Appunti sulla Progettazione Concettuale di Basi di Dati

Basato sulle slide del Prof. Danilo Montesi

17 maggio 2025

Indice

1 Introduzione alla Progettazione Concettuale

L'obiettivo della progettazione concettuale è creare un modello dei dati che sia **indipendente** da qualsiasi specifico Database Management System (DBMS) o tecnologia. È la fase in cui tradiamo i requisiti del mondo reale in una struttura formale.

2 Il Processo di Progettazione di un Database

Il design di un database si articola in diverse fasi:

1. **Raccolta dei Requisiti del DB (**`DB requirements`**)**: Cosa deve fare il database? Quali dati deve memorizzare?
2. **Progettazione Concettuale (**`Conceptual Design`**)**:
    - È la fase di **ANALISI** ("`WHAT?`").
    - Si traduce i requisiti in un modello concettuale (es. Diagramma Entità-Relazione o E-R).
    - L'output è lo **Schema Concettuale (**`Conceptual Schema`**)**. Questo schema è una descrizione astratta della struttura del database, focalizzata sulle entità, gli attributi e le relazioni tra di esse, senza preoccuparsi di come verranno implementate.
3. **Progettazione Logica (**`Logical Design`**)**:
    - È la fase di **PROGETTAZIONE** ("`HOW?`").
    - Si traduce lo schema concettuale in un modello logico, specifico per un tipo di DBMS (es. relazionale, NoSQL).
    - L'output è lo **Schema Logico (**`Logical Schema`**)** (es. tabelle SQL con chiavi primarie/esterne, tipi di dato).
4. **Progettazione Fisica (**`Physical Design`**)**:
    - Si specificano i dettagli di implementazione fisica (es. indici, partizionamento).
    - L'output è lo **Schema Fisico (**`Physical Schema`**)**.

Noi ci concentriamo sulla Progettazione Concettuale.

3 Attività della Progettazione Concettuale e Modellazione dei Dati

Queste attività sono interconnesse e spesso iterative:

- **Elicitazione dei Requisiti (**`Requirements' elicitation`**)**: Raccogliere le informazioni.
- **Analisi dei Requisiti (**`Requirements' analysis`**)**: Capire e chiarire le informazioni.
- **Costruzione dello Schema Concettuale (**`Building the conceptual schema`**)**: Disegnare il modello E-R.
- **Costruzione del Glossario (**`Building the glossary`**)**: Definire i termini chiave.

4 Raccolta dei Requisiti

4.1 Fonti dei Requisiti

- **Utenti e Clienti**:
    - Interviste.
    - Documentazione specifica (*ad hoc*).
- **Documentazione Esistente**:

  - Leggi e regolamenti di settore.
  - Regolamenti interni, processi aziendali.
  - Soluzioni preesistenti.
- **Moduli (`Forms`)**: I moduli cartacei o digitali esistenti sono una miniera d'oro di informazioni sui dati.

4.2 Acquisizione e Analisi

- È un'attività **difficile e non standardizzata**.
- Spesso si parte da requisiti iniziali che necessitano di **raffinamento** attraverso ulteriori acquisizioni.

4.3 Acquisizione tramite Interviste

- **Diverse tipologie di utenti forniscono informazioni diverse**.
- I manager di alto livello hanno una visione più ampia ma meno dettagliata.
- Le interviste portano a raffinamenti successivi.

4.4 Interagire con gli Utenti: Consigli

- **Controlli di comprensione e coerenza frequenti**: "Quindi, se ho capito bene, ogni studente può iscriversi a più corsi, e ogni corso può avere più studenti?"
- **Esempi di casi d'uso (`Use cases`)**: Molto utili, specialmente casi generici e casi limite. "Cosa succede se uno studente si iscrive e poi si ritira? E se un corso non ha iscritti?"
- **Chiedere definizioni e classificazioni**: "Cosa intende esattamente per 'studente attivo'?"
- **Chiedere di evidenziare aspetti essenziali vs. periferici**: "È fondamentale tracciare lo storico degli indirizzi dello studente, o basta l'indirizzo attuale?"

