Database: MySQL

Davide Spano

Università di Cagliari

davide.spano@unica.it

Corso di Amministrazione di Sistema

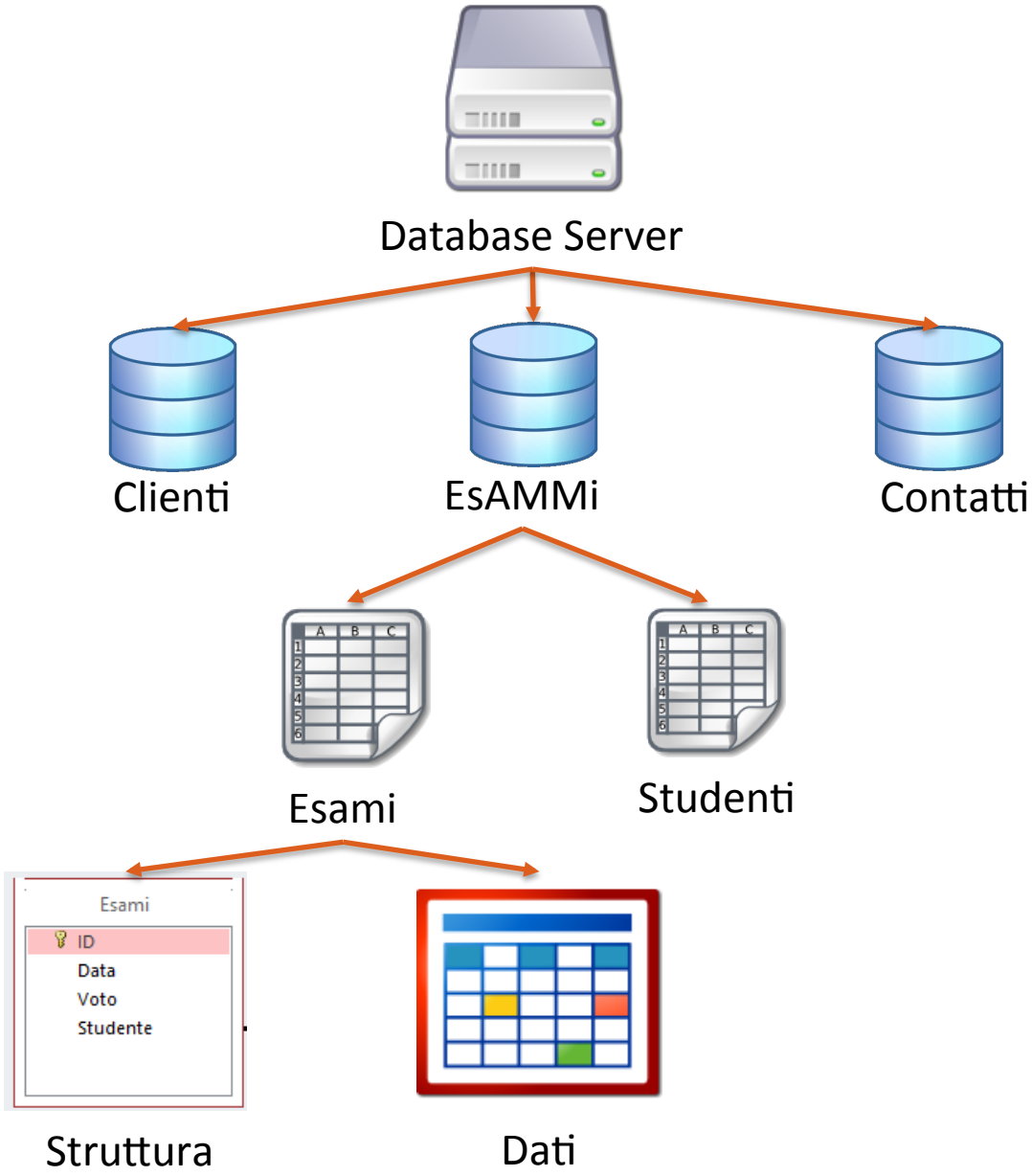
Introduzione ai Database

- In queste lezioni ripassiamo alcune proprietà dei database
- E le sfrutteremo per implementare la persistenza dei dati nelle nostre applicazioni
- **Database**: una collezione strutturata di dati, organizzati in modo che possano essere ricercati in modo efficiente
- Esistono diversi database engine
  - Noi utilizzeremo MySQL
  - Uno dei più diffusi in ambiente web
  - Open Source
  - Ne esistono altri (Oracle, MS SQL Server, Postgres)
- I dati contenuti in un database vengono acceduti e modificati tramite un linguaggio standard (SQL)
  - Noi vedremo le query principali

Il mondo delle tabelle

- I dati di un database sono organizzati utilizzando delle tabelle
- Un **database server** contiene un insieme di **database,** ognuno dei quali è identificato da un *nome*
- Ogni **database** è costituito da un insieme di **tabelle**
- Ogni **tabella** ha una **struttura**
  - Un insieme di colonne
  - Ogni colonna ha un **nome**
  - Ognuna colonna è associata ad un **tipo** di dato (intero, stringa, float ecc.)
- Ogni **tabella** contiene un insieme di dati
  - Una collezione di righe
  - Ogni riga contiene una serie di campi, associati ad una colonna della tabella

Struttura di un database server



Creazione delle tabelle

- Per creare delle tabelle in un database dobbiamo specificare
  - Un nome per la tabella
  - Per ogni colonna, un nome ed un tipo di dato
- È molto importante studiare la struttura del database a tavolino, prima di iniziare l'inserimento dei dati
  - In modo da non doverla modificare durante lo sviluppo
  - In modo da non dimenticare di mantenere dei dati utili
- Esistono semplici tecniche tipiche di design
  - Noi accenneremo a qualcosa, si vedono al corso di database

Organizzare i dati in tabelle

- Sappiamo *cosa* si può inserire all'interno delle celle delle tabelle
- Ma non abbiamo ancora parlato di *come* organizzarle…
- Vediamo giusto un paio di concetti per modellare dati semplici
- Dietro c'è una buona dose di teoria che vedrete a basi di dati
  - Chiavi primarie
  - Forme normali
  - …
- C'è una buona correlazione tra il design delle classi e quella delle tabelle
- … specialmente nei casi semplici
- Tenete presente che ci sono figure professionali che si occupano solo di questo!

Il tabellone

Presidente	Membro1	Membro2	Insegnamento	Matricola	Studente	Voto
Davide Spano	Riccardo Scateni	Gianni Fenu	AMM	12345	Mario Rossi	24
Davide Spano	Riccardo Scateni	Gianni Fenu	AMM	54321	Paola Bianchi	30
Davide Spano	Riccardo Scateni	Gianni Fenu	AMM	23456	Pinco Pallino	27
Riccardo Scateni	Gianni Fenu	Davide Spano	IUM	12345	Mario Rossi	30

- Una tabella unica per tutti i dati
- Tende ad avere tantissime colonne
- Non ha assolutamente struttura
- I dati sono duplicati
- In poche parole **non si deve usare!**

