

# SQL: concetti base

---

PROF. DIOMAIUTA CRESCENZO

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA «*LUIGI VANVITELLI*»

# SQL

---

- Structured Query Language
  - DDL (Data Definition Language)
  - DML (Data Manipulation Language)
  - DCL (Data Control Language).
- I sistemi commerciali spesso offrono una serie di strumenti che estendono le funzionalità definite a livello di standard.
- Un esempio sono i trigger di Oracle per la definizione di DB attivi.

# SQL: Livelli di compatibilità

---

- I sistemi commerciali sono classificati in base all'aderenza allo standard 92
  - Entry Level (SQL-89)
  - Intermediate Level
  - Full SQL (SQL-92)

# Domini

---

- Domini elementari (predefiniti)
- Domini definiti dall'utente (semplici, ma riutilizzabili)

# Domini elementari

---

- **Carattere:** singoli caratteri o stringhe, anche di lunghezza variabile
  - character [varying] [(lunghezza)] *PER INDICARE VARYING (stringhe)*
- **Bit** *L>max del carattere*
  - utilizzato per attributi che possono assumere valori 0 ed 1. Può essere usato per stringhe di bit.
  - bit [varying] [(lunghezza)]
- **Numerici:** permette di rappresentare valori interi o valori in virgola fissa
  - Numeric [(Precisione[, Scala])]
  - Decimal [(Precisione[, Scala])]
  - Integer
  - smallint

# Domini elementari

---

- I domini **integer** e **smallint** si usano quando non è necessario avere una rappresentazione della parte frazionaria.
- L'unico vincolo implementativo è che la precisione del dominio **integer** sia del dominio **smallint**.
- Il dominio **numeric** rappresenta numeri in base decimale;
  - Precisione indica il numero di cifre significative
  - Scala il numero di cifre dopo la virgola.
    - ES: numeric(6,4) [-99.9999, +99.9999]  
6 cifre post virgola
    - La differenza tra **numeric** e **decimal** è che la precisione per **numeric** è un requisito esatto, mentre per **decimal** è un requisito minimo.

↳ Se non è soddisfatto non lo fa inserire, altrimenti no.

# Domini elementari

---

- Tipi numerici approssimati:
  - float [(Precisione)], double precision, real
  - Nel dominio float si può scegliere la precisione, intesa come numero di cifre della mantissa.
  - La precisione di double è doppia rispetto a real.
- Data e Ora:
  - Date : record di valori {year month day}
  - Time: record di valori {hour minute second}
  - Timestamp: date+time
- Intervalli temporali:
  - interval PrimaUnità diTempo [to UltimaUnità diTempo]
  - interval year to month
  - indica che gli intervalli vanno misurati in numero di anni e di mesi.

# Domini elementari

---

- **Data, ora, intervalli di tempo**
- Introdotti in SQL:1999:
  - **Boolean** ↗ per memorizzare file binari.
  - **BLOB, CLOB** (binary/character large object): per grandi immagini e testi

# Domini: Definizione

---

- Istruzione **CREATE DOMAIN**:
  - definisce un dominio (**semplice**), utilizzabile in definizioni di relazioni, anche con vincoli e valori di default

# Create Domain: Esempio

---

```
CREATE DOMAIN Voto  
AS SMALLINT DEFAULT NULL  
CHECK ( value >=18 AND value <= 30 )
```

# Definizione dei dati in SQL

---

- Istruzione **CREATE DATABASE**:
- Crea un nuovo database, che potrà contenere tabelle, viste, trigger o altri tipi di oggetti
- Esempio:
  - **CREATE DATABASE Azienda**

# Definizione dei dati in SQL

---

- Consente la dichiarazione di uno schema di base di dati come collezione di oggetti, cioè domini, tabelle, viste, privilegi ed asserzioni.

*create schema [nome]*

*[ [authorization] Autorizzazione ]*

{ElementiDelloSchema}

- Esempio:

➤ CREATE SCHEMA schema\_azienda

# Definizione dei dati in SQL

---

- Seguito dal parametro **AUTHORIZATION** indica il proprietario dello schema. Nel caso in cui venga omesso, il proprietario sarà l'utente che ha digitato il comando
- Esempio:
  - `CREATE SCHEMA schema_azienza AUTHORIZATION amministratore`

# Creazione di tabelle

Definisce uno schema per una nuova relazione

- Istruzione **CREATE TABLE**:
- definisce uno schema di relazione e ne crea un'istanza vuota
- specifica attributi, domini e vincoli

```
create table nome (  
    NAttr Dom [ValoreDef] [vincoli]  
    {,NAttr Dom [ValoreDef] [vincoli]} [Altri Vincoli]  
)
```

# Creazione di tabelle

include univocità e obbligatorietà



**CREATE TABLE** Impiegato(

Matricola CHAR(6) PRIMARY KEY,  $\Rightarrow$  da qualche parte ci deve essere.

Nome CHAR(20) NOT NULL, nome è campo obbligatorio

Cognome CHAR(20) NOT NULL,

Dipart CHAR(15),

Stipendio NUMERIC(9) DEFAULT 0, valore di default se non inserito

FOREIGN KEY(Dipart) REFERENCES Dipartimento(NomeDip),  $\Rightarrow$  Nome della tabella

UNIQUE (Cognome, Nome)

)

Indica che in questa tabella (cognome, nome) sono una coppia univoca

\* Implica esistenza di NomeDip e la sua sarà chiave primaria.

Chavi esterna fattta con dipart e punta a campo

NomeDip di Tabella Dipartimenti\*

NOTA: Dpart e NomeDip devono avere stesso dominio.

## Creazione di tabelle

---

- Gli attributi possono appartenere ai domini predefiniti o a domini creati con la sintassi
  - `create domain ndom as TipodiDato [ValoreDiDefault] [Vincolo]`
  
- Nella definizione delle tabelle e di domini esistono i valori di **Default** che rappresentano il valore assunto dall' attributo quando viene inserita una riga nella tabella senza specificare il valore per l' attributo stesso.

`default {valore | user | null}`

ES: NumFigli smallint default 0

# Vincoli intrarelazionali

---

- **NOT NULL**: indica che l'attributo deve essere specificato, a meno che ad esso non sia associato un valore di default diverso da null.
- **UNIQUE**: si applica ad un attributo per dire che nella tabella non possono esistere due righe con valori identici su tale attributo.
- Es: Matricola char(6) unique  
Es: cognome char(20),  
      nome char(20), unique(cognome,nome)
  - per indicare che non devono esistere righe che abbiano uguali sia il nome sia il cognome.

