

# Corso Web MVC MySQL

Emanuele Galli

[www.linkedin.com/in/egalli/](http://www.linkedin.com/in/egalli/)

# Database Management System

Applicativo che ci permette di usare regole per dati

- Principali DBMS Relazionali

- Oracle, MySQL, SQL Server, PostgreSQL, DB2

database proprietario di microsoft

alternativa libera ad oracle è molto simile  
piace molto in ambiente scientifico

Database di IBM

versione 8?

- NoSQL

basato su documento document bas, ogni documento può essere usato a modo suo. non è struttura

- MongoDB (doc), ElasticSearch (doc), Redis (k-v)

Chiave valore in pratica è una mappa, il vantaggio è che è velocissimo Odi1 basta avere la chiave

## MySQL

sezione commerciale a pagamento

<https://www.mysql.com/downloads/>

<https://dev.mysql.com/downloads/>

sezione libera con tool gratuiti dev per development/developer

<https://dev.mysql.com/downloads/installer/>

<https://dev.mysql.com/doc/>

Programmi che ci permettono di creare i nostri dati  
mi salva i dati su disco fisso e mi gestisce egli stesso i dati.  
al cuore del database relazionale c'è una tabella (un insieme di righe(record)  
i vari dati del mio elemento sarebbero le sue caratteristiche". La riga mi dovrebbe spiegare bene come è fatto il mio soggetto  
ma colonna mi riporta.  
Mi permette di gestire tutti i miei elementi,

# Alcuni IDE per MySQL

- Quest Toad Edge uno per ogni database ma è a pagamento
- MySQL Workbench
- Database Development per Eclipse  
ci sono dei tool per integrare eclipse
  - Help, Install New Software, Work with (...) → Database Development
- DBeaver (standalone o plugin per Eclipse)  
Versione di eclipse fatta diventare per l'utilizzo solo di database
- Accesso CLI (mysql.exe nella directory MySQL server bin)  
command line interface

```
mysql -u root -p
```

# Database Relazionale

- Colonna: un singolo tipo di dato (campo) memorizzato in una tabella
- Riga (o record): collezione di dati (colonne) che descrivono completamente un'entità
- Tabella: insieme di righe in memoria volatile (result set) o persistente
- Tabelle memorizzate in uno **schema** del database, associato ad un **utente**  
Chiave che sta fuori e la foreign key ha lo stesso numero della primary key se appartiene a quella categoria: relazione MANY TO ONE
- Relazioni tra tabelle: **primary key** (PK) → **foreign key** (FK)
- PK: identifica univocamente (naturale o surrogata) una riga nella tabella corrente (normalmente singola colonna)  
LA chiave surrogata è sicura se è unica è unica  
si preferiscono perchè sono più robuste
- FK: identifica univocamente una riga in un'altra tabella  
identifica una relazione che non deve essere per forza univoca, può essere anche duplicata
- Un utente può avere il permesso di accedere tabelle di altri schemi
- **SQL** è il linguaggio standard per l'accesso a database relazionali

MODO CON CUI  
IDENTIFICHIAMO IN  
MANIERA UNIVOCAMENTE  
UNA RIGA all'interno DELLA  
TABELLA  
non identifichiamo qualcuno  
con il suo nome ma con il suo  
id,

# Relazioni tra tabelle

- **One to many / many to one**
  - Uno stato (PK) → molte città (FK duplicata)
- **Many to many** (implementato via tabella intermedia)
  - Uno stato → molte organizzazioni
  - Una organizzazione → molti stati
- **One to one**
  - Uno stato (PK) → una capitale (FK unique)

database, fare in modo che con tutte le relazioni fra tabelle, mi impedisce di eliminare un dipartimento se esso ha dei lavoratori associati.

È compito del DBMS mantenere l'integrità referenziale

# SQL

- DQL – Data Query Language
  - SELECT
- DML – Data Manipulation Language
  - INSERT, UPDATE, DELETE
- DDL – Data Definition Language
  - CREATE, ALTER, DROP, RENAME, TRUNCATE
- TC – Transaction Control Transazione
  - COMMIT, ROLLBACK, SAVEPOINT
- DCL – Data Control Language DBA database administrator  
per dare e togliere i permessi di modificare le tabelle
  - GRANT, REVOKE assegnano privilegi agli utenti nel database

Le keyword SQL sono  
*case insensitive*

è uguale sia se si usa il minuscolo sia il maiuscolo, se si vuole far presente  
che è una key word allora si mette grande

