# Corso Web MVC SQL Oracle

Emanuele Galli

www.linkedin.com/in/egalli/

#### Oracle Database

- Multi-model DBMS
  - Relazionale
    - MySQL, SQL Server, PostgreSQL, DB2
  - NoSQL
    - MongoDB (doc), ElasticSearch (doc), Redis (k-v)
- https://www.oracle.com/database/
- https://www.oracle.com/database/technologies/xe-downloads.html
- https://www.oracle.com/database/technologies/xe-prior-releases.html
- https://www.oracle.com/database/technologies/appdev/jdbc.html

#### Database Relazionale

- Collezione di informazioni correlate, organizzata in tabelle
- Dati memorizzati in righe e ordinati per colonne
- Tabelle memorizzate in uno schema del database, che viene associato ad un utente
- Relazioni tra tabelle: primary key (PK) → foreign key (FK)
- Un utente può avere il permesso di accedere tabelle di altri schemi
- SQL (Structured Query Language) è il linguaggio standard per l'accesso a database relazionali

#### Relazioni tra tabelle

- One to many / many to one
  - Un continente (PK) → molti stati (FK duplicata)
- Many to many (implementato via tabella intermedia)
  - Ogni stato → molte organizzazioni
  - Ogni organizzazione → molti stati
- One to one
  - Ogni stato (PK) → una capitale (FK unique)

## Structured Query Language (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
  - COMMIT, ROLLBACK, SAVEPOINT
- DCL Data Control Language
  - GRANT, REVOKE

## Attivazione HR via SQL Plus

- Connessione a Oracle, per utente e database
  - sqlplus user/password@host
    - sqlplus / as sysdba
- Da Oracle 12 in avanti va specificato il container (orclpdb, xepdb1, ...)
  - alter session set container = xepdb1;
  - Stato del database corrente
    - select name, open mode from v\$pdbs;
    - alter database open
- alter user hr identified by hr account unlock;

#### Creazione utente "me"

 Connesso con sqlplus come sysdba (sul container) si crea l'utente e lo si attiva

```
create user me identified by password account unlock; grant connect, resource to me; alter user me quota unlimited on users;
```

 Per terminare l'esecuzione di sqlplus exit

#### TNSNAME.ORA

Nella subdirectory /network/admin della home del database

- In caso di cambiamenti: refresh del listener Oracle Isnrctl reload
- connessione via tnsname sqlplus hr/hr@xe

## Alcuni IDE per Oracle

- Quest Toad for Oracle
- Oracle SQL Developer
- Database Development per Eclipse
  - Help, Install New Software, Work with (...) → Database Development
  - Oracle Database Tools per Eclipse
- DBeaver (standalone o plugin per Eclipse)

• ...

## Principali tipi di dati in Oracle

NUMBER(precision, scale)

INTEGER •

CHAR(length)

VARCHAR2(length)

DATE

#### **SELECT**

- select sysdate from dual; -- la tabella dual select \* from regions; -- tutta la tabella
- La selezione può essere filtrata per colonne e righe select region\_name from regions; -- colonne select \* from regions where region\_id = 1; -- righe
- Pseudo colonne select rowid, rownum from regions;

## Dettagli su tabelle

Tabelle dell'utente corrente

```
select table_name
from user tables;
```

Descrizione di una tabella

```
select column_name, nullable, data_type, data_length, data_precision, data_scale from user_tab_columns where table_name = 'JOBS';
```

(in sqlplus e SQL Developer: describe jobs;)

## Operatori aritmetici

Numeri

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

Date

```
select to_date('30-NOV-2019') + 2, to_date('02-NOV-2019') - 3 from dual; select to_date('02-NOV-2019') - to_date('25-MAR-2019') from dual;
```

 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)
   select job\_title, min\_salary as original, min\_salary salary
- Delimitato da doppi apici se include spazi e per mantenere il case (altrimenti maiuscolo)

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

from jobs;

#### Concatenazione

Operatore ||

```
select first_name || ' ' || last_name as "Employee's name" from employees;
```

#### **NULL**

 Valore non presente o non valido, check esplicito con "is null" select first\_name, last\_name, commission\_pct from employees where commission\_pct is null;

- "Assorbe" altri operandi select first\_name, last\_name, 12 \* salary \* commission\_pct from employees;
- Il built-in NVL() evita ambiguità select first\_name, last\_name, 12 \* salary \* nvl(commission\_pct, 0) from employees;

