😷

Capitolo 4 (SQL: concetti base)

SQL

Linguaggio con varie funzionalità:

- Contiene sia il DDL che il DML

Il linguaggio SQL è **semi-dichiarativo**, basato in parte sull'algebra relazionale ed in parte sul calcolo relazionale

Può essere usato **interattivamente** (con maschere del DBMS) o essere **incorporato** in programmi C, Java, ecc.

Storia

Venne inizialmente proposto **SEQUEL** (1974): Structured English Query Language, definito all'IBM Research, basato su termini inglesi che mascherano i difficili concetti dell'algebra relazionale

Prime implementazioni nei DBMS Oracle (Relational Software, Inc. 1979), SQL/Data System (IBM 1981) e DBS (IDBM 1983)

Dal 1983 SQL è diventato "standard di fatto"

Versioni

SQL è implementato dai principali fornitori di DBMS, ed è il linguaggio per database più usato al mondo

L'ANSI e l'ISO hanno sviluppato una serie di standard per SQL

- ANSI SQL-86, SQL2 (1992), SQL3(1999)
- Tuttavia ogni DBMS relazionale, implementa un **dialetto** di SQL che è un'estensione o un sottoinsieme di SQL standard

Definizione di schemi in SQL

- `CREATE SCHEMA nome_schema AUTHORIZATION nome_utente`
  - Crea uno schema `nome_schema`, il cui proprietario è l'utente con account `nome_utente`
- `DROP SCHEMA nome_schema drop-behaviour`
  - Elimina lo schema chiamato `nome_schema`, l'opzione `drop_behaviour` può assumere i valori `CASCADE` o `RESTRICT`
    - `DROP SCHEMA COMPANY CASCADE`: Lo schema del database COMPANY viene rimosso con tutte le tabelle, domini ed altri elementi
    - `DROP SCHEMA COMPANY RESTRICT`: Lo schema viene eliminato solo se non contiene elementi

Creazione del database

Creazione di uno schema:

```
CREATE SCHEMA [NomeSchema] [[authorization]Autorizzazione]
               {DefElementoSchema}
```

- `Autorizzazione` reppresenta il nome dell'utente proprietario dello schema; se viene omesso si assume che il proprietario sial'utente che ha lanciato il comando. Se il nome è omesso , si assume come nome dello schema il nome del proprietario

Definizione delle tabelle

`CREATE TABLE`

    - Una tabella SQL è costituita da una collezione di attributi e da un insieme (eventualmente vuoto) di vincoli
- Definisce uno schema di relazione e ne crea un’istanza vuota
- Specifica attributi, domini e vincoli

```
CREATE TABLE Nometabella (
    NomeAttributo Dominio [ValoreDiDefault] [Vincoli]
    {, NomeAttributo Dominio [ValoreDiDefault] [Vincoli]

    AltriVincoli
    }
)
```

Esempio:

```
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,
    FOREIGN KEY(Dipart) REFERENCES Dipartimento(NomeDip),
    UNIQUE(Cognome, Nome)
)
```


Vincoli:

- Matricola è chiave primaria
- Nome non può avere valore nullo
- Cognome non può essere avere valore nullo
- Se non inserisco il valore di Stipendio verrà impostato a 0 dal DBMS
- Vincolo di integrità referenziale tra Dipart di Impiegato e NomeDip in Dipartimento (automaticamente Dipart non potrà

avere valore nullo perché il vincolo dovrà essere rispettato, a meno che non imposti la possibilità di essere nullo)

- `UNIQUE(Cognome, Nome)` non ci possono essere due persone che hanno lo stesso nome e cognome, tuttavia possono avere stesso nome oppure stesso cognome

Domini

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

Elementari:

- **Carattere**: singoli caratteri o stringhe, anche di lunghezza variabile (così che il DBMS ottimizzi l'utilizzo di spazio)
- **Bit:** singoli booleani o stringhe
- **Numerici:** esatti o approssimati (prevedono parte decimale)
- **Data, ora, intervalli di tempo**
- Introdotti in SQL 1999:
  - **Boolean**
  - **BLOB, CLOB:** binary/character large object, per grandi immagini e testi

