

SQL

- DBMS
- MySQL
- SQL
- Stored Procedures
- Esempi:
 - <https://github.com/egalli64/mpjp> mySql

Database Management System

- Principali DBMS Relazionali
 - Oracle, MySQL, SQL Server, PostgreSQL, DB2
- NoSQL
 - MongoDB (doc), ElasticSearch (doc), Redis (k-v)

un applicativo che permette di usare regole per gestire i dati, relazionali o no (noSQL). MySQL appartiene a Oracle; SQL Server è il database proprietario di Microsoft; Pos è nato come alternativa libera a Oracle; DB2 è il database di IBM.

Mongo si basa non su tabelle ma su documenti, è quindi non troppo rigido; Redis riceve una chiave e ci ritorna un valore.

MySQL

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

<https://dev.mysql.com/downloads/> dev: sezione libera (development/developer)

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



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

troviamo tutta la documentazione

Alcuni IDE per MySQL

- Quest Toad Edge
- MySQL Workbench
- Database Development per Eclipse
 - Help, Install New Software, Work with (...) → Database Development
- DBeaver (standalone o plugin per Eclipse)
- Accesso CLI (mysql.exe nella directory MySQL server bin)

il vantaggio di usare i tools so Eclipse è il poter lavorare direttamente su Eclipse senza aprire troppe finestre.

`mysql -u root -p` - si scrive quando non voglio inserire qualcosa, ad esempio no user e no password

`"C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql" -u root -p`

database: programma che ci permette di gestire i nostri dati, salvandoli su memoria di massa come fa il file system. Cuore del database relazionale è la tabella, vista come un insieme di righe o un insieme di colonne. La riga spiega interamente com'è fatto un oggetto, mentre la colonna riporta un singolo tipo di dato della riga. La tabella quindi mi permette di gestire in blocco tutto ciò mettendolo su RAM (in esecuzione) o disco fisso. In questo caso la gestione avviene tramite relazioni.

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**
- Relazioni tra tabelle: **primary key** (PK) → **foreign key** (FK)
- PK: identifica univocamente (naturale o surrogata) una riga nella tabella corrente (normalmente singola colonna)
- FK: identifica univocamente una riga in un'altra tabella
- Un utente può avere il permesso di accedere tabelle di altri schemi
- **SQL** è il linguaggio standard per l'accesso a database relazionali

schema: equivale a package, mi permette di memorizzare le mie tabelle ed è collegato ad un utente.

la FK può anche non essere univoca perché può fare riferimento a più righe

Relazioni tra tabelle

- **One to many / many to one** il pollo e il salmone che appartengono al reparto gastronomia.
 - Uno stato (PK) → molte città (FK duplicata)
- **Many to many** (implementato via tabella intermedia) ho un pc che non usa solo un impiegato, quando non ci sono lo usa un'altra persona.
 - Uno stato → molte organizzazioni
 - Una organizzazione → molti stati
- **One to one** posto macchina singolo per un solo dipendente
 - Uno stato (PK) → una capitale (FK unique)

È compito del DBMS mantenere l'integrità referenziale

= impedire che succedano cose assurde, scombinando tutti i dati

SQL

- DQL – Data Query Language
 - SELECT seleziona dalle tabelle, mi permette di leggere
- DML – Data Manipulation Language
 - INSERT, UPDATE, DELETE
- DDL – Data Definition Language
 - CREATE, ALTER, DROP, RENAME, TRUNCATE
- TC – Transaction Control
 - COMMIT, ROLLBACK, SAVEPOINT
- DCL – Data Control Language
 - GRANT, REVOKE

Le keyword SQL sono
case insensitive

select = SELECT

Amministrazione del DBMS

Creazione utente e database via CLI - root

- `create user me identified by 'password';` -- password delimitata da apici e case sensitive
- `create database me;` -- database è lo `schema` in cui sono definiti gli oggetti dove metto le mie tabelle
- `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` aggiungi "user" -- eliminazione di un utente sull'istanza locale di MySQL

i commenti su sql sono con --

Gestione dei database

local host: la macchina corrente (o 127.0.0.1). Ogni macchina ha la possibilità di lavorare su più porte

- `show databases;` -- tutti i database disponibili all'utente corrente
- `use me;` -- selezione del database correntemente in uso

Esecuzione di uno script (non funziona su MySQL Workbench, occorre invece aprire il file ed eseguirlo)

- `source migration.sql` clicchi nelle proprietà del file che vuoi fare migrare, ti fa vedere dove si trova, copi l'url in mySQL da "File-open SQL project"

Principali tipi di dato

DECIMAL(precision, scale)

INTEGER, INT

FLOAT, DOUBLE

CHAR(length)

VARCHAR(length)

DATE

TIMESTAMP

se voglio dire che la colonna è un numero devo dirgli precision e scale: il primo sta per il numero di cifre prima della virgola, il secondo i decimali.
CHAR è un array di caratteri, di dimensione fissa: se ci metto che la length deve essere 3 ci saranno sempre dentro 3 caratteri; se ce ne fosse solo uno disponibile, gli altri due vengono automaticamente messi bianchi.
VARCHAR è una stringa, limitata alla dimensione che scrivo tra le parentesi. In entrambe posso mettere sia char che caratteri. != da Java, la stringa è espressa con apici singoli.

