

Basi di dati

SQL: concetti base

SQL

- originariamente "Structured Query Language", ora "nome proprio"
- linguaggio con varie funzionalità:
 - contiene sia il DDL sia il DML
- ne esistono varie versioni
- vediamo gli aspetti essenziali, non i dettagli

SQL: "storia"

- prima proposta **SEQUEL** (1974);
- prime implementazioni in SQL/DS e Oracle (1981)
- dal 1983 ca. "standard di fatto"
- standard (1986, poi 1989, 1992, 1999, 2003, 2006, 2008, ...)
 - recepito solo in parte (!! Vedi <http://troels.arvin.dk/db/rdbms/> per un confronto)

Evoluzione dello standard: SQL-base

- **SQL-86:** primo standard. Possedeva gran parte delle primitive per la formulazione di interrogazioni, ma offriva un supporto limitato per la definizione e manipolazione degli schemi e delle istanze
- **SQL-89:** aggiunge la definizione di integrità referenziale

Evoluzione dello standard: SQL-2

- **SQL-92:** in gran parte compatibile con la versione precedente, introduce nuove funzionalità:
 - Nuovi costrutti (e.g. **coalesce**, **nullif** e **case**)
 - 3 livelli d'implementazione: entry, intermediate, full

Evoluzione dello standard: SQL-3 (1)

Organizzato in:

- SQL:1999:** introduce l'object-relational, trigger e funzioni esterne

- SQL:2003:** estende il modello ad oggetti e introduce l'integrazione con Java ed XML

Evoluzione dello standard: SQL-3 (2)

- **SQL:2006:** estende l'integrazione della parte XML ad altri linguaggi (e.g. XQuery)
- **SQL:2008:** introduce una serie di lievi modifiche (e.g. supporto trigger con `instead of`)

Evoluzione dello standard SQL

Nome Informale	Nome Ufficiale	Caratteristiche
SQL-Base	SQL-86	Costrutti di base
	SQL-89	Integrità referenziale
SQL-2	SQL-92	Modello relazionale Vari costrutti nuovi 3 livelli: entry, intermediate, full
SQL-3	SQL:1999	Modello relazionale ad oggetti Organizzato in diverse parti Trigger, funzioni esterne, ...
	SQL:2003	Estensioni del modello ad oggetti Eliminazione di costrutti non usati Nuove parti: SQL/JRT, SQL/XML, ...
	SQL:2006	Estensione della parte XML
	SQL:2008	Lievi aggiunte (per esempio: trigger instead of)

Definizione dei dati in SQL (1)

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

Esempio:

CREATE DATABASE Azienda

note:

In SQLite **sqlite3_open_v2(Azienda)**

In Mimer **CREATE DATABANK Azienda**

Definizione dei dati in SQL (2)

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

Esempio:

```
CREATE SCHEMA schema_azienza
```

Definizione dei dati in SQL (3)

- Istruzione **CREATE SCHEMA**:
 - 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_azienda  
AUTHORIZATION amministratore
```

Definizione dei dati in SQL (4)

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

CREATE TABLE: 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)  
)
```

Domini

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

Domini elementari

- **Carattere**: singoli caratteri o stringhe, anche di lunghezza variabile
- **Numerici**, esatti e approssimati
- **Data, ora, intervalli di tempo**
- Introdotti in SQL-3:
 - **Boolean**
 - **BLOB, CLOB** (binary/character large object): per grandi immagini e testi

Definizione di domini

- 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 )
```

Vincoli intrarelazionali

- NOT NULL
- UNIQUE definisce chiavi
- PRIMARY KEY: chiave primaria (una sola, implica NOT NULL; DB2 non rispetta lo standard)
- CHECK, vedremo più avanti

UNIQUE e PRIMARY KEY

- due forme:
 - nella definizione di un attributo, se forma da solo la chiave
 - come elemento separato

CREATE TABLE: 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)  
)
```

PRIMARY KEY, alternative

Matricola CHAR(6) PRIMARY KEY

Matricola CHAR(6),

PRIMARY KEY (Matricola)

CREATE TABLE: 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)  
)
```

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

- **CHECK**, vedremo più avanti
- **REFERENCES** e **FOREIGN KEY** permettono di definire vincoli di integrità referenziale
- di nuovo due sintassi
 - per singoli attributi
 - su più attributi
- E' possibile definire politiche di reazione alla violazione

Esempio di integrità referenziale (1)

Infrazioni

<u>Codice</u>	Data	Vigile	Prov	Numero
34321	1/2/95	3987	MI	39548K
53524	4/3/95	3295	TO	E39548
64521	5/4/96	3295	PR	839548
73321	5/2/98	9345	PR	839548

Vigili

<u>Matricola</u>	Cognome	Nome
3987	Rossi	Luca
3295	Neri	Piero
9345	Neri	Mario
7543	Mori	Gino

Esempio di integrità referenziale (2)

Infrazioni

<u>Codice</u>	Data	Vigile	Prov	Numero
34321	1/2/95	3987	MI	39548K
53524	4/3/95	3295	TO	E39548
64521	5/4/96	3295	PR	839548
73321	5/2/98	9345	PR	839548

Auto

<u>Prov</u>	<u>Numero</u>	Cognome	Nome
MI	39548K	Rossi	Mario
TO	E39548	Rossi	Mario
PR	839548	Neri	Luca

CREATE TABLE: esempio