# Vincoli intrarelazionali

- **PRIMARY KEY**: può essere indicato una sola volta per tabella
- Mediante esso si specifica l'attributo (o l'insieme di attributi) che rappresentano la chiave primaria della tabella.
  - E' usato per sottolineare l'attributo (o l'insieme di attributi) più usato per accedere alle righe di una tabella.

```
Create table impiegato (
    Nome char(20),
    Cognome char(20),
    Dipart char(20),
    primary key (Cognome,Nome)
)
```

↑  
imposto NOT NULL e UNIQUE

```
CREATE TABLE Impiegato(
    Matricola CHAR(6) PRIMARY KEY,
    Nome CHAR(20) NOT NULL,
    Cognome CHAR(20) NOT NULL,
    Dipart CHAR(15),
    Stipendio NUMERIC(9) DEFAULT 0,
)
```

# Chiavi su più attributi, attenzione

---

**Nome**                   **CHAR(20) NOT NULL,**  
**Cognome**               **CHAR(20) NOT NULL,**  
**UNIQUE (Cognome, Nome),**

**Nome**                   **CHAR(20) NOT NULL UNIQUE,**  
**Cognome**               **CHAR(20) NOT NULL UNIQUE,**

**Non è la stessa cosa!**

# Vincoli interrelazionali

---

- Tratteremo i vincoli di integrità referenziale (detti anche vincoli di riferimento).
- In SQL esiste il vincolo foreign key. Esso crea un legame tra l'attributo della tabella corrente (interna) e un attributo di un'altra tabella (esterna) (tipicamente alla chiave di questa).
- L'unico requisito è che l'attributo della tabella esterna sia definito unique.
- Tale vincolo può essere specificato in due modi:
  - references
  - foreign key

# Vincoli interrelazionali

---

- **references**: è usato quando si vuole creare il legame con riferimento ad un singolo attributo:

```
create table Impiegato (
    Nome    char(20),
    Cognome char(20),
    Dipart  char(20) references Dipartimento(NomeDip)
)
```

- **foreign key**: è usato quando ci si riferisce ad insieme di attributi:

```
create table Impiegato (
    Nome    char(20),
    Cognome char(20),
    Dipart  char(20) references Dipartimento(NomeDip),
    foreign key (Nome, Cognome) references Anagrafica (nonan, cognan)
)
```

# Vincoli interrelazionali

---

- Violazioni di un vincolo foreign key.

```
create table Impiegato (
    Nome char(20),
    Cognome char(20),
    Dipart char(20) references Dipartimento(NomeDip)
)
```

- Impiegato è la tabella interna.
- Dipartimento è la tabella esterna.
- La violazione può essere a livello di tabella
  - interna (Impiegato)
  - esterna (Dipartimento)

# Vincoli interrelazionali

---

- Violazione sulla tabella interna.
- Il vincolo può essere violato usando (quando si inserisce una nuova riga o quando si modifica una riga esistente) per il campo Dipart un valore non presente nella colonna NomeDip della tabella Dipartimento.
- L'operazione viene semplicemente impedita.

# Vincoli interrelazionali

- Consideriamo la tabella esterna.
- Le operazioni che possono dare problemi sono:
  - Modifiche dell'attributo riferito (on update)
  - Cancellazione di righe (on delete)
- Politiche possibili
  - Cascade**: tutte le righe della tabella interna che corrispondono alla riga cancellata vengono eliminate;
  - set null**: all'attributo referente viene assegnato NULL al posto del valore cancellato nella tabella esterna
  - set default**: all'attributo referente viene assegnato il valore di default al posto del valore cancellato nella tabella esterna *di default ammesso per la chiave esterna*
  - no action**: la cancellazione non è consentita

```
create table Impiegato (
    Nome char(20),
    Cognome char(20),
    Dipart char(20),
    foreign key (Dipart) references Dipartimento(NomeDip)
        on delete set null on update cascade
)
```

# Modifica degli schemi

## □ alter

alter domain *nomeDominio* <set default *valoreDefault*

drop default

add constraint *DefVincolo* |

drop constraint *NomeVincolo* >

Modifica Tabella

alter table *nomeTabella* <

alter column *NomeAttributo* <set default *NuovoDefault*

drop default> |

add constraint | *vincolo*

drop constraint |

add column *NomeAttributo* *colonna*

drop column *NomeAttributo* >

Lui modifica il dominio

Ma per modificare la tabella?

Modifico campo che già c'è (una colonna pre-existing)

□ ES: alter table Dipartimento add column NroUffici numeric(4)

# Modifica degli schemi

---

- Drop:**
  - Permette di rimuovere schemi, tabelle etc.
  - `drop <schema | domain | table | view | assertion | > [restrict | cascade]`
- restrict:** specifica che il comando deve essere eseguito solo in presenza di oggetti vuoti.
- cascade:** tutti gli oggetti specificati sono rimossi, innestando una reazione a catena.
- Data la pericolosità del comando, molto sistemi permettono di analizzare il risultato di un drop cascade prima di eseguirlo effettivamente.

# Definizione degli indici

---

- Gli indici rappresentano uno strumento prezioso per velocizzare l'accesso ai dati.
  - **create [unique] index NomIdx on NomeTable (ListaAttributi)**
- **unique** indica che nella tabella non sono possibili righe che assumono lo stesso valore su **ListaAttributi** (che quindi è una chiave eventualmente non minimale)
- L'ordine in cui sono dati gli attributi è importante.
  - **create unique index ind onto Anagrafica (Cognome, Nome)**
  - **drop index Nomidx**

# Cataloghi relazionali

---

- Si supponga di avere creato uno schema e di aver creato tutta una serie di oggetti. Come faccio a vedere tutti gli oggetti definiti?
- Ogni DBMS mantiene alcune tabelle in cui sono riportate le descrizione degli oggetti creati (ad es. le tabelle create).
- Esiste un Definition\_Schema ed una serie di viste su di esso (TABLES, COLUMNS, DOMAINS, ....)

Schema<sup>↑</sup> che dà info su tabelle create, doms create ecc.

# Interrogazioni in SQL

---

- SQL esprime le interrogazioni in modo dichiarativo attraverso il comando select.
- L' interrogazione è poi automaticamente tradotta in termini procedurali e poi eseguita.
- L' alternativa è costituita dall' uso di linguaggi tradizionali, in cui SQL è immerso.
- Esistono vari modi per formulare una stessa query.
- L' utente, tipicamente, non ha bisogno di preoccuparsi dell' efficienza della query formulata quanto della sua leggibilità e modificabilità.

# Interrogazioni in SQL

---

**SELECT** ListaAttributi

**FROM** ListaTabelle *dove quale tabella*

[ **WHERE** Condizione ]  $\Rightarrow$  o è true o è false.

*La select proietta, la where seleziona*

- clausola **SELECT** (chiamata target list)
- clausola **FROM**
- clausola **WHERE**

# Interrogazioni semplici

- Esempio:
- Impiegato (Nome, Cognome, Stipendio,Dipart, Ufficio)  
Dipartimento(Nome,Indirizzo,Citta)
  - Interrogazione 1: Estrarre lo stipendio degli impiegati di cognome «Rossi»

Tabella Impiegato

```
select Stipendio as Salario  
from Impiegato  
where Cognome='Rossi'
```

Salario
45
80

Nome	Cognome	Dipart	Ufficio	Stipendio
Mario	Rossi	Amministrazione	10	45
Carlo	Bianchi	Produzione	20	36
Giovanni	Verdi	Amministrazione	20	40
Franco	Neri	Distribuzione	16	45
Carlo	Rossi	Direzione	14	80
Lorenzo	Gialli	Direzione	7	73
Paola	Rosati	Amministrazione	75	40
Marco	Franco	Produzione	20	46

# Clausola select

---

- Ritorna una tabella con un unico attributo (Salario) i cui valori sono ottenuti dalle righe di Impiegato aventi Cognome = 'Rossi'.
  - select \* from Impiegato where Cognome='Rossi'

Nome	Cognome	Dipart	Ufficio	Stipendio	Città
Mario	Rossi	Amministrazione	10	45	Milano
Carlo	Rossi	Direzione	14	80	Milano

- La clausola *select* specifica gli elementi dello schema della tabella risultato. Se compare il carattere speciale \* (asterisco), seleziona tutti gli attributi delle tabelle elencate nella clausola *from* (senza proiezione).