5 Documentazione Descrittiva e Gestione dei Termini

5.1 Regole per la Documentazione Descrittiva

- **Scegliere il giusto livello di astrazione**.
- **Struttura delle frasi standard**: Semplifica l'analisi.
- **Dividere frasi troppo lunghe/complesse**.
- **Distinguere frasi sui "dati" da frasi sulle "funzioni"**:
  - Dati: "Uno studente *ha* un nome, un cognome e una matricola."
  - Funzioni: "Il sistema *deve permettere* di iscrivere uno studente a un corso."

5.2 Regole Generali per Termini e Concetti

- **Costruire un glossario dei termini**: Cruciale per evitare ambiguità.
  - *Esempio Pratico*: Se nel tuo team Node.js uno chiama un campo `customerId` e un altro `client_id`, il glossario chiarisce che `Customer` e `Client` sono sinonimi e si userà `customerId`.
- **Omonimi e sinonimi devono essere unificati**: Un solo termine per un concetto.
- **Chiarire esplicitamente le relazioni tra i termini**.
- **Ordinare le frasi per concetti**: Raggruppare requisiti simili.

6 Esempi di Requisiti

6.1 Esempio Database Bibliografico

- Automatizzare riferimenti bibliografici.
- ID di 7 caratteri (iniziali autori, anno, carattere disambiguazione).
- Riferimenti possono essere *monografie* (editore, data, luogo) o *articoli di rivista* (nome rivista, volume, numero, pagine, anno).
- Per entrambi: nomi degli autori.

6.2 Esempio Azienda di Formazione

Questo esempio sarà usato più avanti per la progettazione.

- Gestire corsi, lezioni, insegnanti.
- **Studenti (∼5000)**: ID, codice fiscale, cognome, età, sesso, luogo nascita, nome datori di lavoro (passati e presenti con date), indirizzo, telefono, corsi frequentati (∼200) con voto finale.
- **Workshop/Corsi**: Tracciare workshop frequentati, dove e quando si tengono le lezioni. I corsi hanno codice, titolo, edizioni (con data inizio/fine, numero partecipanti).
- **Studenti Freelance**: Area di interesse, titolo onorifico.
- **Insegnanti (∼300)**: Cognome, età, luogo nascita, nome corso insegnato, set corsi insegnati (passati/futuri), storico telefonate. Possono essere dipendenti interni o collaboratori esterni.

7 Il Glossario

Il glossario è fondamentale. Ecco un esempio basato sull'azienda di formazione:

Termine	Descrizione	Sinonimo	Correlato a
Partecipante	Chi prende parte ai corsi	Studente	Corso, Società
Docente	L'insegnante dei corsi. Potrebbe essere un dipendente interno.	Insegnante	Corso
Corso	Corso interno. Può avere diverse edizioni.	Workshop	Docente
Azienda	Luogo di lavoro attuale (o passato) del partecipante.	Luogo	Partecipante

Esempio Pratico: Nel tuo schema Prisma o MongoDB:

- `Participant` potrebbe diventare `model Student {}` o una collection `students`.
- Il glossario ti aiuta a decidere se `Lecturer` e `Teacher` sono la stessa cosa e come chiamare l'entità/collection (`model Teacher {}`).
- `Course` e `Workshop` sono sinonimi per la stessa entità/collection.

8 Strutturare i Requisiti

Dopo la raccolta, i requisiti vanno organizzati in gruppi omogenei di frasi. L'esempio dell'azienda di formazione viene strutturato:

- **Frasi generali**: "L'azienda richiede un DB per corsi, lezioni, insegnanti."
- **Frasi sui partecipanti**: Dettagli sugli studenti (ID, CF, nome, età, datori lavoro, corsi frequentati, ecc.).

- **Frasi specifiche sui partecipanti**: Dettagli per freelance (area interesse) o dipendenti di organizzazioni (livello gerarchico).
- **Frasi sul datore di lavoro**: Dettagli sui datori di lavoro dei partecipanti (nome, indirizzo, telefono).
- **Frasi sui corsi**: Dettagli sui corsi (codice, titolo, edizioni, date, n° partecipanti, aule, orari).
- **Frasi sui docenti**: Dettagli sugli insegnanti (cognome, età, corsi insegnati, tipo contratto, ecc.).