Domini in SQL2, numeri e stringhe:

- **Numerici esatti:**

```
numeric [(Precisione (i) [,Scala (j))]
decimal [(Precisione [, Scala])]
integer
smallint
```

`numeric` e `decimal` rappresentano numeri in **base decimae**

- Interi (`INTEGER`, `SMALLINT`)
- Numeri formattati (`DECIMAL(i, j)`, `DEC(i,j)`, `NUMERIC(i,j)`)
  - `i`, detta precisione, indica il numero di cifre significative
  - `j`, detta scala, indica il numero di cifre dopo la virgola
  - Per specificare la scala, bisogna specificare anche la precisione

- **Numerici approssimati**
  - Reali (`FLOAT`, `REAL`, `DOUBLE PRECISION`)

```
float [(Precisione)]
real
double precision
```

- **Stringhe di caratteri:**
  - A lunghezza fissa (`CHAR(n)`, `CHARACTER(n)`)
  - A lunghezza variabile (`VARCHAR(n)` o `CHAR VARYING(n)`)
    - Per default `n`, il numero massimo di caratteri, è 1
- **Stringhe di bit:**
  - A lunghezza fissa (`BIT(n)`)
  - A lunghezza variabile (`BIT VARYING(n)`)

Domini in SQL2, date e orari:

```
date
time [(Precisione)] [with time zone]
timestamp[(Precisione)][with time zone]
```

- `DATE`
  - Ha dieci posizioni, con componenti `YEAR`, `MONTH` e `DAY`. Formato YYYY-MM-DD
- `TIME`
  - Ha (almeno) otto posizioni con componenti `HOUR`, `MINUTE` e `SECOND`. Formato HH:MM:SS
- `TIME(i)`
  - `i` = precisione delle frazioni di secondo, specifica `i + 1` posizioni aggiuntive per `TIME`, una per il separatore ed `i` per le frazioni di secondo
- `TIME WITH TIME ZONE`
  - Usa ulteriori 6 posizioni per lo spiazzamente dal `GMT`, con un range da +13:00 a -12:59

Valori di default e definizione di nuovi domini

Se per un attributo di una ennupla non viene fornito un valore, il DBMS assegna un valore di default (solitamente NULL)

La clausola `DEFAULT <valore>` affianco alla definizione di attributo consente di specificare un diverso valore di default

Invece l'istruzione `CREATE DOMAIN` definisce un dominio (semplice) utilizzabile in definizioni di relazioni, anche con vincoli e valori di default

```
CREATE DOMAIN NomeDominio as TipoDiDato
             [ValoreDiDefault]
             [Vincolo] (
```

Esempio:

```
CREATE DOMAIN Voto
    AS SMALLINT DEFAULT NULL
```

```
    CHECK (value >= 18 AND value <= 30)
```

La clausola `CHECK` sta impostando un vincolo di dominio, vi è un insieme di valori ammissibili per per un oggetto

Specifica dei valori di default

Il `ValoreDiDefault` permette di specificate il valore che l'attributo deve assumete quando viene inserita una riga nella tabella senza che sia specificato il valore per l'attributo

Quando il default non è specificato allora si assume come default il valore `null`

```
default <GenericoValore |user|null>
```

- `GenericoValore` è in valore compatibile con il dominio
- `user` imposta come valore di default l'identificativo dell'utente

Vincoli intrarelazioni e interrelazionali

Nella definizione delle tabelle è possibile definire dei vincoli, ovvero delle proprietà che devono esser verificate da ogni istanza della base di dati

Li distinguiamo in:

- Vincolo intrarelazionali
- Vincoli interrelazionali

Vincoli intrarelazionali (di dominio ed ennupla)

- `NOT NULL`

- Indica che il valore `null` ènon è ammesso come valore dell'attributo; il valore dell'attributo deve essere sempre specificato

