Informatica Grafica

Corso di Laurea in Ingegneria Edile – Architettura

Basi di dati

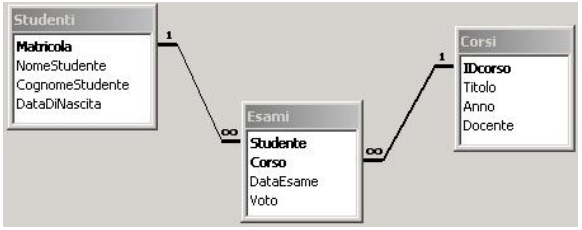
Paolo Torroni

Dipartimento di Elettronica, Informatica e Sistemistica (DEIS)
Università degli Studi di Bologna

Anno Accademico 2011/2012

Basi di dati

	Matricola	NomeStudente	CognomeStudente	DataDiNascita
	1111	Mario	Neri	02/10/1983
	2222	Elena	Rossi	05/07/1982
	3333	Mario	Bianchi	24/01/1983
	4444	Giuseppe	Neri	15/11/1981
▶				



- **Basi di dati**
  - Le informazioni sono tra le risorse strategiche più importanti
  - Processi informativi:
    - raccolta, acquisizione delle informazioni;
    - archiviazione, conservazione delle informazioni;
    - elaborazione delle informazioni;
    - distribuzione, scambio delle informazioni.
  - Come organizzare i dati? Modello relazionale
  - Come elaborare i dati per ottenere informazioni? SQL
  - Sofware per la gestione di basi di dati (⌨ LibreOffice.org)

Parte I

Introduzione

Concetti di base

- **Sistema informativo**: sistema di supporto ai processi informativi di un'organizzazione.
  - Non è legato in alcun modo all'Informatica
- **Sistema informatico**: porzione di sistema informativo gestita in modo automatico mediante tecnologie informatiche.
  - Solo una porzione di gran parte dei sistemi informativi
  - L'informazione da manipolare/gestita deve essere resa omogenea
  - Introduzione di artefatti (es: codice fiscale per identificare le persone)
- Le **informazioni** sono rappresentate per mezzo di **dati**, che devono essere **interpretati** per fornire informazione

Sistemi software dedicati alla gestione delle informazioni

- **Sistema di archiviazione**: memorizzazione e ricerca di informazioni invarianti nel tempo.
  - Enciclopedia, dizionario, . . . (CD, DVD, . . .)
- **Banca dati**: memorizzazione e ricerca di informazioni che crescono nel tempo.
  - Condivisione tra utenti in rete (es. Gazzette Ufficiali).
  - Non equipaggiata per gestire aggiornamenti frequenti delle informazioni.
- **Basi di dati**: collezione di dati.
  - Dimensioni e persistenza: uso della memoria secondaria. I dati vivono più a lungo dei programmi che li gestiscono.
  - Condivisione: garantire accesso coerente di applicazioni e utenti diversi a dati comuni.

Sistemi software dedicati alla gestione delle informazioni

- **Database Management System (DBMS)**: sistema software per gestire collezioni grandi, condivise e persistenti di dati.
  - **Affidabilità**: conservazione dei dati a fronte di guasti. Meccanismi di *backup* e *recovery*.
  - **Privatezza**: più utenti, meccanismi di autorizzazione.
  - **Efficienza**: utilizzo delle risorse accettabile dall'utente.
  - **Efficacia**: capacità di rendere produttivi gli utenti.

Tipologie di basi di dati

- Non esiste una sola tipologia di basi di dati
- Scelta dipende da alcuni parametri di utilizzo previsto

Tabella: Esempi di tipologie di basi di dati

Tipologia	**Dimensioni**	**Utenti**	**Oper/s**	**DBMS**
Personali	10MB-100MB	1-3	< 3	Open/LibreOffice.org, Access, MySQL, PostgreSQL
Gruppo/Dip	100MB-10GB	3-100	< 300	SQLserver, MySQL, PostgreSQL
Aziendali	10GB-100GB	100-1000	< 30.000	SQLServer, Oracle, DB2
Grandi Aziende	100GB-10TB	> 1000	< 300.000	Oracle, DB2

Organizzazione dei dati nel WWW

- Il WWW non è un DBMS
  - Dati non omogenei, non strutturati, non consistenti, ...
- Però: alcune caratteristiche interessanti
  - **Dati**: all'interno di pagine che hanno un **indirizzo** (URL).
  - **Correlazione tra** e **accesso a pagine**: attraverso informazioni aggiuntive (**link**, URL).
- **Modello dei dati**: insieme dei **concetti** utilizzati per
  - **organizzare** i dati di interesse e
  - **descriverne la struttura**, in modo che risulti
  - **comprensibile ad un computer**

Modello dei dati

- Vari tipi di modelli dei dati:
  - **Gerarchico**: uso di **strutture ad albero** (file system)
  - **Reticolare**: uso di **grafi** (come nel Web)
  - **Relazionale**: basato sul costrutto di **relazione**, rappresentato mediante una **tabella**.
- Esempio di base di dati universitaria con tre tabelle:
  - Studenti, Esami, Corsi.

Tabella: Studenti

Matricola	**Cognome**	**Nome**	**DataDiNascita**
6545	Rossi	Maria	15/10/1978
8678	Pinti	Paola	13/12/1976
4567	Verdi	Luigi	02/09/1979
3465	Rossi	Mario	11/12/1978

Modello dei dati

- Vari tipi di modelli dei dati:
  - **Gerarchico**: uso di **strutture ad albero** (file system)
  - **Reticolare**: uso di **grafi** (come nel Web)
  - **Relazionale**: basato sul costrutto di **relazione**, rappresentato mediante una **tabella**.
- Esempio di base di dati universitaria con tre tabelle:
  - Studenti, Esami, Corsi.

Tabella: Esami

Studente	**Voto**	**Corso**
3465	29	03
3465	28	02
6545	30	02
8678	18	01

Modello dei dati

- Vari tipi di modelli dei dati:
  - **Gerarchico**: uso di **strutture ad albero** (file system)
  - **Reticolare**: uso di **grafi** (come nel Web)
  - **Relazionale**: basato sul costrutto di **relazione**, rappresentato mediante una **tabella**.
- Esempio di base di dati universitaria con tre tabelle:
  - Studenti, Esami, Corsi.

Tabella: Corsi

Codice	**Titolo**	**Docente**
01	Matematica	Rossi
02	Storia	Bruni
03	Inglese	Verdi

- Dati correlati tra di loro attraverso **valori comuni**

Schema e Istanza

- **Schema**, invariante nel tempo: **caratteristiche** dei dati

Tabella: Studenti

$\Rightarrow$

Matricola	**Cognome**	**Nome**	**DataDiNascita**
6545	Rossi	Maria	15/10/1978
8678	Pinti	Paola	13/12/1976
4567	Verdi	Luigi	02/09/1979
3465	Rossi	Mario	11/12/1978

Schema e Istanza

- **Schema**, invariante nel tempo: **caratteristiche** dei dati
- **Istanza** o **stato** della base di dati: **valori effettivi**

Tabella: Studenti

	Matricola	**Cognome**	**Nome**	**DataDiNascita**
$\Rightarrow$	6545	Rossi	Maria	15/10/1978
$\Rightarrow$	8678	Pinti	Paola	13/12/1976
$\Rightarrow$	4567	Verdi	Luigi	02/09/1979
$\Rightarrow$	3465	Rossi	Mario	11/12/1978

Parte II

Modello relazionale dei dati

Modello relazionale

- Basato sul concetto matematico di **relazione**
- Rispetto a modelli gerarchico e reticolare:
  - Basato unicamente sui **valori**
    - Valori numerici, o sequenze di caratteri (matricola)
    - Nessun riferimento esplicito tra strutture di memorizzazione per esprimere **dipendenze** tra dati
  - Favorisce l'indipendenza dei dati
    - Lo **schema logico** è fatto usando solo strutture tabellari

Esempio

- Gestione di dati su libri e sui loro autori
- Due classi di oggetti, ciascuna con specifiche **proprietà**, o **attributi**
  - **Libro**: titolo e autore
  - **Autore**: nome, cognome e anno di nascita
- Una **tabella** (**relazione**) per ciascuna classe
  - Struttura **logica**, indipendente dai meccanismi di memorizzazione adottati
  - Identificata da un **nome**
  - Colonne $\Rightarrow$ **attributi**, righe $\Rightarrow$ **istanze**
  - Accesso alle istanze effettuato tramite **valori** degli attributi

Tabella: Libri

Titolo	**Editore**	**Autore**
Il ritorno delle gru	Bompiani	Trevanian
L'estate di Katya	Rizzoli	Trevanian

Definizioni formali

Tabella: Libri

Titolo	**Editore**	**Autore**
Il ritorno delle gru	Bompiani	Trevanian
L'estate di Katya	Rizzoli	Trevanian

Definizione (Dominio di un attributo)

Per ogni attributo, è definito un **dominio**, *cioè l'insieme di tutti i suoi possibili valori*

- $\mathcal{D}(\textbf{Titolo}) =$ stringhe di 50 caratteri
- $\mathcal{D}(\textbf{Disponibile}) = \{$ Sì, No $\}$

Definizioni formali

Tabella: Libri

Titolo	**Editore**	**Autore**
Il ritorno delle gru	Bompiani	Trevanian
L'estate di Katya	Rizzoli	Trevanian

Definizione (Schema di relazione)

Data una relazione, lo **schema della relazione** *è una rappresentazione della sua* **struttura**

- Nome della relazione, attributi, domini.
- $R(A_1, \ldots, A_n)$

Libri	**Titolo**	**Editore**	**Autore**

Definizioni formali

Tabella: Libri

Titolo	**Editore**	**Autore**
Il ritorno delle gru	Bompiani	Trevanian
L'estate di Katya	Rizzoli	Trevanian

Definizione (Ennupla)

*L'***ennupla** *è una* **funzione** *che associa a ciascun attributo un valore preso dal suo dominio*

- È una possibile scelta di valori (uno per attributo).
- $a_1 \in \mathcal{D}(A_1)$, $a_2 \in \mathcal{D}(A_2)$, $\ldots \Rightarrow (a_1, \ldots, a_n)$
- Corrisponde alla **riga** di una tabella

Il ritorno delle gru	Bompiani	Trevanian

Definizioni formali

Tabella: Libri

Titolo	**Editore**	**Autore**
Il ritorno delle gru	Bompiani	Trevanian
L'estate di Katya	Rizzoli	Trevanian

Definizione (Istanza di una relazione)

*L'***istanza di una relazione** *(o semplicemente* **relazione***) è un insieme di ennuple*

- $istanza(R(A_1, \ldots, A_n)) \subseteq \mathcal{D}(A_1) \times \cdots \times \mathcal{D}(A_n)$

Il ritorno delle gru	Bompiani	Trevanian
L'estate di Katya	Rizzoli	Trevanian

Condizioni perché una tabella rappresenti una relazione

Tabella: Libri

Titolo	**Editore**	**Autore**
Il ritorno delle gru	Bompiani	Trevanian
L'estate di Katya	Rizzoli	Trevanian

1. ogni **colonna** della tabella ha una diversa **intestazione**;
2. i **valori** presenti in ogni **colonna** sono **omogenei** fra di loro;
3. l'**ordinamento** tra le colonne è **irrilevante**.
4. le **righe** sono **diverse** fra loro;
5. l'**ordinamento** tra le righe è **irrilevante**.

- Nel seguito: relazione $\leftrightarrow$ tabella

Basi di dati

Definizione (Schema della base di dati)

Uno **schema di una base di dati** *è costituito da un insieme di schemi di relazione. A tale insieme è dato un nome (nome della base di dati)*

Definizione (Istanza di base di dati)

*Dato uno schema di una base di dati R, un'***istanza di una base di dati** *su tale schema è costituita da un insieme di istanze di relazioni I, dove I contiene un'istanza di relazione per ogni schema di relazione presente in R.*

- Lo schema rappresenta le intestazioni delle tabelle.
- L'istanza contiene i dati veri e propri (righe di tutte le tabelle).

Perché il modello relazionale?