**select = SELECT**

# Amministrazione del DBMS

Creazione utente (con password delimitata da apici e case sensitive) e database via CLI - root

```
create user me identified by 'password';
```

```
create database me; -- database è lo schema in cui sono definiti gli oggetti MySQL creare un database
```

```
grant all privileges on me.* to me; -- tutti i privilegi standard sul database me all'utente me
```

```
grant alter routine on me.* to me; -- privilegi per modificare le procedure
```

```
-- drop me@localhost -- eliminazione di un utente sull'istanza locale di MySQL  
user  
se lo usi generico local host non serve non lo riconosco
```

Gestione dei database

```
show databases; -- tutti i database disponibili all'utente corrente
```

```
use me; -- selezione del database correntemente in uso
```

Esecuzione di uno script

```
source migration.sql
```

# Principali tipi di dato

Principali tipi di dati che posso usare in MySQL

DECIMAL(numero di cifre che intendo utilizzareprecision, numero di cifre dopo la virgola ( es. Decimal (4,2)-> 12.34 quattro cifre due dopo la virgolascale) decimale, un numero che è espresso una colonna è un numero e gli devo

INTEGER, INT è un intero possiamo usare entrambi perchè sono equivalenti

FLOAT, DOUBLE 32/64 bit

CHAR(length) è un array di caratteri ha dimensione fissa potrei usarlo per scrivere il nome di una persona array di caratteri. Le parentesi tonde danno un elemento in più dell'array. la lunghezza sarà sempre la stessa. Se ne ho 3 e ne metto uno solo gli altri sono spazio spazio

VARCHAR(length) stringa con una dimensione massima è quello che ci ho messo io è il limite massimo posso riempirlo fino a lì

DATE gestire le date

TIMESTAMP Mi rappresenta un momento specifico

In MySQL il confronto tra  
stringhe è per default  
*case insensitive*



# SELECT

- Selezione di dati (colonne) da una tabella, filtrata per colonne e righe

```
select region_name from regions where region_id = 1;
```

Poichè è una primary key non può essercene uno solo  
se ha lo stesso nome della tabella dovrebbe essere la primary key

- Selezione dei soli valori unici

```
select distinct manager_id from employees;
```

- Modifica i risultati in lettura da tabella

SE NON VOGLIO MODIFICARE i dati ma è solo in lettura, magari voglio solo vedere come sarebbero. Questa query qua non fa modifiche alla tabella

```
select job_title, min_salary, min_salary + 2000, min_salary * 3 + 1000 from jobs;
```

- Alias di colonna, introdotto da AS (opzionale) e delimitato da apici (singoli o doppi)

```
select job_title, min_salary as original, min_salary salary from jobs;
```

```
select job_title, min_salary + 2000 "increased min salary" from jobs;
```

- La tabella DUAL (implicita e fittizia)

```
select 1+2, 3-4, 2*6, 5/2, current_date -- from dual;
```

- Concatenazione

```
select concat(country_id, "...", region_id, '!') from countries;
```

# Informazioni su tabelle e utenti

- Tabelle

`show tables; -- del database corrente`

`select table_name from information_schema.tables; -- generale`

`select * from information_schema.tables where table_schema='me';`

- Descrizione di una tabella

`describe countries;`

`select * from information_schema.columns c where c.table_schema='me' and c.table_name = 'countries';`

- Descrizione degli utenti

`select * from mysql.user;`

# NULL

- Valore non presente o non valido, check esplicito con “is null”

in sql è difficile fare a meno del null perchè indica l'assenza di informazione  
vuol dire due cose: se non c'è o se il valore non è valido

Operatore specifico per fare un check sul null

```
select first_name, last_name  
from employees  
where commission_pct is null;
```

- “Assorbe” altri operandi

commissione annuale per impiegato null\*12=null . Quindi null assorbe gli altri operatori

```
select first_name, last_name, 12 * salary * commission_pct from employees;
```

questo è solo per il prodotto perchè sto modificando il salario per 12

- La funzione IFNULL() permette di decidere il comportamento

```
select first_name, last_name, 12 * salary * ifnull(commission_pct, 0)  
from employees;
```

# Operatori di confronto

=, !=, <, >, <=, >=

Colonne specificate subito dopo la select.

dalla tabella regione dove region id= 1 mi selezionerò tutto quanto della regione di \

select \* from regions where region\_id = 1;

Tutte quelle che non sono la regione 2

select \* from regions where region\_id != 2;

regione 1 e 2

select \* from regions where region\_id < 3;

Con il tre compreso

select \* from regions where region\_id <= 3;

Seleziona tutte le colonne

# Operatori SQL

LIKE, BETWEEN, IN, IS NULL. Per negare il loro risultato: **NOT**

funziona come il punto esclamativo in java e serve per negare il risultato

- **LIKE** wildcard: `_ %` è come "\_ul%" non è esattamente uguale ma assomiglia alla stringa

```
select last_name from employees where last_name like '_ul%';
```

- **BETWEEN**

```
select * from regions where region_id between 2 and 3;
```

tutte le righe che hanno una region id comprese fra due e tre.  
intervallo  
Con le parentesi sono entrambi compresi

```
select * from countries where country_name between 'a' and 'c';
```

Qui c è compreso se ci fosse uno stato con solo c perchè c è compreso ma solo C.