- `UNIQUE`
  - Si applica ad un attributo o un insieme di attributi e va a definire chiavi candidate: righe differenti non possono avere gli stessi valori
    - Se bisogna definirlo solo su un attributo allora si aggiunge la clausola `UNIQUE` di fianco all'attributo
    - Se bisogna definirlo su più attributi si usa la sintassi:

      ```
      unique (Attributo {, Attributi})
      ```

- `PRIMARY KEY`
  - Di norma, per ogni relazione c'è bisogno di definire la **chiave primaria**. Si specifica il vincolo primary key una volta per ogni tabella(mentre si possono usare un numero arbitrario di vole i vincoli `UNIQUE` e `NOT NULL`
  - Può essere definito su un singolo attributo oppure su un elenco di attributi (come unique)
  - Gli attributi che fanno parte della chiave primaria non possono assumere valore nullo: la definizione di primary key implica per tutti gli attributi della chiave una definizione `NOT NULL`

    Esempio:

    ```
    Nome VARCHAR(20),
    Cognome VARCHAR(20),
    PRIMARY KEY (Cognome, Nome)
    ```

Vincoli interrelazionali

- `REFERENCES` e `FOREIGN KEY:`
  - Quelli più diffusi, come già abbiamo detto, sono i *vincoli di integrità referenziale.* Per la loro definizione si usa il vincolo `FOREIGN KEY`, cioè *chiave esterna*
  - Questo vincolo crea un legame tra i valori di un attributo di una tabella su cui è definito (chiamata **interna**) e i valori di un attributo di un'altra tabella (chiamata **esterna**)
    - Il vincolo impone che in ogni riga della tabella interna il valore dell'attributo specificato, se diverso da null, sia presente nelle righe della tabella esterna tra i valori del corrispondente attributo
    - L'unico requisito che è richiesto per l'attributo a cui si fa riferimento nella tabella esterna è che sioa soggetto ad un vincolo `UNIQUE`; molto spesso, infatti, si fa rifermento alla chiave primaria della tabella esterna
  - Se c'è un solo attributo coinvolto, si può usare `REFERENCES`: con esso si specificano la tabella esterna e l'attributo della tabella esterna al quale l'attributo in questione deve essere legata


```
CREATE TABLE Impiegato (
    Matricola character(6) primary key,
    Nome varchar(20) not null,
    Cognome varchar(20) not null,
    Dipart varchar(15)
           REFERENCES Dipartimento (NomeDip),
    Ufficio numeric(3),
    Stipendio numeric(9) default 0,
    unique (Cognome,Nome)
)
```

- Se ci sono più attributi coinvolti, si può usare il costrutto `FOREIGN KEY`, posto al termine della definizione degli attributi. Segue poi il costrutto `REFERENCES` per la definizione della tabella esterna

```
FOREIGN KEY (Nome, Cognome)
           REFERENCES Anagrafica(Nome, Cognome)
```

- Si possono definire politiche di reazione:
  -

Modifiche degli schemi

`ALTER DOMAIN`: modifica un dominio

`ALTER TABLE`: modifica la tabella

`DROP DOMAIN`: cancella il dominio

`DROP TABLE`: cancella la tabella

ALTER TABLE: aggiunta di un attributo

Vogliamo aggiungere l'email di un impiegato nella tabella impiegato:

```
ALTER TABLE Impiegato ADD email VARCHAR(12)
```

Il valore di `email` o si specifica di default o sarà NULL

Con la `ALTER TABLE` non è permessa la clausola `NOT NULL`

ALTER TABLE: eliminazione di un attributo

Quando si elimina una colonna occorre scegliere l'opzione `CASCADE` o `RESTRICT`

- Con `CASCADE` vincoli e viste che referenziano la colonna sono eliminati dallo schema
- Con `RESTRICT` il comando ha successo solo se nessun vincolo o vista referenzia la colonna

Esempio: rimuovere la colonna indirizzo dalla tabella Impiegato:

```
ALTER TABLE Impiegato DROP indirizzo CASCADE
```

ALTER TABLE: modifica vincoli

Modifica di una colonna eliminando una clausola di default o definendone una nuova