- Basato unicamente sui valori
- Notevoli vantaggi rispetto all'organizzazione del Web:
  - Schema indipendente da come vengono memorizzati i dati
  - Trasporto agevole dei dati da un sistema a un altro
  - Rappresentate solo le informazioni rilevanti per l'applicazione
  - Associazioni bidirezionali tra dati

Informazione incompleta

- Data una ennupla, non è detto che abbiamo tutte le informazioni su tutti gli attributi.
- Possibili tre situazioni diverse:
  - Si sa che per l'ennupla esiste il valore di un attributo, ma non si sa qual è
    - Non so chi ha scritto il libro "L'estate di Katya"
    - Valore **sconosciuto**
  - Per l'ennupla si sa che non è definito un certo attributo
    - Nessun editore ha pubblicato il libro "L'estate di Katya"
    - Valore **inesistente**
  - Non si è sicuri se per l'ennupla sia definito o meno l'attributo
    - Valore **senza informazione**
- Necessaria estensione della definizione di ennupla
  - Valore **NULL** (nessun dominio)
  - Unico valore convenzionale per i tre casi sopra

Parte III

Vincoli di integrità

Errori nei dati

- Possibile immettere dati sintatticamente corretti, ma semanticamente privi di senso

Tabella: Esami

Studente	**Corso**	**Voto**	**Lode**
Rossi	Informatica	32	No
Verdi	Matematica	18	Sì
Neri	Oroscopia	25	No

- Prima ennupla: **Voto** $32 \notin [18..30]$
- Seconda ennupla: **Voto** e **Lode** non correlati
- Terza ennupla: **Corso** (probabilmente) errato

Vincoli di integrità

- Si usano per limitare l'immissione di dati scorretti
- Sono formulazioni di **proprietà** che i dati devono soddisfare
- Definiti **a livello di schema**
  - Devono essere soddisfatti da tutte le possibili istanze corrette dello schema
- Modellano alcune caratteristiche rilevanti della realtà che si vuole rappresentare
- Quattro tipi di vincoli di integrità:
  - vincoli su **valori** (o di dominio);
  - vincoli di **ennupla** (o di riga);
  - vincoli di **chiave**;
  - vincoli di **integrità referenziale**.

Vincoli di dominio

- Coinvolgono solo gli attributi di una singola relazione
- **Vincoli intra-relazionali**
- Esprimono condizioni sui valori
  - di **un singolo** attributo (colonna)
  - di una singola ennupla (riga)

Tabella: Esami

Studente	**Corso**	**Voto**	**Lode**
Rossi	Informatica	32	No
Verdi	Matematica	18	Sì
Neri	Oroscopia	25	No

Vincoli di dominio

- Coinvolgono solo gli attributi di una singola relazione
- **Vincoli intra-relazionali**
- Esprimono condizioni sui valori
  - di **un singolo** attributo (colonna)
  - di una singola ennupla (riga)

Tabella: Esami

$\Rightarrow$

Studente	**Corso**	**Voto**	**Lode**
Rossi	Informatica	**32**	No
Verdi	Matematica	18	Sì
Neri	Oroscopia	25	No

- Prima ennupla: $32 \notin \mathcal{D}(\textbf{Voto})$

Vincoli di ennupla

- Coinvolgono solo gli attributi di una singola relazione
- **Vincoli intra-relazionali**
- Esprimono condizioni sui valori
  - di **più attributi** (colonne)
  - di una singola ennupla (riga)

Tabella: Esami

Studente	**Corso**	**Voto**	**Lode**
Rossi	Informatica	32	No
Verdi	Matematica	18	Sì
Neri	Oroscopia	25	No

- *Se* **Lode**=Sì *allora* **Voto**=30

Vincoli di ennupla

- Coinvolgono solo gli attributi di una singola relazione
- **Vincoli intra-relazionali**
- Esprimono condizioni sui valori
  - di **più attributi** (colonne)
  - di una singola ennupla (riga)

Tabella: Esami

	Studente	**Corso**	**Voto**	**Lode**
	Rossi	Informatica	32	No
$\Rightarrow$	Verdi	Matematica	**18**	**Sì**
	Neri	Oroscopia	25	No

- *Se* **Lode**=Sì *allora* **Voto**=30
- Seconda ennupla: vincolo violato

Chiave

- È un attributo che permette di **identificare univocamente le ennuple** di una tabella
  - Esempi: ISBN, CF, Matricola (Studenti), Codice (Corsi)
  - Anche più attributi: (Studente, Corso) nella tabella Esami
- Costituita da insieme **minimale** di attributi
  - Esempio: (CF, ~~Nome~~) in una tabella Contribuenti

Tabella: Esami

Studente	**Corso**	**Voto**	**Lode**
Rossi	Informatica	32	No
Verdi	Matematica	18	Sì
Rossi	Informatica	25	No

Chiave

- È un attributo che permette di **identificare univocamente le ennuple** di una tabella
  - Esempi: ISBN, CF, Matricola (Studenti), Codice (Corsi)
  - Anche più attributi: (Studente, Corso) nella tabella Esami
- Costituita da insieme **minimale** di attributi
  - Esempio: (CF, ~~Nome~~) in una tabella Contribuenti

Tabella: Esami

	Studente	**Corso**	**Voto**	**Lode**
⇒	Rossi	Informatica	32	No
	Verdi	Matematica	18	Sì
⇒	Rossi	Informatica	25	No

- Prima e terza ennupla: stessa chiave!

Vincoli di chiave

- Properietà della chiave:
  - Esiste sempre (tabelle: **insiemi** di ennuple)
  - Definita considerando insiemi **minimali** di attributi
  - Garantisce accessibilità a ciascun dato della base di dati
  - Attributo **essenziale** perché accesso solo tramite valori
  - **Chiave primaria**, scelta tra tutte le chiavi:
    - Non sono permessi valori nulli
    - Non sono permessi duplicati
- Vincoli di chiave:
  - Coinvolgono solo gli attributi di una singola relazione
  - **Vincoli intra-relazionali**
  - Vincoli di **unicità** e **presenza** della chiave nella relazione

Vincoli di integrità referenziale

- Coinvolgono gli attributi di più relazioni
- **Vincoli inter-relazionali**
- Permettono di **correlare** i dati memorizzati in **tabelle diverse**
- Impongono che i valori degli attributi di **una relazione** compaiano nella **chiave primaria di un'altra relazione**
- Un vincolo di integrità referenziale:
  - fra gli attributi X di una relazione R_1 e un'altra relazione R_2
  - impone ai valori di X in R_1
  - di comparire come valori della chiave primaria di R_2
- Esempio: vincolo tra Studente (Esami) e Matricola (Studenti)

Tabella: Esami

Studente	**Corso**	**Voto**
0142	Matematica	28
8392	Informatica	30L
1022	Informatica	23

Tabella: Studenti

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Vincoli di integrità referenziale

- Coinvolgono gli attributi di più relazioni
- **Vincoli inter-relazionali**
- Permettono di **correlare** i dati memorizzati in **tabelle diverse**
- Impongono che i valori degli attributi di **una relazione** compaiano nella **chiave primaria di un'altra relazione**
- Un vincolo di integrità referenziale:
  - fra gli attributi X di una relazione R_1 e un'altra relazione R_2
  - impone ai valori di X in R_1
  - di comparire come valori della chiave primaria di R_2
- Esempio: vincolo tra Studente (Esami) e Matricola (Studenti)

Tabella: Esami

	Studente	**Corso**	**Voto**
	0142	Matematica	28
	8392	Informatica	30L
⇒	**1022**	Informatica	23

Tabella: Studenti

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Parte IV

Linguaggi delle basi di dati

Linguaggi delle basi di dati

- Due tipologie di linguaggi
  - **Linguaggi testuali**: comandi per la definizione e manipolazione di dati impartiti mediante sequenze di caratteri
  - **Linguaggi grafici**: uso di interfacce grafiche
- Per **progettare** un db relazionale bisogna definire le relazioni
  1. **Progetto logico** della base di dati
     - Si fa solo riferimento al modello dei dati
     - Prescinde dal software (DBMS) che si vuole usare
  2. **Definizione nel contesto di un DBMS**
     - Si usa un **Data Definition Language** (DDL)
     - Insieme di costrutti per definire i dati (tabelle)
     - Linguaggi testuali: es. `CREATE TABLE Studenti`
     - Spesso: interfaccia grafica
- Per **utilizzare** il database:
  - **Accesso ai dati** (**interrogazioni**, o **query**)
  - Uso di un **Data Manipulation Language** (DML)
- Spesso: stesso linguaggio fa da DDL e DML (come *SQL*)

Query

- Di norma le query sono definiti dai progettisti
- L'utente finale usa il db attraverso **maschere** in cui vengono inseriti dei parametri in base a cui selezionare/aggiornare dati
- DML basati su **algebra relazionale**
  - Insieme di operatori che manipolano relazioni
  - Argomenti: relazioni
  - Risultato: una nuova relazione
  - Costrutti principali:
    - **operatori insiemistici**: $\cup, \setminus, \cap$
    - **ridenominazione**
    - **proiezione**
    - **selezione**
    - **join**

Operatori insiemistici

- Solo su tabelle con gli stessi attributi
- Operatori **binari** di unione, differenza, intersezione

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: Stud_Sci

Matricola	**Nome**	**Cognome**
8279	Fabrizia	Bianchi
0989	Stefano	Neri
0142	Giulia	Bianchi

Operatori insiemistici

- Solo su tabelle con gli stessi attributi
- Operatori **binari** di unione, differenza, intersezione

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: Stud_Sci

Matricola	**Nome**	**Cognome**
8279	Fabrizia	Bianchi
0989	Stefano	Neri
0142	Giulia	Bianchi

- **Unione**

Tabella: Stud_Ing $\cup$ Stud_Sci

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
8392	Giulia	Bianchi
0142	Giulia	Bianchi
8279	Fabrizia	Bianchi
0989	Stefano	Neri

Operatori insiemistici

- Solo su tabelle con gli stessi attributi
- Operatori **binari** di unione, differenza, intersezione

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: Stud_Sci

Matricola	**Nome**	**Cognome**
8279	Fabrizia	Bianchi
0989	Stefano	Neri
0142	Giulia	Bianchi

- Unione
- **Differenza**

Tabella: Stud_Ing $\setminus$ Stud_Sci

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
8392	Giulia	Bianchi

Operatori insiemistici

- Solo su tabelle con gli stessi attributi
- Operatori **binari** di unione, differenza, intersezione

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: Stud_Sci

Matricola	**Nome**	**Cognome**
8279	Fabrizia	Bianchi
0989	Stefano	Neri
0142	Giulia	Bianchi

- Unione
- Differenza
- **Intersezione**

Tabella: Stud_Ing $\cap$ Stud_Sci

Matricola	**Nome**	**Cognome**
0142	Giulia	Bianchi

Operatore di ridenominazione

- Operatore unario
- Modifica l'**intestazione** di una tabella

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: $\text{Ridenomina}_{(Matricola \rightarrow ID)}(\text{Stud\_Ing})$

ID	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Operatore di proiezione

- Operatore unario di decomposizione verticale
- Seleziona un **sottoinsieme degli attributi**

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: $\text{Proietta}_{(Nome, Cognome)}(\text{Stud\_Ing})$

Nome	**Cognome**
Mario	Rossi
Giulia	Bianchi

Operatore di selezione

- Operatore unario di decomposizione orizzontale
- Seleziona un **sottoinsieme delle tuple** in base a una **condizione**

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: $\text{Seleziona}_{(Matricola<1000)}(\text{Stud\_Ing})$

Matricola	**Nome**	**Cognome**
0142	Giulia	Bianchi

Operatore di θ-join

- Operatore **binario**
- Serve a **correlare** i dati presenti **in più tabelle**

Tabella: Studenti

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: Esami

Studente	**Corso**	**Voto**
8392	Informatica	28
0989	Informatica	30
0142	Analisi 1	24
8392	Geometria	23

- Esempio:
  - Join tra Studenti ed Esami
  - Condizione: Studenti.Matricola = Esami.Studente

Tabella: Studenti Join$_{(Studenti.Matricola=Esami.Studente)}$(Esami)

Matricola	**Nome**	**Cognome**	**Studente**	**Corso**	**Voto**
0142	Giulia	Bianchi	0142	Analisi 1	24
8392	Giulia	Bianchi	8392	Informatica	28
8392	Giulia	Bianchi	8392	Geometria	23

Esempio di uso degli operatori per eseguire query

- Database per una biblioteca con 3 tabelle
  - Libri(**Titolo**, **Autore**, **ISBN**)
  - Prestiti(**Libro**, **Utente**, **DataPrestito**)
  - Utenti(**Nome**, **Indirizzo**, **CodiceFiscale**)

Q1. *tutti i titoli il cui autore è Pavese*

- **Proietta**$_{(Titolo)}$(
  Seleziona$_{(Autore=\text{“}Pavese\text{“})}$(Libri)
)

Q2. *tutte le date in cui Rossi ha preso in prestito libri*

- **Proietta**$_{(DataPrestito)}$(
  Seleziona$_{(Nome=\text{“}Rossi\text{“})}$(
  Prestiti **Join**$_{(Utenti.CodiceFiscale=Prestiti.Utente)}$ (Utenti)
)
)

Parte V

SQL

Structured Query Language

- Contiene le funzionalità di DDL e DML
- Tipi di dati memorizzati in un db $\Rightarrow$ **domini**

Structured Query Language

- Contiene le funzionalità di DDL e DML
- Tipi di dati memorizzati in un db $\Rightarrow$ **domini**

1. **caratteri**: singoli caratteri e stringhe con lunghezza max
   - `CHARACTER`, es: Sesso (M/F)
   - `CHARACTER(16)`, es: CodiceFiscale
   - `VARCHAR(30)`, es: Cognome, Indirizzo

Structured Query Language

- Contiene le funzionalità di DDL e DML
- Tipi di dati memorizzati in un db $\Rightarrow$ **domini**