↳ Non posso usare la ridenominazione

# Interrogazioni semplici

---

- Interrogazione 3: Estrarre lo stipendio mensile dell'impiegato che ha cognome 'Bianchi'

select Stipendio/12 as StipMens      *Ophazionen arithmetiche su dati che interrogo*  
from Impiegato  
where Cognome='Bianchi'

StipMens
3,00

# Clausola from

- Interrogazione 4: Estrarre i nomi degli impiegati e le città in cui lavorano

```
select Impiegato.Nome, Impiegato.Cognome, Citta'  
from Impiegato, Dipartimento  
where Impiegato.Dipart = Dipartimento.Nome
```

Impiegato.Nome	Impiegato.Cognome	Dipartimento.Città
Mario	Rossi	Milano
Carlo	Bianchi	Torino
Giovanni	Verdi	Milano
Franco	Neri	Roma
Carlo	Rossi	Milano
Lorenzo	Gialli	Milano
Paola	Rosati	Milano
Marco	Franco	Torino

La virgola indica che  
sono facendo  
operazione di  
Join

Immagino ci sia Vincolo di un'oggetto  
referenziale o che almeno si possa fare  
il Join. Farlo per il matching.

Tabella Dipartimento

Nome	Indirizzo	Città
Amministrazione	Via Tito Livio, 27	Milano
Produzione	P.Le Lavater, 3	Torino
Distribuzione	Via Segre, 9	Roma
Direzione	Via Tito Livio, 27	Milano
Ricerca	Via Venosa, 6	Milano

In maniera più compatta:

```
select I.nome, D.nome, D.Città
```

```
from Impiegato as I, Dipartimento as D
```

```
where I.Dipart = D.Nome
```

Oppure  
from Impiegato I

⇒ Qui ci possono essere ambiguità nei nomi, quindi specifico  
con la notazione punto/tilde

# Clausola where

---

- Nella clausola where è possibile usare connettivi logici (and, or, not) ed espressioni di confronto.

- Interrogazione 6: Estrarre il nome e il cognome degli impiegati che lavorano nell'ufficio 20 del dipartimento Amministrazione

```
select Nome, Cognome  
from Impiegato  
where Ufficio=20 and dipart='Amministrazione'
```

Nome	Cognome
Giovanni	Verdi

# Clausola where

---

- Interrogazione 7: Estrarre i nomi e i cognomi degli impiegati che lavorano nel dipartimento Amministrazione o nel dipartimento Produzione

```
select Nome, Cognome  
from Impiegato  
where Dipart='Amministrazione' OR Dipart = 'Produzione'
```

Nome	Cognome
Mario	Rossi
Carlo	Bianchi
Giovanni	Verdi
Paola	Rosati
Marco	Franco

# Clausola where

---

- Interrogazione 8: Estrarre i nomi propri degli impiegati di cognome ‘Rossi’ che lavorano nei dipartimenti Amministrazione o Produzione

```
select Nome
```

```
from Impiegato
```

```
where Cognome = 'Rossi' AND (Dipart='Amministrazione' OR Dipart = 'Produzione')
```

Nome
Mario

# Operatore like

---

- E' usato nel confronto di stringhe e si appoggia su due caratteri speciali:
  - \_ (indica un qualsiasi carattere)
  - % (indica una stringa, anche vuota)

```
select nome  
from Impiegato  
where cognome like '_o%'
```

- Ritornerà tutti nomi di Impiegato il cui cognome ha una "o" in seconda posizione e termina per "i" (Rossi, Doncelli, Loi).

# Gestione dei valori nulli

---

- Esempio Stipendio > 40
  - è soddisfatta dalle righe in cui Stipendio è noto e maggiore di 40.
- Per i valori nulli si usa *Stipendio is null* oppure *Stipendio is not null*
- In SQL-2 ci si è orientati verso una logica a 3 valori.
- L'unico predicato che ritorna solo valore vero o falso è
- *is [not] null*
- Non ci sono particolari stravolgimenti, se non in interrogazioni molto complesse

# Interpretazione algebrica di interrogazioni SQL

---

- Una interrogazione generica:

```
select T_1.Attributo_11, ..., T_h.Attributo_nm
```

```
from Tabella_1 T_1, ..., Tabella_n T_n
```

```
where Condizione
```

- Corrisponde in algebra relazionale a:

- $\pi_{T_1.Attributo_11, \dots, T_h.Attributo_nm} (\sigma_{Condizione} (Tabella_1 \bowtie \dots \bowtie Tabella_n))$

# Duplicati

---

- Abbiamo detto che il risultato del select è una relazione.
- Questo non è del tutto vero, in quanto tale risultato potrebbe avere righe uguali.
- Ciò è dovuto a ragioni di efficienza, perché eliminare i duplicati può essere costoso in termini di tempo e deve essere espressamente richiesto.
- `select distinct Città from ...`

# Join

- Quando si formula una interrogazione su 2 o più tabelle, si agisce sul prodotto cartesiano. E' possibile allora specificare nella condizione where gli attributi leganti per ottenere un theta join.
- La sintassi SQL-2 ha però introdotto un'altra possibilità per specificare questo operatore con diverse varianti
  - from t1 tipojoin join t2 on condizione
  - con tipojoin
  - inner, right (outer), left (outer), full (outer)

Select I.Nome, Cognome, D.Città

From Impiegato as I join Dipartimento as D

On Dipart=D.nome

Su quali valori devo effettuare il match? I.Dipart = D.Nome

Se scrivo Join, la clausola on è obbligatoria.

Specifico gli attributi su cui fare il Join.

# Outer Join

---

- Outer join
  - left
  - right
  - full
- Il join esterno mantiene tutte le righe della tabella di sinistra (left), della tabella di destra (right) o di entrambe (full).

# Outer Join

- Interrogazione: Estrarre i guidatori con le automobili loro associate, mantenendo nel risultato anche i guidatori senza automobile

GUIDATORE

Nome	Cognome	NroPatente
Mario	Rossi	VR 2030020Y
Carlo	Bianchi	PZ 1012436B
Marco	Neri	AP 4544442R

Targa	Marca	Modello	NroPatente
KB 574 WW	Fiat	Punto	VR 2030020Y
GA 652 FF	Fiat	Panda	VR 2030020Y
BJ 747 XX	Lancia	Ypsilon	PZ 1012436B
ZB 421 JJ	Fiat	Uno	MI 2020030U

AUTOMOBILE

Nome	Cognome	NroPatente	Targa	Marca	Modello
Mario	Rossi	VR 2030020Y	KB 574 WW	Fiat	Punto
Mario	Rossi	VR 2030020Y	GA 652 FF	Fiat	Panda
Carlo	Bianchi	PZ 1012436B	BJ 747 XX	Lancia	Ypsilon
Marco	Neri	AP 4544442R	NULL	NULL	NULL

Select Nome, Cognome, G.NroPatente, Targa,  
Marca, Modello

From Guidatore G left join Automobile A on  
(G.NroPatente = A.NroPatente)

↳ su quale attributo fare join

# Full Join

- Interrogazione: Estrarre tutti i guidatori e tutte le auto, mostrando tutte le relazioni esistenti tra essi

GUIDATORE

Nome	Cognome	NroPatente
Mario	Rossi	VR 2030020Y
Carlo	Bianchi	PZ 1012436B
Marco	Neri	AP 4544442R