- **IN**

```
select * from regions where region_id not in (2, 3);
```

c'è il null che non fa funzionare nulla se non uso l'operatore is null non va bene. dobbiamo stare attenti che non ci sia dentro un null

```
select * from regions where region_id not in (2, 3, null); -- !! NOT IN(..., NULL) → FALSE !!
```

- **IS NULL**

```
select * from employees where manager_id is null;
```

In MySQL il confronto tra stringhe è per default *case insensitive*

Se io lo voglio far diventare case sensitive uso LIKE BINARY

# Operatori logici

- **AND**

```
select * from employees  
where salary < 3000 and employee_id > 195;
```

- **OR** (disgiunzione inclusiva)

```
select * from employees  
where salary > 20000 or last_name = 'King';
```

- **NOT**

```
select * from employees  
where not department_id > 20;
```

# Ordinamento via ORDER BY

- ORDER BY segue FROM – WHERE

```
select * from employees
```

```
order by last_name;
```

- ASC (ascending, default) / DESC (descending)

```
select * from employees
```

```
order by last_name desc, first_name asc;
```

ordinarlo in modo ascendente o discendente  
sotto vanno ordinati per nome quello che  
metto per primo è quello che comanda  
Se metto il numero e le colonne specificate  
da select sono due sul select posso

- notazione posizionale

```
select first_name, last_name from employees
```

```
order by 2;
```

# Esercizi

- Employees
  - Tutti i nomi, cognomi, email, telefoni, date di assunzione, ordinati per cognome e nome
  - Chi ha nome David o Peter
  - Chi appartiene al dipartimento 60. Chi appartiene ai dipartimenti 30, 50
  - Chi ha salario
    - maggiore di 10000
    - minore di 4000 o maggiore di 15000
    - minore di 4000 o maggiore di 15000, ma solo per i dipartimenti 50 e 80



# Esercizi

- Employees
  - Chi è stato assunto nel 2005
  - Quali job\_id sono presenti, in ordine naturale
  - Chi ha una commissione
  - Chi ha una 'a' nel nome o cognome
- Departments
  - Nomi, in ordine naturale
- Locations
  - Indirizzi delle sedi italiane

# JOIN

- Selezione di dati provenienti da due tabelle
- **INNER JOIN** – viene creata una riga nel risultato per ogni regola di join soddisfatta se non viene soddisfatta verrebbe buttata via la riga
- **OUTER JOIN** – se la regola non è soddisfatta, si preservano comunque i dati di una tabella di partenza Lo metto comunque dentro anche se non soddisfa tutti i campi
- **self JOIN** – left e right nella JOIN sono la stessa tabella particolare-> faccio una join della stessa tabella se ho messo in employees ho messo troppe informazioni
- **non-equi JOIN** – usano operatori diversi da “=” di solito facciamo una join con un'uguaglianza sono quelle che usano forme diverse di equijoin che è la self join.

# INNER JOIN

- Selezione dati correlati su diverse tabelle

se non avessi i join dovrei fare due join diversi

```
select region_name from regions where region_id = 1;
```

```
select country_name from countries where region_id = 1;
```

```
-- region_id = 1 .. 4
```

- Equi-join “classica” sulla relazione PK → FK è il metodo vecchio che non è più utilizzato

fammi vedere la regione e la naz

```
select region_name, country_name
```

```
from regions, countries
```

metti insieme riga per riga

```
where regions.region_id = countries.region_id;
```

# Alias per tabelle

- Si possono definire nel FROM alias per tabelle validi solo per la query corrente

```
select r.region_name, c.country_name
    per fare questo alias non si mette as ma:
from regions r, countries c
where r.region_id = c.region_id;
```

# JOIN – USING vs NATURAL JOIN

- INNER JOIN standard SQL/92

```
select region_name, country_name
```

```
from regions join countries -- join è “inner” per default
```

```
using(region_id);
```

chiamano region\_id sia il primary key di regions  
country la avrà come foreign key