```
CREATE TABLE Infrazioni(  
    Codice CHAR(6) NOT NULL PRIMARY KEY,  
    Data DATE NOT NULL,  
    Vigile INTEGER NOT NULL  
        REFERENCES Vigili(Matricola),  
    Provincia CHAR(2),  
    Numero CHAR(6) ,  
    FOREIGN KEY(Provincia, Numero)  
        REFERENCES Auto(Provincia, Numero)  
)
```

Politiche di reazione

- Specificata immediatamente dopo il vincolo di integrità consente di associare politiche diverse ai diversi eventi (delete, update) secondo la seguente sintassi:

```
on < delete | update >  
    < cascade | set null |  
    set default | no action >
```

Politiche di reazione: delete

- `cascade`: si propagano le cancellazioni.
- `set null`: all'attributo referente viene assegnato il valore nullo al posto del valore cancellato nella tabella
- `set default`: all'attributo referente viene assegnato il valore di default al posto del valore cancellato nella tabella esterna
- `no action`: la cancellazione non viene consentita

Politiche di reazione: update

- `cascade`: il nuovo valore viene propagato nell'altra tabella.
- `set null`: all'attributo referente viene assegnato il valore nullo al posto del valore modificato nella tabella.
- `set default`: all'attributo referente viene assegnato il valore di default al posto del valore modificato nella tabella esterna.
- `no action`: l'azione di modifica non viene consentita.

Modifiche degli schemi

- ALTER DOMAIN
- ALTER TABLE
- DROP DOMAIN
- DROP TABLE

ALTER DOMAIN

- Istruzione **ALTER DOMAIN**:
 - Permette di effettuare modifiche sui domini creati in precedenza
 - Deve essere utilizzato assieme ad uno di questi costrutti: **SET DEFAULT**, **DROP DEFAULT**, **ADD CONSTRAINT** o **DROP CONSTRAINT**

ALTER DOMAIN: esempio 1

- ALTER DOMAIN Voto SET DEFAULT 30
 - Imposta il valore predefinito del dominio con nome **Voto** a **30**
 - Il valore predefinito viene applicato solo ai nuovi attributi senza valore, inseriti dopo l'invocazione di tale comando
- ALTER DOMAIN Voto DROP DEFAULT
 - Elimina il valore predefinito del dominio **Voto**

ALTER DOMAIN: esempio 2

- ALTER DOMAIN Voto
SET CONSTRAINT votoValido CHECK
(value >=60 AND value <=100)
 - Aggiunge il vincolo **votoValido** all'interno del dominio **Voto**
- ALTER DOMAIN Voto
DROP CONSTRAINT votoValido
 - Elimina il vincolo legato al check

ALTER TABLE

- Istruzione **ALTER TABLE**:
 - Permette di effettuare modifiche su tabelle create in precedenza
 - Deve essere utilizzato assieme ad uno di questi parametri: **ALTER COLUMN**, **ADD COLUMN**, **DROP COLUMN**, **DROP CONSTRAINT** o **ADD CONSTRAINT**

ALTER TABLE: esempio 1

- ALTER TABLE Impiegato
ALTER COLUMN Matricola SET NOT NULL
 - Impone che l'attributo **Matricola** della tabella **Impiegato** non contenga valori nulli
- ALTER TABLE Impiegato
ADD COLUMN Livello CHARACTER(10)
 - Aggiunge l'attributo **Livello** alla tabella **Impiegato**.

ALTER TABLE: esempio 2

- ALTER TABLE Impiegato
DROP COLUMN Livello RESTRICT
 - Elimina l'attributo Livello dalla tabella Impiegato solo se questi non contiene valori
- ALTER TABLE Impiegato
DROP COLUMN Livello CASCADE
 - Elimina l'attributo Livello dalla tabella Impiegato ed i valori in esso contenuti

ALTER TABLE: esempio 3

- ALTER TABLE Impiegato
ADD CONSTRAINT matrValida CHECK
(char_length(Matricola) = 10)
 - Aggiunge il vincolo **matrValida** all'attributo **Matricola** della tabella **Impiegato**
- ALTER TABLE Impiegato
DROP CONSTRAINT matrValida
 - Elimina il vincolo **matrValida** dalla tabella **Impiegato**

DROP DOMAIN

- Istruzione **DROP DOMAIN**:
 - cancella un domino definito da un utente

Esempio:

DROP DOMAIN Voto

DROP TABLE

- Istruzione **DROP TABLE**:
 - consente di distruggere una tabella, eliminando i dati contenuti in essa.

Esempio:

DROP TABLE Infrazioni

Definizione degli indici

- è rilevante dal punto di vista delle prestazioni
- ma è a livello fisico e non logico
- in passato era importante perché in alcuni sistemi era l'unico mezzo per definire chiavi
- **CREATE INDEX**

CREATE INDEX: esempio

- CREATE INDEX idx_Cognome
ON Vigili (Cognome)
- Crea l'indice idx_Cognome sull'attributo
Cognome della tabella Vigili

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**
- modifica:
 - **INSERT, DELETE, UPDATE**

Istruzione SELECT (versione base)

SELECT ListaAttributi
FROM ListaTabelle
[WHERE Condizione]

- "target list"
- clausola FROM
- clausola WHERE

SELECT: come leggerla