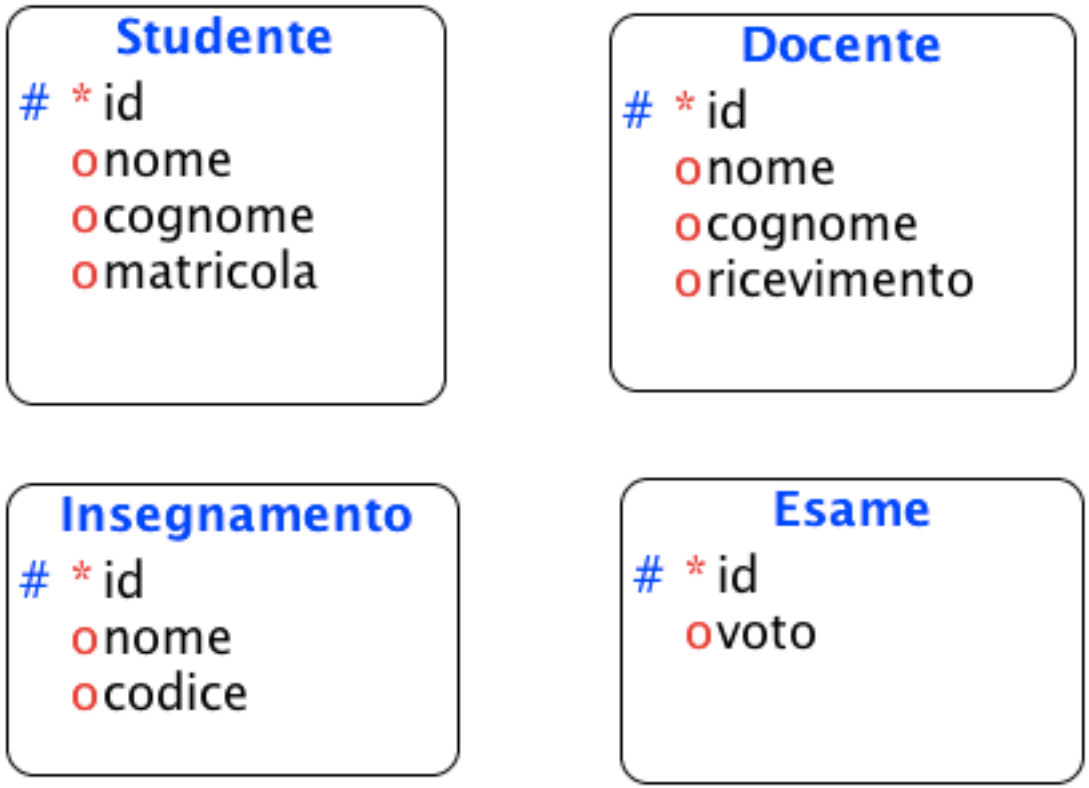
Regole per un buon design

- Includere un identificatore unico per ogni elemento
- Ogni colonna deve contenere un singolo valore
  - Spesso invece vengono inseriti nella stessa colonna utilizzando caratteri separatori
- Non ci devono essere colonne che ripetono lo stesso tipo di dato
  - Nel nostro esempio Membro1 e Membro2
  - Spesso se aggiungete numeri in coda al nome è segnale di un cattivo design
- Non ci devono essere dati ripetuti
  - Nel nostro esempio non è sicuro che alla stessa matricola corrisponda lo stesso nome dello studente

Passo 1: identificare le entità

- Le entità sono ciò che dovete salvare all'interno del database
- Tecnicamente, un'entità è tutto ciò che può essere riconosciuto in modo indipendente
- E quindi può essere identificato da un identificatore unico
- Le entità possiedono degli attributi che le caratterizzano
  - Almeno l'identificatore unico è un attributo
- Gli attributi possono avere diversi tipi
  - Intero
  - Stringa
  - Float
  - …
- Ovviamente per la stessa applicazione è possibile identificare entità diverse

Entità esempio



Le chiavi primarie

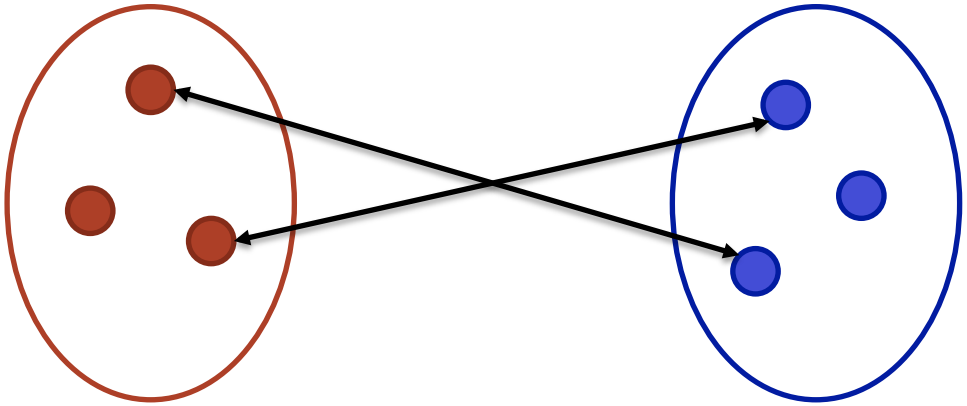
- Una chiave è un insieme dei campi di una tabella che identificano in modo unico una riga
- Di solito si utilizza un campo apposito che funzioni come chiave, a cui si da un valore intero progressivo
- Questo valore viene incrementato in automatico dal DB (esistono dei tipi appositi)
- In alcune tabelle però è necessario utilizzare più di un campo per identificare una riga
- Una chiave primaria è una chiave con il numero minimo di campi
- In teoria ne esistono più di una per tabella
- Ma per noi di solito sarà solo una

Relazioni

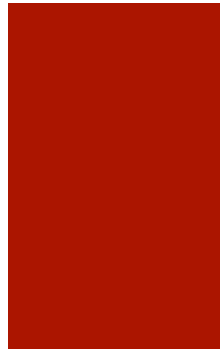
- Oltre agli attributi, le entità possono essere collegate ad altre entità
- Quando una entità è collegata ad un'altra, si dice che tra loro esiste una **relazione**
- Una relazione si classifica in base alla cardinalità
- **Uno ad Uno**
- **Uno a Molti**
- **Molti a Molti**

Relazione uno ad uno

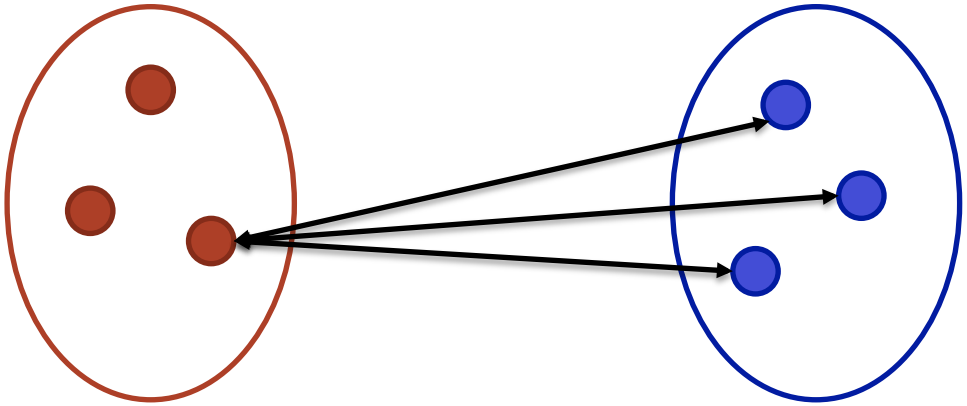
- Una relazione dove ad un entità di un tipo ne corrisponde al più una dell'altro
- Alcune relazioni possono non essere specificate per alcuni elementi
- Per altre invece è necessario che lo siano (esattamente uno)
- Esempi:
  - Il conducente di un'auto
  - Il coniuge (attuale)



Relazione uno a molti

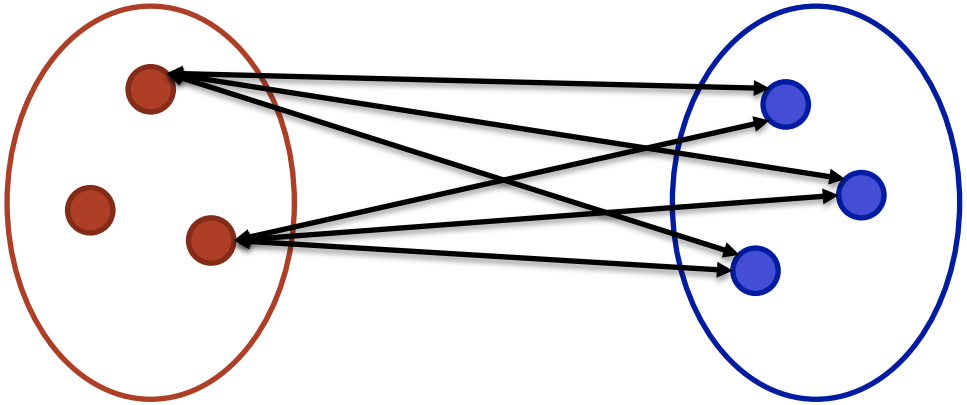


- Una relazione uno a molti si ha quando una entità di un tipo è collegata a molte entità del secondo
- Anche in questo caso, alcune relazioni sono specificate per zero o più valori o per uno o più valori
- Esempio:
  - Padre o madre
  - Autori di un libro
  - Acquisto di un oggetto