1. **caratteri**: `CHARACTER`, `CHARACTER(16)`, `VARCHAR(30)`
2. **flag**: la presenza di una o più proprietà
   - `BIT`, es: Lode (solo valori `'0'` e `'1'`)
   - `BIT(10)` (stringa di bit, es: `'0010011101'`)
   - `BIT VARYING(25)` (stringa di bit)

Structured Query Language

- Contiene le funzionalità di DDL e DML
- Tipi di dati memorizzati in un db $\Rightarrow$ **domini**

1. **caratteri**: `CHARACTER`, `CHARACTER(16)`, `VARCHAR(30)`
2. **flag**: `BIT`, `BIT(10)`, `BIT VARYING(25)`
3. **tipi numerici**
   - `INTEGER`, 32 bit, dominio: [-2G, +2G]
   - `NUMERIC(5,2)`, numeri decimali: xxx,xx
   - `REAL`, `FLOAT`, `DOUBLE PRECISION`: virgola mobile

Structured Query Language

- Contiene le funzionalità di DDL e DML
- Tipi di dati memorizzati in un db $\Rightarrow$ **domini**

1. **caratteri**: `CHARACTER`, `CHARACTER(16)`, `VARCHAR(30)`
2. **flag**: `BIT`, `BIT(10)`, `BIT VARYING(25)`
3. **tipi numerici**: `INTEGER`, `NUMERIC(5,2)`, `REAL`, `FLOAT`, `DOUBLE PRECISION`
4. **informazioni temporali**
   - `DATE`, strutturato in `yyyy-mm-dd`
     - Es: `'1973-04-03'`
   - `TIME`, strutturato in `hh:mm:ss`
     - Es: `'15:05:59'`
   - `TIMESTAMP`: comprende tutti i campi di `DATE` e `TIME`
     - Es: `'1973-04-03 15:05:59'`

Structured Query Language

- Contiene le funzionalità di DDL e DML
- Tipi di dati memorizzati in un db $\Rightarrow$ **domini**

1. **caratteri**: `CHARACTER`, `CHARACTER(16)`, `VARCHAR(30)`
2. **flag**: `BIT`, `BIT(10)`, `BIT VARYING(25)`
3. **tipi numerici**: `INTEGER`, `NUMERIC(5,2)`, `REAL`, `FLOAT`, `DOUBLE PRECISION`
4. **informazioni temporali**: `DATE`, `TIME`, `TIMESTAMP`

- Definizione di **vincoli di integrità**
  1. Vincoli **intra-relazionali di chiave**: `PRIMARY KEY`
  2. Vincoli **intra-relazionali di dominio**: `NOT NULL`
  3. Vincoli **intra-relazionali di dominio/ennupla**: `CHECK`
  4. Vincoli **inter-relazionali**: `FOREIGN KEY(...) REFERENCES...`

Creazione di una tabella: `CREATE TABLE Studenti`

Tabella: Studenti

Matricola	**Nome**	**Cognome**	**DataNascita**
1234	Mario	Rossi	02/02/1990
0142	Giulia	Bianchi	28/10/1991
8392	Giulia	Bianchi	17/10/1943

- Studenti(**Matricola**, **Nome**, **Cognome**, **DataNascita**)
- Domini delle colonne:
  - **Matricola**: testo (4 caratteri). Chiave primaria.
  - **Nome**, **Cognome**: testo (al più 50 caratteri)
  - **DataNascita**: data

Creazione di una tabella: `CREATE TABLE Studenti`

Tabella: Studenti

Matricola	**Nome**	**Cognome**	**DataNascita**
1234	Mario	Rossi	02/02/1990
0142	Giulia	Bianchi	28/10/1991
8392	Giulia	Bianchi	17/10/1943

- Studenti(**Matricola**, **Nome**, **Cognome**, **DataNascita**)

```
CREATE TABLE studenti
(
        matricola CHAR(4) PRIMARY KEY,
        nome VARCHAR(50),
        cognome VARCHAR(50),
        data_nascita DATE
)
```

Creazione di una tabella: `CREATE TABLE Corsi`

Tabella: Corsi

ID	**Denominazione**	**Anno**	**Docente**
145	Analisi Matematica 1	1	Arcozzi
317	Informatica grafica	1	Torroni
161	Sociologia	1	Piccoli
957	Estimo	2	Minghini

- Corsi(**ID**, **Denominazione**, **Anno**, **Docente**)
- Domini delle colonne:
  - **ID**: intero. Chiave primaria.
  - **Denominazione**: testo (al più 50 caratteri)
  - **Anno**: intero
  - **Docente**: testo (al più 20 caratteri)

Creazione di una tabella: `CREATE TABLE Corsi`

Tabella: Corsi

ID	**Denominazione**	**Anno**	**Docente**
145	Analisi Matematica 1	1	Arcozzi
317	Informatica grafica	1	Torroni
161	Sociologia	1	Piccoli
957	Estimo	2	Minghini

- Corsi(**ID**, **Denominazione**, **Anno**, **Docente**)
- Requisiti aggiuntivi:
  - Non possono esistere due corsi con lo stesso titolo
  - Anno di corso $\in [1..5]$
  - Il docente deve essere specificato obbligatoriamente

Creazione di una tabella: `CREATE TABLE Corsi`

Tabella: Corsi

ID	**Denominazione**	**Anno**	**Docente**
145	Analisi Matematica 1	1	Arcozzi
317	Informatica grafica	1	Torroni
161	Sociologia	1	Piccoli
957	Estimo	2	Minghini

- Corsi(**ID**, **Denominazione**, **Anno**, **Docente**)

```
CREATE TABLE corsi
(
        ID INTEGER PRIMARY KEY,
        denominazione CHAR(50) UNIQUE,
        anno INTEGER,
        docente CHAR(20) NOT NULL,
        CHECK (anno>0 AND anno<6)
)
```

Creazione di tabelle con integrità referenziale

Tabella: Studenti

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: Esami

Studente	**Corso**	**DataEsame**	**Voto**
8392	318	09/12/2009	28
0989	317	09/12/2009	30
0142	145	14/12/2009	24
8392	161	12/01/2010	23

- Esistono **correlazioni** tra campi di **tabelle diverse**
  - Esami.Studente $\leftrightarrow$ Studenti.Matricola
  - Esami.Corso $\leftrightarrow$ Corsi.ID
- Possibili problemi (da evitare):
  - In Esami compare un codice di un **corso** sconosciuto
  - In Esami compare un codice di uno **studente** sconosciuto
- Soluzione: **vincoli di integrità referenziale**

Creazione di tabelle con integrità referenziale

Tabella: Studenti

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: Esami

Studente	**Corso**	**DataEsame**	**Voto**
8392	**318**	09/12/2009	28
0989	317	09/12/2009	30
0142	145	14/12/2009	24
8392	161	12/01/2010	23

- Occorre distinguere tra 2 ruoli di una tabella
  - Tabella **interna** (es: Esami)
  - Tabella **esterna** (es: Studenti, Corsi)
- Vincolo di integrità referenziale:
  - per ogni ennupla della **tabella interna**,
  - il valore dell'**attributo specificato**, se non nullo,
  - deve trovarsi in almeno una ennupla della **tabella esterna**
  - tra i valori del **corrispondente attributo**.
- `FOREIGN KEY(...) REFERENCES ...`

Creazione di tabelle con integrità referenziale

Tabella: Studenti

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: Esami

Studente	**Corso**	**DataEsame**	**Voto**
8392	~~318~~	09/12/2009	28
~~0989~~	317	09/12/2009	30
0142	145	14/12/2009	24
8392	161	12/01/2010	23

```
CREATE TABLE esami
(
      studente CHAR(4), corso INTEGER,
      data_esame DATE, voto INTEGER,
      CHECK (voto>17 AND voto<31),
      PRIMARY KEY(studente, corso),
      FOREIGN KEY(studente) REFERENCES
            studenti(matricola),
      FOREIGN KEY(corso) REFERENCES corsi(id)
)
```

Vincoli SQL

`NOT NULL`	Presenza
`UNIQUE`	Unicità
`PRIMARY KEY`	Chiave primaria
`FOREIGN KEY`	Chiave forestiera
`CHECK`	Vincoli di ennupla
`DEFAULT`	Valore predefinito

- Esempio:

```
CREATE TABLE ordini
(
    id INT PRIMARY KEY,
    numero INT NOT NULL UNIQUE,
    data DATE DEFAULT CURRENT_DATE
)
```

Query SQL

- Interrogazioni o query: operazioni per selezionare, combinare, aggiornare, visualizzare dati del db
  - Serve un Data Manipulation Language (SQL)
- SQL è un linguaggio **dichiarativo**
  - Specifica l'**obiettivo** delle interrogazioni, non come raggiungerlo
  - Una volta **specificata** una query in SQL, viene passata a un **ottimizzatore** che la esegue nel modo migliore.
- Conseguenze positive:
  - non bisogna pensare agli aspetti di ottimizzazione
  - esistono molti modi diversi di esprimere la stessa query
  - molti produrranno lo stesso risultato
  - quando si scrive una query basta concentrarsi sulla sua correttezza, leggibilità, modificabilità

Manipolazione dei dati con SQL

- **Inserimento**, **eliminazione** e **aggiornamento** di una ennupla
- Operatori `INSERT`, `DELETE`, `UPDATE`

Tabella: Corsi

ID	**Denominazione**	**Anno**	**Docente**
145	Analisi Matematica 1	1	Arcozzi
317	Informatica grafica	1	Torroni
161	Sociologia	1	Piccoli
957	Estimo	2	Minghini

Manipolazione dei dati con SQL

- **Inserimento**, **eliminazione** e **aggiornamento** di una ennupla
- Operatori `INSERT`, `DELETE`, `UPDATE`

Tabella: Corsi

ID	**Denominazione**	**Anno**	**Docente**
145	Analisi Matematica 1	1	Arcozzi
317	Informatica grafica	1	Torroni
161	Sociologia	1	Piccoli
957	Estimo	2	Minghini

1. **Inserimento** di una ennupla

```
INSERT INTO corsi
VALUES (237,'Meccanica Razionale',1,'Muracchini')
```

Manipolazione dei dati con SQL

- **Inserimento**, **eliminazione** e **aggiornamento** di una ennupla
- Operatori `INSERT`, `DELETE`, `UPDATE`

Tabella: Corsi

ID	**Denominazione**	**Anno**	**Docente**
145	Analisi Matematica 1	1	Arcozzi
317	Informatica grafica	1	Torroni
161	Sociologia	1	Piccoli
957	Estimo	2	Minghini
237	Meccanica Razionale	1	Muracchini

1. **Inserimento** di una ennupla

```
INSERT INTO corsi
VALUES (237,'Meccanica Razionale',1,'Muracchini')
```

Manipolazione dei dati con SQL

- **Inserimento**, **eliminazione** e **aggiornamento** di una ennupla
- Operatori `INSERT`, `DELETE`, `UPDATE`

Tabella: Corsi

ID	**Denominazione**	**Anno**	**Docente**
145	Analisi Matematica 1	1	Arcozzi
317	Informatica grafica	1	Torroni
161	Sociologia	1	Piccoli
957	Estimo	2	Minghini
237	Meccanica Razionale	1	Muracchini

1. **Inserimento** di una ennupla: `insert`
2. **Eliminazione** di una ennupla

```
DELETE FROM corsi
WHERE id=317
```

Manipolazione dei dati con SQL

- **Inserimento**, **eliminazione** e **aggiornamento** di una ennupla
- Operatori `INSERT`, `DELETE`, `UPDATE`

Tabella: Corsi

ID	**Denominazione**	**Anno**	**Docente**
145	Analisi Matematica 1	1	Arcozzi
161	Sociologia	1	Piccoli
957	Estimo	2	Minghini
237	Meccanica Razionale	1	Muracchini

1. **Inserimento** di una ennupla: `insert`
2. **Eliminazione** di una ennupla

```
DELETE FROM corsi
WHERE id=317
```

Manipolazione dei dati con SQL

- **Inserimento**, **eliminazione** e **aggiornamento** di una ennupla
- Operatori `INSERT`, `DELETE`, `UPDATE`

Tabella: Corsi

ID	**Denominazione**	**Anno**	**Docente**
145	Analisi Matematica 1	1	Arcozzi
161	Sociologia	1	Piccoli
957	Estimo	2	Minghini
237	Meccanica Razionale	1	Muracchini

1. **Inserimento** di una ennupla: `insert`
2. **Eliminazione** di una ennupla: `delete`
3. **Modifica** di una ennupla

```
UPDATE corsi
SET docente='Virgilio'
WHERE id=957
```

Manipolazione dei dati con SQL

- **Inserimento**, **eliminazione** e **aggiornamento** di una ennupla
- Operatori `INSERT`, `DELETE`, `UPDATE`

Tabella: Corsi

ID	**Denominazione**	**Anno**	**Docente**
145	Analisi Matematica 1	1	Arcozzi
161	Sociologia	1	Piccoli
957	Estimo	2	Virgilio
237	Meccanica Razionale	1	Muracchini