```
3 SELECT Matricola, Nome  
1 FROM Vigili  
2 WHERE Cognome = 'Rossi'
```

- 1 Dalla relazione 'Vigili'
- 2 Filtra tutte le tuple con valore 'Rossi' nell'attributo Cognome
- 3 Mostrandomi per ciascuna tupla trovata la Matricola e il Nome

Base dati di esempio 2

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

Maternita

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

Paternita

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

Selezione e proiezione

- Nome e reddito delle persone con meno di trent'anni

$\text{PROJ}_{\text{Nome, Reddito}}(\text{SEL}_{\text{Eta} < 30}(\text{Persone}))$

```
select nome, reddito  
from persone  
where eta < 30
```

Persone

Nome	Reddito
Andrea	21
Aldo	15
Filippo	30

SELECT, abbreviazioni

```
select nome, reddito  
from persone  
where eta < 30
```

```
select p.nome as nome,  
       p.reddito as reddito  
from persone as p  
where p.eta < 30
```

Selezione, senza proiezione

- Nome, età e reddito delle persone con meno di trent'anni

$SEL_{Eta < 30}(Persone)$

```
select *  
from persone  
where eta < 30
```

Persone

Nome	Età	Reddito
Andrea	27	21
Aldo	25	15
Filippo	26	30

Proiezione, senza selezione

Persone

- Nome e reddito di tutte le persone

PROJ_{Nome, Reddito}(Persone)

select nome, reddito
from persone

Nome	Reddito
Andrea	21
Aldo	15
Maria	42
Anna	35
Filippo	30
Luigi	40
Franco	20
Olga	41
Sergio	35
Luisa	87

SELECT, abbreviazioni (1)

```
select *  
from persone  
where eta < 30
```

```
select nome, età, reddito  
from persone  
where eta < 30
```

SELECT, abbreviazioni (2)

- R(A,B)

```
select *  
from R
```

equivale (intuitivamente) a

```
select X.A as A, X.B as B  
from R X  
where true
```

Espressioni nella target list

```
select Reddito/2 as redditoSemestrale  
from Persone  
where Nome = 'Luigi'
```

Persone

redditoSemestrale

20

Condizione complessa

```
select *  
from persone  
where reddito > 25 and (eta < 30 or eta > 60)
```

Persone

Nome	Età	Reddito
Filippo	26	30
Sergio	85	35

Condizione “LIKE”

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

```
select *  
from persone  
where nome like 'A_d%'
```


Condizione “LIKE”: esempio

Persone

Nome	Età	Reddito
Andrea	27	21
Aldo	25	15

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 (Età > 40) OR (Età IS NULL) (Impiegati)

Esempio

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

SEL $\text{Età} > 40 \text{ OR } \text{Età IS NULL}$ (Impiegati)

```
select *  
from impiegati  
where eta > 40 or eta is null
```

Proiezione, osservazione:

Cognome	Filiale
Neri	Napoli
Neri	Milano
Rossi	Roma

- cognome e filiale di tutti gli impiegati

PROJ _{Cognome, Filiale} (Impiegati)

Distinct

```
select  
    cognome, filiale  
from impiegati
```

Cognome	Filiale
Neri	Napoli
Neri	Milano
Rossi	Roma
Rossi	Roma

```
select distinct  
    cognome, filiale  
from impiegati
```

Cognome	Filiale
Neri	Napoli
Neri	Milano
Rossi	Roma

Selezione, proiezione e join

- Istruzioni **SELECT** con una sola relazione nella clausola **FROM** permettono di realizzare:
 - selezioni, proiezioni, ridenominazioni
- con più relazioni nella **FROM** si realizzano join (e prodotti cartesiani)

SQL e algebra relazionale (1)

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