Relazione molti a molti

- È una relazione che collega più di una entità di un tipo a più di un tipo dell'altra
- Vuol dire che uno stesso elemento del tipo rosso ha associato più di un elemento blu
- E che uno stesso elemento del tipo blu ha associato più di un rosso
- Anche in questo caso si possono avere delle relazioni che per alcuni elementi non sono specificate
- Esempio: fratelli e sorelle



Stabilire il tipo di relazione

- C'è una domanda magica che ci aiuta a stabilire la cardinalità della relazione fra una entità A ed una B
- ***Per ogni A, quanti sono i possibili B?***
- Le risposte possibili sono due: **uno** o **molti**
- Nei casi con un numero esatto maggiore di uno (es. 3), di solito si riporta al caso "Molti"
- Inoltre, bisogna stabilire in ogni caso se la relazione debba sempre esistere, oppure se sia opzionale
  - Uno
    - Zero o Uno
    - Esattamente uno
  - Molti
    - Zero o più
    - Uno o più

Stabilire il tipo di relazione

- La risposta alla domanda precedente è la cardinalità della relazione da A a B
- Ma dobbiamo stabilire anche la cardinalità da B a A
- Si opera nello stesso modo rifacciamo la domanda al contrario
- ***Per ogni B, quanti sono i possibili A?***
- Da qui otteniamo quattro casi
  - A -> B = Uno, B -> A = Uno => **Relazione uno ad uno**
  - A -> B = Molti, B -> A = Uno => **Relazione uno a molti**
  - A -> B = Uno, B -> A = Molti => **Relazione uno ad molti**
  - A -> B = Molti, B -> A = Molti => **Relazione molti a molti**

Attributi nelle relazioni

- È possibile che vi siano da specificare degli attributi anche per le relazioni
  - Un esempio tipico è una data o una quantità
    - Data di matrimonio
    - Data di associazione ad un gruppo
    - Numero di istanze di un'associazione
- Nelle relazioni uno ad uno, uno a molti o molti a uno si seleziona la tabella con cardinalità uno
  - E si aggiunge un campo che specifica l'attributo della relazione
  - In pratica si fa come per la chiave esterna
- Nelle relazioni molti a molti, il campo si aggiunge nella tabella che rappresenta la relazione

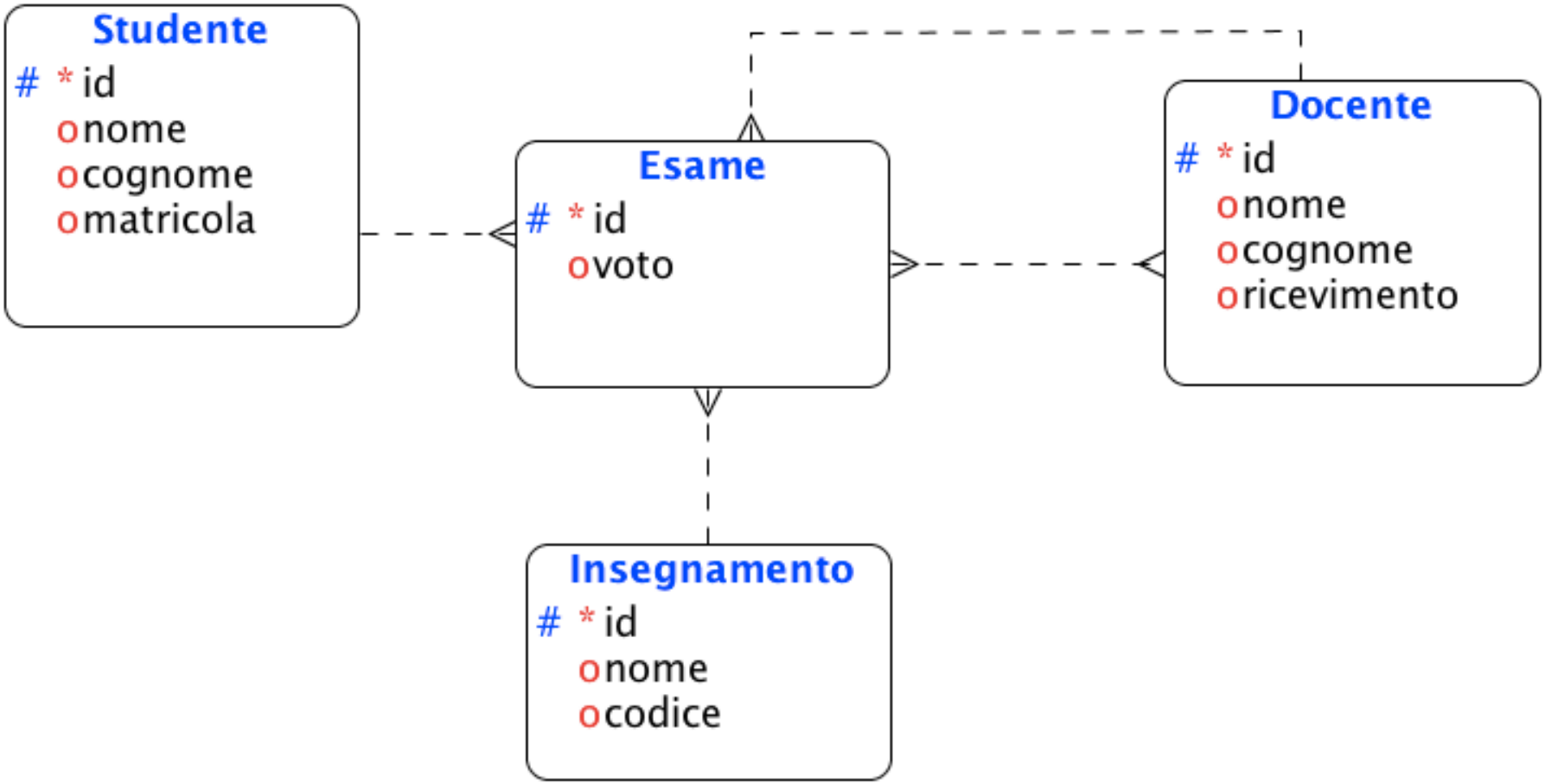
Esempio: Esami

- Un esame ha un docente presidente di commissione e due docenti membri
- E ovviamente uno studente che lo sostiene
- Quattro relazioni:
  - Esame e Docente (presidente)
  - Esame e Docente (membro commissione)
  - Esame e Insegnamento
  - Esame e Studente
- Poniamoci le domandine magiche

Esempio: Esami (2)