- Se la relazione è “naturale” → NATURAL JOIN

```
select region_name, country_name
```

```
from regions natural join countries;
```

guarda tu quale è la relazione naturale tra region e country

# JOIN – ON

- NATURAL JOIN e JOIN – USING implicano una relazione equi-join per PK e FK con lo stesso nome
- JOIN – ON ci permette una maggior libertà

```
select region_name, country_name
```

```
from regions join countries
```

```
on(regions.region_id = countries.region_id);
```

specifichiamo anche come è fatta  
la relazione è anche utile se la  
relazione non è di equijoin

# <sup>inner</sup> JOIN – WHERE

- JOIN – ON

```
select region_name, country_name
from regions r join countries c
on(r.region_id = c.region_id)
where r.region_id = 1;
```

- JOIN – USING



```
select region_name, country_name
from regions join countries
using(region_id)
where region_id = 1;
```

- NATURAL JOIN

```
select region_name, country_name
from regions natural join countries
where region_id = 1;
```

- query classica equivalente

```
select region_name, country_name
from regions r, countries c
where r.region_id = c.region_id
and r.region_id = 1;
```

# Prodotto Cartesiano

- Se manca la condizione in una JOIN, ogni riga della prima tabella viene abbinata con tutte le righe della seconda

```
select region_name, country_name  
from regions, countries;
```

- SQL/92 CROSS JOIN, richiede che sia esplicito

```
select region_name, country_name  
from regions cross join countries;
```

è il prodotto cartesiano se si dimenticano di mettere on per fare un prodotto cartesiano

- **Ma** MySQL interpreta JOIN senza ON o USING come CROSS



# Self JOIN

- La FK si riferisce alla PK della stessa tabella

è una query voglio leggere il cognome degli impiegati e il cognome del manager

```
select e.last_name as employee, m.last_name as manager
```

as è per far stampare sopra employee e manager alias ma a livello di intestazione

```
from employees e join employees m
```

```
on (e.manager_id = m.employee_id);
```

Come metto in relazione due righe di queste tabelle il manager id deve essere uguale all'employee id

- Versione “classica”

```
select e.last_name as employee, m.last_name as manager
```

```
from employees e, employees m
```

```
where e.manager_id = m.employee_id;
```

# JOIN su più tabelle

dobbiamo trovare una colonna che integri le due tabelle

- JOIN – ha solo una tabella left e una right → 2 JOIN per 3 tabelle

```
select employee_id, city, department_name
from employees join departments using(department_id)
join locations using(location_id);
```

e location non hanno una relazione primary key e foreign key quindi bisogna fare così  
qui non servono gli alias

- Versione “classica” → 2 condizioni nel WHERE per 3 tabelle

```
select employee_id, city, department_name
from employees e, departments d, locations l
where d.department_id = e.department_id and d.location_id = l.location_id;
```

# Non-equi JOIN

- JOIN basate su operatori diversi da "=", poco usate

dalla tabella empl. prendo il nome e il salario minimo e quello corrente

```
select e.last_name, e.salary, j.min_salary
```

c'è job id che collega employee a jobs job id è una foreign key

```
from employees e join jobs j
```

lo voglio fare tra i salary (non sto usando una key per fare il confronto ma una join) tra il minimo salario e il minimo salario +100

```
on(e.salary between j.min_salary and j.min_salary + 100)
```

```
where(e.job_id = j.job_id);
```

solo quelli dove il job id è uguale da entrambe le parti

- Versione "classica"

```
select e.last_name, e.salary, j.min_salary
```

```
from employees e, jobs j
```

```
where e.salary between j.min_salary and j.min_salary + 100
```

```
and e.job_id = j.job_id;
```

# LEFT OUTER JOIN

- Genera un risultato anche se la FK nella tabella left alla tabella right è NULL. I valori non disponibili relativi alla tabella right sono messi a NULL.

```
select first_name, department_name
```

```
from employees supporta anche il null sul department_id left outer join departments
```

```
using(department_id) come chiave relazione viene fatta su department_id  
se non posso metterle in relazione xk c'è un null con il join normale non avrebbe stampato la relazione che in una delle due colonne ha il null
```

```
where last_name = 'Grant';
```

# RIGHT OUTER JOIN

- Genera un risultato per le righe nella tabella right anche se non c'è corrispondenza con righe nella tabella left

La relazione PK FK è asimmetrica in employees abbiamo la foreign key che mi dice

`select first_name, last_name, department_name`

la query è uguale ma il risultato cambia i department name vengono stampati tutti anche se non ci sono persone che lavorano

per quel dipartimento

`from employees right outer join departments`

`using(department_id)`

`where department_id between 110 and 120;`

# Esercizi

- Nome degli employees e del loro department
- Nome degli employees e job title (da JOBS)
- Nome degli employees che hanno il salario minimo o massimo previsto per il loro job title
- Nome degli employees basati in UK (LOCATIONS)
- Nome dei departments e manager associato