Questa strutturazione aiuta a identificare le future entità e le loro proprietà.

9 Dai Requisiti agli Schemi Concettuali (E-R)

Come tradurre i termini identificati nei costrutti del modello Entità-Relazione (E-R)?

- **Entità (`Entity`)**:
  - Se il termine ha **proprietà rilevanti** e descrive **oggetti autonomi**.
  - *Esempio*: `Studente`, `Corso`, `Docente`.
  - *Prisma/SQL*: Diventeranno tabelle (`model Student {}`, `CREATE TABLE Student (...)`).
  - *MongoDB*: Diventeranno collections (`db.students`).
- **Attributo (`Attribute`)**:
  - Se è un termine **semplice senza ulteriori specificazioni** (proprietà di un'entità).
  - *Esempio*: `Nome` dello Studente, `Titolo` del Corso.
  - *Prisma/SQL*: Diventeranno colonne nelle tabelle (`name: String`, `title: String`).
  - *MongoDB*: Diventeranno campi nei documenti (`{ name: "Mario", title: "Database 101" }`).
- **Relazione (`Relationship`)**:
  - Quando un termine **collega altri termini** (entità).
  - *Esempio*: "Studente *si iscrive a* Corso".
  - *Prisma/SQL*: Spesso implementate con chiavi esterne e tabelle di join.

```
model Student {
        id          Int      @id @default(autoincrement())
        // ... altri attributi
        enrollments Enrollment[]
}
model Course {
        id          Int      @id @default(autoincrement())
        // ... altri attributi
        enrollments Enrollment[]
}
model Enrollment { // Tabella di join
        studentId Int
        courseId  Int
        student   Student @relation(fields: [studentId], references: [id])
        course    Course  @relation(fields: [courseId], references: [id])
        enrollmentDate DateTime
        @@id([studentId, courseId])
}
```

  - *MongoDB*: Spesso implementate con DBRefs, array di ID, o embedding (se la relazione è 1-a-pochi e i dati sono strettamente legati).
- **Generalizzazione (`Generalization / ISA Relationship`)**:

- Quando un termine è un **caso più generale di un altro**.
- *Esempio*: `Persona` è una generalizzazione di `Studente` e `Docente`. Sia studenti che docenti sono persone e condividono attributi comuni (nome, cognome, CF) ma hanno anche attributi specifici.
- *Prisma/SQL*: Ci sono varie strategie:
  1. Tabella unica con un campo "tipo" (es. `Person` con `personType: "Student" | "Teacher"`).
  2. Tabelle separate per le specializzazioni che referenziano una tabella base comune.
  3. Tabelle separate che duplicano gli attributi comuni (meno ideale per la consistenza).
- *MongoDB*: Spesso si usa un campo `type` in una singola collection `people`, oppure collections separate se le differenze sono marcate.

10 Design Pattern E-R Comuni

Sono "best practices" per risolvere problemi comuni di modellazione.

10.1 Reificazione di Attributi in Entità

- **Problema**: Un attributo di un'entità ha esso stesso delle proprietà o partecipa ad altre relazioni.
- **Esempio**: Inizialmente `Company` è un attributo (es. `companyName`) di `Employee`.
  - Se `Company` deve avere un suo indirizzo, partita IVA, o essere collegata ad altri `Employee` o a `Projects`, allora `Company` va "reificata" (resa concreta) come un'entità separata.
  - Si crea l'entità `Company` e una relazione `Job` (o `WorksFor`) tra `Employee` e `Company`.
- **Cardinalità**: Un `Employee` (`1,1`) lavora per una `Company`. Una `Company` (`1,N`) può avere molti `Employee`.
  - (`1,1`): Esattamente uno.
  - (`1,N`): Da uno a molti.
  - (`0,1`): Zero o uno.
  - (`0,N`): Da zero a molti.
- *Esempio Pratico*:
  - *Prima*: `model Employee { id Int @id; name String; companyName String; }`
  - *Dopo*:

```
model Employee {
        id        Int     @id
        name      String
        companyId Int
        company   Company @relation(fields: [companyId], references: [id])
}
model Company {
        id        Int     @id
        name      String
        address   String? // Company ha i suoi attributi
        employees Employee[]
}
```

10.2 Relazioni "Part-of" (Composizione e Aggregazione)

- Relazioni (`1,N`) che rappresentano "parte di".
- **Composizione (`Composition`)**: Forte dipendenza. La parte non può esistere senza il tutto.
  - Esempio: `Cinema` (1) è composto da `Hall` (N). Ogni `Hall` (`1,1`) appartiene a un solo `Cinema`. Se il cinema viene distrutto, le sale non esistono più.