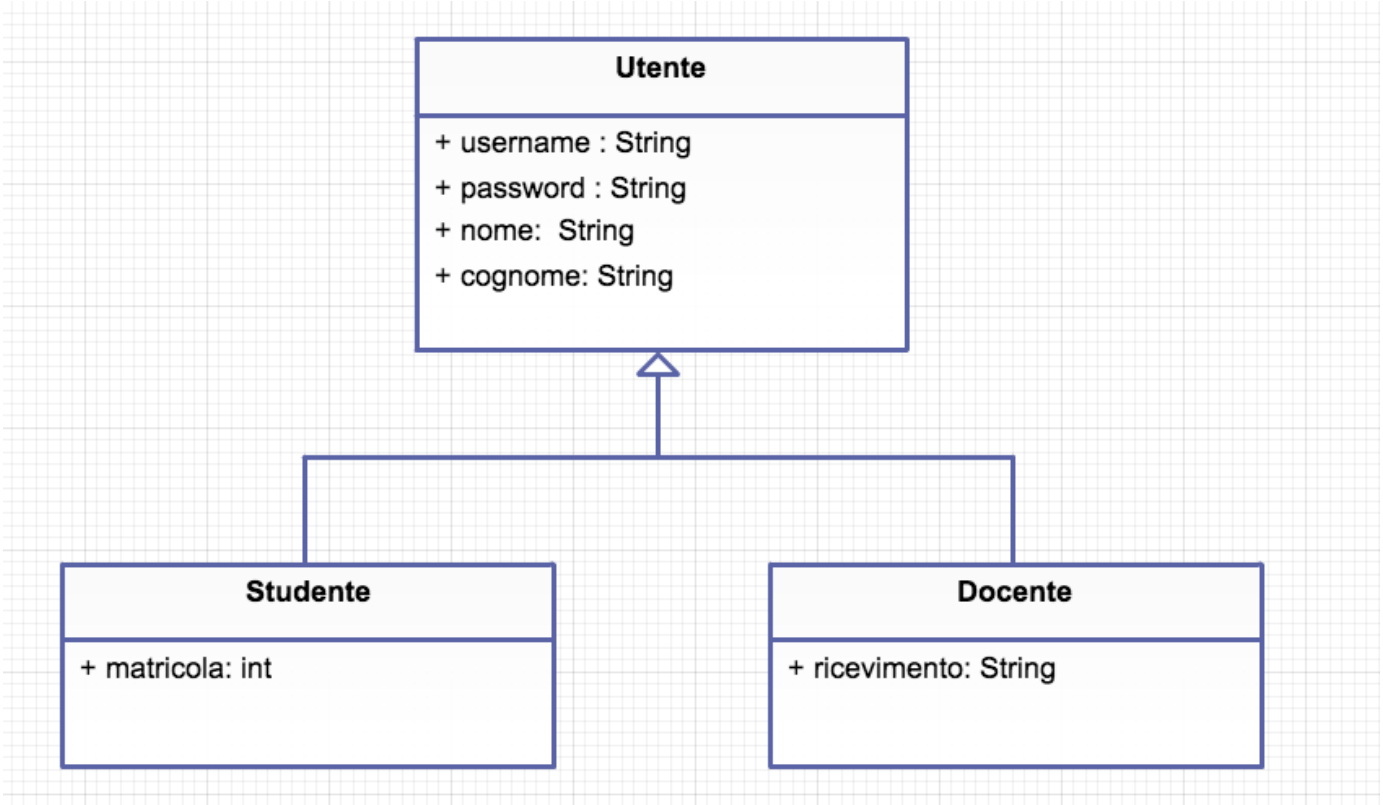
- Presidente
  - Per ogni Esame, quanti Docenti (presidenti)? **Uno**
  - Per ogni Docente, quanti Esami (da presidente)? **Molti**
  - **Relazione uno a molti**
- Membro
  - Per ogni Esame, quanti Docenti (membri)? **Molti**
  - Per ogni Docente, quanti Esami (da membro)? **Molti**
  - **Relazione molti a molti**
- Studente
  - Per ogni Esame, quanti Studenti? **Uno**
  - Per ogni Studente, quanti Esami? **Molti**
  - **Relazione uno a molti**
- Non abbiamo relazioni uno ad uno
  - Un esempio facile è la relazione coniuge tra persone

Esempio: Esami (3)



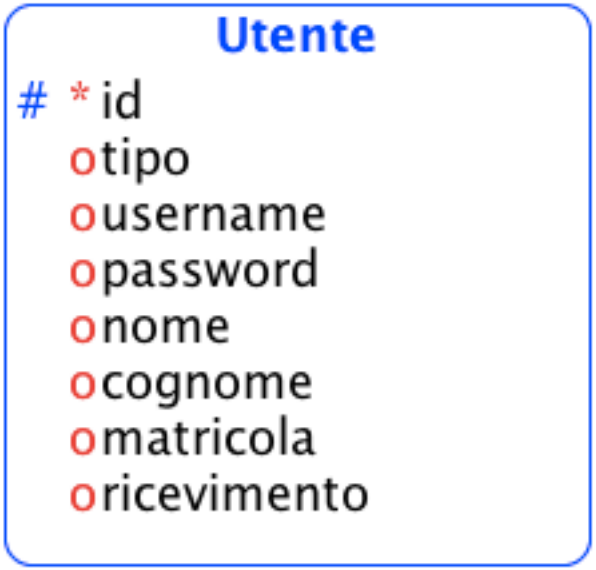
Ereditarietà

- È possibile modellare l'ereditarietà delle classi con le entità e le relazioni
- Non c'è una soluzione unica per questo problema
  - Ne vedremo tre differenti
  - Sono tutte corrette
  - Hanno svantaggi e vantaggi
  - Vanno valutate caso per caso



Ereditarietà soluzione 1

- Entità unica per superclasse e sottoclassi
- Gli attributi delle sottoclassi sono opzionali
- Si aggiunge un campo per distinguere gli Studenti dai Docenti

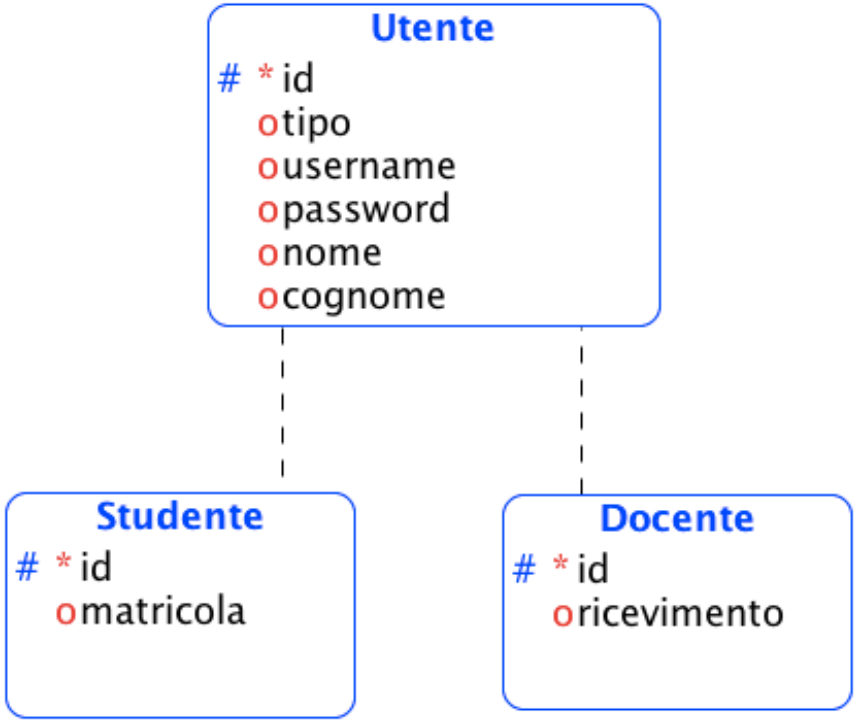


Soluzione 1: Pro e Contro

- Pro
  - Molto semplice da implementare
  - Tutto si trova in una tabella unica
  - Utenti, Docenti e Studenti hanno la stessa chiave primaria
- Contro
  - Potenzialmente la tabella ha tante colonne
  - Spesso sono vuote perché non hanno significato
    - Es. l'orario di ricevimento di uno studente…

Ereditarietà soluzione 2

- Una entità per la superclasse ed una per ogni sottoclasse
- La sottoclasse è collegata con una relazione uno ad uno con la superclasse

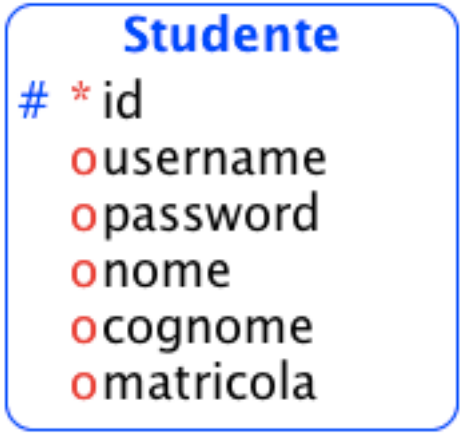


Soluzione 2: Pro e Contro

- Pro
  - Corrispondenza uno ad uno con le classi
  - Nessuna colonna senza dati
- Contro
  - La struttura è più complessa
  - L'estrazione dei dati richiede delle query più complesse