1. **Inserimento** di una ennupla: `insert`
2. **Eliminazione** di una ennupla: `delete`
3. **Modifica** di una ennupla

```
UPDATE corsi
SET docente='Virgilio'
WHERE id=957
```

Query SQL: il costrutto SELECT

- Molti operatori relazionali:
  - **operatori insiemistici**: $\cup, \setminus, \cap$
  - **ridenominazione**
  - **proiezione**
  - **selezione**
  - **join**
- Un unico, semplice costrutto:

  `SELECT ... FROM ... WHERE ...`

- `SELECT column_name(s)`
  `FROM table_name`

Query SQL: il costrutto SELECT

- Molti operatori relazionali:
  - **operatori insiemistici**: $\cup, \setminus, \cap$
  - **ridenominazione**
  - **proiezione**
  - **selezione**
  - **join**
- Un unico, semplice costrutto:

  `SELECT ... FROM ... WHERE ...`

- ```
  SELECT column_name(s)
  FROM table_name
  UNION
  SELECT column_name(s)
  FROM table_name2
  ```
  ```

Query SQL: il costrutto SELECT

- Molti operatori relazionali:
  - **operatori insiemistici**: $\cup, \setminus, \cap$
  - **ridenominazione**
  - **proiezione**
  - **selezione**
  - **join**
- Un unico, semplice costrutto:

  ```
  SELECT ... FROM ... WHERE ...
  ```

- ```
  SELECT column_name(s)
  INTO a_new_table
  FROM table_name
  ```
  ```

Query SQL: il costrutto SELECT

- Molti operatori relazionali:
  - **operatori insiemistici**: $\cup, \setminus, \cap$
  - **ridenominazione**
  - **proiezione**
  - **selezione**
  - **join**
- Un unico, semplice costrutto:

  ```
  SELECT ... FROM ... WHERE ...
  ```

- ```
  SELECT column_name(s)
  INTO a_new_table
  FROM table_name
  WHERE column_name operator value
  ```
  ```

Query SQL: il costrutto SELECT

- Molti operatori relazionali:
  - **operatori insiemistici**: $\cup, \setminus, \cap$
  - **ridenominazione**
  - **proiezione**
  - **selezione**
  - **join**
- Un unico, semplice costrutto:

  `SELECT ... FROM ... WHERE ...`

- ```
  SELECT *
  FROM table_name
  WHERE column_name operator value
  ```
  ```

Query SQL: il costrutto SELECT

- Molti operatori relazionali:
  - **operatori insiemistici**: $\cup, \setminus, \cap$
  - **ridenominazione**
  - **proiezione**
  - **selezione**
  - **join**
- Un unico, semplice costrutto:

  ```
  SELECT ... FROM ... WHERE ...
  ```

- ```
  SELECT *
  FROM table_name
  WHERE column_name operator value
  AND column_name operator value
  ```
  ```

Query SQL: il costrutto SELECT

- Molti operatori relazionali:
  - **operatori insiemistici**: $\cup, \setminus, \cap$
  - **ridenominazione**
  - **proiezione**
  - **selezione**
  - **join**
- Un unico, semplice costrutto:

  ```
  SELECT ... FROM ... WHERE ...
  ```

- ```
  SELECT *
  FROM table_name
  WHERE column_name operator value
  OR column_name operator value
  ```
  ```

Query SQL: il costrutto SELECT

- Molti operatori relazionali:
  - **operatori insiemistici**: $\cup, \setminus, \cap$
  - **ridenominazione**
  - **proiezione**
  - **selezione**
  - **join**
- Un unico, semplice costrutto:

  ```
  SELECT ... FROM ... WHERE ...
  ```

- ```
  SELECT *
  FROM table_name
  WHERE column_name operator value
  OR column_name operator value
  ORDER BY column_name(s) ASC | DESC
  ```
  ```

Query SQL: il costrutto SELECT

- Molti operatori relazionali:
  - **operatori insiemistici**: $\cup, \setminus, \cap$
  - **ridenominazione**
  - **proiezione**
  - **selezione**
  - **join**
- Un unico, semplice costrutto:

  ```
  SELECT ... FROM ... WHERE ...
  ```

- ```
  SELECT *
  FROM table_name
  WHERE column_name IN (value1,value2,...)
  ```
  ```

Query SQL: il costrutto SELECT

- Molti operatori relazionali:
  - **operatori insiemistici**: $\cup, \setminus, \cap$
  - **ridenominazione**
  - **proiezione**
  - **selezione**
  - **join**
- Un unico, semplice costrutto:

  `SELECT ... FROM ... WHERE ...`

- `SELECT *`
  `FROM table_name`
  `WHERE column_name BETWEEN value1 AND value2`
  (solo per valori numerici)

Query SQL: il costrutto SELECT

- Molti operatori relazionali:
  - **operatori insiemistici**: $\cup, \setminus, \cap$
  - **ridenominazione**
  - **proiezione**
  - **selezione**
  - **join**
- Un unico, semplice costrutto:

  ```
  SELECT ... FROM ... WHERE ...
  ```

- ```
  SELECT *
  FROM table_name
  WHERE column_name IS NULL
  ```
  ```

Query SQL: il costrutto SELECT

- Molti operatori relazionali:
  - **operatori insiemistici**: $\cup, \setminus, \cap$
  - **ridenominazione**
  - **proiezione**
  - **selezione**
  - **join**
- Un unico, semplice costrutto:

  `SELECT ... FROM ... WHERE ...`

- Possibile utilizzo di **funzioni esterne**

  `SELECT NOW(),CURDATE(),CURTIME()`

Query SQL: il costrutto SELECT

- Molti operatori relazionali:
  - **operatori insiemistici**: $\cup, \setminus, \cap$
  - **ridenominazione**
  - **proiezione**
  - **selezione**
  - **join**
- Un unico, semplice costrutto:

```
SELECT ... FROM ... WHERE ...
```

- Possibile utilizzo di **funzioni esterne**
- Funzioni **aggregate**

```
SELECT COUNT(*)
FROM table_name
WHERE ...
```

Query SQL: il costrutto SELECT

- Molti operatori relazionali:
  - **operatori insiemistici**: $\cup, \setminus, \cap$
  - **ridenominazione**
  - **proiezione**
  - **selezione**
  - **join**
- Un unico, semplice costrutto:

```
SELECT ... FROM ... WHERE ...
```

- Possibile utilizzo di **funzioni esterne**
- Funzioni **aggregate**

```
SELECT SUM(column_name)
FROM table_name
WHERE ...
```

Query SQL: il costrutto SELECT

- Molti operatori relazionali:
  - **operatori insiemistici**: $\cup, \setminus, \cap$
  - **ridenominazione**
  - **proiezione**
  - **selezione**
  - **join**
- Un unico, semplice costrutto:

```
SELECT ... FROM ... WHERE ...
```

- Possibile utilizzo di **funzioni esterne**
- Funzioni **aggregate**

```
SELECT MIN(column_name)
FROM table_name
WHERE ...
```

Query SQL: il costrutto SELECT

- Molti operatori relazionali:
  - **operatori insiemistici**: $\cup, \setminus, \cap$
  - **ridenominazione**
  - **proiezione**
  - **selezione**
  - **join**
- Un unico, semplice costrutto:

```
SELECT ... FROM ... WHERE ...
```

- Possibile utilizzo di **funzioni esterne**
- Funzioni **aggregate**

```
SELECT MAX(column_name)
FROM table_name
WHERE ...
```

Query SQL: il costrutto SELECT

- Molti operatori relazionali:
  - **operatori insiemistici**: $\cup, \setminus, \cap$
  - **ridenominazione**
  - **proiezione**
  - **selezione**
  - **join**
- Un unico, semplice costrutto:

```
SELECT ... FROM ... WHERE ...
```

- Possibile utilizzo di **funzioni esterne**
- Funzioni **aggregate**

```
SELECT AVG(column_name)
FROM table_name
WHERE ...
```

Operatore di unione

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: Stud_Sci

Matricola	**Nome**	**Cognome**
8279	Fabrizia	Bianchi
0989	Stefano	Neri
0142	Giulia	Bianchi

- **Unione** insiemistica

```
SELECT column_name(s) FROM table_name1
UNION
SELECT column_name(s) FROM table_name2
```

Tabella: Stud_Ing $\cup$ Stud_Sci

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
8392	Giulia	Bianchi
0142	Giulia	Bianchi
8279	Fabrizia	Bianchi
0989	Stefano	Neri

Operatore di unione

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: Stud_Sci

Matricola	**Nome**	**Cognome**
8279	Fabrizia	Bianchi
0989	Stefano	Neri
0142	Giulia	Bianchi

- **Unione** insiemistica

```
SELECT * FROM Stud_Ing
UNION
SELECT * FROM Stud_Sci
```

Tabella: Stud_Ing $\cup$ Stud_Sci

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
8392	Giulia	Bianchi
0142	Giulia	Bianchi
8279	Fabrizia	Bianchi
0989	Stefano	Neri

Operatore di ridenominazione

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: Ridenomina$_{(Matricola \rightarrow ID)}$(Stud_Ing)

ID	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

- `SELECT Matricola AS ID, Nome, Cognome FROM Stud_Ing`

Operatore di ridenominazione

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: Studenti

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

- ```
  SELECT *
  FROM Stud_Ing
  AS Studenti
  ```
  ```

Operatore di proiezione

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

- `SELECT column_name(s)`
  `FROM table_name`

Tabella: Proietta$_{(Nome, Cognome)}$(Stud_Ing)

Nome	**Cognome**
Mario	Rossi
Giulia	Bianchi

Operatore di proiezione

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

- `SELECT Nome,Cognome`
  `FROM Stud_Ing`

Tabella: $\text{Proietta}_{(Nome, Cognome)}(\text{Stud\_Ing})$

Nome	**Cognome**
Mario	Rossi
Giulia	Bianchi

Operatore di selezione

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

- ```
  SELECT *
  FROM table_name
  WHERE column_name operator value
  ```

Tabella: Seleziona$_{(Matricola<1000)}$(Stud_Ing)

Matricola	**Nome**	**Cognome**
0142	Giulia	Bianchi
  ```

Operatore di selezione

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

- ```
  SELECT *
  FROM Stud_Ing
  WHERE Matricola<1000
  ```

Tabella: $\text{Seleziona}_{(Matricola<1000)}(\text{Stud\_Ing})$

Matricola	**Nome**	**Cognome**
0142	Giulia	Bianchi
  ```

Operatore di selezione

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

- ```
  SELECT *
  FROM Stud_Ing
  WHERE Cognome='Bianchi'
  ```

Tabella: $\text{Seleziona}_{(Cognome='Bianchi')}(\text{Stud\_Ing})$

Matricola	**Nome**	**Cognome**
0142	Giulia	Bianchi
8392	Giulia	Bianchi
  ```

Operatore di selezione

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

- ```
  SELECT *
  FROM Stud_Ing
  WHERE Cognome IN ('Bianchi','Rossi')
  ```

Tabella: $\text{Seleziona}_{(Cognome \in \{'Bianchi','Rossi'\})}(\text{Stud\_Ing})$

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi
  ```

Operatore di selezione

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

- ```
  SELECT *
  FROM Stud_Ing
  WHERE Cognome LIKE '%i'
  ```

Tabella: $\text{Seleziona}_{(Cognome \in \{\ldots i\})}(\text{Stud\_Ing})$

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi
  ```

Operatore di selezione

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

- ```
  SELECT *
  FROM Stud_Ing
  WHERE Cognome LIKE '%i' AND Matricola<2000
  ```

Tabella: Seleziona$_{(Cognome \in \{...i\} \wedge Matricola < 1000)}$(Stud_Ing)

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
  ```

Operatori di selezione

=	Uguaglianza
<>	Disuguaglianza
>	Maggioranza
<	Minoranza
>=	Maggiore o uguale
<=	Minore o uguale
`BETWEEN`	Intervallo di valori (*range*)
`LIKE`	*Pattern* (*espressioni regolari*[1])
`IN`	Insieme di valori
`AND`	Congiunzione logica
`OR`	Disgiunzione logica
`IS NULL`	Valore non definito

(1) Uso di *wildcards*: %, _, `[charlist]`, `[!charlist]`

```
SELECT * FROM Stud_Ing
WHERE Cognome LIKE '[bsp]%'
```

Operatori di selezione

=	Uguaglianza
<>	Disuguaglianza
>	Maggioranza
<	Minoranza
>=	Maggiore o uguale
<=	Minore o uguale
`BETWEEN`	Intervallo di valori (*range*)
`LIKE`	*Pattern* (*espressioni regolari*[1])
`IN`	Insieme di valori
`AND`	Congiunzione logica
`OR`	Disgiunzione logica
`IS NULL`	Valore non definito

(1) Uso di *wildcards*: %, _, `[charlist]`, `[!charlist]`

```
SELECT * FROM Stud_Ing
WHERE Cognome LIKE '[!bsp]%'
```