```
select distinct R1.A1, R2.A4  
from   R1, R2  
where  R1.A2 = R2.A3
```

- prodotto cartesiano (**FROM**)
- selezione (**WHERE**)
- proiezione (**SELECT**)

SQL e algebra relazionale (2)

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

Select distinct R1.A1, R2.A4
from R1, R2
where R1.A2 = R2.A3

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

SQL, alias e ridenominazione

- possono essere necessarie ridenominazioni
 - nel prodotto cartesiano
 - nella target list

```
select X.A1 AS B1, ...  
from   R1 X, R2 Y, R1 Z  
where  X.A2 = Y.A3 AND ...
```

Equivalenza tra SQL e algebra relazionale

```
select distinct X.A1 AS B1, Y.A4 AS B2  
from   R1 X, R2 Y, R1 Z  
where  X.A2 = Y.A3 AND Y.A4 = Z.A1
```

$$\text{REN}_{B1, B2 \leftarrow A1, A4} \left(\text{PROJ}_{A1, A4} \left(\text{SEL}_{A2 = A3 \text{ AND } A4 = C1} \left(\text{R1 JOIN R2 JOIN REN}_{C1, C2 \leftarrow A1, A2} (\text{R1}) \right) \right) \right)$$

SQL: esecuzione delle interrogazioni

- Le espressioni SQL sono dichiarative e noi ne stiamo vedendo la semantica
- In pratica, i DBMS eseguono le operazioni in modo efficiente, ad esempio:
 - eseguono le selezioni al più presto
 - se possibile, eseguono join e non prodotti cartesiani

SQL: specifica delle interrogazioni

- La capacità dei DBMS di "ottimizzare" le interrogazioni, rende (di solito) non necessario preoccuparsi dell'efficienza quando si specifica un'interrogazione
- È perciò più importante preoccuparsi della chiarezza (anche perché così è più difficile sbagliare ...)

Base dati di esempio 2

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

Maternita

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

Paternita

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

Selezione, proiezione e join: esempio 1

- I padri di persone che guadagnano più di 20

$\text{PROJ}_{\text{Padre}}(\text{paternita}$
 $\text{JOIN}_{\text{Figlio} = \text{Nome}}$
 $\text{SEL}_{\text{Reddito} > 20}(\text{persone}))$

```
select distinct padre
from persone, paternita
where figlio = nome and reddito > 20
```

Selezione, proiezione e join: esempio 2

- Le persone che guadagnano più dei rispettivi padri; mostrare nome, reddito e reddito del padre

$\text{PROJ}_{\text{Nome, Reddito, RP}} (\text{SEL}_{\text{Reddito} > \text{RP}} (\text{REN}_{\text{NP, EP, RP} \leftarrow \text{Nome, Eta, Reddito}} (\text{paternita JOIN}_{\text{NP=Padre}} (\text{JOIN}_{\text{Figlio = Nome}} (\text{persone}))))$

```
select f.nome, f.reddito, p.reddito
from persone p, paternita, persone f
where p.nome = padre and
      figlio = f.nome and
      f.reddito > p.reddito
```

SELECT, con ridenominazione del risultato

```
select figlio, f.reddito as reddito,  
           p.reddito as redditoPadre  
from persone p, paternita, persone f  
where p.nome = padre and figlio = f.nome  
and f.reddito > p.reddito
```


Join esplicito

- Padre e madre di ogni persona

```
select paternita.figlio, padre, madre  
from maternita, paternita  
where paternita.figlio = maternita.figlio
```

```
select madre, paternita.figlio, padre  
from maternita join paternita on  
    paternita.figlio = maternita.figlio
```

SELECT con join esplicito, sintassi

```
SELECT ...  
FROM Tabella { ... JOIN Tabella ON CondDiJoin }, ...  
[ WHERE AltraCondizione ]
```

SELECT con join esplicito: esempio

- Le persone che guadagnano più dei rispettivi padri; mostrare nome, reddito e reddito del padre

```
select f.nome, f.reddito, p.reddito  
from persone p, paternita, persone f  
where p.nome = padre and  
       figlio = f.nome and  
       f.reddito > p.reddito
```

```
select f.nome, f.reddito, p.reddito  
from (persone p join paternita on p.nome = padre)  
     join persone f on figlio = f.nome  
where f.reddito > p.reddito
```

Ulteriore estensione: join naturale (meno diffuso)

$\text{PROJ}_{\text{Figlio, Padre, Madre}}(\text{paternita JOIN}_{\text{Figlio = Nome}} \text{REN}_{\text{Nome=Figlio}}(\text{maternita}))$

paternita JOIN maternita

```
select madre, paternita.figlio, padre
from maternita join paternita on
    paternita.figlio = maternita.figlio
```