# Esercizi /2

- Nome di ogni department e, se esiste, del relativo manager
- Nome dei department che non hanno un manager associato
- Nome degli employees e del loro manager

# Funzioni su riga singola ci danno un risultato solo

(LIBERE)

- Operano su e ritornano una singola riga
  - Caratteri e stringhe
  - Numeri
  - Date
  - Espressioni regolari
  - Conversione: **CAST()** non è l'unica
    - `select cast(12345.67 as char), cast('2019-05-01' as date);`  
lo fa automaticamente trasformami questa stringa in un date  
ci permette di trasformare dati in qualcosa di diverso



# Alcune funzioni su stringhe

- **ASCII()**: codice ASCII di un carattere, **CONVERT()** + **CHR()**: da codice ASCII a carattere  
`select ascii('A') as A, convert(char(90) using utf8) as '90';`
- **CONCAT()**: concatenazione di stringhe  
`select concat(first_name, ' ', last_name) from employees;`
- **UPPER()**: tutto maiuscolo, **LOWER()**: tutto minuscolo  
`select upper('upper') up, lower('LOWER') low;`
- **POSITION()**, **LOCATE()**: sub, target [, start] → [1..n], 0 not found  
`select position('ba' in 'crab' ) as "not found", position('ra' in 'crab' ) as pos;`  
`select locate('ab', 'crab abba rabid cab', 13) as pos;`
- **LENGTH()**: per string e numeri, convertiti implicitamente in stringhe  
`select length('name'), length(42000);` mi dice la lunghezza della stringa solo dopo averlo trasformato può contare le cifre, ma con - e . viene contato nella lunghezza

i numeri vengono convertiti automaticamente in stringhe

# Alcune funzioni su stringhe /2

principalmente per stampare a linea di comando

- **LPAD()**, **RPAD()**: padding. Stringa → dimensione, con eventuale pad specificato  
gli diremo come mettere in colonna i vari dati l a sx r a dx  
`select lpad('tom', 30, '.') tom, rpad('tim', 30, '_- _') tim;`
- **LTRIM()**, **RTRIM()**, **TRIM()**: rimozione di caratteri dall'input  
reim per togliere il padding a dx sx o dx e sx  
questo è l'alias  
`select ltrim(' Hi!') "left", concat('[', rtrim('Hi! '), ']') "right", concat('[', trim(' Hi! '), ']') "both";`  
`select trim(leading 'xy' from 'xy!xy') "left", trim(trailing 'xy' from 'xy!xy') "right", trim(both 'xy' from 'xy!xy') "both";`
- **RIGHT()**: estrae da una stringa n caratteri a destra  
`select right('discardedXYZ', 3);`
- **REPLACE()**: sostituzione di substring, **SUBSTR()**: estrazione di substring  
`select replace('Begin here', 'Begin', 'End'), substr('ABCDEFGFG', 3, 4);`  
vai sulla posizione 3 e prendimi 4 caratteri

# Alcune funzioni numeriche

- **ABS()**: valore assoluto
- **CEIL()**: 'soffitto', **FLOOR()**: 'pavimento'  
5,7 voglio ottenere 6 e con floor 5
- **MOD()**: modulo, resto di divisione intera
- **POWER()**: potenza; **EXP()**:  $e^x$ ; **SQRT()**: radice 2; **LN()**, **LOG()**: logaritmi
- **ROUND()**, **TRUNCATE()**: arrotonda/tronca a decimali (-) o potenze di 10 (-)
- **SIGN()**: -1, 0, 1 per numeri negativi, zero, positivi
- **PI()**: pi greco
- **SIN()**, **COS()**, **TAN()**<sup>tangenti</sup>,...: funzioni trigonometriche

# Alcune funzioni su date

- **CURDATE()**, **NOW()**: data, data e time corrente
- **DAYNAME()**, **MONTHNAME()**: Nome del giorno e nome del mese nome del giorno o del mese
- **DATE\_FORMAT()**, **STR\_TO\_DATE()**: conversione tra data e stringa
- **DATE\_ADD**(date, INTERVAL expr unit), **DATE\_SUB**(): data +/- intervallo  
date\_add(curdate(), interval 1 day)
- **EXTRACT**(unit FROM date): estrae parte della data(-time)  
select extract(year from now());
- **DATEDIFF**(): giorni di distanza tra due date(-time)
- **LAST\_DAY**(date): ultimo giorno del mese

```
set lc_time_names = 'it_IT';  
ma str_to_date()  
usa sempre 'en_US'  
per settare in una certa lingua per local date
```