Operatore di θ-join

Tabella: Studenti

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: Esami

Studente	**Corso**	**Voto**
8392	Informatica	28
0989	Informatica	30
0142	Analisi 1	24
8392	Geometria	23

- Esempio:
  - Join tra Studenti ed Esami
  - Condizione: Studenti.Matricola = Esami.Studente

Tabella: Studenti Join$_{(Studenti.Matricola=Esami.Studente)}$(Esami)

Matricola	**Nome**	**Cognome**	**Studente**	**Corso**	**Voto**
0142	Giulia	Bianchi	0142	Analisi 1	24
8392	Giulia	Bianchi	8392	Informatica	28
8392	Giulia	Bianchi	8392	Geometria	23

Operatore di θ-join

Tabella: Studenti

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: Esami

Studente	**Corso**	**Voto**
8392	Informatica	28
0989	Informatica	30
0142	Analisi 1	24
8392	Geometria	23

- ```
  SELECT column_name(s)
  FROM table_name1
  INNER JOIN table_name2
  ON table_name1.column_name=table_name2.column_name
  ```

Tabella: Studenti Join$_{(Studenti.Matricola=Esami.Studente)}$(Esami)

Matricola	**Nome**	**Cognome**	**Studente**	**Corso**	**Voto**
0142	Giulia	Bianchi	0142	Analisi 1	24
8392	Giulia	Bianchi	8392	Informatica	28
8392	Giulia	Bianchi	8392	Geometria	23
  ```

Operatore di θ-join

Tabella: Studenti

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: Esami

Studente	**Corso**	**Voto**
8392	Informatica	28
0989	Informatica	30
0142	Analisi 1	24
8392	Geometria	23

- ```
  SELECT *
  FROM Studenti
  INNER JOIN Esami
  ON Studenti.Matricola=Esami.Studente
  ```

Tabella: Studenti Join$_{(Studenti.Matricola=Esami.Studente)}$(Esami)

Matricola	**Nome**	**Cognome**	**Studente**	**Corso**	**Voto**
0142	Giulia	Bianchi	0142	Analisi 1	24
8392	Giulia	Bianchi	8392	Informatica	28
8392	Giulia	Bianchi	8392	Geometria	23
  ```

Tipi di join

`INNER JOIN`	θ-join
`LEFT JOIN`	Include tutte le righe della prima tabella (anche senza corrispettivi nella seconda)
`RIGHT JOIN`	Include tutte le righe della seconda tabella (anche senza corrispettivi nella prima)
`FULL JOIN`	Include tutte le righe di tutte le tabelle

LEFT JOIN

Tabella: Studenti

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: Esami

Studente	**Corso**	**Voto**
8392	Informatica	28
0989	Informatica	30
0142	Analisi 1	24
8392	Geometria	23

- ```
  SELECT Matricola, Nome, Cognome, Corso, Voto
  FROM Studenti
  LEFT JOIN Esami
  ON Studenti.Matricola=Esami.Studente
  ```

Tabella: Studenti L-Join$_{(Studenti.Matricola=Esami.Studente)}$(Esami)

Matricola	**Nome**	**Cognome**	**Corso**	**Voto**
1234	Mario	Rossi		
0142	Giulia	Bianchi	Analisi 1	24
8392	Giulia	Bianchi	Informatica	28
8392	Giulia	Bianchi	Geometria	23
  ```

RIGHT JOIN

Tabella: Studenti

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: Esami

Studente	**Corso**	**Voto**
8392	Informatica	28
0989	Informatica	30
0142	Analisi 1	24
8392	Geometria	23

- ```
  SELECT Matricola, Nome, Cognome, Corso, Voto
  FROM Studenti
  RIGHT JOIN Esami
  ON Studenti.Matricola=Esami.Studente
  ```

Tabella: Studenti R-Join$_{(Studenti.Matricola=Esami.Studente)}$(Esami)

Matricola	**Nome**	**Cognome**	**Corso**	**Voto**
8392	Giulia	Bianchi	Informatica	28
			Informatica	30
0142	Giulia	Bianchi	Analisi 1	24
8392	Giulia	Bianchi	Geometria	23
  ```

FULL JOIN

Tabella: Studenti

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

Tabella: Esami

Studente	**Corso**	**Voto**
8392	Informatica	28
0989	Informatica	30
0142	Analisi 1	24
8392	Geometria	23

- ```
  SELECT Matricola, Nome, Cognome, Corso, Voto
  FROM Studenti
  FULL JOIN Esami
  ON Studenti.Matricola=Esami.Studente
  ```

Tabella: Studenti F-Join$_{(Studenti.Matricola=Esami.Studente)}$(Esami)

Matricola	**Nome**	**Cognome**	**Corso**	**Voto**
1234	Mario	Rossi		
0142	Giulia	Bianchi	Analisi 1	24
8392	Giulia	Bianchi	Informatica	28
8392	Giulia	Bianchi	Geometria	23
			Informatica	30
  ```

Funzioni aggregate

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

- `SELECT COUNT(*) AS TotalStudents FROM Stud_Ing`

Tabella: Uso di `COUNT(*)`

TotalStudents
3

Funzioni aggregate

Tabella: Stud_Ing

Matricola	**Nome**	**Cognome**
1234	Mario	Rossi
0142	Giulia	Bianchi
8392	Giulia	Bianchi

- ```
  SELECT Nome, Cognome, Matricola
  FROM Stud_Ing
  WHERE Matricola=(
      SELECT MIN(Matricola)
      FROM Stud_Ing
  )
  ```

Tabella: Uso di `MIN(*)`

Nome	**Cognome**	**Matricola**
Giulia	Bianchi	0142
  ```

Funzioni aggregate

`AVG()`	Valore medio
`COUNT()`	Numero di righe
`FIRST()`	Primo valore
`LAST()`	Ultimo valore
`MAX()`	Valore massimo
`MIN()`	Valore minimo
`SUM()`	Somma dei valori

- Spesso usati assieme a `GROUP BY` e `HAVING`

Funzioni aggregate

`AVG()`	Valore medio
`COUNT()`	Numero di righe
`FIRST()`	Primo valore
`LAST()`	Ultimo valore
`MAX()`	Valore massimo
`MIN()`	Valore minimo
`SUM()`	Somma dei valori

- Spesso usati assieme a `GROUP BY` e `HAVING`

```
SELECT customer, SUM(order_price) FROM orders
GROUP BY customer
```

Funzioni aggregate

`AVG()`	Valore medio
`COUNT()`	Numero di righe
`FIRST()`	Primo valore
`LAST()`	Ultimo valore
`MAX()`	Valore massimo
`MIN()`	Valore minimo
`SUM()`	Somma dei valori

- Spesso usati assieme a `GROUP BY` e `HAVING`

```
SELECT customer, SUM(order_price) FROM orders
GROUP BY customer
HAVING SUM(order_price)<2000
```

Esempi di uso di funzioni aggregate

Tabella: **bbc** — BBC Country Profiles (`SQLzoo.net`)

name	**region**	**area**	**population**	**gdp**
Afghanistan	South Asia	652225	26000000	
Albania	Europe	28728	3200000	6656000000
...	...	...	...	...

1a. Mostra la popolazione mondiale
1b. Mostra tutte le regioni
1c. Mostra il PIL (*gdp*) totale dell'*Africa*
1d. Quanti paesi hanno un'area compresa tra 100K e 1M km^2?
1e. Qual è la popolazione totale di *France*, *Germany* e *Spain*?
1f. Di quanti paesi non conosciamo il PIL?
2a. Quanti paesi ha ciascuna regione?
2b. Per ogni regione, mostra la regione e il numero di paesi con almeno 10 ML di abitanti
2c. Elenca le regioni con popolazione di almeno 100 ML

Esempi di uso di funzioni aggregate

Tabella: **bbc** — BBC Country Profiles (`SQLzoo.net`)

name	**region**	**area**	**population**	**gdp**
Afghanistan	South Asia	652225	26000000	
Albania	Europe	28728	3200000	6656000000
...	...	...	...	...

1a. Mostra la popolazione mondiale

```
SELECT SUM(population)
FROM bbc
```

Esempi di uso di funzioni aggregate

Tabella: **bbc** — BBC Country Profiles (`SQLzoo.net`)

name	**region**	**area**	**population**	**gdp**
Afghanistan	South Asia	652225	26000000	
Albania	Europe	28728	3200000	6656000000
...	...	...	...	...

1b. Mostra tutte le regioni

```
SELECT DISTINCT region
FROM bbc
```

Esempi di uso di funzioni aggregate

Tabella: **bbc** — BBC Country Profiles (`SQLzoo.net`)

name	region	area	population	gdp
Afghanistan	South Asia	652225	26000000	
Albania	Europe	28728	3200000	6656000000
...	...	...	...	...

1c. Mostra il PIL (*gdp*) totale dell'*Africa*

```
SELECT SUM(gdp)
FROM bbc
WHERE region='Africa'
```

Esempi di uso di funzioni aggregate

Tabella: **bbc** — BBC Country Profiles (`SQLzoo.net`)

name	**region**	**area**	**population**	**gdp**
Afghanistan	South Asia	652225	26000000	
Albania	Europe	28728	3200000	6656000000
...	...	...	...	...

1d. Quanti paesi hanno un'area compresa tra 100.000 e 1 ML di km^2?

```
SELECT COUNT(*)
FROM bbc
WHERE area BETWEEN 1E5 AND 1E6
```

Esempi di uso di funzioni aggregate

Tabella: **bbc** — BBC Country Profiles (`SQLzoo.net`)

name	region	area	population	gdp
Afghanistan	South Asia	652225	26000000	
Albania	Europe	28728	3200000	6656000000
...	...	...	...	...

1e. Qual è la popolazione totale di *France*, *Germany* e *Spain*?

```
SELECT SUM(population)
FROM bbc
WHERE name IN ('France','Germany','Spain')
```

Esempi di uso di funzioni aggregate

Tabella: **bbc** — BBC Country Profiles (`SQLzoo.net`)

name	region	area	population	gdp
Afghanistan	South Asia	652225	26000000	
Albania	Europe	28728	3200000	6656000000
...	...	...	...	...

1f. Di quanti paesi non conosciamo il PIL?

```
SELECT COUNT(*)
FROM bbc
WHERE gdp IS NULL
```

Esempi di uso di funzioni aggregate

Tabella: **bbc** — BBC Country Profiles (`SQLzoo.net`)

name	region	area	population	gdp
Afghanistan	South Asia	652225	26000000	
Albania	Europe	28728	3200000	6656000000
...	...	...	...	...

2a. Quanti paesi ha ciascuna regione?

```
SELECT region, COUNT(*)
FROM bbc
GROUP BY region
```

Esempi di uso di funzioni aggregate

Tabella: **bbc** — BBC Country Profiles (`SQLzoo.net`)

name	region	area	population	gdp
Afghanistan	South Asia	652225	26000000	
Albania	Europe	28728	3200000	6656000000
...	...	...	...	...

2b. Per ogni regione, mostra la regione e il numero di paesi con almeno 10 ML di abitanti

```
SELECT region, COUNT(*)
FROM bbc
WHERE population>=1E7
GROUP BY region
```

Esempi di uso di funzioni aggregate

Tabella: **bbc** — BBC Country Profiles (`SQLzoo.net`)

name	region	area	population	gdp
Afghanistan	South Asia	652225	26000000	
Albania	Europe	28728	3200000	6656000000
...	...	...	...	...

2c. Elenca le regioni con popolazione di almeno 100 ML

```
SELECT region
FROM bbc
GROUP BY region
HAVING SUM(population)>=1E8
```

Esercitazione di laboratorio

- Mercoledì 14 dicembre, ore 15:00, Lab4
  - Capitolo 6.7 e Modulo E

Parte VI

Progettazione dei sistemi informativi

Ciclo di vita dei sistemi informativi

- Insieme di attività: da **specifica informale** delle caratteristiche desiderate a **realizzazione** del S.I.
  1. **Studio di fattibilità**. Definire **costi** delle alternative e **priorità** delle componenti.
  2. **Raccolta e analisi dei requisiti**. Individuazione di **proprietà** e **funzionalità** del futuro sistema.
     - Fonti: **utenti**, **documentazione** esistente, **realizzazioni preesistenti**.
  3. **Progettazione**. Individuazione di struttura e organizzazione dei **dati**, e caratteristiche degli **applicativi** che li useranno.
     - Produce uno **schema relazionale** della base di dati.
  4. **Implementazione**. Realizzazione del S.I., seguendo uno **schema fisico** dei dati.
  5. **Validazione e collaudo**. Verifica di **funzionamento** e **qualità** del S.I.
  6. **Funzionamento**. Il S.I. è operativo: **gestione**, **manutenzione**, **formazione**.