#### SELECT DISTINCT

- Tutti i manager con dipendenti associati select manager\_id from employees;
- DISTINCT ci permette di filtrare i duplicati select distinct manager\_id from employees;

## Operatori di confronto

```
• =, !=, <, >, <=, >=, ANY (almeno uno), ALL
    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;
    select * from regions where region_id > any(1, 2, 3);
    select * from regions where region id > all(1, 2, 3);
```

## Operatori SQL

- LIKE, IN, BETWEEN, IS NULL. Per negare il loro risultato: NOT
- LIKE wildcard: \_ %
   select last\_name from employees where last\_name like '\_ul%';
- IN

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

- BETWEEN
   select \* from regions where region\_id between 2 and 3;
- IS NULL
  - select \* from employees where manager\_id is null;

## 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 (descinding)
 select \* from employees
 order by last\_name desc, first\_name asc;

 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
- OUTER JOIN anche se la regola non è soddisfatta, si preservano i dati di una (o entrambe) le tabelle di partenza
- self JOIN left e right nella JOIN sono la stessa tabella
- non-equi JOIN usano operatori diversi da "="

#### **INNER JOIN**

- Selezione dati correlati su diverse tabelle 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 select region\_name, country\_name from regions, countries
   where regions.region\_id = countries.region\_id;

## Alias per tabelle

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

F

```
select r.region_name, 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);
- Se la relazione è "naturale" → NATURAL JOIN select region\_name, country\_name from regions natural join countries;

#### JOIN - ON

- NATURAL JOIN e JOIN USING mplicano 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 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;

#### Self JOIN

- La FK si riferisce alla PK della stessa tabella select e.last\_name as employee, m.last\_name as manager from employees e join employees 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 ON ha solo una tabella left e una right → 2 JOIN per 3 tabelle select employee\_id, city, department\_name from employees e join departments d on d.department\_id = e.department\_id join locations I on d.location\_id = I.location\_id;
- Versione "classica" → 2 condizioni nel WHERE per 3 tabelle select employee\_id, city, department\_name from employees e, departments d, locations I 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 between j.min\_salary and j.min\_salary + 100)
 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.job\_id = j.job\_id
 and e.salary between j.min\_salary and j.min\_salary + 100;

#### 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';
```

#### 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;
```

#### **FULL OUTER JOIN**

 Preserva i valori esistenti in entrambe le tabelle, inserendo null nelle eventuali celle relative all'assenza di relazione select e.last\_name, d.department\_name from employees e full outer join departments d on (e.department\_id = d.department\_id) where last\_name = 'Grant' or d.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
- Nome degli employees che non sono manager

# Funzioni su riga singola

- Operano su e ritornano una singola riga
  - Caratteri e stringhe
  - Numeri
  - Conversione
  - Date
  - Espressioni regolari

# Alcune funzioni su stringhe

- ASCII(): codice ASCII di un carattere, CHR(): da codice ASCII a carattere
  - select ascii('A') as A, chr(90) as "90" from dual;
- CONCAT(): concatenazione di stringhe, cfr. operatore ||
  - select concat(first\_name, last\_name) from employees;
- INITCAP(): iniziali maiuscole, UPPER(): tutto maiuscolo, LOWER(): tutto minuscolo
  - select initcap('a new thing') as initcap, lower('NEW') low, upper('old') up from dual;
- INSTR(): x, target, start, occurrence → [1..n], 0 not found
  - select instr('crab', 'ba') as "not found", instr('crab abba rabid cab', 'ab', 2, 3) as pos from dual;
  - select instr(sysdate, '19') as pos from dual;
- LENGTH(): per string e numeri, convertiti implicitamente in stringhe
  - select length('name'), length(42000) from dual;

# Alcune funzioni su stringhe /2

- LPAD(), RPAD(): padding. Stringa → dimensione, con eventuale pad
  - select lpad('tom', 30, '.') tom, rpad('tim', 30, '\_- -\_') tim from dual;
- LTRIM(), RTRIM(), TRIM(): rimozione di caratteri dall'input
  - select ltrim(' Hi!'), rtrim('Hi!abab', 'ab'), trim('0' from '00Hi!000') from dual;
- NVL(): null to value, se null → secondo parametro
  - select employee\_id, nvl(commission\_pct, 0) from employees;
- NVL2(): se non è null → secondo parametro, altrimenti il terzo
  - select employee id, nvl2(commission pct, 'value', 'no value') from employees;
- REPLACE(): sostituzione di substring, SUBSTR(): estrazione di substring
  - select replace('Begin here', 'Begin', 'End'), substr('ABCDEFG',3,4) from dual;

#### Alcune funzioni numeriche

- ABS(): valore assoluto
- CEIL(): 'soffitto', FLOOR(): 'pavimento'
- MOD(): modulo, resto di divisione intera
- POWER(): potenza, EXP(): ex, SQRT(): radice quadrata, LOG(), LN(): logaritmi
- ROUND(), TRUNC(): arrotonda/tronca a n decimali (o potenze di 10 se n è negativo)
- SIGN(): -1, 0, 1 per numeri negativi, zero, positivi select abs(10), abs(-10), ceil(5.8), ceil(-5.2), floor(5.8), floor(-5.2), mod(8, 3), mod(8, 4), power(2, 1), power(2, 3), exp(1), sqrt(25), sqrt(5), log(10, 100), ln(exp(1)), round(5.75), round(5.75, 1), round(5.75, -1), trunc(5.75), trunc(5.75, 1), trunc(5.75, -1) from dual;

#### Alcune funzioni di conversione

- TO\_CHAR(), TO\_NUMBER(): convertono (formattando) a VARCHAR2 e NUMBER
- CAST(), converte ad un tipo supportato da ORACLE, se compatibile