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;` se la tabella si chiama regions e la mia key ha lo stesso nome, sarà quasi sicuramente la primary key
se non specifico il WHERE lui intende tutte le righe e tutte le colonne

- Selezione dei soli valori unici

`select distinct manager_id from employees;` distinct mi elimina i duplicati

- Modifica i risultati in lettura da 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) è un dato di sistema riconosciuto da MySQL, che crea una tabella fittizia per poterlo stampare; stessa cosa se voglio fare le operazioni. From dual si può anche omettere, tanto è l'unica tabella di questo tipo.

`select 1+2, 3-4, 2*6, 5/2, current_date -- from dual;` infatti se trovo uno statement in cui c'è solo Select ... vuol dire che si riferisce a 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”

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

- “Assorbe” altri operandi voglio sapere la commissione totale per impiegato, quindi chi ha null come commissione darà come risultato un null. Vedi anche sotto.

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

- La funzione IFNULL() permette di decidere il comportamento

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

see è diverso da null procedi con l'operazione. La funzione è stata creata perché SQL non è un linguaggio turing completo, quindi non ha if al suo interno.

Operatori di confronto

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

seleziona tutte le colonne dalla tabella regione, dove il region id è = a 1

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

```
select * from regions where region_id != 2;
```

```
select * from regions where region_id < 3;
```

```
select * from regions where region_id <= 3;
```

Operatori SQL

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

- **LIKE** wildcard: `_ %` seleziona da employees last name, dove last name è come (assomiglia a) la stringa che contenga "ul" ma prima deve avere un carattere e dopo può esserci qualsiasi cosa.
`select last_name from employees where last_name like '_ul%';`
_ cioè 1 solo carattere; % cioè 0,1,1000 caratteri che lo seguono.

- **BETWEEN**

`select * from regions where region_id between 2 and 3;` in java sarebbe stato (2,3]

`select * from countries where country_name between 'a' and 'c';`
se voglio che comprenda anche il Canada devo mettere tra a e d

- **IN**

`select * from regions where region_id not in (2, 3);` le righe che non hanno come id 2 e 3

`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*
cfr: **LIKE BINARY**

se aggiungo "Like binary" il confronto maiusc minusc diventa fondamentale

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;
```

vuol dire che deve essere minore o uguale, dato che non deve essere maggiore

Ordinamento via ORDER BY

- ORDER BY segue FROM – WHERE

select * from employees

order by last_name;

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

partendo dalla Z ad esempio

select * from employees

order by last_name desc, first_name asc;

- notazione posizionale

select first_name, last_name from employees

order by 2; 2 in questo caso è la seconda colonna che ci interessa, in questo caso last name

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

Voglio avere un risultato estraendo dati da tabelle che sono in relazione. Nella inner se ho un dato che non mi fa mettere insieme le due tabelle, la riga (e quindi il dato) sparisce. Outer join mi permette invece di preservare i dati di partenza.

- Selezione di dati provenienti da due tabelle
- INNER JOIN – viene creata una riga nel risultato per ogni regola di join soddisfatta
- OUTER JOIN – se la regola non è soddisfatta, si preservano comunque i dati di una tabella di partenza
- self JOIN – left e right nella JOIN sono la stessa tabella
- non-equi JOIN – usano operatori diversi da “=”

equijoin= join normale, primary key e foreign key sono legate da un'uguaglianza. (slide 19)

INNER JOIN

- Selezione dati correlati su diverse tabelle questo è il ragionamento che si deve seguire

```
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 codice vecchio

```
select region_name, country_name  
from regions, countries  
where regions.region_id = countries.region_id;
```

Alias per tabelle non si mette "as"

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

```
select region name che prendo da R r.region_name, country name che prendo da c c.country_name  
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);
```

torna meglio se la primary e la foreign key si chiamano allo stesso modo

- Se la relazione è “naturale” → NATURAL JOIN

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

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);
```

JOIN – WHERE

- **JOIN – ON** deve specificare la relazione tra le due tabelle