Modello relazionale dei dati

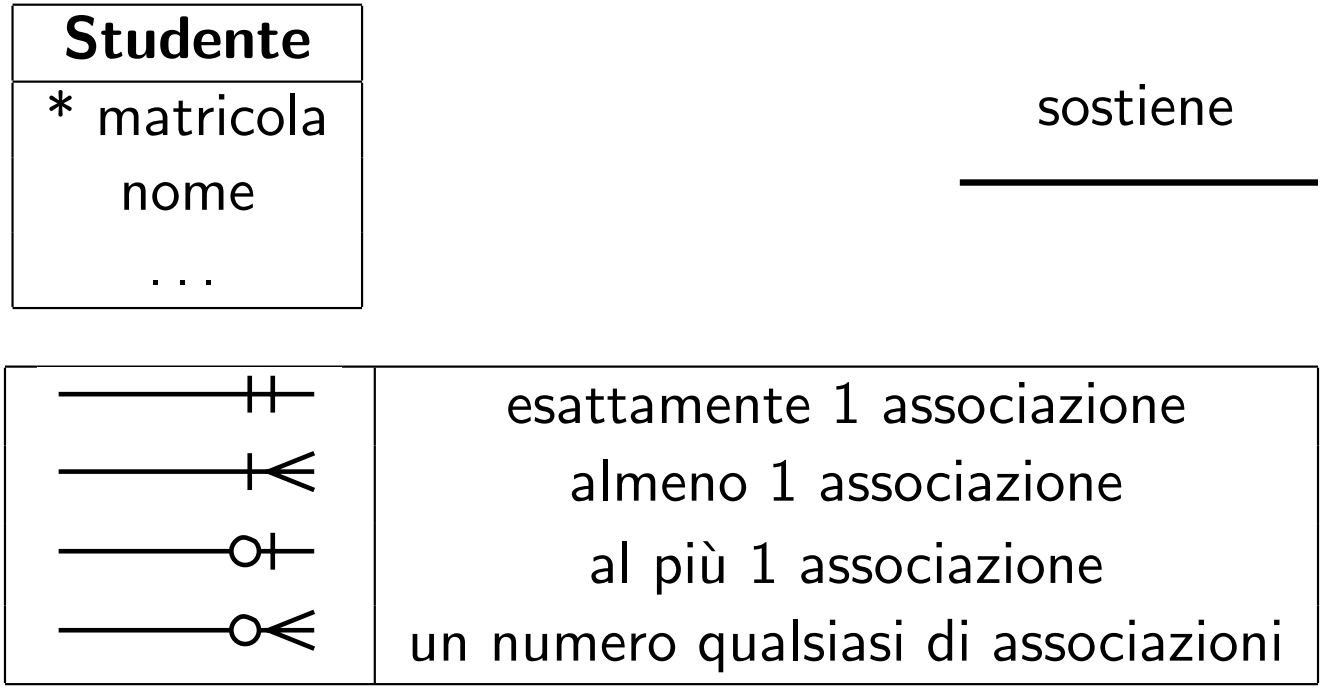
- Modello **Entity-Relationship**.
- **Entità**: una persona, luogo o cosa di cui si vogliono raccogliere e conservare istanze multiple di dati.
  - Ha un **nome** (un sostantivo) e degli **attributi** che descrivono i dati di interesse.
  - Ha un **identificatore** per individuare le istanze in modo univoco.
  - L'attributo (o gli attributi) che fanno da identificatore sono segnati da asterischi.

Modello relazionale dei dati

- Modello **Entity-Relationship**.
- **Entità**: una persona, luogo o cosa di cui si vogliono raccogliere e conservare istanze multiple di dati.
- **Relazione**: una associazione tra due entità.
  - Ha un **nome** (un verbo).

Modello relazionale dei dati

- Modello **Entity-Relationship**.
- **Entità**: una persona, luogo o cosa di cui si vogliono raccogliere e conservare istanze multiple di dati.
- **Relazione**: una associazione tra due entità.
  - Ha un **nome** (un verbo).
  - Ha **cardinalità** (min) e **modalità** (max).



Modello relazionale dei dati

- Anche possibile definire **tassonomie**
  - Relazioni "IS-A"
    - Ad esempio: Entità Persona, Studente, Docente.
    - Studente IS-A Persona,
    - Docente IS-A Persona.
- Infine, anche le relazioni possono avere attributi
  - Esempio: Studente supera Esame (Voto, Data)

Esercizio: DB di paesi e regioni geografiche

- DB della BBC ("country profile") disponibile su `http://sql.org/` (`http://www.sqlzoo.net/`)

1. **Fattibilità**: OK
2. **Analisi dei requisiti**:
3. **Progetto**:
4. **Implementazione**:
5. **Validazione e collaudo**:
6. **Funzionamento**:

Esercizio: DB di paesi e regioni geografiche

- DB della BBC ("country profile") disponibile su `http://sql.org/` (`http://www.sqlzoo.net/`)

1. **Fattibilità**:
2. **Analisi dei requisiti**:
   - capacità di registrare dati presi da SQLzoo per eseguire semplici esperimenti
   - capacità di rispondere alle seguenti query:
     - Query 1: totale PIL, abitanti, numero di paesi, e PIL/abitante di ciascuna regione geografica
     - Query 2: nome del paese con il PIL minimo
     - Query 3: nome dei paesi europei con una popolazione superiore alla media
   - dati a disposizione: paesi, regioni, area, popolazione, PIL
   - entità dei dati: poche centinaia di record
3. **Progetto**:
4. **Implementazione**:
5. **Validazione e collaudo**:
6. **Funzionamento**:

Esercizio: DB di paesi e regioni geografiche

- DB della BBC ("country profile") disponibile su `http://sql.org/` (`http://www.sqlzoo.net/`)

1. **Fattibilità**:
2. **Analisi dei requisiti**:
3. **Progetto**:
   - organizzazione e struttura dei dati:
   - caratteristiche degli applicativi:
4. **Implementazione**:
5. **Validazione e collaudo**:
6. **Funzionamento**:

Esercizio: DB di paesi e regioni geografiche

- DB della BBC ("country profile") disponibile su `http://sql.org/` (`http://www.sqlzoo.net/`)

1. **Fattibilità**:
2. **Analisi dei requisiti**:
3. **Progetto**:
   - organizzazione e struttura dei dati:
     - **Entità**: paesi, regioni
     - **Relazioni**: paese appartiene a regione. Paese ha 1 regione. Regione ha almeno 1 paese.
     - **Proprietà**: area, popolazione, PIL $\longrightarrow$ paese
   - caratteristiche degli applicativi:
4. **Implementazione**:
5. **Validazione e collaudo**:
6. **Funzionamento**:

Esercizio: DB di paesi e regioni geografiche

- DB della BBC ("country profile") disponibile su `http://sql.org/` (`http://www.sqlzoo.net/`)

1. **Fattibilità**:
2. **Analisi dei requisiti**:
3. **Progetto**:
   - organizzazione e struttura dei dati:
     - **Entità**: paesi, regioni
     - **Relazioni**: paese appartiene a regione. Paese ha 1 regione. Regione ha almeno 1 paese.
     - **Proprietà**: area, popolazione, PIL $\longrightarrow$ paese
   - caratteristiche degli applicativi:
     - DB contiene i dati, presi da SQLzoo
     - possibilità di inserire query
4. **Implementazione**:
5. **Validazione e collaudo**:
6. **Funzionamento**:

Esercizio: DB di paesi e regioni geografiche

- DB della BBC ("country profile") disponibile su `http://sql.org/` (`http://www.sqlzoo.net/`)

1. **Fattibilità**:
2. **Analisi dei requisiti**:
3. **Progetto**:
   - organizzazione e struttura dei dati:
     - **Entità**: paesi, regioni
     - **Relazioni**: paese appartiene a regione. Paese ha 1 regione. Regione ha almeno 1 paese.
     - **Proprietà**: area, popolazione, PIL $\rightarrow$ paese
   - caratteristiche degli applicativi:
     - DB contiene i dati, presi da SQLzoo
     - possibilità di inserire query
   - **Schema relazionale**
   - **Schema fisico**
4. **Implementazione**:
5. **Validazione e collaudo**:
6. **Funzionamento**:

Esercizio: DB di paesi e regioni geografiche

- DB della BBC ("country profile") disponibile su `http://sql.org/` (`http://www.sqlzoo.net/`)

1. **Fattibilità**:
2. **Analisi dei requisiti**:
3. **Progetto**:
4. **Implementazione**:
   - DBMS: PostgreSQL 8.4
   - piattaforma: Mac OS X 10.5
   - implementazione delle tabelle segue lo schema fisico; uso di query SQL di tipo `CREATE TABLE`
   - caricamento dati tramite copia da sito Web, foglio elettronico e query SQL di tipo `INSERT`
5. **Validazione e collaudo**:
6. **Funzionamento**:

Esercizio: DB di paesi e regioni geografiche

- DB della BBC ("country profile") disponibile su `http://sql.org/` (`http://www.sqlzoo.net/`)

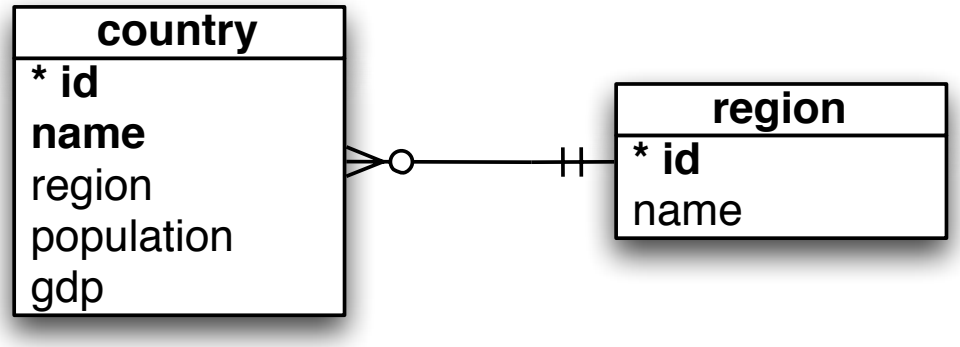
1. **Fattibilità**:
2. **Analisi dei requisiti**:
3. **Progetto**:
4. **Implementazione**:
5. **Validazione e collaudo**:
   - tramite query SQL di tipo `SELECT`
6. **Funzionamento**:

Esercizio: DB di paesi e regioni geografiche

- DB della BBC ("country profile") disponibile su `http://sql.org/` (`http://www.sqlzoo.net/`)

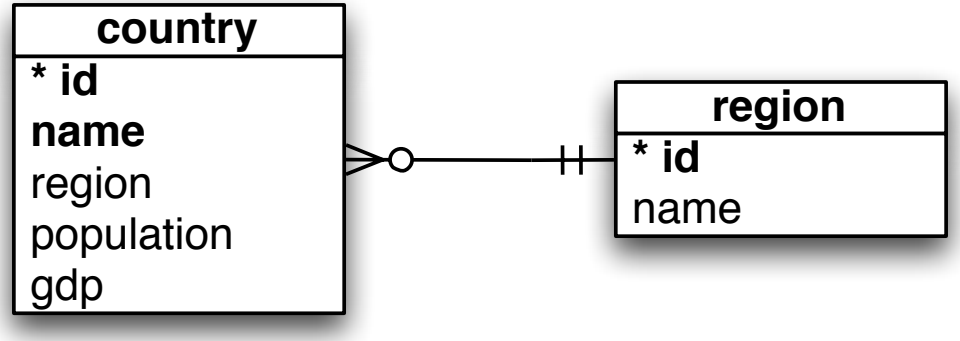
1. **Fattibilità**:
2. **Analisi dei requisiti**:
3. **Progetto**:
4. **Implementazione**:
5. **Validazione e collaudo**:
6. **Funzionamento**:
   - Dati conservati nella propria installazione di PostgreSQL per futuro riferimento
   - Eventuale aggiornamento software PostgreSQL se necessario
   - Portabilità ad altre architetture supportate tramite funzionalità di importazione di PostgreSQL

Progetto logico del DB “country profile”



- Entità:
  - regions(**id**, name)
    - Chiave primaria: **id**
    - Vincoli di unicità: name
  - countries(**id**, name, region, population, gdp)
    - Chiave primaria: **id**
    - Chiave forestiera: **region** (riferimento: regions.id)
    - Vincoli di unicità: name

Progetto fisico del DB "country profile"



- regions(**id**, name)
  - Numero di regioni: < 10
  - **id**: tipo `CHAR` (1 carattere da 0 a 9), chiave primaria
  - **name**: tipo `VARCHAR(20)`, *unico*, $\neq \emptyset$
- countries(**id**, name, region, population, gdp)
  - **id**: tipo `DECIMAL(3,0)` (3 cifre, da 0 a 999), chiave primaria
  - **name**: tipo `VARCHAR(50)` *unico*, $\neq \emptyset$
  - **region**: chiave forestiera (rif.: regions.id, tipo `CHAR`), $\neq \emptyset$
  - **area**: numerico (km^2), val. max $\sim 10^{10}$, prec: 3-4 cifre $\Rightarrow$ `REAL` (tipo `INTEGER` a 32 bit, max $\sim 2^9 < 2^{10}$)
  - **population**: numerico (N. abitanti), val. max $\sim 10^8$, prec: 3-4 cifre $\Rightarrow$ `REAL`
  - **gdp**: numerico (US$), val. max $\sim 10^{13}$, prec: 3-4 cifre $\Rightarrow$ `REAL`