Ereditarietà soluzione 3

- Una entità solo per ognuna delle sottoclassi
- I dati della superclasse si “spalmano” su ognuna delle tabelle



Soluzione 3: Pro e Contro

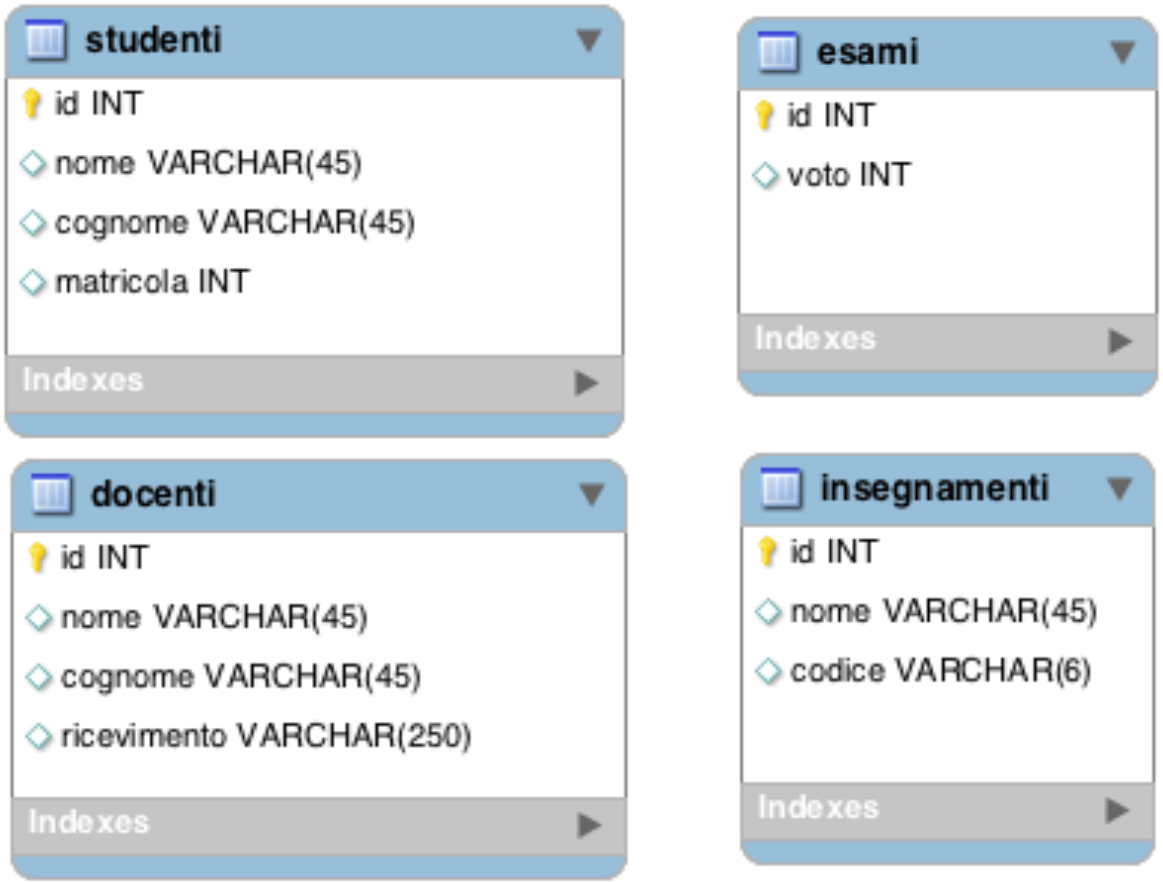
- Pro
  - Tentativo di prendere i vantaggi della soluzione 1 e 2
  - Non ci sono attributi non validi
  - Non si introducono nuove relazioni
  - Le query sono semplici
- Contro
  - Si elimina la comodità di avere i dati comuni concentrati sulla superclasse
    - Per esempio se voglio tutti i nomi e cognomi degli utenti devo controllare due entità
  - C'è comunque un po' di ridondanza

Dalle entità-relazioni alle tabelle

- Abbiamo creato la struttura logica del nostro database
- Ma il database non era costituito da tabelle?
- E le relazioni dove le mettiamo?
- C'è un modo praticamente automatico per tradurre il nostro modello relazionale in uno fisico (con tabelle)

Entità

- Per ogni entità identificata, si crea una tabella
- La tabella ha una colonna per ogni attributo identificato
- Ovviamente con il tipo giusto

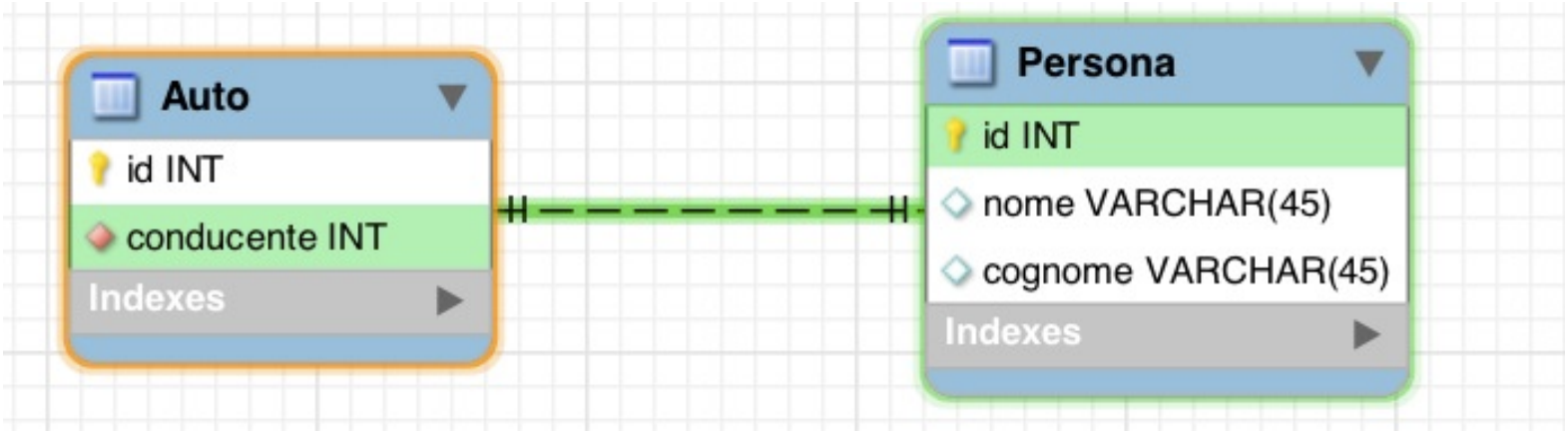


Relazioni

- Le relazioni si traducono in campi da inserire nelle tabelle
- Oppure in nuove tabelle
- La modalità di traduzione dipende dal tipo di relazione che dobbiamo modellare
- Vediamole una per una