AUTOMOBILE

Targa	Marca	Modello	NroPatente
KB 574 WW	Fiat	Punto	VR 2030020Y
GA 652 FF	Fiat	Panda	VR 2030020Y
BJ 747 XX	Lancia	Ypsilon	PZ 1012436B
ZB 421 JJ	Fiat	Uno	MI 2020030U

Nome	Cognome	NroPatente	Targa	Marca	Modello
Mario	Rossi	VR 2030020Y	KB 574 WW	Fiat	Punto
Mario	Rossi	VR 2030020Y	GA 652 FF	Fiat	Panda
Carlo	Bianchi	PZ 1012436B	BJ 747 XX	Lancia	Ypsilon
Marco	Neri	AP 4544442R	NULL	NULL	NULL
NULL	NULL	MI 2020030U	ZB 421 JJ	Fiat	Uno

Select Nome, Cognome, G.NroPatente, Targa,  
Marca, Modello

From Guidatore G full join Automobile A on  
(G.NroPatente = A.NroPatente)

# Variabili (Alias)

---

- Esempio

```
select I.nome, D.nome, D.Citta'  
From Impiegato as I, Dipartimento as D  
where I.Dipart = D.Nome
```

- In questo caso la variabile è da interpretarsi semplicemente come un ridenominazione (nome alternativo).
- Utilizzando gli Alias è però possibile anche far riferimento a più esemplari della stessa tabella.
- In questo caso l'interpretazione è che, per ogni alias, si introduce una variabile di tipo tabella che viene inizializzata con la tabella in questione.

# Variabili (Alias)

- Interrogazione: Estrarre tutti gli impiegati che hanno lo stesso cognome (ma diverso nome) di impiegati del dipartimento di Produzione

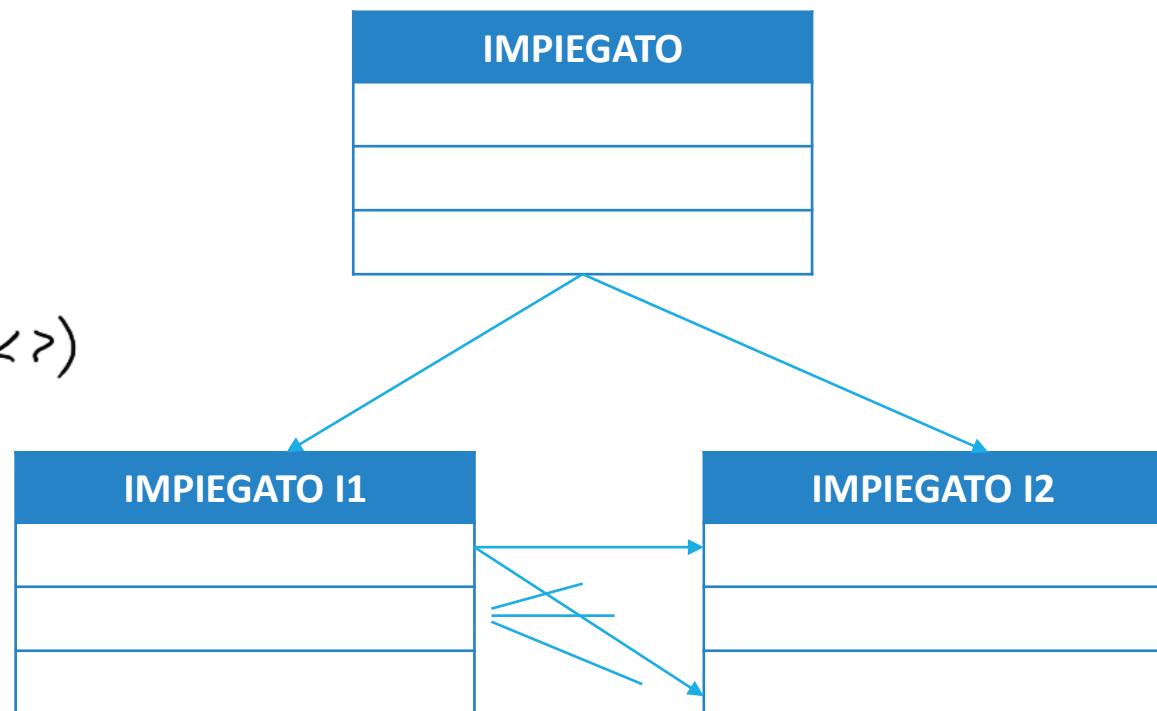
Select I1.Cognome, I1.Nome

From Impiegato I1, Impiegato I2

Where I1.Cognome = I2.Cognome AND

I1.Nome <> I2.Nome AND (differenza = <>)

I2.Dipart = 'Produzione'



# Variabili (Alias)

---

- Interrogazione: Trovare il nome e lo stipendio dei capi degli impiegati che guadagnano più di 40

Select I1.Nome as NomeC, I1.Stipendio as StipC

From Impiegato I1, Supervisione, Impiegato I2

Where I1.Matricola = Supervisione.Capo AND

I2.Matricola = Supervisione.Impiegato AND

I2.Stipendio > 40

# Ordinamento

---

- Il risultato di una interrogazione è una relazione e come tale non è ordinata.  
*order by AttrDiOrdinamento [ asc | desc ] (se non specificato, default è descending)  
{, AttrDiOrdinamento [ asc | desc ] }*
- Nelle applicazioni è comunque utile avere i risultati ordinati secondo una certa strategia
- Interrogazione: Estrarre il contenuto della tabella automobile ordinato in base alla marca (in modo discendente e al modello).  
*select \*  
from Automobili  
order by Marca desc, Modello*
- L'ordinamento introduce, in molti casi, un overhead significativo.

Targa	Marca	Modello	NroPatente
BJ 747 XX	Lancia	Ypsilon	PZ 1012436B
GA 652 FF	Fiat	Panda	VR 2030020Y
KB 574 WW	Fiat	Punto	VR 2030020Y
ZB 421 JJ	Fiat	Uno	MI 2020030U

# Operatori aggregati

---

- Principale estensione di SQL rispetto all'algebra relazionale.
- Agiscono non a livello di tupla ma a livello di relazione.
  - count
  - sum
  - max
  - min
  - Avg
- Distinct: elimina i duplicati
- All: trascura solo i valori nulli

# Operatore count

\* , oppure distinct cognome , all cognome con  
tutte attributi

- count restituisce il numero di righe o distinti valori;

sintassi: count(< \* | [ distinct | all ] ListaAttributi >)

- Interrogazione: Estrarre il numero di impiegati del dipartimento Produzione:

```
select count(*)  
from Impiegato  
Where Dipart = 'Produzione'
```

- Interrogazione: Estrarre il numero di diversi valori dell'attributo Stipendio fra tutte le righe di Impiegato

```
select count(distinct Stipendio)  
from Impiegato
```

- Interrogazione: Estrarre il numero di righe che possiedono un valore non nullo per l'attributo Nome

```
select count(all Nome)  
from Impiegato
```

# Operatori aggregati

---

- <sum | max | min | avg> ([distinct | all] AttrEspr)
- sum e avg ammettono solo espressioni che rappresentano valori numerici o di tipo intervallo temporale.
- max e min possono riferirsi a qualsiasi attributo sul cui dominio sia definito un ordinamento. (*strutture più lunghe*)
- Interrogazione: Estrarre la somma degli stipendi del dipartimento Amministrazione

```
select sum(Stipendio) as SommaStipendio  
from Impiegato  
where Dipart = 'Amministrazione'
```

SommaStipendio
125

# Operatori aggregati

**select max(Stipendio)**

**from Impiegato**

**where Età <35**

l'unico più  
→ quale

Operatore aggregato e operatore di campo.