Progetto fisico del DB “country profile”

- Codice SQL per creazione tabelle:

```
CREATE TABLE regions
(
    id CHAR PRIMARY KEY,
    name VARCHAR(20) UNIQUE NOT NULL
);

CREATE TABLE countries
(
    id DECIMAL(3) PRIMARY KEY,
    name VARCHAR(50) UNIQUE NOT NULL,
    region CHAR NOT NULL,
    area REAL,
    population REAL,
    gdp REAL,
    FOREIGN KEY(region) REFERENCES regions(id)
);
```

Progetto fisico del DB “country profile”

- Codice SQL per inserimento dati:

```
/* regioni */
INSERT INTO regions VALUES('0','Africa');
INSERT INTO regions VALUES('1','Americas');
INSERT INTO regions VALUES('2','Asia-Pacific');
/* ... */
INSERT INTO regions VALUES('7','South Asia');

/* paesi */
INSERT INTO countries VALUES(1,'Afghanistan','7',
    652225,26000000,NULL);
INSERT INTO countries VALUES(2,'Albania','3',
    28728,3200000,6656000000);
/* ... */
INSERT INTO countries VALUES(193,'Zimbabwe','0',
    390759,12900000,6192000000);
```

Query al DB “country profile”

- **Query1** — mostra dati aggregati sulle regioni geografiche: totale PIL, abitanti, numero di paesi, e PIL/abitante.

```
/* query1 */
SELECT
    SUM(gdp) AS total_gdp,
    SUM(population) AS total_pop,
    COUNT(*) AS n_regions,
    SUM(gdp)/SUM(population) AS welfare,
    regions.name
FROM countries
JOIN regions
ON countries.region=regions.id
GROUP BY regions.name
ORDER BY welfare DESC;
```

Query al DB “country profile”

- **Query2** — mostra nome del paese con il PIL minimo

```
SELECT countries.name, regions.name FROM regions
JOIN countries
ON countries.region=regions.id
WHERE countries.gdp=(
    SELECT min(gdp) FROM countries
);
```

Query al DB “country profile”

- **Query3** — mostra nome dei paesi europei con una popolazione superiore alla media

```
SELECT countries.name FROM countries
JOIN regions
ON countries.region=regions.id
WHERE countries.population>(
    SELECT avg(population) FROM countries
    JOIN regions
    ON countries.region=regions.id
    WHERE regions.name='Europe'
)
AND regions.name='Europe';
```

Esercizio: Hogwarts database

- Sviluppo di un database su Hogwarts

1. **Fattibilità**: OK
2. **Analisi dei requisiti**:
   - capacità di registrare dati su HP e rispondere alle query:
     - Query 1: mostra tutte le *house*
     - Query 2: mostra gli studenti e *head of house* di Griffindor
     - Query 3: mostra gli studenti, per ciascuna *house*
     - Query 4: mostra i docenti che sono (stati) anche *headmaster*
     - Query 5: mostra i docenti e gli studenti che sono nell'Ordine
     - Query 6: mostra i docenti di *Defence against the Dark Arts*
     - Query 7: mostra i personaggi non umani
     - Query 8: mostra i personaggi che non insegnano a Hogwarts
   - dati a disposizione: dai libri
   - entità dei dati: poche centinaia di record

Esercizio: Hogwarts database

- Sviluppo di un database su Hogwarts

3. **Progetto**:
   - organizzazione e struttura dei dati:
     - **Entità**, **Relazioni Proprietà** ???
   - caratteristiche degli applicativi:
     - possibilità di inserire query SQL
     - OK interfaccia testuale

   $\Rightarrow$ **Schema relazionale**

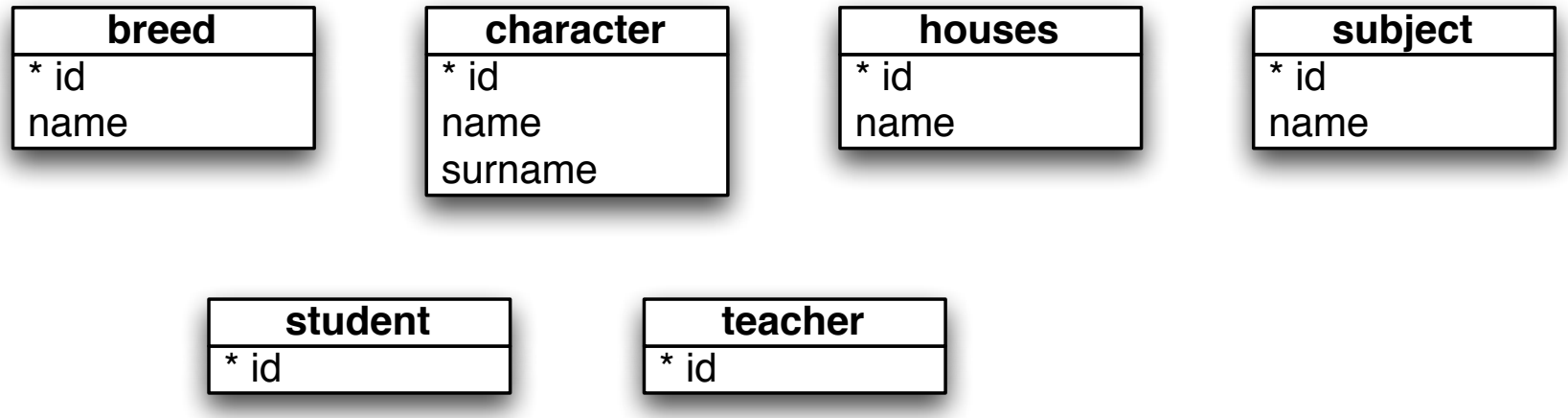
   $\Rightarrow$ **Schema fisico**

Esercizio: Hogwarts database

- Sviluppo di un database su Hogwarts

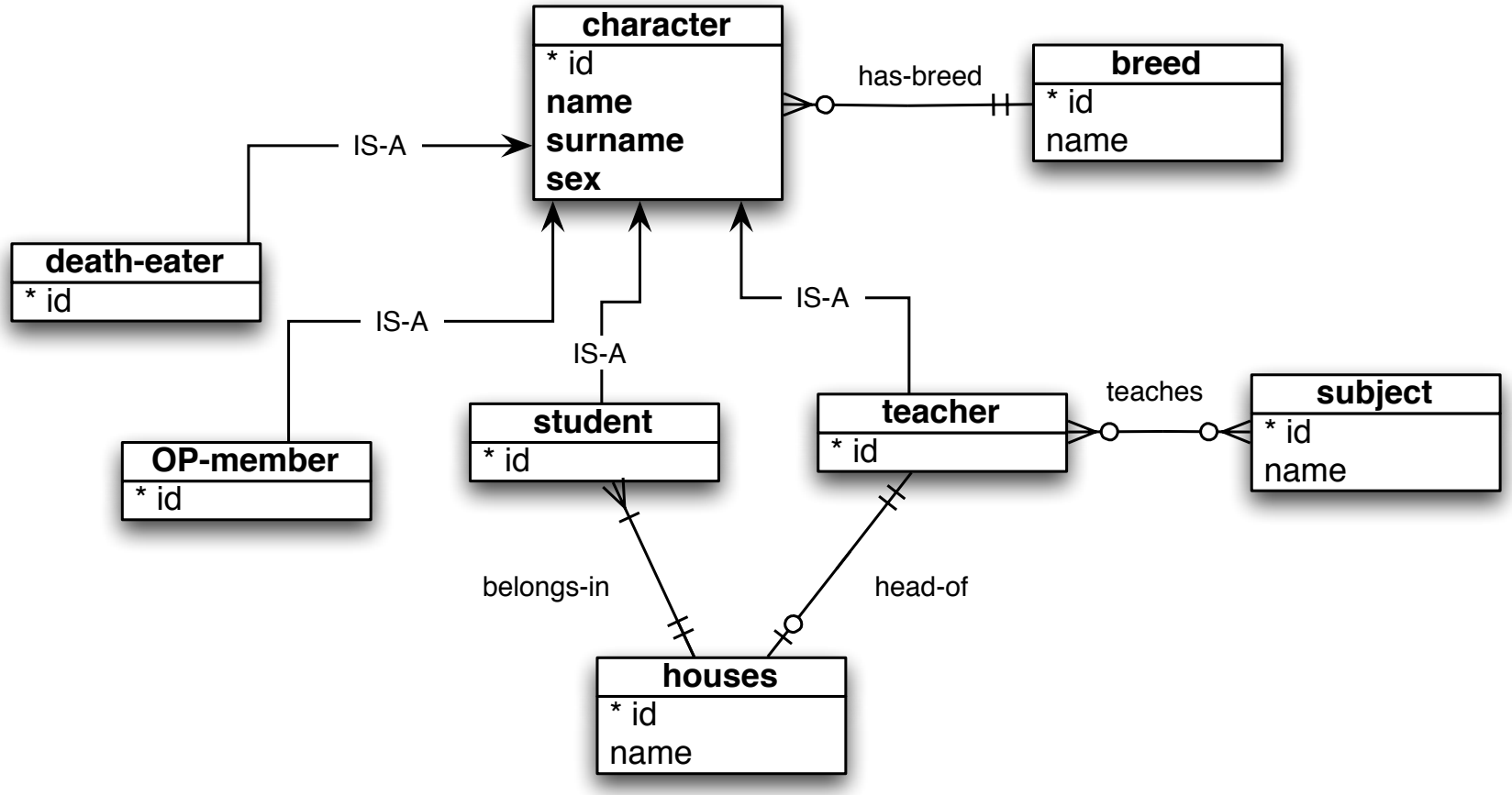
4. **Implementazione**:
   - DBMS: PostgreSQL 8.4
   - piattaforma: Mac OS X 10.5
   - ...
5. **Validazione e collaudo**:
   - tramite query SQL di tipo `SELECT`
6. **Funzionamento**:
   - Dati conservati nella propria installazione di PostgreSQL ...
   - Eventuale aggiornamento software ...
   - Portabilità ad altre architetture ...

Schema Logico



- Entità di cui bisogna tenere traccia
- Proprietà?
- Relazioni?

