCharter
- Quando si accede ad un volo (charter o no) si vuole spesso conoscere tutte le proprietà di tale volo (durata, tipo aereo se volo charter, aeroporto di arrivo e aeroporto di partenza):
  - **accorpamento** di DatiVoloCharter e VoloCharter e, successivamente, della relazione risultante con ArrPartVoloCharter
  - **accorpamento** di VoloNonCharter e ArrPartVoloNonCharter e, successivamente, della relazione risultante con VoloNonCharter
- Quando si accede alla compagnia si accede anche ai dati relativi alla sua sede:
  - **accorpamento** di Compagnia e CittàSede
- Si eliminano le relazioni inutili Sede e Telefono mediante **accorpamento**

Esercizio 8: soluzione – schema ristrutturato

VoloNonCharter(Codice, Comp, Durata, Arrivo, Partenza)
  foreign key: VoloNonCharter[Comp] ⊆ Compagnia[Nome]
  foreign key: VoloNonCharter[Arrivo] ⊆ Aeroporto[Codice]
  foreign key: VoloNonCharter[Partenza] ⊆ Aeroporto[Codice]
  chiave: Comp, Arrivo, Partenza

VoloCharter(Codice, Comp, TipoAereo, Durata, Arrivo, Partenza)
  foreign key: VoloCharter[Comp] ⊆ Compagnia[Nome]
  foreign key: VoloCharter[Arrivo] ⊆ Aeroporto[Codice]
  foreign key: VoloCharter[Partenza] ⊆ Aeroporto[Codice]
  chiave: Comp, Arrivo, Partenza

Aeroporto(Codice, Nome)
  foreign key: Aeroporto[Codice] ⊆ LuogoAeroporto[Aeroporto]

LuogoAeroporto(Aeroporto, NomeCittà, NazCittà)
  foreign key: LuogoAeroporto[Aeroporto] ⊆ Aeroporto[Codice]
  foreign key: LuogoAeroporto[NomeCittà, NazCittà] ⊆ Città[Nome, Nazione]

Città(Nome, Nazione, NumAbitanti)

Compagnia(Nome, AnnoFond, NomeCittà, NazCittà)
  foreign key: Compagnia[NomeCittà, NazCittà] ⊆ Città[Nome, Nazione]
  inclusione: Compagnia[Nome] ⊆ TelefonoComp[Comp]

TelefonoCom(Numero, Comp)
  foreign key: TelefonoComp[Comp] ⊆ Compagnia[Nome]

Tappa(CodVoloCharter, Comp, Aeroporto, Ordine)
  foreign key: Tappa[CodVoloCharter, Comp] ⊆ VoloCharter[Codice, Comp]
  foreign key: Tappa[Aeroporto] ⊆ Aeroporto[Codice]

Esercizio 8: soluzione – vincoli e viste

Vincoli:

- VoloNonCharter e VoloCharter sono disgiunti:
  VoloNonCharter[Codice, Comp] ∩ VoloCharter[Codice, Comp] = ∅
  VoloNonCharter[Comp, Arrivo, Partenza] ∩ VoloCharter[Comp, Arrivo, Partenza] = ∅
- vincolo su Ordine in Tappa

Viste per ricostruire le relazioni dello schema originario:

view Volo = $\text{PROJ}_{\text{Codice,Comp,Durata}}$(VoloNonCharter) ∪ $\text{PROJ}_{\text{Codice,Comp,Durata}}$(VoloCharter)

view ArrPart = $\text{PROJ}_{\text{Codice,Comp,Arrivo,Partenza}}$(VoloNonCharter) ∪ $\text{PROJ}_{\text{Codice,Comp,Arrivo,Partenza}}$(VoloCharter)

view Sede = $\text{PROJ}_{\text{Nome}}$(Compagnia)

view CompagniaOriginaria = $\text{PROJ}_{\text{Nome, AnnoFond}}$(Compagnia)

view CittàSede = $\text{PROJ}_{\text{Nome,NomeCittà,NazCittà}}$ (Compagnia)

view Telefono = $\text{PROJ}_{\text{Numero}}$(TelefonoComp)

Esercizio 9: ristrutturare il seguente schema ...

Officina(Nome, NumDip, Indirizzo)
  foreign key: Officina[Nome] ⊆ Dirige[Officina]
  inclusione: Officina[Nome] ⊆ Lavora[Officina]

Persona(CodFis, Indirizzo)

Direttore(CodFis, Età, AnniAnz)
  foreign key: Direttore[CodFis] ⊆ Persona[CodFis]
  foreign key: Direttore[CodFis] ⊆ Dirige[Direttore]

Dipendente(CodFis, AnniAnz)
  foreign key: Dipendente[CodFis] ⊆ Persona[CodFis]
  inclusione: Dipendente[CodFis] ⊆ Lavora[Dipendente]

Dirige(Officina, Direttore)
  foreign key: Dirige[Officina] ⊆ Officina[Nome]
  foreign key: Dirige[Direttore] ⊆ Direttore[CodFis]
  chiave: Direttore

Lavora(Officina, Dipendente, AnniServizio)
  foreign key: Lavora[Officina] ⊆ Officina[Nome]
  foreign key: Lavora[Dipendente] ⊆ Dipendente[CodFis]

TelPer(CodFis, Telefono)
  foreign key: TelPer[CodFis] ⊆ Persona[CodFis]
  foreign key: TelPer[Telefono] ⊆ Telefono[Numero]

Esercizio 9: ristrutturare il seguente schema...(parte 2)