- **Aggregazione (`Aggregation`)**: Debole dipendenza. La parte può esistere indipendentemente dal tutto.
  - Esempio: `Team` (1) è composto da `Expert` (N). Un `Expert` (`0,1`) può appartenere a un `Team` (o a nessuno). Se il team si scioglie, gli esperti esistono ancora.

10.3 "Instance-of"

- **Problema**: Distinguere una rappresentazione astratta/modello da una sua istanza concreta.
- **Esempi**:
  - `Flight` (astratto: rotta, orario generico) vs. `ScheduledFlight` (istanza: volo specifico di un giorno con data, aereo assegnato).
  - `Tournament` (astratto: nome del torneo) vs. `Edition` (istanza: edizione 2024 del torneo, con date specifiche).
- *Esempio Pratico*:
  - `ProductTemplate` (nome, descrizione generica) vs `ProductInstance` (SKU specifico, colore, taglia, data di produzione).
  - `CourseDefinition` (codice, nome, crediti) vs `CourseOffering` (anno accademico, semestre, docente, aula).

10.4 Reificazione di Relazioni Binarie

- **Problema**: Una relazione tra due entità ha essa stessa degli attributi o partecipa ad altre relazioni.
- **Esempio**: Studente - Esame - Lezione.
  - Inizialmente, `Exam` potrebbe essere una relazione tra `Student` e `Lecture`.
  - Se l'esame ha attributi come `Grade` (voto) e `Date`, allora `Exam` viene reificata come entità.
  - Si creano due relazioni binarie: `Student-takes-Exam (S-E)` e `Exam-is_for-Lecture (E-L)`.
- *Esempio Pratico (Prisma, vedi sopra per `Enrollment`)*: Se la relazione "studente si iscrive a corso" ha una data di iscrizione, un voto, ecc., la tabella di join `Enrollment` diventa un'entità reificata.

10.5 Reificazione di Relazioni Ricorsive

- **Problema**: Una relazione tra istanze della stessa entità ha attributi.
- **Esempio**: `Team` gioca una `Match` contro un altro `Team`.
  - Una relazione `PlaysAgainst` tra `Team` e `Team`.
  - Se la partita (`Match`) ha attributi come `Date`, `Score`, allora `Match` viene reificata come entità.
  - Si creano due relazioni: `Team_Home-plays-Match` e `Team_Visiting-plays-Match`.
- *Esempio Pratico*: Un `Employee` può essere `manager` di altri `Employee`. Se questa relazione di management ha una `startDate` o un `roleDescription`, si potrebbe reificare in un'entità `ManagementRelationship`.

10.6 Reificazione di Attributi di Relazioni

- **Problema**: Un attributo di una relazione molti-a-molti ha esso stesso delle proprietà.
- **Esempio**: `Player` (musicista) - `Plays` (suona) - `Orchestra`. La relazione `Plays` ha un attributo `Instrument`.
  - Se `Instrument` (es. "Violino Stradivari Modello X") deve avere attributi propri (es. `Trademark`, `Type`, anno di fabbricazione) o essere suonato da più musicisti in diverse orchestre, allora `Instrument` va reificato.
  - Si crea l'entità `Instrument` e la relazione `Plays` diventa ternaria (o si reifica `Plays` in un'entità che collega `Player`, `Orchestra`, `Instrument`).

10.7 Caso Specifico (Generalizzazione/ISA)

- **Problema**: Una sottocategoria di un'entità ha caratteristiche o relazioni aggiuntive.
- **Esempio**: `Manager` è un caso specifico di `Employee`.
  - Tutti i `Manager` sono `Employee`, ma solo i `Manager` gestiscono (`Manage`) dei `Project`.
  - Non tutti gli `Employee` gestiscono progetti.
- Si usa una freccia di generalizzazione (concettualmente) da `Manager` a `Employee`.