```
select madre, figlio, padre
from maternita natural join paternita
```

Outer join

- Con join e natural join, che possiamo chiamare anche inner join, puo' capitare che alcuni valori di attributi delle tuple di partenza, non appaiano piu' nel risultato finale.
- Per evitare questa perdita di informazione, se pur parziale, si puo' usare un:

left/right/full OUTER join

- Left e right join sono esterni per definizione, quindi la keyword OUTER puo' essere omessa.

Left join

- Padre e, se nota, madre di ogni persona

```
select paternita.figlio, padre, madre  
from paternita left join maternita  
on paternita.figlio = maternita.figlio
```

paternita.figlio	padre	madre
Franco	Sergio	NULL
Olga	Luigi	Anna
Filippo	Luigi	Anna
Andrea	Franco	Maria
Aldo	Franco	Maria

Left outer join

- Padre e, se nota, madre di ogni persona

```
select paternita.figlio, padre, madre  
from paternita left outer join maternita  
on paternita.figlio = maternita.figlio
```

paternita.figlio	padre	madre
Franco	Sergio	NULL
Olga	Luigi	Anna
Filippo	Luigi	Anna
Andrea	Franco	Maria
Aldo	Franco	Maria

- Outer** è opzionale in quanto equivalente al **left join** che è sempre un join esterno

Outer join

```
select paternita.figlio, padre, madre  
from maternita join paternita  
on maternita.figlio = paternita.figlio
```

```
select paternita.figlio, padre, madre  
from maternita left outer join paternita  
on maternita.figlio = paternita.figlio
```

```
select paternita.figlio, padre, madre  
from maternita full outer join paternita  
on maternita.figlio = paternita.figlio
```

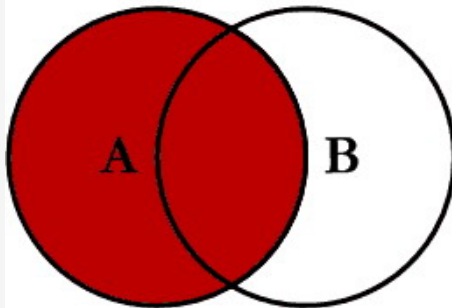
- Che cosa produce l'ultima select?

Full Outer join: esempio

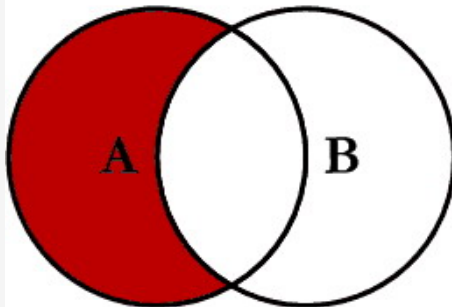
paternita.figlio	padre	madre
NULL	NULL	Luisa
NULL	NULL	Luisa
Olga	Luigi	Anna
Filippo	Luigi	Anna
Andrea	Franco	Maria
Aldo	Franco	Maria
Franco	Sergio	NULL

Il **full outer join** restituisce il join interno esteso con le righe escluse di entrambe le tabella

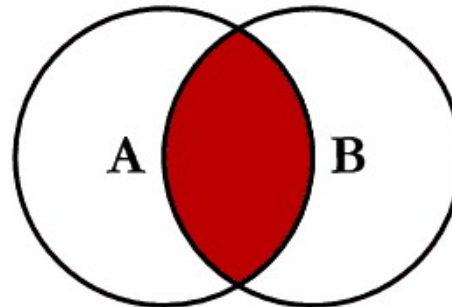
SQL JOINS



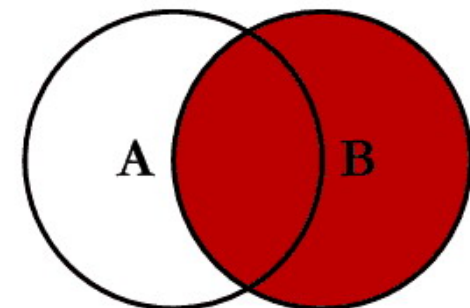
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



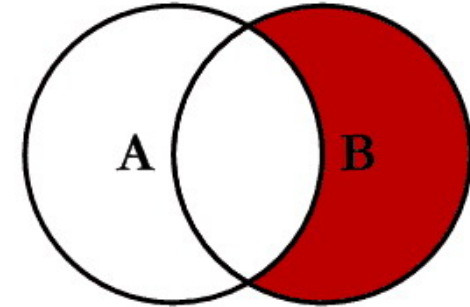
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



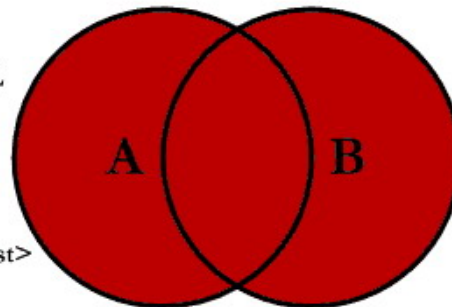
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



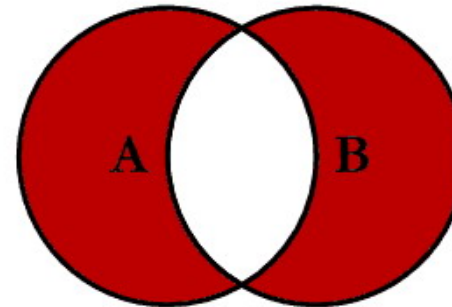
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

Ordinamento del risultato

- Nome e reddito delle persone con meno di trenta anni **in ordine alfabetico**