# Espressioni regolari

- **REGEXP\_LIKE()** versione estesa di LIKE

- Es: cognomi che iniziano per A o E:

```
select last_name
```

```
from employees
```

```
where regexp_like(last_name, '^[ae].*');
```

che hanno il cognome secondo questo pattern  
troverai un carattere a scelta tra a ed e e poi un carattere  
qualsiasi (.) lunghezza quanta ne vuoi

# Altre funzioni

- **VERSION()**
  - versione di MySQL in esecuzione
- **USER()**
  - utente connesso
- **DATABASE()**
  - il database corrente

# Esercizi

- Employees
  - Qual è il salario corrente, quale sarebbe con un incremento dell'8.5%, qual è il delta come valore assoluto
  - Quanti giorni sono passati dall'assunzione a oggi
  - Quant'è la commissione di ognuno o 'no value'

# Funzioni aggregate

- Ignorano i NULL
- Uso di DISTINCT per filtrare duplicati
- **AVG()**: media
- **COUNT()**: numero di righe
- **MAX()**: valore massimo
- **MIN()**: minimo
- **SUM()**: somma
- **STDDEV()**: deviazione standard
- **VARIANCE()**: varianza



# Raggruppamento via GROUP BY

- Divide il risultato della select in gruppi
- È possibile applicare funzioni aggregate sui gruppi  
select department\_id, truncate(avg(salary), 0)  
from employees  
group by department\_id  
order by 1;

# GROUP BY – HAVING

- HAVING filtra i risultati di GROUP BY
- È possibile filtrare prima le righe della SELECT con WHERE, e poi il risultato della GROUP BY con HAVING

```
select manager_id, round(avg(salary))
```

```
from employees
```

```
where salary < 8000
```

```
group by manager_id
```

```
having avg(salary) > 6000
```

```
order by 2 desc;
```

# Subquery

- In WHERE:

```
select first_name, last_name from employees
where employee_id = (select manager_id from employees where last_name = 'Chen');
```

- In FROM (inline view):

```
select max(e.salary)
from (select employee_id, salary from employees where employee_id between 112 and 115) e;
```

- In HAVING:

```
select department_id, round(avg(salary)) from employees group by department_id
having avg(salary) < (select max(x.sal) from
(select avg(salary) sal from employees group by department_id) x)
order by 2 desc;
```

# JOIN con subquery

- Subquery genera una tabella temporanea → join  
select region\_name, c.country\_count  
from regions natural join (  
select region\_id, count(\*) country\_count  
from countries  
group by region\_id) c;

# subquery multirighe in WHERE

- Uso dell'operatore IN

es: nome di EMPLOYEES che sono manager

```
select first_name, last_name from employees
```

```
where employee_id in (
```

```
    select distinct manager_id
```

```
    from employees where manager_id is not null)
```

```
order by 2;
```

# Esercizi

- Employees
  - Salary: maggiore, minore, somma, media
    - Come sopra, ma per ogni job\_id
  - Quanti dipendenti per ogni job\_id
    - Quanti sono gli IT\_PROG
  - Quanti sono i manager
  - Nome dei dipendenti che non sono manager
  - Qual è la differenza tra il salario maggiore e il minore
    - Come sopra, ma per ogni job\_id, non considerando dove non c'è differenza
  - Qual è il salario minimo con i dipendenti raggruppati per manager, non considerare chi non ha manager, né i gruppi con salario minimo inferiore a 6.000€

# Esercizi /2

- Indirizzi completi, tra locations e countries
- Employees
  - Nome di tutti i dipendenti e nome del loro department
    - Come sopra, ma solo per chi è basato a Toronto
  - Chi è stato assunto dopo David Lee
  - Chi è stato assunto prima del proprio manager
  - Chi ha lo stesso manager di Lisa Ozer
  - Chi lavora in un department in cui c'è almeno un employee con una 'u' nel cognome
  - Chi lavora nel department Shipping
  - Chi ha come manager Steven King

# INSERT

```
INSERT INTO table (columns...) VALUES (values...);
```

```
insert into regions(region_id, region_name)  
values (11, 'Antarctica');
```

- I valori NULLABLE, se NULL, sono impliciti  
insert into regions(region\_id) values (12);
- Il nome delle colonne è opzionale (cfr. DESCRIBE)  
insert into regions values (13, null);