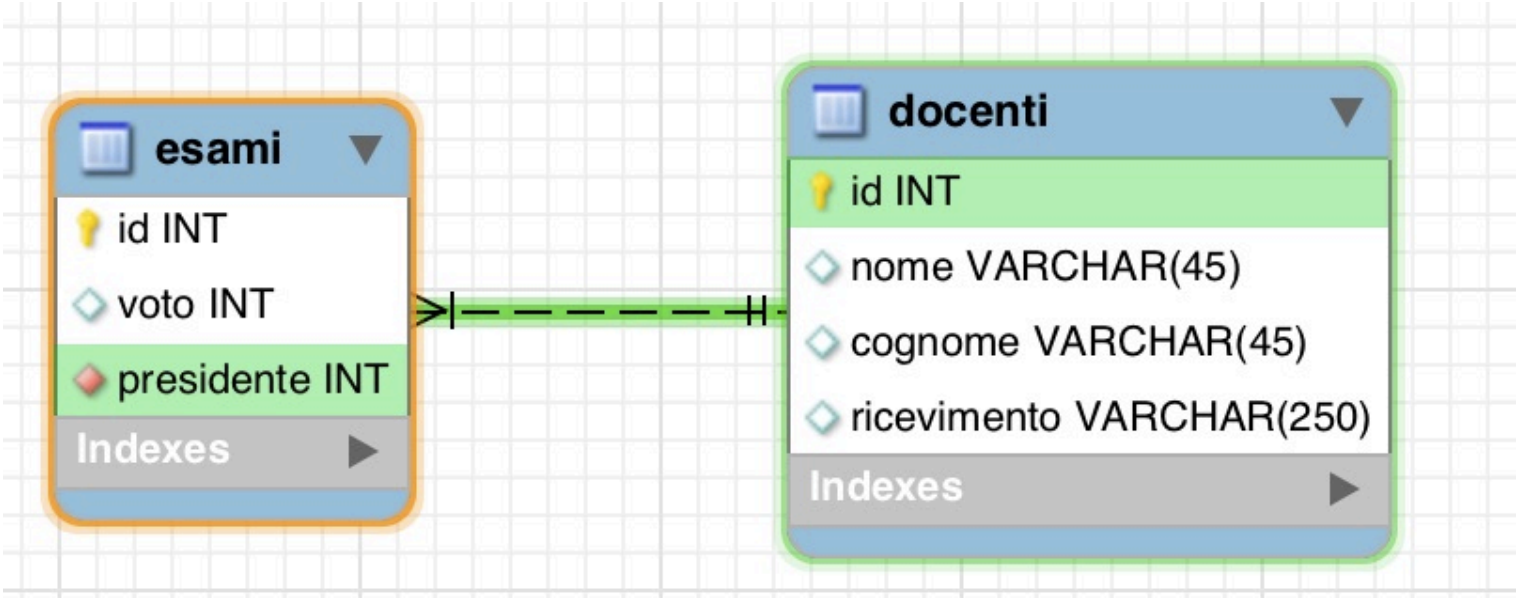
Relazione uno ad uno

- Si crea una colonna della prima che “punta” alla chiave primaria della seconda (**chiave esterna**)
- Scegliere la prima o la seconda tabella per ospitare la colonna è indifferente
- Di solito si usa come nome della colonna quello della relazione
- Si permette che non ci sia un valore specificato nella colonna nel caso la relazione possa essere non specificata



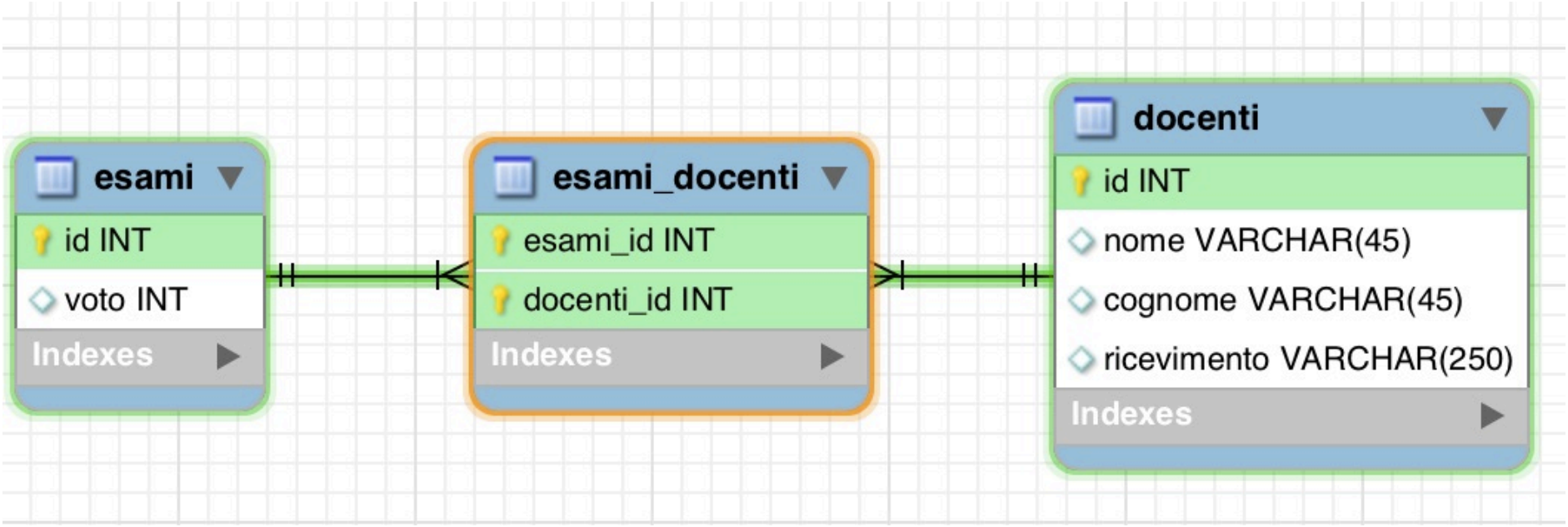
Relazione uno a molti

- Si seleziona la tabella che rappresenta l'entità con cardinalità "molti"
- E si aggiunge una colonna che punta alla tabella dell'entità che ha cardinalità "uno" (**chiave esterna**)
- Si permette che non ci sia un valore specificato nella colonna nel caso la relazione possa essere non specificata



Relazione molti a molti

- Si aggiunge una tabella che contiene un riferimento alla prima tabella ed un riferimento alla seconda
- Questa tabella non contiene entità, serve solo a modellare la relazione
- La tabella contiene una riga per ogni coppia di entità associate
- La chiave primaria di questa tabella è la coppia di chiavi esterne verso le altre due (esami_id e docenti_id nell'esempio)



Accesso ad un database

- Il database server mantiene una lista di utenti che possono accedere ai vari database
- Ha la possibilità di specificare diversi diritti di lettura/ scrittura sui diversi database
- Per questo è necessario specificare le credenziali per connettersi
  - Anche quando si accede tramite PHP

Inviare comandi ad un database

- Ci si può connettere al database server MySQL in 3 modi
  - Command Line (comando mysql)
  - Tramite interfaccia grafica (PhpMyAdmin)
  - Tramite un API di un linguaggio di programmazione (PHP)
- Una volta connessi, si possono inviare dei comandi al database
- Si specificano tramite un linguaggio standard: **SQL**
  - Structured Query Language
  - Viene utilizzato da tutti i database relazionali (con qualche variante)
- Tipi di comando:
  - *Data Definition*: definizione della struttura dei dati
  - *Data Manipulation*: inserimento, cancellazione e modifica dei dati
  - *Query*: interrogazioni sui dati
  - *Control*: controllo su accessi ai dati

CRUD

- Per ognuno degli elementi del database server (database, tabelle, righe) è utile pensare a chi ha diritto di eseguire le quattro operazioni possibili
- **Create**: creare un nuovo elemento
- **Read**: leggere lo stato corrente di un elemento
- **Update**: aggiornare lo stato corrente di un elemento
- **Delete**: cancellare lo stato corrente di un elemento
- Queste quattro operazioni sono in gergo riferite con l'acronimo CRUD
- Di solito, alcuni utenti non hanno nessun diritto, altri hanno solo il read, altri li hanno tutti

SQL: creazione di un database

- Creazione di un database:
  ```
  CREATE DATABASE nomedatabase;
  ```
  - `nomedatabase` è il nome del database
- Creazione di un utente:
  ```
  CREATE USER 'nome'@'hostname' IDENTIFIED BY 'password';
  ```
  - `nome` è il nome dell'utente
  - `hostname` è il nome del server dove viene ospitato il database (solitamente localhost)
  - `password` è la password di accesso
  - Es. `CREATE USER 'davide'@'localhost' IDENTIFIED BY 'spano';`
- Come tutti i comandi che vedremo anche in seguito, possono fallire per vari motivi
  - Il database ci specifica un codice di errore, che possiamo utilizzare per identificarne la causa
  - Google in questo caso è un fedele alleato