Esempi:

```
ALTER TABLE Impiegato ALTER dipart DROP DEFAULT
```

```
ALTER TABLE Infrazione ALTER vigile SET DEFAULT "3334"
```


Si può eliminare un vincolo solo se gli si è dato un nome nella `CREATE TABLE` tramite la keyword `CONSTRAINT`

Definizione degli indici

Sono rilevanti dal punto di vista delle prestazioni

Sono presenti a livello fisico e non logico

In passato erano importanti perché in alcuni sistemi erano l’unico mezzo per definire chiavi

Per creare un indice: `CREATE INDEX on <attributo>`

Per cancellare un indice: `DROP INDEX <attributo>`

DDL, in pratica

In molti sistemi si utilizzano strumenti diversi dal codice SQL per definire lo schema della base di dati

SQL, operazioni sui dati

Interrogazione:

- `SELECT`

Inserimento, cancellazione e aggiornamento di ennuple:

- `INSERT`, `DELETE`, `UPDATE`

Operazioni di aggiornamento

Si può operare sia su singole ennuple che su gruppi di ennuple di una relazione

```
-- Le parentesi quadre dicono che l'attributo è opzionale
INSERT INTO Tabella [(Attributi)]
  VALUES(Valori)
```

Durante l’inserimento è necessario seguire l’ordine di assegnazione, inserire uno spazio vuoto per lasciare i valori di

default

Si può usare un sistema non posizionale specificando il nome degli attributi

Per inserire insiemi di ennuple la forma alternativa è:

```
INSERT INTO Tabella ([ATTRIBUTI])
  SELECT -- Proviene da un'interrogazione
```

Esempio di inserimento:

```
-- Sistema posizionale
INSERT INTO Persone VALUES('Mario', 25, 52)
```

```
-- Sistema non posizionale
INSERT INTO Persone(Nome, Eta, Reddito) VALUES('Pino', 25, 52)
```

```
-- Eta verrà impostato al valore di default
INSERT INTO Persone(Nome, Reddito) VALUES('Lino',55)
```


SQL, operazioni sui dati

Interrogazione:

- `SELECT`

Modifica:

- `INSERT`, `DELETE`, `UPDATE`

Istruzione SELECT (versione base)

```
SELECT ListaAttributi FROM ListaTabelle [WHERE Condizione]
```

Le due clausole `SELECT` e `FROM` sono obbligatorie

`WHERE` è una clausola opzionale

Persone

Nome	Età	Reddito
Andrea	27	21
Aldo	25	15
Maria	55	42
Anna	50	35
Filippo	26	30
Luigi	50	40
Franco	60	20
Olga	30	41
Sergio	85	35
Luisa	75	87

Maternità

Madre	Figlio
Luisa	Maria
Luisa	Luigi
Anna	Olga
Anna	Filippo
Maria	Andrea
Maria	Aldo

Paternità

Padre	Figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

Selezione e proiezione di nome e reddito delle persone con meno di trenta anni:

$PROJ_{Nome,Reddito}(SEL_{Eta<30}(Persone))$

```
SELECT nome, reddito
FROM persone
```

```
WHERE eta < 30
```

SELECT, abbreviazioni

Durante la selezione è possibile specificare degli alias

```
SELECT p.nome as nome, p.reddito as reddito
FROM persone p
WHERE p.eta < 30
```

Selezione, senza proiezione

Nome, età e reddito delle persone con meno di trenta anni

$SEL_{Eta<30}(Persone)$

```
SELECT *
FROM persone
WHERE eta < 30
```


Proiezione, senza selezione

Nome e reddito di tutte le persone

$PROJ_{Nome,Reddito}(Persone)$

```
SELECT nome, reddito
FROM persone
```

Espressioni nella target list

```
SELECT reddito/2 as redditoSemestrale
FROM Persone
WHERE Nome = 'Luigi'
```

Condizione complessa

```
SELECT *
FROM Persone
WHERE reddito > 25 AND (eta < 30 or eta > 60)
```