10.8 Storicizzazione di un Concetto

- **Problema**: Necessità di tracciare i cambiamenti di un concetto nel tempo.
- **Esempi**:
  - `Anagraphic` (dati anagrafici) può avere una versione `Historic` e una `Current`. Si usano attributi come `StartDate`, `ExpiryDate`.
  - `Software` può avere una versione `Legacy` e una `Updated`.
  - **Impiego (`Employment`)**: Si può modellare `CurrentEmployment` e `PastEmployment`.
    - *Opzione 1*: Due relazioni separate (`CurrentEmployment`, `PastEmployment`) tra `Employee` e `Company`.
    - *Opzione 2*: Reificare `Employment` come entità con `BeginDate`, `EndDate` e poi generalizzarla in `CurrentEmployment` e `PastEmployment` (o usare un attributo di stato).
- *Esempio Pratico*: Per tracciare lo storico degli indirizzi di un cliente:

```
model Customer {
        id              Int       @id
        // ...
        addressHistory Address[]
}
model Address {
        id          Int       @id
        street      String
        city        String
        customerId Int
        customer    Customer  @relation(fields: [customerId], references: [id])
        startDate  DateTime
        endDate    DateTime? // Null se è l'indirizzo corrente
}
```

10.9 Estensione di un Concetto (Generalizzazione/ISA)

- **Problema**: Un concetto esistente viene esteso con nuove informazioni per casi specifici.
- **Esempio**: `Project` è un concetto generale. Un `AcceptedProject` (progetto accettato) richiede informazioni aggiuntive come `Founding` (finanziamento) e `StartDate`, che non sono necessarie per progetti in attesa o rifiutati.
- Si usa una generalizzazione (concettualmente) da `AcceptedProject` a `Project`.

10.10 Relazioni Ternarie e Loro Reificazione

- **Relazione Ternaria**: Coinvolge tre entità.
  - Esempio: `Employee` lavora su un `Task` in un `Office`. La relazione `Work` collega queste tre.
  - Le cardinalità indicano che un impiegato può lavorare su più task in più uffici, un ufficio può ospitare più impiegati su più task, e un task può essere svolto da più impiegati in più uffici.

- **Reificazione di Relazione Ternaria (1)**:
  - La relazione ternaria `Work` viene trasformata in un'entità `Work`.
  - L'entità `Work` è collegata a `Employee`, `Office`, e `Task` tramite tre relazioni binarie (E-W, O-W, T-W).
  - Questo è utile se l'evento "Work" ha attributi propri (es. `duration`, `status`).
- **Reificazione di Relazione Ternaria (2) - Semplificata**:
  - Se ci sono vincoli specifici (es. "un task può essere eseguito da un solo operatore e in un solo ufficio"), il modello può essere semplificato.
  - Nell'esempio delle slide, `Task` diventa centrale, con una relazione (`1,1`) verso `Employee` (tramite O-S, probabilmente "Operator-for-Service") e (`1,1`) verso `Office` (tramite S-I, probabilmente "Service-at-Location").
  - *Nota*: Le etichette delle relazioni (O-S, S-I) sono un po' criptiche, ma il concetto è la semplificazione basata su vincoli.

11 Strategie di Progettazione dello Schema E-R

Come si affronta la creazione dello schema E-R?

11.1 Strategia Top-Down

1. Si parte dai concetti più generali (entità principali).
2. Si raffinano progressivamente aggiungendo dettagli:
   - Identificare attributi.
   - Identificare relazioni.
   - Scomporre entità complesse.
   - Introdurre generalizzazioni/specializzazioni.

Esempi:

- `Exam` (iniziale) $\rightarrow$ `Student` - `Exam` (relazione) - `Lecture`.
- `Employee` (iniziale) $\rightarrow$ `Employee` con attributi `Surname`, `Age`, `Wage`.
- `People` (iniziale) $\rightarrow$ Generalizzazione in `Man` e `Woman`.