**select Matricola, max(Stipendio)**

**from Impiegato**

l'uni dà un valore

**where Età <35**

ERRATA!! Nella clausola Select è presente una selezione ed un operatore aggregato.

# Interrogazioni con raggruppamento

- In alcune applicazioni sorge l'esigenza di applicare gli operatori aggregati distintamente a sottoinsiemi di righe.
- Interrogazione: Estrarre la somma degli stipendi di tutti gli impiegati dello stesso dipartimento

select Dipart, sum(Stipendio)

from Impiegato

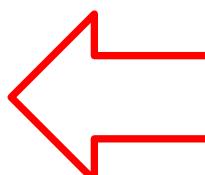
group by Dipart

↑  
Raggruppa  
i dipartimenti  
comuni e fa  
la somma

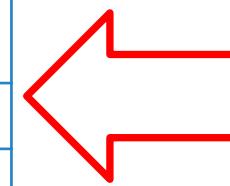
sum(Stipendio)	
Dipart	sum(Stipendio)
Amministrazione	125
Distribuzione	45
Direzione	153
Produzione	82

(Non esiste nell'algabra relazionale)

group by Dipart



Dipart	Stipendio
Amministrazione	45
Amministrazione	40
Amministrazione	40
Produzione	36
Produzione	46
Distribuzione	45
Direzione	80
Direzione	73
Amministrazione	40
Produzione	46



select Dipart, Stipendio

from Impiegato

Dipart	Stipendio
Amministrazione	45
Produzione	36
Amministrazione	40
Distribuzione	45
Direzione	80
Direzione	73
Amministrazione	40
Produzione	46

# Interrogazioni con raggruppamento

---

select Ufficio

from Impiegato

group by Dipart

- sintatticamente errata e priva di senso. Poiché ad ogni valore di Dipart corrisponderanno diversi valori di Ufficio

select Dipart, count (\*), Città

from Impiegato I inner join Dipartimento D on (I.Dipart=D.Nome)

group by Dipart

- sintatticamente errata ma significativa

select Dipart, count (\*), Città

from Impiegato I inner join Dipartimento D on (I.Dipart=D.Nome)

group by Dipart , Città

# Predicati sui gruppi

- La clausola group by consente di lavorare su sottoinsiemi.
- La clausola having consente di selezionare i sottoinsiemi.
- Interrogazione: Estrarre i dipartimenti che spendono più di 100 in stipendi

select Dipart, sum(Stipendio) as SommaStipendi

from Impiegati

group by Dipart

having sum(Stipendio) > 100

= Somma Stipendi

↑ faccio selezioni su gruppi

Dipart	SommaStipendi
Amministrazione	125
Direzione	153

- Solo i predicati che contengono operatori aggregati possono apparire nell'argomento della clausola having *né Stipendio*

*Where non posso usarlo perché ho raggruppato*

# Sintassi generale di una query SQL

---

```
select ListaAttributoOEspressioni  
from ListaTabelle  
[ where CondizioniSemplici ]  
[ group by ListaAttributiDiRaggruppamento ]  
[ having CondizioniAggregate ]  
[ order by ListaAttributiDiOrdinamento ]
```

# Interrogazioni di tipo insiemistico

- union, intersect, except (minus) [all] *per le differenze*
- Possono essere solo al livello più esterno di una query, operando sul risultato di un select.
- Eseguono sempre una eliminazione di duplicati (se non si esplicita la keyword all).
- E' richiesto che gli attributi siano compatibili.

select Nome

from Impiegato

union

select Cognome

from impiegato

Interrogazione: Estrarre i nomi e i cognomi degli impiegati

Nome
Mario
Carlo
Giovanni
Franco
Lorenzo
Paola
Marco
Rossi
Bianchi
Verdi
Neri
Gialli
Rosati
Franco

# Interrogazioni di tipo insiemistico

- Interrogazione: Estrarre i cognomi degli impiegati che sono anche nomi

```
select Nome,  
      from Impiegato  
      intersect  
      select Cognome  
      from Impiegato
```

Equivalent to

```
select I1.Nome  
      from Impiegato I1, Impiegato I2  
      where I1.Nome = I2.Cognome
```

Nome

Franco

# Interrogazioni di tipo insiemistico

- Interrogazione: Estrarre i nomi degli impiegati che non sono cognomi di qualche impiegato

```
select Nome,  
      from Impiegato  
  except  
  select Cognome  
      from Impiegato
```

Nome
Mario
Carlo
Giovanni
Lorenzo
Paola
Marco

Equivalent to

```
SELECT Nome  
FROM Impiegato  
WHERE Nome not in (SELECT Cognome  
                      FROM Impiegato)
```

non presente  
in  
Query nm and query

# Interrogazioni nidificate

---

- In SQL è possibile scrivere interrogazioni che presentano al loro interno altre interrogazioni.
- La nidificazione può avvenire in tutte e tre le clausole (Select, From, Where)
  - Where è la più comune *all'interno del where: un any è un predicato che dà vero se succede quello.*
  - any: il predicato è vero se viene restituita almeno una riga
  - all: il predicato è vero se tutte le righe vengono restituite

# Interrogazioni nidificate

---

- In SQL è possibile scrivere interrogazioni che presentano al loro interno altre interrogazioni.
- La nidificazione può avvenire in tutte e tre le clausole (Select, From, Where)
  - Where è la più comune
- In pratica abbiamo delle condizioni che si basano sul risultato di altre interrogazioni

# Interrogazioni nidificate: Esempio - BD riferimento

Imp

<u>CodImp</u>	Nome	Sede	Ruolo	Stipendio
E001	Rossi	S01	Analista	2000
E002	Verdi	S02	Sistemista	1500
E003	Bianchi	S01	Programmatore	1000
E004	Gialli	S03	Programmatore	1000
E005	Neri	S02	Analista	2500
E006	Grigi	S01	Sistemista	1100
E007	Violetti	S01	Programmatore	1000
E008	Aranci	S02	Programmatore	1200

Sedi

<u>Sede</u>	Responsabile	Citta
S01	Biondi	Milano
S02	Mori	Bologna
S03	Fulvi	Milano