```
select to_char(12345.67), to_char(12345.67, '99,999.99'),
to_char(2019, 'RN'), to_number('970,13') * 2,
cast('05-APR-19' as date) + 2, cast(12345.678 as number(10,2))
from dual;
```

#### Alcune funzioni su date

- ADD\_MONTHS(): aggiunge mesi alla data
- MONTHS\_BETWEEN(): mesi tra le due date
- NEXT\_DAY(): giorno della settimana successivo al corrente
- LAST\_DAY(): ultimo giorno del mese
- ROUND(), TRUNC(): arrotonda/tronca il giorno
   select add\_months(sysdate, 3), months\_between(sysdate, '01-FEB-2019'),
   last\_day(sysdate), next\_day(sysdate, 'lun')
   round(sysdate, 'year'), round(sysdate, 'month'),
   trunc(sysdate, 'year'), trunc(sysdate, 'month')
   from dual;

# 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].\*');

### Esercizi

#### Employees

- Qual è il salario corrente, quale sarebbe con un incremento dell'8.5%, qual è il delta come valore assoluto
- Chi ha 'a' nel nome o cognome
- Quanti mesi sono passati dall'assunzione a oggi
- Salario mostrato come una serie di asterischi (1 = 1000€)
- Quant'è la commissione di ognuno o 'no value'



**₽** 

- Ignorano i NULL
- Uso di DISTINCT per filtrare duplicati
- AVG(): media
- COUNT(): numero di righe
- MAX(): valore massimo
- MEDIAN(): mediana

- MIN(): minimo
- STDDEV(): deviazione standard
- SUM(): somma
- VARIANCE(): varianza

# Raggruppamento via GROUP BY

- Divide il risultato della select in gruppi
- È possibile applicare funzioni aggregate sui gruppi select department\_id, trunc(avg(salary)) 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, trunc(avg(salary))
from employees
where salary < 8000
group by manager_id
having avg(salary) > 6000
order by 2 desc;
```

# Subquery

• In WHFRF:

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

• In HAVING:

```
select department_id, trunc(avg(salary)) from employees
group by department_id having avg(salary) < (
select max(avg(salary)) from employees group by department_id);
```

• In FROM (inline view):

```
select employee_id from (select employee_id from employees where employee_id between 112 and 115);
```

# JOIN con subquery

 Subquery genera una tabella temporanea → join select region name, country count from regions natural join ( select region id, count(rowid) country count from countries group by region id);

# subquery multirighe in WHERE

 Uso degli operatori IN, ANY, ALL 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
- 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
  - Name e department name
    - 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 = '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

### 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 esegue una SELECT due volte, risultati diversi a causa di un INSERT di T2
  - Non repeatable: T1 SELECT, T2 UPDATE, T1 SELECT non ripetibile
  - Dirty: T1 UPDATE, T2 SELECT, T1 ROLLBACK, valore per T2 è invalido
- Garanzie fornite dal DBMS

**READ COMMITTED**: no dirty read ← default Oracle

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 varchar2(20),
  coder_id integer);
```

#### CREATE TABLE AS SELECT

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

```
create table coders
as
select employee_id as coder_id, first_name, last_name, hire_date, salary
from hr.employees
where department_id = 60;
```

#### ALTER TABLE

ADD / DROP COLUMN