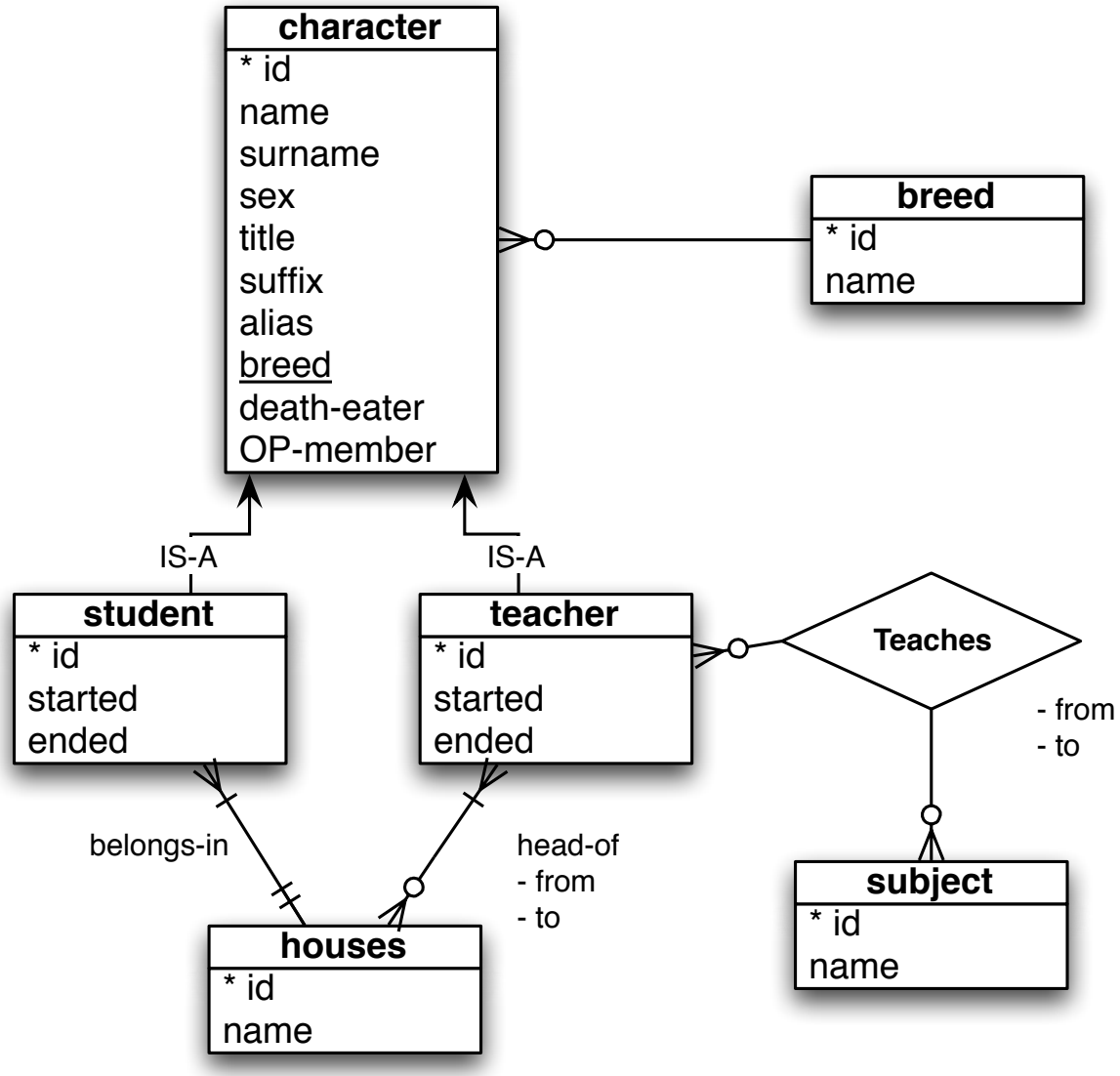
Schema Logico



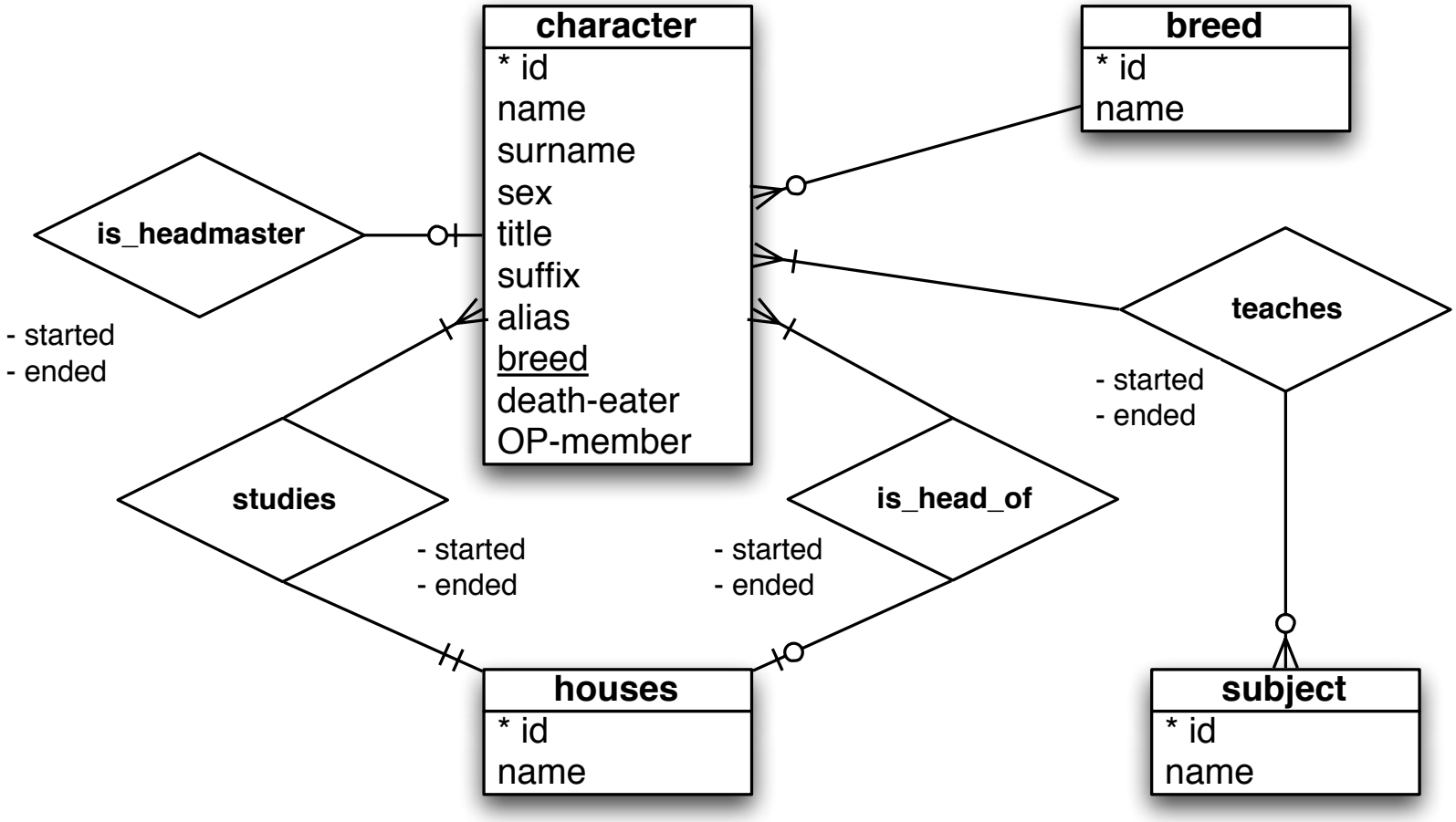
- Una prima possibilità

Schema Logico

- Una seconda possibilità
- *Breed*, *death-eater* e *OP-member* sono in realtà delle proprietà dei personaggi
- Espressa la relazione tra *teacher* e *subject*

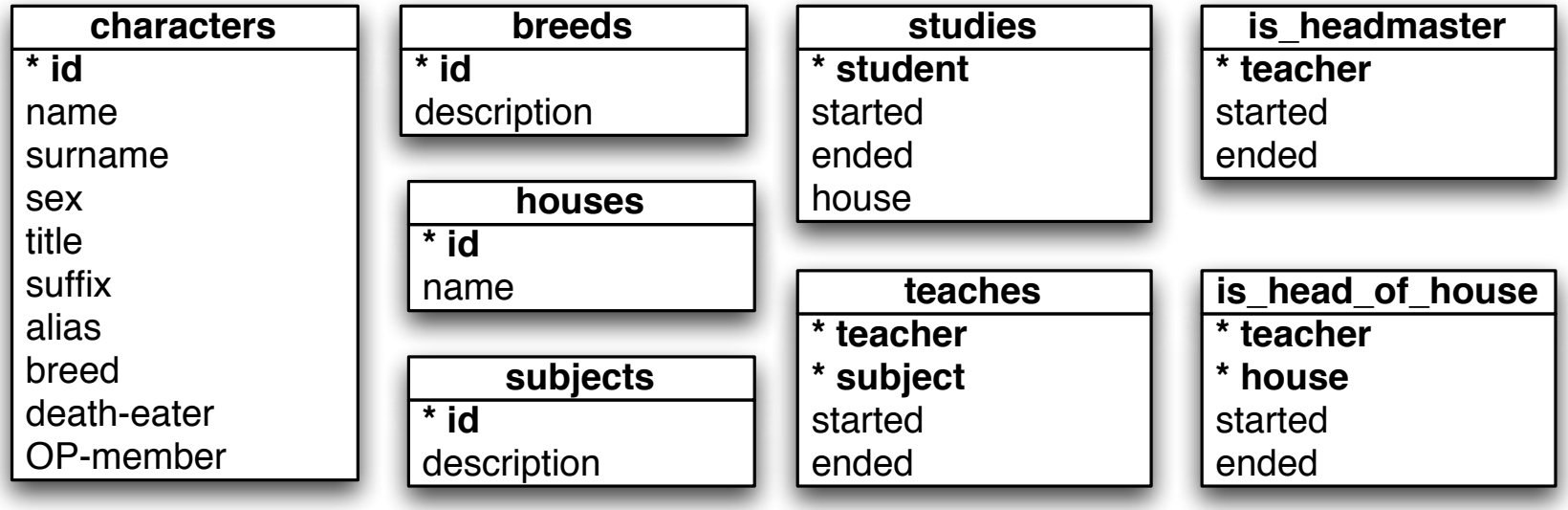


Schema Logico



- Infine . . .
  - Eliminate le relazioni IS-A
  - Selezionate le relazioni molti-molti, e quelle che hanno associate delle proprietà di interesse

Relazioni



- Tutte le entità e relazioni identificate (schema precedente) vengono mappate in relazioni

$\Rightarrow$ Definizione di chiavi principali, forestiere, vincoli di integrità

Schema fisico

```
CREATE TABLE breeds
(
    id NUMERIC(3) PRIMARY KEY,
    description VARCHAR(20)
);

CREATE TABLE houses
(
    id NUMERIC(1) PRIMARY KEY,
    name VARCHAR(20) NOT NULL
);

CREATE TABLE subjects
(
    id NUMERIC(2) PRIMARY KEY,
    description VARCHAR(30) NOT NULL
);
```

Schema fisico

```
CREATE TABLE characters
(
    id NUMERIC(3) PRIMARY KEY,
    name VARCHAR(20),
    surname VARCHAR(30),
    sex CHAR,
    title VARCHAR(10),
    suffix VARCHAR(2),
    alias VARCHAR(20),
    breed NUMERIC(2),
    death_eater BIT,
    OP_member BIT,
    FOREIGN KEY(breed) REFERENCES breeds(id)
);
```

Schema fisico

```
CREATE TABLE teaches
(
    teacher NUMERIC(3) NOT NULL,
    subject NUMERIC(2) NOT NULL,
    started DATE,
    ended DATE,
    PRIMARY KEY(teacher,subject),
    FOREIGN KEY(teacher) REFERENCES characters(id),
    FOREIGN KEY(subject) REFERENCES subjects(id)
);
```

Schema fisico

```
CREATE TABLE studies
(
    student NUMERIC(3) PRIMARY KEY,
    started DATE,
    ended DATE,
    house NUMERIC(1),
    FOREIGN KEY(student) REFERENCES characters(id),
    FOREIGN KEY(house) REFERENCES houses(id)
);
```

Schema fisico

```
CREATE TABLE is_headmaster
(
    teacher NUMERIC(3) PRIMARY KEY,
    started DATE,
    ended DATE,
    FOREIGN KEY(teacher) REFERENCES characters(id)
);
```

Schema fisico

```
CREATE TABLE is_head_of_house
(
    teacher NUMERIC(3),
    house NUMERIC(1),
    started DATE,
    ended DATE,
    PRIMARY KEY(teacher,house),
    FOREIGN KEY(teacher) REFERENCES characters(id),
    FOREIGN KEY(house) REFERENCES houses(id)
);
```

Query al DB “Hogwarts”

- **Query1** — mostra tutte le *house*.

```
SELECT name
FROM houses
```

Query al DB “Hogwarts”

- **Query2** — mostra gli studenti e *head of house* di Griffindor.

```
SELECT characters.name,characters.surname
FROM characters
JOIN studies
ON studies.student=characters.id
JOIN houses
ON houses.id=studies.house
WHERE houses.name='Griffindor'
UNION
SELECT characters.name,characters.surname
FROM characters
JOIN is_head_of_house
ON characters.id=is_head_of_house.teacher
JOIN houses
ON houses.id=is_head_of_house.house
WHERE houses.name='Griffindor';
```

Query al DB “Hogwarts”

- **Query3** — mostra gli studenti, per ciascuna *house*.

```
SELECT characters.name,characters.surname, houses.name
FROM studies
JOIN characters
ON studies.student=characters.id
JOIN houses
ON houses.id=studies.house
ORDER BY houses.id;
```

Query al DB “Hogwarts”

- **Query4** — mostra i docenti che sono (stati) anche *headmaster*.

```
SELECT characters.name,characters.surname
FROM characters
JOIN is_headmaster
ON characters.id=is_headmaster.teacher
JOIN teaches
ON characters.id=teaches.teacher;
```

Query al DB “Hogwarts”

- **Query5** — mostra i docenti e gli studenti che sono nell’Ordine.

```
SELECT characters.name,characters.surname
FROM characters
JOIN studies
ON characters.id=studies.student
WHERE characters.OP_member='1'
UNION
SELECT characters.name,characters.surname
FROM characters
JOIN teaches
ON characters.id=teaches.teacher
WHERE characters.OP_member='1';
```

Query al DB “Hogwarts”

- **Query6** — mostra i docenti di *Defence against the Dark Arts*.

```
SELECT name, surname
FROM characters
JOIN teaches
ON characters.id=teaches.teacher
JOIN subjects
ON teaches.subject=subjects.id
WHERE subjects.description='Defence Against the Dark Arts'
```

Query al DB “Hogwarts”

- **Query7** — mostra i personaggi non umani.

```
SELECT title, name, surname, suffix, alias, description
FROM characters
JOIN breeds
ON characters.breed=breeds.id
WHERE description<>'Witch or Wizard'
AND description<>'Muggle'
AND description<>'Squib'
ORDER BY breed
```

Query al DB “Hogwarts”

- **Query8** — mostra i personaggi che non insegnano a Hogwarts.

```
SELECT DISTINCT title, name, surname, suffix, alias
FROM characters
LEFT JOIN teaches
ON characters.id=teaches.teacher
WHERE teaches.teacher IS NULL
ORDER BY surname, name
```