Telefono(Numero)
  inclusione: Telefono[Numero] ⊆ TelPer[Telefono]
Veicolo(Targa, Modello, Tipo, AnnoImm)
  foreign key : Veicolo[Targa] ⊆ Possiede[Veicolo]
Possiede(Veicolo, Proprietario)
  foreign key: Possiede[Veicolo] ⊆ Veicolo[Targa]
  foreign key: Possiede[Proprietario] ⊆ Persona[CodFis]
Riparazione(Codice, Officina, OraAcc, DataAcc)
  inclusione: Riparazione[Officina] ⊆ Officina[Nome]
  foreign key: Riparazione[Codice,Officina] ⊆ Relativa[Codice,Officina]
Relativa(Codice, Officina, Veicolo)
  foreign key: Relativa[Codice,Officina] ⊆ Riparazione[Codice,Officina]
  foreign key: Relativa[Veicolo] ⊆ Veicolo[Targa]
Terminata(Codice, Officina, OraRic, DataRic)
  foreign key: Terminata[Codice,Officina] ⊆ Riparazione[Codice,Officina]

Vincoli esterni:
- riconsegna dopo accettazione
- vincolo che lega Officina[NumDip] alle istanze in Lavora
- vincolo su AnniAnz di Direttore e Dipendente derivante dall'eliminazione ISA

... tenendo conto delle seguenti specifiche

- Quando si accede ai direttori, interessano anche tutti i dati relativi all'officina che dirigono e viceversa, quando si accede alle officine, interessano anche i dati relativi all'età e alla anzianità del loro direttore
- Ai dipendenti si accede spesso separatamente rispetto ai direttori
- Quando si accede ai dipendenti interessano anche i dati relativi all'indirizzo

Esercizio 9: soluzione – ristrutturazioni

- Quando si accede ai direttori, interessano anche tutti i dati relativi all'officina che dirigono e viceversa quando si accede alle officine, interessano anche tutti i dati relativi al loro direttore:
  - **accorpamento** di Direttore e Dirige e, successivamente, della relazione risultante con Officina
- Ai dipendenti si accede spesso separatamente rispetto ai direttori:
  - **decomposizione orizzontale** di Persona in dipendenti e non
- Quando si accede ai dipendenti interessano anche i loro dati anagrafici:
  - **accorpamento** tra la relazione e Dipendente e PersonaDipendente
- Si elimina la relazione inutile Telefono mediante **accorpamento**

Esercizio 9: soluzione – schema ristrutturato

OfficinaDirettore(Nome, NumDip, Indirizzo, Direttore, EtaDir, AnniAnzDir)
  chiave: Direttore
  inclusione: Officina[Nome] ⊆ Lavora[Officina]
PersonaNonDip(CodFis, Indirizzo)
Dipendente(CodFis, AnniAnz, Indirizzo)
  inclusione: Dipendente[CodFis] ⊆ Lavora[Dipendente]
Lavora(Officina, Dipendente, AnniServizio)
  foreign key: Lavora[Officina] ⊆ OfficinaDirettore[Nome]
  foreign key: Lavora[Dipendente] ⊆ Dipendente[CodFis]
TelPer(CodFis, Telefono)
Veicolo(Targa, Modello, Tipo, AnnoImm)
Possiede(Veicolo, Proprietario)
  foreign key: Possiede[Veicolo] ⊆ Veicolo[Targa]
Riparazione(Codice, Officina, OraAcc, DataAcc,)
  inclusione: Riparazione[Officina] ⊆ OfficinaDirettore[Nome]
  foreign key: Riparazione[Codice,Officina] ⊆ Relativa[Codice,Officina]
Relativa(Codice, Officina, Veicolo)
  foreign key: Relativa[Codice,Officina] ⊆ Riparazione[Codice,Officina]
  foreign key: Relativa[Veicolo] ⊆ Veicolo[Targa]
Terminata(Codice, Officina, OraRic, DataRic)
  foreign key: Terminata[Codice,Officina] ⊆ Riparazione[Codice,Officina]

Esercizio 9: soluzione – vincoli e viste

Vincoli:

- PersonaNonDip e Dipendente sono disgiunti:
  - $\text{PersonaNonDip}[\text{CodFis}] \cap \text{Dipendente}[\text{CodFis}] = \varnothing$
- Vincoli risultanti dai vincolo di foreign key verso Persona
  - $\text{Officina}[\text{Direttore}] \subseteq \text{PersonaNonDip}[\text{CodFis}] \cup \text{Dipendente}[\text{CodFis}]$
  - $\text{Veicolo}[\text{Proprietario}] \subseteq \text{PersonaNonDip}[\text{CodFis}] \cup \text{Dipendente}[\text{CodFis}]$
  - $\text{TelPer}[\text{CodFis}] \subseteq \text{PersonaNonDip}[\text{CodFis}] \cup \text{Dipendente}[\text{CodFis}]$
- Vincoli esterni:
  - riconsegna dopo accettazione
  - vincolo che lega Officina[NumDip] alle istanze in Lavora
  - vincolo su Officina[AnniAnzDir] e Dipendente[AnniAnz] derivante dall'eliminazione ISA

Viste per ricostruire le relazioni dello schema originario:

view Persona = $\text{PersonaNonDip} \cup \text{PROJ}_{\text{CodFis,Indirizzo}}(\text{Dipendente})$

view OfficinaOrig = $\text{PROJ}_{\text{Nome, NumDip, Indirizzo}}(\text{Officina})$

view Direttore = $\text{PROJ}_{\text{Direttore, EtaDir, AnniAnzDir}}(\text{Officina})$

view Dirige = $\text{PROJ}_{\text{Nome, Direttore}}(\text{Officina})$

view Telefono = $\text{PROJ}_{\text{Telefono}}(\text{TelPer})$