Prog

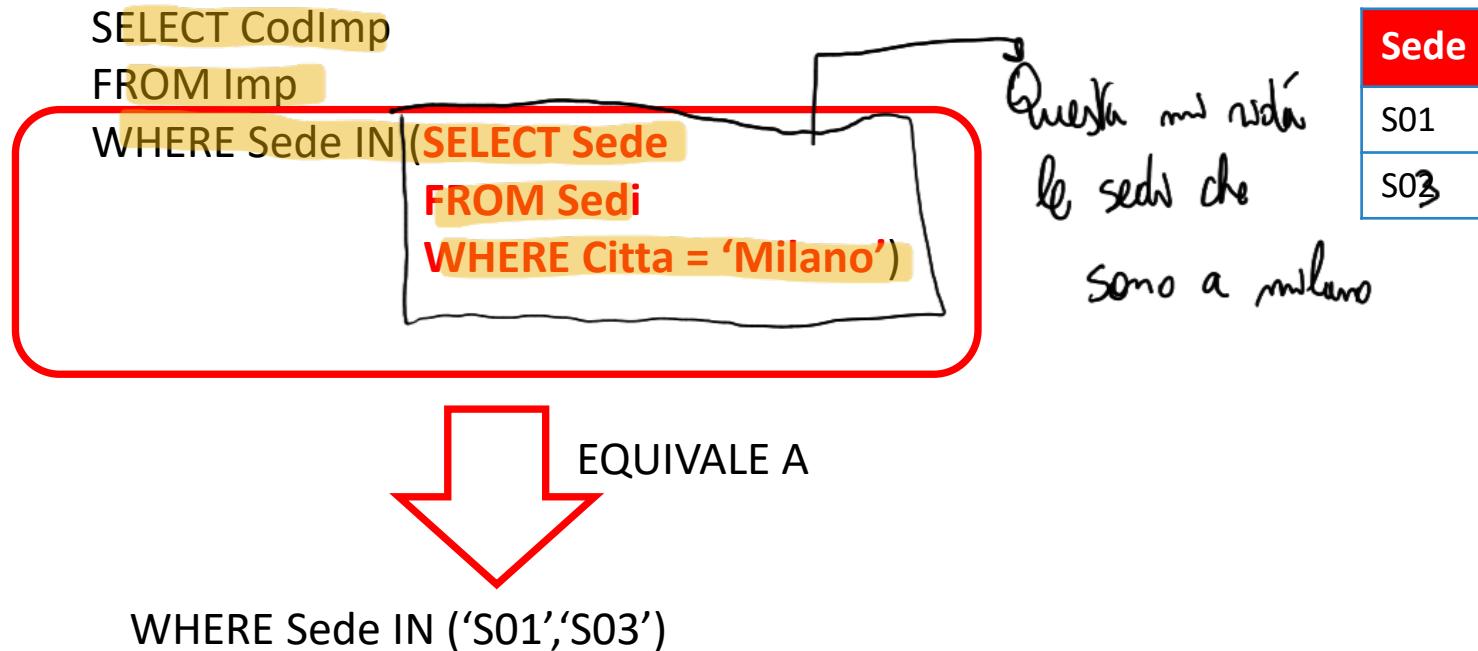
<u>CodProg</u>	Citta
P01	Milano
P01	Bologna
P02	Bologna

Vincolo di legge/ruolo referenziale

Leygue su campo non chiave

# Interrogazioni nidificate: Esempio

- Interrogazione: Impiegati delle sedi di Milano



# Interrogazioni nidificate: Esempio

- Nel caso generale è possibile confrontare tra loro tuple, anziché singoli valori (questo si può fare anche senza avere subquery!)

```
SELECT CodImp  
FROM Imp  
WHERE Sede <> 'S01'  
AND (Ruolo, Stipendio) IN  
      (SELECT Ruolo, Stipendio  
       FROM Imp  
       WHERE Sede = 'S01')
```

CodImp
E004

- La query trova gli impiegati delle sedi S02 e S03 che hanno stesso ruolo e stipendio di qualche impiegato della sede S01

# Interrogazioni nidificate ‘Scalari’

- Gli operatori di confronto  $=, <,...$  si possono usare solo se la subquery restituisce non più di una tupla (subquery “scalare”)
  - Interrogazione: Impiegati con stipendio minimo

```
SELECT CodImp  
FROM Imp  
WHERE Stipendio = (SELECT MIN(Stipendio)
```

↑  
**FROM Imp**)

*Se restituisce un solo valore posso ragionare così.*

CodImp
E004

# Interrogazioni nidificate ‘Scalari’

---

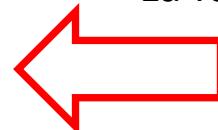
- ATTENZIONE : Se in un predicato si confronta un attributo con il risultato di un'interrogazione, sorge il problema di disomogeneità dei termini del confronto. Infatti, da una parte si ha il risultato di un'interrogazione SQL (in generale un insieme di valori), mentre dall'altra abbiamo il valore di un attributo per la particolare riga
- Tale problema viene risolto da SQL tramite l'utilizzo di alcune parole chiave (all, any, in, not in, exists, not exists) che estendono i normali operatori di confronto relazionale (=, <>, <, >, <=, >=)

# Interrogazioni nidificate ‘Scalari’

- Se la subquery può restituire più di un valore si devono usare le forme
  - <op> ANY**: la relazione **<op>** vale per almeno uno dei valori
  - <op> ALL** : la relazione **<op>** vale per tutti i valori

```
SELECT Responsabile  
FROM Sedi  
WHERE Sede = ANY (SELECT Sede  
                  FROM Imp  
                  WHERE Stipendio > 1500)
```

La forma = **ANY** equivale a **IN**



- Interrogazione: **Impiegati con stipendio minimo**

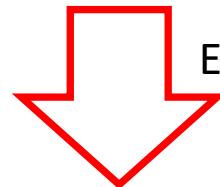
```
SELECT CodImp  
FROM Imp  
WHERE Stipendio <= ALL (SELECT Stipendio  
                         FROM Imp)
```

# Interrogazioni nidificate: Esempio

- Interrogazione: Estrarre gli impiegati che lavorano in dipartimenti situati a Firenze

```
SELECT *  
FROM Impiegato  
WHERE Dipart = ANY (SELECT Nome  
                    FROM Dipartimento  
                    WHERE Città = 'Firenze')
```

Selezione le righe di **Impiegato** per cui il valore di **Dipart** è uguale ad almeno uno dei valori dell'attributo **Nome** delle righe di **Dipartimento**



EQUIVALENTE A

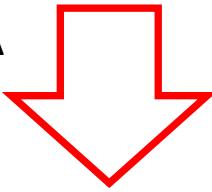
```
SELECT *  
FROM Impiegato, Dipartimento D  
WHERE Dipart = D.Nome AND D.Città = 'Firenze'
```

# Interrogazioni nidificate: Esempio

- Interrogazione: Estrarre i dipartimenti in cui non lavorano persone di cognome 'Rossi'

```
SELECT Nome  
FROM Dipartimento  
WHERE Nome <> all (SELECT Dipart  
                      FROM Impiegato  
                      WHERE Cognome = 'Rossi')
```

EQUIVALENTE A



```
SELECT Nome  
FROM Dipartimento  
except  
Select Dipart  
From Impiegato  
Where Cognome = 'Rossi'
```

L'interrogazione nidificata seleziona i valori di Dipart di tutte le righe in cui il cognome vale 'Rossi'

La condizione è quindi soddisfatta da quelle righe di Dipartimento per cui il valore di Nome non fa parte dai nomi prodotti dalla subquery

$$(\pi_{Nome}(Dipartimento) - \pi_{Dipart}(\sigma_{Cognome = 'Rossi'}(Impiegato)))$$

# Interrogazioni nidificate

---

- Per rappresentare l'appartenenza o l'esclusione
- Operatore `in` è identico a `=any`
- Operatore `not in` è identico a `<> all`

# Interpretazione semplice

---

- Un'interpretazione molto semplice delle interrogazioni nidificate consiste nell'assumere che l'interrogazione nidificata (o interna) venga eseguita prima di analizzare le righe dell'interrogazione esterna
- Si può ipotizzare che il risultato dell'interrogazione nidificata venga salvato in una tabella temporanea; il controllo sulle righe dell'interrogazione esterna può essere fatto accedendo direttamente al risultato temporaneo
- Questa interpretazione (detta semplice), in cui l'interrogazione nidificata viene eseguita una sola volta, è corretta nel caso in cui le variabili di range definite nell'interrogazione più esterna non vengano utilizzate nell'ambito dell'interrogazione più interna

# Interpretazione semplice: Esempio

- Considerando l'interrogazione precedente

```
SELECT Nome  
FROM Dipartimento  
WHERE Nome <> all (SELECT Dipart  
                      FROM Impiegato  
                      WHERE Cognome = 'Rossi')
```

1.