```
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** solo una cosa specificata tra parentesi

```
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

def: l'associazione di ogni elemento di un insieme con tutti gli altri del secondo insieme. Se non specifico a sql con che criterio voglio unire le tabelle, automaticamente fa un prodotto cartesiano. Ha senso se voglio elencare in quanti colori è disponibile un vestito ad esempio (gonna blu, gonna verde, gonna rossa. maglione giallo, maglione verde, ...

- 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;
```

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

Self JOIN

- La FK si riferisce alla PK della stessa tabella

voglio tutti i cognomi dei dipendenti e il cognome del loro manager associato. as è per fargli stampare sulla colonna "employee" e "manager"

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

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

i manager sono anche loro employees, è tutto in una tabella. qui faccio finta che le tabelle siano 2 e nell'on specifico cosa trovo in "e" e "m"

```
on (e.manager_id = m.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

- 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);
- 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

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

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

```
on(e.salary al posto dell = dentro l'on between j.min_salary and j.min_salary + 100)
```

metto il salario del dipendente tra il minimo e il minimo+100, ma solo se vale la relazione sotto, cioè che il job id è lo stesso

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

- 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 left outer join departments  
using(department_id)  
where last_name = 'Grant';
```

nonostante io abbia un null come foreign key (quindi ho solo la primary), stampo lo stesso quello che posso stampare. Il null è nell'esempio in tabella right, ovvero department name. Quella che "vince", che ha la priorità, è la tabella employees, perché in questa query l'ho lasciata a sinistra.

RIGHT OUTER JOIN

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

```
select first_name, last_name, department_name  
from employees right outer join departments  
using(department_id)  
where department_id between 110 and 120;
```

in sql gli estremi sono sempre inclusi

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

- Operano su e ritornano una singola riga
 - Caratteri e stringhe
 - Numeri
 - Date
 - Espressioni regolari
 - Conversione: **CAST()** permette di trasformare un tipo di dato in un altro tipo, glielo dico io cosa voglio con "as"
 - `select cast(12345.67 as char), cast('2019-05-01' as date);`

Alcune funzioni su stringhe

- **ASCII()**: codice ASCII di un carattere, **CONVERT()** + ^{CHAR}~~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 la posizione parte da 1, !=Java
select position('ba' in 'crab') as "not found", position('ra' in 'crab') as pos;
select locate('ab', 'crab abba rabid cab', 13) as pos; gli dico cosa cerco, dove lo cerco, da che posizione voglio partire
- **LENGTH()**: per string e numeri, convertiti implicitamente in stringhe
select length('name'), length(42000); conta anche i punti come caratteri

Alcune funzioni su stringhe /2

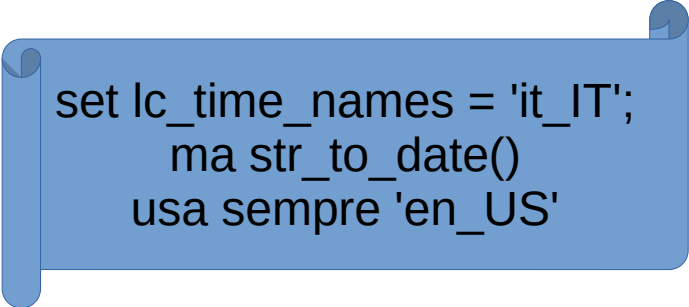
- **LPAD()**, **RPAD()**: padding. Stringa → dimensione, con eventuale pad specificato
left
right
mette 27 puntini
`select lpad('tom', 30, '.') tom, rpad('tim', 30, '_-_-') tim;` per riempire le stringhe
- **LTRIM()**, **RTRIM()**, **TRIM()**: rimozione di caratteri dall'input
right and left
`select 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('ABCDEFGH', 3, 4);`
sostituisci begin con End nella stringa vai sulla posizione 3 e prendi 4 caratteri

Alcune funzioni numeriche

- **ABS()**: valore assoluto
- **CEIL()**: 'soffitto', **FLOOR()**: 'pavimento' approssimare per eccesso o per difetto da un decimale
- **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()**,...: funzioni trigonometriche

Alcune funzioni su date

- `CURDATE()`, `NOW()`: data, data e time corrente
- `DAYNAME()`, `MONTHNAME()`: 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'
```

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].*');

tutti i cognomi che hanno questo pattern: il circonflesso indica l'inizio, il punto qualsiasi carattere dopo e l'asterisco quanti ne voglio

Altre funzioni

- **VERSION()** per essere sicuri della versione in cui sono
 - 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 quanto sto aggiungendo?
 - 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