```
select nome, reddito  
from persone  
where eta < 30  
order by nome ASC
```

Con order by i campi vengono ordinati in ordine alfabetico ascendente (order by ASC) o discendente (order by DISC)

Ordinamento del risultato: base dati d'esempio

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

Ordinamento del risultato

```
select nome, reddito  
from persone  
where eta < 30
```

Persone

Nome	Reddito
Andrea	21
Aldo	15
Filippo	30

```
select nome, reddito  
from persone  
where eta < 30  
order by nome
```

Persone

Nome	Reddito
Aldo	15
Andrea	21
Filippo	30

Se non si specifica un verso per l'ordinamento (ASC o DISC), **order by** ordina i dati in ordine ascendente

Ordinamento del risultato

```
select nome, reddito  
from persone  
where eta < 30  
order by nome ASC
```

Persone

Nome	Reddito
Aldo	15
Andrea	21
Filippo	30

```
select nome, reddito  
from persone  
where eta < 30  
order by nome DISC
```

Persone

Nome	Reddito
Filippo	30
Andrea	21
Aldo	15

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**).

Unione senza duplicati

Effettua l'unione tra gli attributi della prima **SELECT** con quelli della seconda, senza duplicare gli stessi valori

Esempio:

```
Select Figlio  
From Maternita  
union  
Select Figlio  
From Paternita
```

Maternita UNION Paternita

Figlio

Maria

Luigi

Olga

Franco

Filippo

Unione con duplicati

Effettua l'unione tra gli attributi della prima **SELECT** con quelli della seconda, duplicando gli attributi con lo stesso valore

Esempio:

```
Select Figlio  
From Maternita  
union all  
Select Figlio  
From Paternita
```

Maternita UNION ALL Paternita

Figlio
Maria
Luigi
Olga
Maria
Olga
Filippo

Notazione posizionale (1)

select padre, figlio

from paternita

union

select madre, figlio

from maternita

- Quali nomi per gli attributi del risultato?
 - inventati o nessuno
 - quelli del primo operando
 - padre + madre

Notazione posizionale: il primo operando

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

Notazione posizionale (2)

```
select padre, figlio  
from paternita  
union
```

```
select figlio, madre  
from maternita
```

```
select padre, figlio  
from paternita  
union
```

```
select madre, figlio  
from maternita
```

- In entrambi i casi il risultato dell'unione ci darà sempre gli attributi **padre** e **figlio**

Differenza

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

- vedremo che si può esprimere con `select` nidificate

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
```

Base dati di esempio 2

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

Interrogazioni nidificate

- le condizioni atomiche permettono anche
 - il confronto fra un attributo (o più attributi, vedremo poi) e il risultato di una sottointerrogazione
 - quantificazioni esistenziali

Interrogazioni nidificate: esempio 1

- nome e reddito del padre di Franco

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

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

Interrogazioni nidificate, commenti

- La forma nidificata è “meno dichiarativa”, ma talvolta più leggibile (richiede meno variabili)
- La forma piana e quella nidificata possono essere combinate
- Le sottointerrogazioni non possono contenere operatori insiemistici (“l’ unione si fa solo al livello esterno”); la limitazione non è significativa

Interrogazioni nidificate: any, all

- Le sottointerrogazioni possono utilizzare gli operatori **ANY** e **ALL**. Con op= (>, <, =, >=, ..)
 - **Attributo op ANY (Sottoespressione)**
- Una riga soddisfa la condizione se risulta vero il confronto fra il valore dell'attributo per la riga e almeno uno degli elementi restituiti dalla sottoespressione
 - **Attributo op ALL(Sottoespressione)**
- Una riga soddisfa la condizione se risulta vero il confronto fra il valore dell'attributo per la riga e tutti gli elementi restituiti dalla sottoespressione

Interrogazioni nidificate: in

- **Attributo IN**(Sottoespressione)
- Una riga soddisfa la condizione se il valore dell'attributo per la riga e' contenuto negli elementi restituiti dall'interrogazione
- **ANY**, **ALL** e **IN** possono anche apparire negati, preceduti da **NOT**