Il sistema può eseguire dapprima l'interrogazione nidificata che estrae il valore dell'attributo Dipart per tutti gli impiegati di cognome "Rossi"

2.

A questo punto, per ciascun dipartimento, si controlla che il valore dell'attributo Nome non sia incluso nella tabella prodotta, utilizzando l'operatore <> all

# Interrogazioni con correlazione

- Talvolta l'interrogazione nidificata fa riferimento al contesto dell'interrogazione esterna che la racchiude; tipicamente ciò accade tramite una variabile di range definita nell'interrogazione più esterna ed usata nell'ambito della interrogazione nidificata
- Si parla in questo caso di Interrogazioni nidificate con correlazione
- ESEMPIO : Estrarre gli impiegati che hanno degli omonimi (stesso nome e cognome, ma diverso codice fiscale) con relazione Persona(CodFiscale, Nome, Cognome, Città)

```
SELECT *  
FROM Persona P  
WHERE exists (SELECT *  
              FROM Persona P1  
              WHERE P1.Nome = P.Nome AND  
                    P1.Cognome=P.Cognome AND  
                    P1.CodFiscale <> P.CodFiscale)
```

l'operatore **exists** ammette come parametro un'interrogazione nidificata e restituisce il valore VERO solo se l'interrogazione nidificata fornisce un risultato non vuoto

# Interrogazioni con correlazione

---

- Nel caso di interrogazioni nidificate con correlazione, l'interpretazione semplice fornita precedentemente non è più valida
- In questo caso è necessario che l'interrogazione nidificata venga valutata separatamente per ogni riga prodotta nella valutazione dell' interrogazione esterna. La nuova interpretazione è la seguente:
  - per ogni riga esaminata nell'ambito dell'interrogazione esterna, si deve valutare l'interrogazione nidificata (che quindi, in questo caso, non può essere calcolata a priori, ma deve essere ricalcolata per ogni riga dell'interrogazione esterna)
  - tale processo può essere ripetuto un numero arbitrario di volte, pari al numero arbitrario di nidificazioni che possono essere utilizzate nell' interrogazione

# Interrogazioni con correlazione

- ATTENZIONE : Per quanto riguarda la visibilità delle variabili di range, vale la restrizione che una variabile è usabile solo nell'ambito dell'interrogazione in cui è definita o nell'ambito di un'interrogazione nidificata (a qualsiasi livello) all'interno di essa

```
SELECT *  
FROM Persona P  
WHERE exists (SELECT *  
              FROM Persona P1  
              WHERE P1.Nome = P.Nome AND  
                    P1.Cognome=P.Cognome AND  
                    P1.CodFiscale <> P.CodFiscale)
```

L'interrogazione nidificata utilizza una **variabile di range** nell' interrogazione più esterna. In questo caso, per ogni riga esaminata nell'ambito dell' interrogazione esterna, si deve valutare l'interrogazione nidificata.

Nell'esempio vengono considerate una ad una le righe della variabile I; per ognuna di queste righe, viene eseguita l'interrogazione nidificata che restituisce o meno l'insieme vuoto a seconda che vi siano o meno degli omonimi della persona

# Interrogazioni nidificate not exists

---

- ESEMPIO : Estrarre gli impiegati che non hanno degli omonimi

```
SELECT *
FROM Persona P
WHERE not exists (SELECT *
                   FROM Persona P1
                   WHERE P1.Nome = P.Nome AND
                         P1.Cognome=P.Cognome AND
                         P1.CodFiscale <> P.CodFiscale
```

# Interrogazioni nidificate – Esempio

Questa interrogazione è corretta?

```
SELECT Dipart
FROM Impiegato
WHERE Dipart in (SELECT Nome
                  FROM Dipartimento D1
                  WHERE Nome = 'Produzione') or
          Dipart in (SELECT Nome
                  FROM Dipartimento D2
                  WHERE D1.Città = D2.Città)
```

**NO** -> se un'interrogazione possiede sotto-interrogazioni annidate allo stesso livello, le variabili introdotte nella clausola `FROM` di una interrogazione non potranno essere usate nell'ambito di un'altra interrogazione allo stesso livello (mentre potranno essere usate in interrogazioni situate a livelli inferiori)

# Interrogazioni nidificate – Esempio

ESEMPIO : Date le relazioni Cantante ed Autore in figura, estrarre i cantautori, ovvero i cantanti che hanno eseguito alcune canzoni di cui erano anche autori

- Cantante(Nome,Canzone)
- Autore(Nome,Canzone)

```
SELECT Nome  
FROM Cantante  
WHERE Nome not in  
      (SELECT Nome  
       FROM Cantante C  
       WHERE Nome not in  
             (SELECT Nome  
              FROM Autore  
              WHERE Autore.Canzone = C.Canzone))
```

La prima interrogazione nidificata non ha alcun legame con l'interrogazione esterna e quindi può essere eseguita in modo del tutto indipendente

La seconda interrogazione nidificata presenta invece un legame con l'interrogazione esterna ( $\text{Autore.Canzone} = \text{C.Canzone}$ ).... Quindi....

# Interrogazioni nidificate – Esempio

---

- L'interrogazione relativa all'esempio precedente avviene seguendo queste fasi
  1. L'interrogazione (SELECT Nome FROM Cantante C...) legge tutte le righe della tabella Cantante
  2. Per ognuna delle righe di C viene valutata l'interrogazione più interna (SELECT Nome FROM Autore...), che restituisce i nomi degli autori della canzone il cui titolo compare nella riga di C che viene considerata. Se il nome del cantante non compare tra gli autori (quindi non è un cantautore puro), allora il nome viene selezionato
  3. Dopo che l'interrogazione nidificata ha terminato di analizzare le righe di C (costruendo la tabella contenente i nomi dei cantanti che non sono cantautori puri), viene eseguita l'interrogazione più esterna, la quale restituirà tutti i nomi di cantanti che non compaiono nella tabella ottenuta come risultato dell'interrogazione nidificata

# Modifica dei dati in SQL - Inserimento

---

- **insert into NomeTabella [ListaAttributi]**

**<values (ListaValori) | SelectSQL>**

- Esempio forma 1:

- **insert into Dipartimento(Nome,Citta) values('prod','Milano')**

- Esempio forma 2:

- **insert into ProdottiMilanesi (select Codice, Descrizione**

**from Prodotti**

**where LuogoProd='Milano')**

↑ Se non mettersi prod, allora viene messo Null o default

# Modifica dei dati in SQL - Inserimento

---

- Se in un inserimento non vengono specificati i valori di tutti gli attributi della tabella, a quelli mancanti viene assegnato il valore di default o il valore NULL.
- È importante l'ordine della ListaAttributi con la ListaValori

# Modifica dei dati in SQL - Cancellazione

---

- **delete from NomeTabella [where Condizione]**
- Se la condizione where manca allora la tabella viene **svuotata** (mantiene lo schema).
  - **delete from Dipartimento**  
*es: Impiegati cancella sede in shde 63*
- **Invece:**  
**drop table Dipartimento cascade** (**elimina schema e contenuto**)  
  