```
alter table items add counter number(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);
```

MODIFY column CONSTRAINT NOT NULL

```
alter table items modify name constraint items_name_nn not null;
```

ADD CONSTRAINT PRIMARY KEY

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

DROP CONSTRAINT

```
alter table items drop constraint items_name_nn;
```

# ALTER TABLE per FK

- Creazione di una colonna come FK
  - alter table items drop column coder\_id;
  - alter table items add constraint items\_coder\_id\_fk
     coder\_id references coders(coder\_id);
- FK con rimozione degli orfani
  - ... coder id references coders(coder id) on delete cascade;
- FK con rimozione della relazione
  - ... coder\_id references coders(coder\_id) on delete set null;

#### CREATE TABLE con CONSTRAINT

```
create table details (
  detail id integer
     constraint detail pk 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')),
  name varchar2(20),
     -- constraint detail name nn not null,
     -- constraint detail_name_uq unique,
  coder id integer
     constraint detail coder id fk references coders (coder id) on delete cascade,
  constraint detail name status uq unique(name, status);
```

### TRUNCATE / DROP TABLE

- 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
- Andrebbero creati in un proprio tablespace, informazioni in USER INDEXES
- B-Tree
  - consigliato per colonne con valori unici, usato da Oracle per PK
     create index coders\_last\_name\_ix on coders(last\_name); -- indice semplice
     create index coders\_name\_ix on coders(first\_name, last\_name); -- indice composto drop index coders last name ix;
- Bitmap
  - più efficienti per colonne con pochi valori
     create bitmap index coders\_gender\_ix on coders(gender);

## SEQUENCE

- Oggetto di database che genera una sequenza di interi create sequence my\_seq; -- inizia da 1, incremento 1
- nextval: incrementa e ritorna il valore della sequenza select my\_seq.nextval from dual;
- currval: ritorna il valore corrente, senza incremento select my\_seq.currval from dual;
- Le sequenze possono essere modificate o eliminate alter sequence my\_seq increment by 2; drop sequence my\_seq;

## SEQUENCE /2

Sequenza con custom start e increase:
 create sequence my seq start with 201 increment by 2;

- Altre proprietà definibili su di una sequenza: minvalue, maxvalue, cycle, order, etc.
- PK: si delega alla sequenza la generazione di valori univoci insert into coders values(my\_seq.nextval, 'Bertrand', 'Meyer', SYSDATE, 8000);
- Info nella tabella USER\_SEQUENCES

#### **VIEW**

- Occorre avere il diritto di crearle (GRANT CREATE VIEW TO ...)
- 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
with read only;
```

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

# PL/SQL

- Estensione procedurale di Oracle a SQL
- Basato sulla definizione di blocco

```
[DECLARE
...] variabili locali

BEGIN
...
Ogni istruzione è terminata da un punto e virgola

[EXCEPTION gestione degli errori
...]

END;
Richiesta di esecuzione del buffer
```

# Hello PL/SQL

- Un blocco minimale
  - non sono necessarie DECLARE e EXCEPTION
- By default l'output su console non è attivo

```
set serveroutput on
begin
  dbms_output.put_line('Hello PL/SQL');
end;
/
```

## Variabili

- Le variabili sono dichiarate nel blocco DECLARE
- Tutti i tipi SQL di Oracle sono supportati da PL/SQL, più alcuni tipi aggiuntivi, come BOOLEAN
- La loro definizione non è obbligatoria
- Per convenzione iniziano per "v\_"

```
declare
   v_width integer;
   v_height integer := 2;
   v_area integer := 6;
begin
   v_width := v_area / v_height;
   dbms_output.put_line(
        'v_width = ' || v_width);
end;
/
```

## Eccezioni

- Gestione degli errori di esecuzione
- Simile al meccanismo try/catch di Java

```
begin
   dbms_output.put_line(6 / 0);
exception
   when zero_divide then
    dbms_output.put_line('Zero divide!');
end;
/
```

```
begin
   dbms_output.put_line(1 / 0);
exception
   when others then
   dbms_output.put_line('Exception!');
end;
/
```

#### IF - ELSIF - ELSE - END IF

```
declare
  v a integer := 1;
begin
  if v a > 0 then
     dbms output.put_line('v_a is positive');
  elsif v a = 0 then
     dbms output.put_line('v_a is zero');
  else
     dbms output.put line('v a is negative');
  end if:
end;
```