# UPDATE (WHERE!)

UPDATE table

SET column = value

[WHERE condition];

update regions

set region\_name = concat('Region ', region\_id)

where region\_id > 10;

# DELETE (WHERE!)

```
DELETE FROM table [WHERE condition];
```

```
delete from regions  
where region_id > 10;
```

# Transazioni

- Inizio: prima istruzione DML (INSERT, UPDATE, DELETE) in assoluto, o dopo la chiusura di una precedente transazione
- Fine: COMMIT, ROLLBACK, istruzione DDL, DCL, EXIT (implicano COMMIT o ROLLBACK in caso di failure)
- Buona norma: COMMIT o ROLLBACK esplicite
  - Eclipse Database Development: Window, Preferences, Data Management, SQL Development, SQL Editor, SQL Files / Scrapbooks, Connection Commit Mode → Manual
  - MySQL Workbench Query → Auto-Commit Transactions

# COMMIT, ROLLBACK, SAVEPOINT

SAVEPOINT: punto intermedio in una transazione

```
insert into regions(region_id, region_name) values (11, 'Antarctica');  
savepoint sp;
```

```
insert into regions(region_id, region_name) values (12, 'Oceania');
```

```
rollback to sp; -- keep Antarctica, rollback Oceania
```

```
commit; -- persist Antarctica
```

# Livelli di isolamento nelle transazioni

- Transazioni concorrenti possono causare problemi in lettura:
  - **Phantom read**: T1 SELECT su più righe; T2 INSERT o DELETE nello stesso intervallo; T1 riesegue la stessa SELECT, nota un fantasma (apparso o scomparso) nel risultato
  - **Non repeatable read**: T1 SELECT, T2 UPDATE, T1 SELECT non ripetibile
  - **Lost update**: T1 UPDATE, T2 UPDATE. Il primo update è perso
  - **Dirty read**: T1 UPDATE, T2 SELECT, T1 ROLLBACK, valore per T2 è invalido
- Garanzie fornite da DBMS
  - READ UNCOMMITTED**: tutti comportamenti leciti
  - READ COMMITTED**: impedisce solo dirty read
  - REPEATABLE READ**: phantom read permesse ← default MySQL
  - SERIALIZABLE**: nessuno dei problemi indicati ← default SQL

# CREATE TABLE (on ME)

- Nome tabella, nome e tipo colonne, constraint, ...

```
create table items (  
    item_id integer primary key,  
    status char,  
    name varchar(20),  
    coder_id integer);
```

# CREATE TABLE AS SELECT

- Se si hanno i privilegi in lettura su una tabella (GRANT SELECT ON ... TO ...) si possono copiare dati e tipo di ogni colonna

```
create table coders
```

```
as
```

```
select employee_id as coder_id, first_name, last_name, hire_date, salary  
from employees  
where department_id = 60;
```

# ALTER TABLE

- ADD / DROP COLUMN

```
alter table items add counter decimal(38, 0);
```

```
alter table items drop column counter;
```

- ADD CONSTRAINT CHECK / UNIQUE

```
alter table items add constraint items_status_ck check(status in ('A', 'B', 'X'));
```

```
alter table coders add constraint coders_name_uq unique(first_name,  
last_name);
```

- ADD CONSTRAINT PRIMARY KEY / senza o con AUTO\_INCREMENT

```
alter table coders add constraint primary key(coder_id);
```

```
alter table coders modify coder_id int primary key auto_increment;
```



# CREATE TABLE con CONSTRAINT

```
create table details (  
    detail_id integer primary key  
        constraint detail_id_ck check (mod(detail_id, 2) = 1),  
    status char default 'A'  
        constraint detail_status_ck check (status in ('A', 'B', 'X')),  
    -- alternativa: status enum('A', 'B', 'X') default 'A'  
    name varchar(20),  
        -- not null,  
        -- unique,  
    coder_id integer references coders(coder_id), -- on delete cascade / set null  
  
    constraint detail_name_status_uq unique(name, status)  
);
```

# TRUNCATE / DROP TABLE

MySQL Workbench ha “safe mode” che limita le funzionalità standard (Edit → Preferences → SQL Editor → Safe Updates)

- `delete from` table\_name; -- DML → rollback
- `truncate table` table\_name; -- no rollback!
- `drop table` table\_name; -- no rollback!