*elimina la tabella ed in più, a causa dell' opzione cascade, tutti i componenti dello schema che ad essa fanno riferimento).*

# Modifica dei dati in SQL - Modifica

---

**update NomeTabella**

Modifichiamo nel database, non la tabella (quello era alter)

**set Attributo = <Espressione | SelectSql | null | default>**

{,Attributo=<Espressione | SelectSql | null | default>}

**[where Condizione]**

- Problema. Si voglia aumentare del 10% lo stipendio degli impiegati che guadagnano meno di 30 e del 15% gli stipendi superiori.
  - **update impiegato set stipendio = stipendio\*1.1 where stipendio <= 30**
  - **update impiegato set stipendio = stipendio\*1.15 where stipendio > 30**

---

# ESERCIZI

# Esercizio 1

---

Dare le definizioni SQL delle tre tabelle

**FONDISTA**(Nome, Nazione, Età)

**GAREGGIA**(NomeFondista, NomeGara, Piazzamento)

**GARA**(Nome, Luogo, Nazione, Lunghezza)

rappresentando in particolare i vincoli di foreign key della tabella GAREGGIA.

# Esercizio 1

---

```
Create Table FONDISTA
```

```
(  
    Nome character(20) primary key,  
    Nazione character(30),  
    Età smallint  
)
```

```
Create table GARA
```

```
(  
    Nome character(20) primary key,  
    Luogo character(20),  
    Nazione character(20),  
    Lunghezza integer  
)
```

```
Create table GAREGGIA
```

```
(  
    NomeFondista character(20) references FONDISTA(Nome),  
    NomeGara character(20),  
    Piazzamento smallint,  
    primary key (NomeFondista, NomeGara),  
    foreign key (NomeGara) references GARA(Nome)  
)
```

# Esercizio 2

---

Dare le definizioni SQL delle due tabelle

**AUTORE** (Nome, Cognome, DataNascita, Nazionalità)

**LIBRO** (TitoloLibro, NomeAutore, CognomeAutore, Lingua)

Per il vincolo foreign key specificare una politica di cascade sulla cancellazione e di set null sulle modifiche.

# Esercizio 2

---

```
Create table LIBRO
(
    TitoloLibro character(30) primary key,
    NomeAutore character(20),
    CognomeAutore character(20),
    Lingua character(20),
    foreign key (NomeAutore, CognomeAutore) references AUTORE(Nome, Cognome)
                                                on delete cascade
                                                on update set NULL
)
```

```
Create table AUTORE
(
    Nome character(20),
    Cognome character(20),
    DataNascita date,
    Nazionalità character(20),
    primary key (Nome, Cognome)
)
```

# Esercizio 3

---

Dato il seguente schema relazionale che descrive il calendario di una manifestazione sportiva a squadre nazionali

**STADIO(Nome, Città, Capienza)**

**INCONTRO(NomeStadio, Data, Ora, Squadra1, Squadra2)**

**NAZIONALE(Paese, Continente, Categoria)**

Esprimere le seguenti interrogazioni in SQL

- Estrarre i nomi degli stadi in cui non gioca nessuna nazionale europea
- Estrarre la capienza complessiva degli stadi in cui si giocano le partite che hanno come prima squadra una nazione sudamericana (ai fini della valutazione della capienza complessiva si sommino le capienze associate a ciascuna gara, anche se più gare si svolgono nello stesso stadio)

# Esercizio 3

Estrarre i nomi degli stadi in cui non gioca nessuna nazionale europea

```
select Nome
from Stadio
where Nome not in (select NomeStadio
                    from Incontro
                    where (Squadra1 in
                            (select Paese
                                from Nazionale
                                where Continente = 'Europa'))
                    or
                    (Squadra2 in
                            (select Paese
                                from Nazionale
                                where Continente = 'Europa'))
```

# Esercizio 3

Estrarre la capienza complessiva degli stadi in cui si giocano le partite che hanno come prima squadra una nazione sudamericana (ai fini della valutazione della capienza complessiva si sommino le capienze associate a ciascuna gara, anche se più gare si svolgono nello stesso stadio)

```
select sum(Capienza)
from Stadio join Incontro on Nome = NomeStadio
Where Squadra1 in (select Paese
                    from Nazionale
                    where Continente = 'Sudamerica' )
```

# Esercizio 4

---

Dato il seguente schema relazionale

**MOTO(Targa, Cilindrata, Marca, Nazione, Tasse)**  
**PROPRIETARIO(Nome, Targa)**

Esprimere le seguenti interrogazioni in SQL

- a) Estrarre i nomi dei proprietari di solo moto giapponesi di almeno due marche diverse

# Esercizio 4

Estrarre i nomi dei proprietari di solo moto giapponesi di almeno due marche diverse

```
select Nome
from Proprietario join Moto on Proprietario.Targa = Moto.Targa
where Nome not in (select Nome
                    from Proprietario join Moto on Proprietario.Targa = Moto.Targa
                    where Nazione <> 'Giappone')
group by Nome
having count(distinct Marca) >= 2
```

# Esercizio 4

Estrarre i nomi dei proprietari di solo moto giapponesi di almeno due marche diverse

```
select P1.Nome  
from Proprietario P1, Moto M1, Proprietario P2, Moto M2  
where P1.Nome not in (select Nome  
                      from Proprietario join Moto on Proprietario.Targa = Moto.Targa  
                      where Nazione <> 'Giappone') and  
      P1.Targa = M1.Targa and  
      P2.Targa = M2.Targa and  
      P1.Nome = P2.Nome and  
      M1.Marca <> M2.Marca
```

# Esercizio 5

---

Dato il seguente schema relazionale

NEGOZI (IDNegozio, Nome, Città)

PRODOTTI (CodProdotto, NomeProdotto, Marca)

LISTINO (Negozio, Prodotto, Prezzo)

Esprimere le seguenti interrogazioni in SQL

1. l'interrogazione che fornisce nome e città dei negozi che vendono prodotti della marca ABC
2. l'interrogazione che trova, per ciascun prodotto, la città in cui viene venduto al prezzo più basso
3. l'interrogazione che trova i prodotti che vengono venduti in una sola città

# Esercizio 5

---

l'interrogazione che fornisce nome e città dei negozi che vendono prodotti della marca ABC

```
select distinct Nome, Citta  
from Negozi, Prodotti, Listino  
where IDNegozio = Negozio and Prodotto = CodProdotto and Marca = 'ABC';
```

# Esercizio 5

l'interrogazione che trova, per ciascun prodotto, la città in cui viene venduto al prezzo più basso

```
select distinct P.NomeProdotto, P.Citta  
from Negozi, Listino, Prodotti P  
where prezzo <= all (select Prezzo, NomeProdotto  
                      from Prodotti P2, Listino  
                      where P.NomeProdotto = P2.NomeProdotto);
```

# Esercizio 5

---

l'interrogazione che trova i prodotti che vengono venduti in una sola città

```
select distinct P1.NomeProdotto
from Prodotti P1
where NomeProdotto not in (select P1.NomeProdotto
                             from Negozi N1, Negozi N2, Prodotti P2, Listino L1, Listino L2
                             where N1.IdNegozio = L1.Negozio
                                   and N2.IdNegozio = L2.Negozio
                                   and L1.Prodotto = L2.Prodotto
                                   and L1.Prodotto = P2.CodProdotto
                                   and N1.Citta <> N2.Citta);
```