## LOOP

Loop semplice: EXIT (WHEN), CONTINUE (WHEN)

```
v_x := 0;
loop
    v_x := v_x + 1;
    if v_x = 3 then exit;
    end if;
end loop;
```

```
v_x := 0;
loop
  v_x := v_x + 1;
  exit when v_x = 5;
end loop;
```

```
v_x := 0;
loop
  v_x := v_x + 1;
  if v_x = 3 then
    -- something special
    continue;
  end if;
  exit when v_x = 5;
end loop;
```

```
v_x := 0;
loop
v_x := v_x + 1;
continue when v_x = 3;
-- something normal
exit when v_x = 5;
end loop;
```

#### WHILE e FOR

- WHILE LOOP, finché la condizione è vera
- FOR LOOP, per ogni valore indicato (REVERSE)

```
v_x := 0;
while v_x < 5 loop
  v_x := v_x + 1;
end loop;</pre>
```

```
for i in 1..5 loop
dbms_output.put_line('for loop: ' || i);
end loop;
```

```
for i in reverse 1..5 loop
   dbms_output.put_line('for loop: ' || i);
end loop;
```

### SELECT INTO

Lettura di una singola riga

```
declare
  v first name coders.first name %type;
                                                        Tipo di una colonna
  v last name coders.last name%type;
begin
  select first name, last name
  into v_first_name, v_last_name
  from coders
  where coder id = 103;
  dbms_output.put_line('[' || v_first_name || ' ' || v_last_name || ']');
end;
```

### **CURSOR**

- Lettura di più righe
- Si definisce un CURSOR associato a SELECT
- OPEN CURSOR esegue la SELECT
- FETCH INTO legge la riga corrente
- EXIT WHEN %NOTFOUND termina la lettura del cursore
- CLOSE CURSOR rilascia le risorse associate

```
declare
  v last name coders.last name%type;
  v hire date coders.hire date%type;
  cursor v coder cursor is
    select last name, hire date from coders;
begin
open v coder cursor;
  loop
  into v last name, v hire date;
  exit when v coder cursor%notfound;
    dbms output.put line(
      '[' || v_last_name || ', ' || v_hire_date || ']');
  end loop;
close v coder cursor;
end:
```

## **CURSOR in FOR LOOP**

gestione implicita, codifica semplificata

#### CREATE PROCEDURE

Parametri IN / OUT

PROCEDURE body

```
create or replace procedure get_coder_salary(
    p_coder_id in coders.coder_id%type,
    p_salary out coders.salary%type) is
begin
    select salary
    into p_salary
    from coders
    where coder_id = p_coder_id;
end get_coder_salary;
/
```

IS/AS

Stored Procedure

drop procedure get\_coder\_salary;

# Esecuzione di una procedura

```
declare
  v id coders.coder_id%type := 105;
  v salary coders.salary%type;
begin
  get coder salary(v id, v salary);
  dbms_output_line('Salary is ' || v_salary);
exception
  when others then
    dbms_output.put_line('Can"t get salary for ' || v_id);
end;
```

#### CREATE FUNCTION

Parametri IN / OUT

Return type

FUNCTION body

```
create or replace function
    p_coder_id in coders.coder_id%type)
return number as
    v_salary coders.salary%type;
begin
    select salary
    into v_salary from coders
    where coder_id = p_coder_id;
    return v_salary;
end get_salary;
/
```

IS/AS

Variabili locali

**Stored Procedure** 

drop function get\_salary;

### Esecuzione di una funzione

```
declare
  v id coders.coder id%type := 105;
  v salary coders.salary%type;
begin
  v salary := get salary(v id);
  dbms output.put_line('Salary is ' || v_salary);
exception
  when others then
     dbms_output.put_line('Can"t get salary for ' || v_id);
end;
```

## **TRIGGER**

- Procedura eseguita automaticamente in relazione (prima, dopo, o invece) all'esecuzione di un comando DML
- Row-level
  - Eseguito per ogni riga coinvolta
  - In update, accesso a stato precedente e successivo
  - Esecuzione condizionale
- Statement-level
  - Eseguito una volta per tutte le righe



# Un esempio di trigger

Tabella di output del trigger

Trigger

```
create table coder_salaries (
   coder_id number(6, 0)
     references coders(coder_id),
   old_salary number(8, 2),
   new_salary number(8, 2)
);
```

```
create or replace trigger salary_update
before update of salary on coders
for each row
begin
  insert into coder_salaries values(
        :old.coder_id, :old.salary, :new.salary);
end salary_update;
/
```

Generazione di eventi che scatenano il trigger

update coders set salary = salary \* 1.3 where coder\_id > 103;

# 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