Utilizzo di un database per query

- Assegnare diritti di lettura e scrittura ad un utente
  `GRANT ALL ON esami.* TO 'davide'@'localhost'`
  - ALL sta per tutti i diritti (lettura/scrittura)
  - Lo * sta per tutte le tabelle contenute nel database
- Selezionare un database per le query successive:
  `USE nomedatabase;`
  - `nomedatabase` è il nome del database

MySQL: Tipi di dato

- MySQL può salvare all'interno delle tabelle diversi tipi di dato:
- **Stringhe e testi**
- **Numeri**
- **Date**
- **Formati binari**
- In aggiunta, possiede dei tipi per gestire gli identificatori unici per le righe di una tabella
  - **Serial**

Dati testuali

- Per i dati testuali si utilizzano principalmente due tipi:
- **CHAR** e **VARCHAR**
- Per entrambi si specifica il numero di caratteri che possono contenere
- **CHAR** utilizza *sempre tutti i caratteri*
  - Se specificate una stringa più corta viene riempita con spazi
- **VARCHAR** usa *al più i caratteri specificati*
  - Non riempie la stringa con spazi
- Entrambe troncano la stringa se troppo lunga

Data type	Bytes used	Examples
CHAR(*n*)	Exactly *n* (<= 255)	CHAR(5): "Hello" uses 5 bytes
		CHAR(57): "New York" uses 57 bytes
VARCHAR(*n*)	Up to *n* (<= 65535)	VARCHAR(100): "Greetings" uses 9 bytes plus 1 byte overhead
		VARCHAR(7): "Morning" uses 7 bytes plus 1 byte overhead

Dati numerici

Data type	Bytes used	Minimum value (signed/unsigned)	Maximum value (signed/unsigned)
TINYINT	1	–128	127
		0	255
SMALLINT	2	–32768	32767
		0	65535
MEDIUMINT	3	–8388608	8388607
		0	16777215
	4	–2147483648	2147483647
INT or INTEGER		0	4294967295
BIGINT	8	–9223372036854775808	9223372036854775807
		0	18446744073709551615
FLOAT	4	–3.402823466E+38	3.402823466E+38
		(no unsigned)	(no unsigned)
DOUBLE or REAL	8	–1.7976931348623157E+308	1.7976931348623157E+308
		(no unsigned)	(no unsigned)

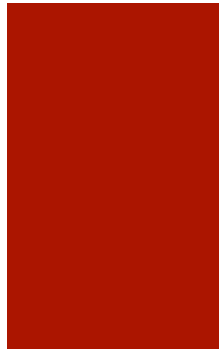
- Per i boolean c'è BOOL, ma è un alias di TINYINT
- SERIAL è un alias di BIGINT UNSIGNED che viene utilizzato per gli id unici

Date

Data type	Time/date format
DATETIME	'0000-00-00 00:00:00'
DATE	'0000-00-00'
TIMESTAMP	'0000-00-00 00:00:00'
TIME	'00:00:00'
YEAR	0000 (Only years 0000 and 1901 - 2155)

- Piccole differenze
  - TIMESTAMP arriva massimo al 2038
  - Ma può essere utilizzato per includere la data corrente in automatico
  - DATETIME può assumere valori più lontani nel tempo

Dati binari



Data type	Bytes used	Attributes
TINYBLOB(*n*)	Up to *n* (<= 255)	Treated as binary data—no character set
BLOB(*n*)	Up to *n* (<= 65535)	Treated as binary data—no character set
MEDIUMBLOB(*n*)	Up to *n* (<= 16777215)	Treated as binary data—no character set
LONGBLOB(*n*)	Up to *n* (<= 4294967295)	Treated as binary data—no character set

- È possibile salvare all'interno del database dei file di tipo binario
  - Immagini
  - Video
  - …
- Si utilizzano le diverse varianti del tipo blob

SQL: Manipolazione di tabelle

- Una volta che abbiamo definito la struttura delle tabelle, è possibile crearle tramite la sintassi SQL

- Creazione di una tabella

```
CREATE TABLE studenti (
    id SERIAL,
    nome VARCHAR(128),
    cognome VARCHAR(128),
    matricola INT
);
```

  - Si specifica prima il nome della tabella
  - Poi si elencano fra parentesi tonde i nomi delle colonne con i rispettivi tipi di dato
  - Utilizzare il tipo `SERIAL` imposta in modo automatico anche la chiave primaria della tabella

- Una tabella si cancella con la seguente sintassi

```
DROP TABLE studenti;
```

SQL: Manipolazione delle tabelle (2)

- Una volta create, possiamo visualizzare la struttura di una determinata tabella con il comando
  `DESCRIBE studente`

```
+-----------+---------------------+------+-----+---------+----------------+
| Field     | Type                | Null | Key | Default | Extra          |
+-----------+---------------------+------+-----+---------+----------------+
| id        | bigint(20) unsigned | NO   | PRI | NULL    | auto_increment |
| nome      | varchar(128)        | YES  |     | NULL    |                |
| cognome   | varchar(129)        | YES  |     | NULL    |                |
| matricola | int(11)             | YES  |     | NULL    |                |
+-----------+---------------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

SQL: Manipolazione delle tabelle (3)

- Una volta creata, è possibile modificare una tabella tramite il comando **ALTER**
- Aggiungere una colonna via (varchar) alla tabella studente
```
ALTER TABLE studenti ADD via VARCHAR(250);
```
- Cambiare il tipo di dato di una colonna (qui aumentiamo i caratteri del varchar)
```
ALTER TABLE studenti MODIFY via VARCHAR(128);
```
- Rinominare una colonna
```
ALTER TABLE studenti CHANGE via indirizzo
VARCHAR(128);
```
- Eliminare una colonna
```
ALTER TABLE studenti DROP via;
```
- Specifica della chiave primaria dopo la creazione
```
ALTER TABLE studenti ADD PRIMARY KEY(id);
```

SQL: Manipolazione delle tabelle (4)

```
/* creiamo una tabella che abbia una chiave
 * esterna verso la tabella studente
 *
 * Passo 1 creare la tabella
 */
CREATE TABLE esami (
    id SERIAL,
    voto INT,
    studente_id BIGINT UNSIGNED
);

/*
 * Passo 2 specificare il vincolo
 * di chiave esterna
 */
ALTER TABLE esami ADD FOREIGN KEY studenti_fk (studente_id)
REFERENCES studenti (id) ON UPDATE CASCATE;
```

SQL: Manipolazione delle tabelle (4)

- La query precedente prima crea la tabella esami
- Poi aggiunge un vincolo di chiave esterna:
  - **`studenti_fk`** è il nome del vincolo di chiave esterna (mantenuto dal database)
  - **`studente_id`** è il nome del campo sulla tabella esami che contiene il riferimento alla tabella **`studenti`**
  - **`studenti`** è il nome della tabella verso la quale la chiave esterna punta
  - **`id`** è il nome del campo della tabella **`studenti`** che viene puntato da **`esami`**
- La parte **`ON UPDATE`** specifica cosa succeda nel caso venga modificato il campo id di uno studente puntato, o cancellata la riga
  - **CASCADE:** la modifica viene effettuata anche nella tabella esami, in cascata
  - **SET NULL:** il campo della tabella esami viene messo a NULL (non specificato
  - **NO ACTION:** non si fa nulla

SQL: Inserimento dei dati

- Una volta che la struttura dei dati è stata tradotta in un insieme di tabelle, bisogna iniziare a popolarle con dei dati
- Ovviamente esistono dei comandi SQL appositi
- Inserimento di una riga

```
INSERT INTO studenti
(id, indirizzo, cognome, matricola, nome)
VALUES
( default, "Via di qua",
 "Spano", 123456, "Davide");
```

- Le colonne possono essere specificate in qualsiasi ordine
- `default` è una parola chiave che inserisce il valore di default per il tipo
  - Zero per gli interi, NULL per le stringe ecc.
  - Per i `SERIAL`, inserisce il prossimo identificatore progressivo valido

Ricerca di dati