Condizione LIKE

Le persone che hanno un nome che inizia per ‘A’ e ha una ‘d’ come terza lettera

Viene utilizzata per fare confronti approssimati tra stringhe

```
SELECT *
FROM persone
WHERE nome LIKE 'A_d%'
-- Underscore significa qualsiasi carattere
-- % significa che ci possono essere 0 caratteri o un
-- numero di caratteri di qualsiasi tipo
```


Gestione dei valori nulli

Impiegati

Matricola	Cognome	Filiale	Età
5998	Neri	Milano	45
9553	Bruni	Milano	NULL

Gli impiegati la cui età è o potrebbe essere maggiore di 40

$SEL_{Eta>40\ OR\ Eta\ IS\ NULL}(Impiegati)$

```
SELECT *
FROM impiegati
WHERE eta > 40 OR eta IS NULL
```

Esiste anche una condizione `IS NOT NULL`

Selezione, proiezione e join

Istruzioni `SELECT` con una relazione nella clausola `FROM` permettono di realizzare:

- Selezioni, proiezioni, ridenominazioni

Con più relazioni nella `FROM` si realizzano join (e prodotti cartesiani)

$R1(A1, A2)\ R2(A3, A4)$

$PROJ_{A1,A4}(SEL_{A2=A3}(R1\ JOIN\ R2))$

```
SELECT R1.A1, R2.A4
FROM R1, R2
```

```
WHERE R1.A2 = R2.A3
```

- Prodotto cartesiano (`FROM`)
- Selezione (`WHERE`)
- Proiezione (`SELECT`)

In questo caso è stata utilizzata una equi join

Condizioni sui gruppi

I padri i cui figli hanno un reddito medio maggiore di 25

```
select padre, avg(f.reddito)
from persone f join paternita on figlio = nome
group by padre
having avg(f.reddito) > 25
```

La `where` si usa per esaminare se le ennuple soddisfano il risultato

La `having` esamina i gruppi

I padri i cui figli sotto i 30 anni hanno un reddito medio maggiore di 20

```
select padre, avg(f.reddito)
from persone f join paternita on figlio = nome
where eta < 30
group by padre
having avg(f.reddito) > 25
```

Sintassi

```
SelectSQL:=
    select ListaAttributiOEspressioni
    from ListaTabelle
    [where CondizioniSemplici]
    [group by ListaAttributiDiRaggruppamento]
    [having CondizioniAggregate]
    [order by ListaAttributiDiOrdinamento]
```

Unione, intersezione e differenza

La `select` da sola non permette di fare unioni; serve un costrutto esplicito:

```
select ...
union [all]
select ...
```

I duplicati vengono eliminati (a meno che si usi `all`); anche dalle proiezioni

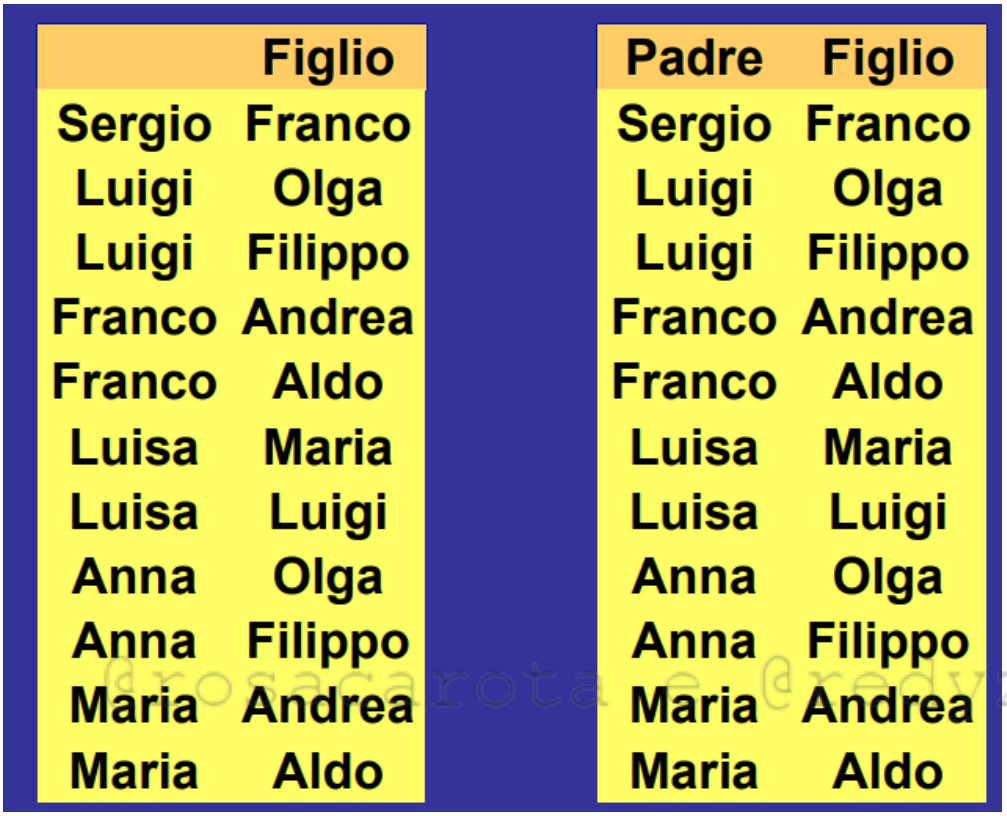
La notazione è **posizionale**

```
select padre
from paternita
union
select madre
from maternita
```