# INDEX

- Possono velocizzare l'accesso alle tabelle, riducendo gli accessi alla memoria di massa

- B-Tree by default

- indice semplice

- `create index coders_last_name_ix on coders(last_name);`

- indice composto

- `create index coders_name_ix on coders(first_name, last_name);`

- `drop index coders_last_name_ix on coders;`

# VIEW

- Query predefinita su una o più tabelle, acceduta come se fosse una tabella
- Semplifica e controlla l'accesso ai dati

```
create or replace view odd_coders_view as
```

```
select * from coders
```

```
where mod(coder_id, 2) = 1;
```

```
drop view odd_coders_view;
```

# Esercizi

- Coders
  - Inserire come assunti oggi:
    - 201, Maria Rossi, 5000€ e 202, Franco Bianchi, 4500€
  - Cambiare il nome da Maria a Mariangela
  - Aumentare di 500€ i salari minori di 6000€
  - Eliminare Franco Bianchi
  - Committare i cambiamenti

# Stored procedure

Funzionalità gestita dal DBMS, introdotte in MySQL dalla versione 5

**procedura**: accetta parametri (in/out)

**funzione**: procedura che ritorna un valore

**trigger**: procedura eseguita in seguito ad una operazione DML su una tabella

# La vita di una stored procedure

In quest'area si  
usano estensioni  
proprietarie MySQL

```
drop procedure if exists hello;  
  
delimiter //  
create procedure hello()  
begin  
    select "Hello!" as greetings;  
end;  
// delimiter ;  
  
call hello();
```

# Variabili

```
declare v_a varchar(20);  
declare v_b int default 42;  
  
set v_a = "hello";  
  
select concat(v_a, ": ", v_b) as greetings;
```



# Condizioni

```
if v_a > 0 then
    set v_b = 'v_a is positive';
elseif v_a = 0 then
    set v_b = 'v_a is zero';
else
    set v_b = 'v_a is negative';
end if;
```

```
case v_a
    when -1 then
        set v_c = 'v_a is minus one';
    when 0 then
        set v_c = 'v_a is zero';
    when 1 then
        set v_c = 'v_a is plus one';
    else
        set v_c = 'v_a is unknown';
end case;
```

# Loop

```
my_loop : loop
    set loop_message = concat(loop_message, ' ', v_i);
    set v_i = v_i + 1;
    if v_i > 6 then
        leave my_loop;
    end if;
end loop my_loop;
```

```
while v_i < 7 do
    set while_message = concat(while_message, ' ', v_i);
    set v_i = v_i + 1;
end while;
```

```
repeat
    set repeat_message = concat(repeat_message, ' ', v_i);
    set v_i = v_i + 1;
until v_i > 6 end repeat;
```

# Esempio di procedura

```
delimiter //  
create procedure total_salaries_coders()  
begin  
    declare v_total decimal(8, 2);  
  
    select sum(salary) into v_total from coders;  
  
    if v_total > 0 then  
        select v_total as "total salary for coders";  
    else  
        select "no salary information available for coders!" as warning;  
    end if;  
end;  
// delimiter ;
```

# Cursor

```
declare cur_coders cursor for  
    select first_name, last_name from coders;  
declare continue handler for not found  
    set v_done = true;
```

definizione di  
cursore e terminatore

uso del  
cursore

```
open cur_coders;  
while not v_done do  
    fetch cur_coders into v_first_name, v_last_name;  
    -- ...  
end while;  
  
-- ...  
  
close cur_coders;
```

# Procedure con parametri

IN (default)

OUT

INOUT

```
create procedure get_coder_salary(  
    in p_coder_id integer,  
    out p_salary decimal(8, 2)  
) begin  
    select salary  
    into p_salary  
    from coders  
    where coder_id = p_coder_id;  
end;
```

```
call get_coder_salary(9104, @result);  
select @result;
```

user-defined variable  
estensione MySQL  
session scoped

# Function

Solo parametri 'in'

```
create function get_salary(  
    p_coder_id integer  
) returns decimal(8, 2)  
deterministic  
begin  
    declare v_result decimal(8, 2);  
  
    -- ...  
  
    return v_result;  
end;
```

Return type

```
select get_salary(104) as salary;
```

# TRIGGER

- Introdotto in MySQL 5
- Procedura eseguita automaticamente prima o dopo un comando DML
- Row-level
  - Eseguito per ogni riga coinvolta
  - Accesso a stato precedente e successivo via OLD e NEW

# Un esempio di trigger

```
create trigger before_update_salary  
  before update on coders  
  for each row  
begin  
  set new.salary = round(new.salary, -1);  
end;
```

Generazione di eventi che scatenano il trigger



```
update coders  
set salary = salary + 3;
```



# Esercizi

- Scrivere e invocare la procedura tomorrow() che stampa la data di domani
- Modificare tomorrow() per fargli accettare come parametro un nome da stampare
- Scrivere e invocare la procedura get\_coder() che ritorna nome e cognome di un coder identificato via id