Interrogazioni nidificate: esempio 2a

- Nome e reddito dei padri di persone che guadagnano più di 20

```
select distinct P.Nome, P.Reddito  
from Persone P, Paternita, 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))
```

notare la **distinct**

Interrogazioni nidificate: esempio 2b

- Nome e reddito dei padri di persone che guadagnano più di 20

```
select distinct P.Nome, P.Reddito  
from Persone P, Paternita, 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, Persone  
                where Figlio = Nome  
                and Reddito > 20)
```

Interrogazioni nidificate: esempio 3

- Nome e reddito dei padri di persone che guadagnano più di 20, con indicazione del reddito del figlio

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

- Questa alternativa produce lo stesso risultato?

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

Soluzione

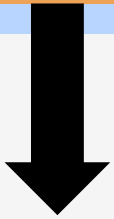
```
select Nome, Reddito  
from Persone  
where Nome in
```

```
(select Padre  
from Paternita  
where Figlio = any
```

```
(select Nome  
from Persone  
where Reddito > 20) )
```



Nome	Reddito
Franco	20
Luigi	40
Luigi	40



Padre
Franco
Luigi
Luigi



Nome
Andrea
Maria
Anna
Filippo
Luigi
Olga
Sergio
Luisa

- **Risposta:** no, in quanto per ciascun padre non visualizziamo il reddito del figlio

Interrogazioni nidificate, commenti, 3

- 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, vedremo presto perchè

Quantificazione esistenziale

- Ulteriore tipo di condizione
 - **EXISTS** (Sottoespressione)
- Il predicato e' vero se la sottoespressione restituisce almeno una tupla

Quantificazione esistenziale: esempio 1

- 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)
```

Quantificazione esistenziale: esempio 2

- I padri i cui figli guadagnano tutti più di 20

```
select distinct Padre
from Paternita Z
where not exists ( select *
                   from Paternita W, Persone
                   where W.Padre = Z.Padre
                   and W.Figlio = Nome
                   and Reddito <= 20 )
```

Quantificazione esistenziale: errore

- I padri i cui figli guadagnano tutti più di 20

```
select distinct Padre  
from Paternita  
where not exists ( select *  
                  from Persone  
                  where Figlio = Nome  
                  and Reddito <= 20 )
```

La variabile **Figlio** non ha una relazione di riferimento

Semantica delle espressioni “correlate”

- L'interrogazione interna viene eseguita una volta per ciascuna ennupla dell'interrogazione esterna

Visibilità

- Scorretta:

```
select *  
from Impiegato  
where Dipart in ( select Nome  
                  from Dipartimento D1  
                  where Nome = 'Produzione')  
                or  
                Dipart in ( select Nome  
                  from Dipartimento D2  
                  where D2.Citta = D1.Citta )
```

Perché nell'ultima select, D1 di Citta non è visibile

Differenza e nidificazione

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

```
select Nome  
from Impiegato I  
where not exists (select *  
                  from Impiegato  
                  where Cognome = I.Nome)
```


Operatori aggregati

- Nelle espressioni della target list possiamo avere anche espressioni che calcolano valori a partire da insiemi di ennuple:
 - conteggio, minimo, massimo, media, totale
 - sintassi base (semplificata):

Funzione ([DISTINCT] *)

Funzione ([DISTINCT] Attributo)

Operatori aggregati: COUNT

- Il numero di figli di Franco

```
select count(*) as NumFigliDiFranco  
from Paternita  
where Padre = 'Franco'
```

- l'operatore aggregato (**count**) viene applicato al risultato dell'interrogazione:

```
select *  
from Paternita  
where Padre = 'Franco'
```

Operatori aggregati: esempio di COUNT

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

NumFigliDiFranco
2

COUNT DISTINCT

Persone

Nome	Età	Reddito
Andrea	27	30
Aldo	25	24
Maria	55	36
Anna	50	36

select count(*)
from persone

4

select count(distinct reddito)
from persone

3

Altri operatori aggregati

- SUM, AVG, MAX, MIN
- Media dei redditi dei figli di Franco

```
select avg(reddito)
from persone join paternita on nome=figlio
where padre='Franco'
```

COUNT e valori nulli (1)

Persone

Nome	Età	Reddito
Andrea	27	30
Aldo	25	NULL
Maria	55	36
Anna	50	36

select count(*)
from persone

4

select count(reddito)
from persone

3

COUNT e valori nulli (2)

Persone

Nome	Età	Reddito
Andrea	27	21
Aldo	25	NULL
Maria	55	21
Anna	50	35

```
select count(distinct reddito)  
from persone
```

2

Operatori aggregati e valori nulli

Persone

Nome	Età	Reddito
Andrea	27	21
Aldo	25	NULL
Maria	55	21
Anna	50	35