Quali nomi per gli attributi del risultato?

- Nessuno

- Quelli de primo operando
- …



	Figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo
Luisa	Maria
Luisa	Luigi
Anna	Olga
Anna	Filippo
Maria	Andrea
Maria	Aldo

Padre	Figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo
Luisa	Maria
Luisa	Luigi
Anna	Olga
Anna	Filippo
Maria	Andrea
Maria	Aldo

Gli attributi compatibili devono rispettare la posizione e avere lo stesso nome

Soluzione corretta:

```
select padre as genitore, figlio from paternita
union
select madre as genitore, figlio from maternita
```

Differenza

```
select Nome
from Impiegato
except
select Cognome as Nome
from Impiegato
```

Intersezione

```
select Nome
from Impiegato
intersect
select Cognome as Nome
from Impiegato
```

Equivale a:


```
select I.Nome
from Impiegato I, Impiegato J
where I.Nome = J.Cognome
```

Interrogazioni nidificate

Le condizioni atomiche permettono anche il confronto fra un attributo (o più) e il risultato di una sottointerrogazioni

Si può quindi usare una specie di quantificatore esistenziale

Esempio, nome e reddito del padre di Franco:

```
select Nome, Reddito
from Persone, Paternità
where Nome = Padre and Figlio = 'Franco'
```

```
select Nome, Reddito
from Persone
where Nome = (
    select Padre
    from Paternita
    where Figlio = 'Franco'
)
```

```
select distinct P.nome, P.Reddito
from Persone P, Paternità, Persone F
where P.Nome = Padre and Figlio = F.Nome
and F.Reddito > 20
```

```
select Nome, Reddito
from Persone
where Nome in (
    select Padre from Paternita
    where Figlio = any (
        select Nome from Persone where Reddito > 20
    )
)
```


In quest'ultima query non si può utilizzare l'uguale dopo `where` perché il confronto avviene tra uno scalare e una lista di nomi, si devono usare `= any` oppure `in`

Questa forma dichiarativa è limitata, non si possono includere nella target list attributi di relazioni nei blocchi interni

Regole di visibilità:

- Non è possibile fare riferimenti a variabili definite in blocchi più interni
- Se un nome di variabile è omesso, si assume riferimento alla variabile più “vicina”

In un blocco si può fare riferimento a variabili definite in blocchi più esterni; la semantica base (prodotto cartesiano, selezione, proiezione) non funziona più

Quantificazione esistenziale

Ulteriore tipo di condizione

- `exists (Sottoespressione)`

Le persone che hanno almeno un figlio:

```
select *
from Persone
where exists (
    select *
    from Paternita
    where Padre = Nome
) or
    exists (
    select *
    from Maternita
    where Madre = Nome
)
```


Può essere utilizzato anche `not exists`