- ```
  SELECT id, nome, cognome, matricola
  FROM studenti
  WHERE nome = "Davide" AND id = 1;
  ```
- Restituisce una lista di righe
  - Nessuna nel caso non vi siano dati che soddisfano la ricerca
- Nome della tabella su cui si vuole cercare dopo il **FROM**
- La lista delle colonne da selezionare dopo la **SELECT**
  - Si utilizza * per specificare tutte le colonne della tabella
- La clausola **WHERE** è un predicato che deve essere soddisfatto da tutte le righe che vengono restituite
  - In pratica filtra le righe che non lo soddisfano
  - È un'espressione booleana
  - Che effettua dei test sui valori delle colonne
  ```

Modifica di un dato

- La modifica di un dato procede logicamente in due passi:
  - Prima si ricerca una lista di righe
  - Poi, su tutte le righe trovate, si modifica il contenuto di una o più colonne

- ```
  UPDATE studenti SET
  matricola = 253662, nome = "Mario"
  WHERE id = 1 AND cognome = "Spano";
  ```

- La tabella da modificare si specifica dopo la parola chiave

- Dopo di che si specifica una lista di *nome-colonna* = *valore*, separati da virgole
  - Rappresentano la lista di modifiche da fare alle righe

- Come nella `SELECT` , la `WHERE` filtra le righe della tabella con un criterio booleano
  ```

Cancellazione di dati

- Come per la update, si procede in due passi:
  - Si seleziona una lista di righe
  - E si cancella
- ```
  DELETE FROM studenti
  WHERE id = 2 AND cognome = "Spano";
  ```
- La tabella dove si vuole cancellare viene specificata dopo `DELETE FROM`
- Le righe si selezionano specificando la clausola `WHERE`
- La cancellazione ed in generale la modifica dei dati devono rispettare i vincoli dello schema del database+
  - Altrimenti viene segnalato un errore
  - Per esempio, solitamente non si può cancellare una riga “puntata” da una chiave esterna
  ```

Ordinamento

- È possibile farsi restituire le righe di una tabella, ordinate per una certa colonna o gruppo di colonne
- L'ordinamento si effettua dopo che gli elementi sono stati filtrati
- `SELECT * FROM studenti ORDER BY cognome;`
  - Restituisce tutti gli studenti ordinati per cognome in modo ascendente, dal più piccolo al più grande (dalla A alla Z)
  - Ovviamente vale per qualsiasi tipo su cui sia definito un ordinamento (es. interi)
- `SELECT * FROM studenti ORDER BY cognome, nome;`
  - Restituisce tutti gli studenti ordinati prima per cognome e poi per nome
- `SELECT * FROM studenti ORDER BY cognome DESC;`
  - Restituisce tutti gli studenti ordinati per cognome in modo discendente (dalla Z alla A)

Contare elementi

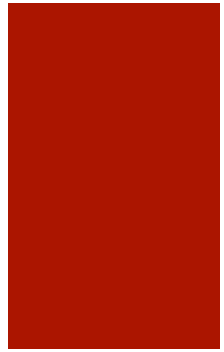
- Delle volte, è necessario semplicemente contare quante righe corrispondono ad una certa ricerca
- Per esempio vogliamo conoscere quanti studenti abbiamo sul database che abbiano la matricola superiore a 20000
- Per questo si utilizza **COUNT**
- ```
  SELECT COUNT(*) FROM studenti
  WHERE matricola > 20000;
  ```
  - Restituisce una sola riga con il numero di studenti
  - Al posto di *, si può specificare il nome di una colonna
  ```

Raggruppare righe

- La raggruppa più righe che abbiano lo stesso contenuto in una colonna
- Spesso è utilizzata insieme alla funzione
- Per esempio, supponiamo di voler contare la frequenza dei voti negli esami sul nostro database
- ```
  SELECT voto, COUNT(voto)
  FROM studenti GROUP BY voto;
  ```
  - Restituisce una riga per ogni gruppo (ogni voto presente sulla tabella), con il numero di elementi del gruppo

```
+------+-------------+
| voto | count(voto) |
+------+-------------+
|   24 |           4 |
|   27 |           3 |
+------+-------------+
```
```

Limitare il numero di righe



- Soprattutto quando si devono visualizzare dei risultati su una pagina web, è inutile elencarli con un listone unico
  - In alcuni casi i risultati potrebbero essere migliaia
- Spesso si usano una lista di pagine, che l'utente può scorrere in avanti e indietro
- In questi casi è inutile farsi restituire tutte le righe dal database per usarne una piccola parte
- Google…

Limitare il numero di righe (2)

- È possibile specificare dei limiti al numero di righe nella ricerca
- `SELECT * FROM studenti LIMIT 15;`
- Restituisce le prime 15 righe della tabella studenti
- `SELECT * FROM studenti LIMIT 10,20;`
- Restituisce le prime 20 righe della tabella studenti, a partire dalla undicesima riga
  - La prima riga ha indice 0 (quindi quella di indice 10 è la undicesima)
  - Il primo numero è l'offset (indice di riga da cui partire)
  - Il secondo numero è il numero di righe da restituire

Ricerche e relazioni

- Abbiamo visto che quando ci sono delle relazioni, i dati sono suddivisi su più tabelle
- Come si fa a selezionare questi dati?
- La sintassi `JOIN ... ON` permette di unire più tabelle per effettuare delle query sull'unione
- Per esempio consideriamo queste due chiavi esterne della tabella esami:
  - La prima verso la tabella insegnamenti ()
  - La seconda verso la tabella studenti ()
- Vogliamo l'elenco di tutti gli studenti che hanno superato l'esame di AMM, con il relativo voto
  - Supponiamo che l'insegnamento AMM abbia id=19

Ricerche e relazioni (2)

```
SELECT studenti.cognome,
       studenti.nome,
       esami.voto
FROM studenti
JOIN esami ON studenti.id = esami.studente_id
WHERE esame.insegnamento_id = 19;
```

- Si seleziona una tabella su cui fare la ricerca delle due (o più) da unire
  - si preferisce la più piccola dal punto di vista delle righe
- Nella **JOIN** , si specifica quale sia l'altra tabella da unire
- Con **ON** , si specifica quali siano i valori da utilizzare per l'unione
- Questo crea una "tabella virtuale" unica su cui poi si esegue il filtro della **WHERE**
- Per distinguere tra i campi delle due tabelle si può far precedere il nome del campo dal nome della tabella, seguito dal punto

Ricerche e relazioni (3)

1

Studenti

id	nome	cognome	matricola
1	Davide	Spano	123456
2	Pinco	Pallino	654321

X

Esami

id	studente_id	Insegnamento_id	voto
1	1	19	24
2	2	19	27
3	1	23	30
4	1	25	18
5	2	23	30

Ricerche e relazioni (3)

2

studenti id	studenti nome	studenti cognome	studenti matricola	esami id	esami. studente_id	esami insegnamento_id	esami. voto
1	Davide	Spano	123456	1	1	19	24
2	Pinco	Pallino	654321	2	2	19	27
1	Davide	Spano	123456	3	1	23	30
1	Davide	Spano	123456	4	1	25	18
2	Pinco	Pallino	654321	5	2	23	30

Ricerche e relazioni (4)

3

studenti id	studenti nome	studenti cognome	studenti matricola	esami id	esami. studente_id	esami insegnamento_id	esami. voto
1	Davide	Spano	123456	1	1	19	24
2	Pinco	Pallino	654321	2	2	19	27

4

studenti nome	studenti cognome	esami. voto
Davide	Spano	24
Pinco	Pallino	27

Ricerche e relazioni (5)

- È possibile fare la join con più di una tabella
- Selezioniamo per ogni esame il nome dell'insegnamento, nome e cognome dello studente ed il voto
-
```
SELECT insegnamento.nome,
       studente.cognome,
       studente.nome,
       esami.voto
FROM esami
JOIN insegnamento ON esami.insegnamento_id =
insegnamento.id
JOIN studenti ON studenti.id =
esami.studente_id
```

Riferimenti

- Robin Nixon *Learning PHP, MySQL, JavaScript and CSS* O'Reilly,
  - Cap. 8
  - Cap. 9
  - Cap. 10
- Ereditarietà nei modelli ER
  - http://www2.mokabyte.it/cms/article.run?articleId=F2J-573-HF7-3L8_7f000001_10911033_851e831c