```
select avg(reddito) as redditomedio  
from persone
```

redditomedio
25,6

Operatori aggregati e target list

- un' interrogazione scorretta:

```
select nome, max(reddito)  
from persone
```

- di chi sarebbe il nome? La target list deve essere omogenea

```
select min(eta), avg(reddito)  
from persone
```

Massimo e nidificazione

- La persona (o le persone) con il reddito massimo

```
select *  
from persone  
where reddito = ( select max(reddito)  
                  from persone )
```

Operatori aggregati e raggruppamenti

- Le funzioni possono essere applicate a partizioni delle relazioni
- Clausola **GROUP BY**:
GROUP BY listaAttributi

Operatori aggregati e raggruppamenti

- Il numero di figli di ciascun padre

```
select Padre, count(*) AS NumFigli  
from paternita  
group by Padre
```

Paternita

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

Padre	NumFigli
Sergio	1
Luigi	2
Franco	2

Semantica di interrogazioni con operatori aggregati e raggruppamenti

1. interrogazione senza **group by** e senza operatori aggregati

select *

from paternita

2. si raggruppa e si applica l'operatore aggregato a ciascun gruppo

Raggruppamenti e target list

scorretta

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

corretta

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

Condizioni sui gruppi

- I padri i cui figli hanno un reddito medio maggiore di 25; mostrare padre e reddito medio dei figli

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

WHERE o HAVING?

- 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) > 20
```


Group by e valori nulli

R

A	B
1	11
2	11
3	null
4	null

select B, count (*)
from R group by B

B	
11	2
null	2

select A, count (*)
from R group by A

A	
1	1
2	1
3	1
4	1

select A, count (B)
from R group by A

A	
1	1
2	1
3	0
4	0

Sintassi SELECT: risassunto

```
SELECT ListaAttributiOEspressioni  
FROM ListaTabelle  
[ WHERE CondizioneSemplice]  
[ GROUP BY ListaAttributiDiRaggrup]  
[ HAVING CondizioniAggregate]  
[ ORDER BY ListaAttributiDiOrdinamento]
```

Operazioni di aggiornamento

- operazioni di
 - inserimento: `insert`
 - eliminazione: `delete`
 - modifica: `update`
- di una o più ennuple di una relazione
- sulla base di una condizione che può coinvolgere anche altre relazioni

Inserimento

```
INSERT INTO Tabella [ ( Attributi ) ]  
VALUES( Valori )
```

oppure

```
INSERT INTO Tabella [ ( Attributi )]  
SELECT ...
```

Inserimento: alcuni esempi

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

```
INSERT INTO Persone(Nome, Eta, Reddito)  
VALUES('Pino',25,52)
```

```
INSERT INTO Persone(Nome, Reddito)  
VALUES('Lino',55)
```

```
INSERT INTO Persone ( Nome )  
SELECT Padre  
FROM Paternita  
WHERE Padre NOT IN (SELECT Nome  
FROM Persone)
```

Inserimento, commenti

- L'ordinamento degli attributi (se presente) e dei valori è significativo
- le due liste debbono avere lo stesso numero di elementi
- se la lista di attributi è omessa, si fa riferimento a tutti gli attributi della relazione, secondo l'ordine con cui sono stati definiti
- se la lista di attributi non contiene tutti gli attributi della relazione, per gli altri viene inserito un valore nullo (che deve essere permesso) o un valore di default

Eliminazione di ennuple

DELETE FROM Tabella
[WHERE Condizione]

Eliminazione di ennuple: alcuni esempi

```
DELETE FROM Persone  
WHERE Eta < 35
```

```
DELETE FROM Paternita  
WHERE Figlio NOT in ( SELECT Nome  
                        FROM Persone)
```

```
DELETE FROM Paternita
```


Eliminazione, commenti

- elimina le ennuple che soddisfano la condizione
- può causare (se i vincoli di integrità referenziale sono definiti con politiche di reazione **cascade**) eliminazioni da altre relazioni
- ricordare: se la where viene omessa, si intende **where true**

Modifica di ennuple

```
UPDATE NomeTabella  
SET Attributo = < Espressione |  
                SELECT ... |  
                NULL |  
                DEFAULT >  
[ WHERE Condizione ]
```

Modifica di ennuple: alcuni esempi

```
UPDATE Persone  
SET Reddito = 45  
WHERE Nome = 'Piero'
```

Persone

Nome	Età	Reddito
Andrea	27	30
Aldo	25	15
Piero	55	45

```
UPDATE Persone  
SET Reddito = Reddito * 1.1  
WHERE Eta < 30
```

Persone

Nome	Età	Reddito
Andrea	27	33
Aldo	25	16,5
Piero	55	36