11.2 Strategia Bottom-Up

1. Si parte dai dettagli: attributi e concetti specifici.
2. Si raggruppano per formare entità e relazioni.
3. Si integrano i vari "pezzi" di schema per formare lo schema completo.

Esempi:

- Requisito su `Employee` $\rightarrow$ Entità `Employee`.
- Concetti `Student`, `Exam`, `Lecture` $\rightarrow$ Schema `Student-Exam-Lecture`.
- Entità `Man`, `Woman` $\rightarrow$ Generalizzazione `People`.

11.3 Strategia Inside-Out

1. Si identifica un concetto centrale e ben compreso.
2. Si espande lo schema "verso l'esterno", aggiungendo concetti (entità, attributi, relazioni) direttamente collegati a quelli già identificati.

Esempio (dalle slide):

1. Inizio: `Employee`.
2. Aggiungo attributi a `Employee`: `Surname`, `Code`.
3. Aggiungo `Dept` (Dipartimento) e le relazioni `Supervision` (Employee supervisiona Dept) e `Belonging` (Employee appartiene a Dept, con attributo `Date`).
4. Aggiungo `Project` e la relazione `Enrollment` (Employee partecipa a Project).
5. Aggiungo `Office` e la relazione `Composition` (Dept è composto da Office, con attributo `Addr` complesso).

12 Regola Pratica e Metodologia

12.1 Regola Pratica: Usare uno Stile Misto!

1. **Crea uno "schizzo" (`sketch`)**: Identifica le entità più rilevanti.
2. **Decomponi lo schema**: Dividi il problema se complesso.
3. **Raffina (top-down), integra (bottom-up), espandi (inside-out)**.

12.2 Sketching dello Schema E-R

- Parti dalle entità più rilevanti (più citate o esplicitamente indicate come tali).
- Crea un primo schema E-R di base.

12.3 Metodologia "Best Practice"

1. **Analisi dei Requisiti**:
   - Analizza, risolvi ambiguità.
   - Crea un glossario.
   - Raggruppa requisiti simili.
2. **Caso Base (`Base case`)**:
   - Definisci uno schema "abbozzato" con i concetti più rilevanti.
3. **Caso Iterativo (`Iterative case`)** (ripeti finché non va bene):
   - Raffina i concetti base usando i requisiti.
   - Aggiungi concetti per descrivere requisiti non ancora coperti.
4. **Analisi di Qualità (`Quality analysis`)** (ripeti durante tutto il processo):
   - Controlla la qualità dello schema e modificalo.

13 Qualità dello Schema E-R

Misure di qualità per uno schema E-R:

- **Correttezza (`Correctness`)**: Lo schema rappresenta accuratamente i requisiti? Usa i costrutti E-R in modo appropriato?
- **Completezza (`Completeness`)**: Tutti i requisiti sono stati rappresentati nello schema? Tutti i dati necessari sono modellati?
- **Chiarezza (`Clarity`)**: Lo schema è facile da capire? È ambiguo?
- **Minimalità (`Minimality`)**: Ci sono elementi ridondanti (entità, attributi, relazioni non necessarie)? Si potrebbe rappresentare la stessa informazione in modo più semplice?

14 Best Practice e Integrazione di Schemi

Per sistemi complessi, si può decomporre il problema:

14.1 Approccio 1

1. Analisi dei Requisiti.
2. Caso Base (schema "scheletro" generale).
3. **Decomposizione**: Suddividi i requisiti complessi secondo lo schema scheletro.
4. Caso Iterativo per ogni **sotto-schema**.
5. **Integrazione**: Unisci i sotto-schemi in uno schema totale, usando lo schema scheletro come riferimento.
6. Analisi di Qualità.

14.2 Approccio 2

1. Analisi dei Requisiti.
2. **Decomposizione**: Identifica aree di interesse e partiziona i requisiti (o acquisiscili separatamente per area).
3. **Per ogni area**:
    - Caso Base.
    - Caso Iterativo.
4. **Integrazione**: Unisci gli schemi delle varie aree.
5. Analisi di Qualità.

15 Esempio Finale: Azienda di Formazione

Questo è un'applicazione pratica della metodologia all'esempio dell'azienda di formazione.

15.1 Affermazione Generale

"Azienda di formazione richiede DB per corsi, lezioni, insegnanti."

15.2 Schema Abbozzato (Sketched Schema)

- Entità: Participant, Lecture (Lezione/Corso), Lecturer (Docente).
- Relazioni: Presence (Participant - Lecture), Teaching (Lecturer - Lecture).

15.3 Raffinamento: Partecipanti e Datori di Lavoro

- Requisiti: ID, CF, dati anagrafici, datori di lavoro (passati/presenti), corsi frequentati. Freelance vs. Dipendenti.
- **Schema Parziale (1)** (basato sulla slide 70):
  - Entità Participant con attributi (Tax, Code, ...).
  - Generalizzazione: Participant è generalizzazione di Employee (con attributi Level, Position) e Freelance (con Title, Area).
  - Entità Employer (Datore di lavoro) con attributi (Name, ...).
  - Relazioni: CurrEmpl (tra Participant e Employer per impiego attuale, 1-a-N), PastEmpl (tra Participant e Employer per impieghi passati, N-a-N, reificata).

15.4 Raffinamento: Corsi

- Requisiti: Corsi ($\sim$200) con codice, titolo, edizioni (con data inizio/fine, n° partecipanti), lezioni (giorno, aula, orario).
- **Schema Parziale (2)** (basato sulla slide 72):
  - Pattern "Instance-of": Lecture (Corso generico) e Edition (Edizione specifica del corso).
    - Lecture (Attributi: Title, Code).
    - Edition (Attributi: Start, End, #Part. - numero partecipanti).
    - Relazione KindOf (1,1) tra Edition e Lecture (un'edizione è di un solo tipo di corso).
  - Pattern "Part-of": Edition è composta da Lesson (singola lezione).
    - Lesson (Attributi: Time, Room, Day).
    - Relazione MadeOf (1,N) tra Edition e Lesson.

15.5 Raffinamento: Docenti

- Requisiti: Docenti ($\sim$300) con dati anagrafici, corsi insegnati (passati/futuri), storico telefonate. Dipendenti interni vs. Esterni.
- **Schema Parziale (3)** (basato sulla slide 74):
  - Entità Lecturer con attributi (Tax, Surname, Age, Place of Birth, Phone (multi-valore, 1,N)).
  - Generalizzazione: Lecturer è generalizzazione di Independent (esterno) e Home (interno).

15.6 Integrazione dello Schema

Si parte dallo schema abbozzato e si integrano i raffinamenti.

- **Schema Intermedio (1)** (basato sulla slide 76): Integrazione di Participant e Lecture.
  - Relazioni PastPresence (0,N)-(0,N) e CurrentPresence (0,1)-(0,N) tra Participant e Lecture. (Questo modella la frequenza ai corsi/lezioni).
- **Schema Intermedio (2)** (basato sulla slide 77): Integrazione di Lecturer, Lecture, Edition.
  - Relazioni Past (0,1)-(0,N) e Current (0,1)-(0,N) tra Edition e Lecturer (per insegnamento).
  - Relazione Duty (1,N)-(0,N) tra Lecturer e Lecture (per indicare i corsi che un docente *può* insegnare o *ha insegnato* in generale, separato dalle specifiche edizioni).

15.7 Schema Finale (Solo Entità e Relazioni)

La slide 78 mostra la struttura complessiva integrando tutti i pezzi, omettendo gli attributi per chiarezza. Si vedono chiaramente:

- `Participant` generalizzato in `Employee` e `Freelance`.
- `Employee` collegato a `Employer`.
- `Lecturer` generalizzato in `Independ.` e `Home`.
- Il nucleo `Lecture` $\rightarrow$ `Edition` $\rightarrow$ `Lesson`.
- Le relazioni che collegano `Participant` a `Lecture`/`Edition` (frequenza).
- Le relazioni che collegano `Lecturer` a `Lecture`/`Edition` (insegnamento).

Questo processo iterativo di sketching, raffinamento e integrazione, guidato dai requisiti e supportato da un glossario e da pattern di progettazione, porta a uno schema concettuale robusto.