Capitolo 4

SQL

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 "de jure" ANSI
  - 1986, esteso con integrità referenziali nel 1989 (SQL-89)
  - 1992: molte funzionalità (SQL-2)
  - 1999: trigger, oggetti, viste ricorsive ... (SQL:1999, o SQL-3)
- Il più diffuso: SQL-2, ma recepito solo in parte (!!)

Definizione dei dati in SQL

- 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
- Bit: singoli booleani o stringhe
- Numerici, esatti e approssimati
- Data, ora, intervalli di tempo
- Introdotti in SQL:1999:
  - 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)
- CHECK, vedremo più avanti

UNIQUE e PRIMARY KEY

- due forme:
  - nella definzione 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

Infrazioni

Codice	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

Matricola	Cognome	Nome
3987	Rossi	Luca
3295	Neri	Piero
9345	Neri	Mario
7543	Mori	Gino

Infrazioni

Codice	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

Prov	Numero	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)
 )
```


Modifiche degli schemi

ALTER DOMAIN
ALTER TABLE
DROP DOMAIN
DROP TABLE
...

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

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

Istanze di esempio

- Useremo nei nostri esempi queste istanze delle relazioni Velisti e Prenota
- Se la chiave per la relazione Prenota contenesse solo gli attributi *vid* e *bid*, in che modo la semantica sarebbe diversa?

P1

vid	bid	giorno
22	101	10/10/96
58	103	11/12/96

V1

vid	vnome	espe-rienza	età
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

V2

vid	vnome	espe-rienza	età
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Interrogazioni SQL di base

```
SELECT [DISTINCT]   lista-attributi
FROM                lista-relazioni
[WHERE              qualificazioni]
```

- *Lista-relazioni*. Una lista di nomi di relazioni (eventualmente con una *variabile di range* dopo ciascun nome)
- *Lista-attributi*. Una lista di attributi delle relazioni in *lista-relazioni*
- *Qualificazioni*. Confronti (attr *op* cost oppure attr1 *op* attr2, dove *op* è uno tra <, >, =, ≤, ≥¸ ≠) combinati usando AND, OR e NOT
- DISTINCT è una parola chiave opzionale che indica che la risposta non dovrebbe contenere duplicati. Per impostazione predefinita i duplicati *non* sono eliminati!

Strategia di valutazione concettuale

- La semantica di una interrogazione SQL è definita in termini della seguente strategia di valutazione concettuale:
  - calcolare il prodotto scalare di *lista-relazioni*
  - scartare le tuple risultanti se non passano le *qualificazioni*
  - cancellare gli attributi che non sono in *lista-attributi*
  - se è specificato DISTINCT, eliminare le righe duplicate
- Questa strategia è probabilmente il modo meno efficiente di calcolare una interrogazione! Un ottimizzatore troverà strategie più efficienti per calcolare *le stesse risposte*.

Esempio di valutazione concettuale

```
SELECT      V.vnome
FROM        Velisti V, Prenota P
WHERE       V.vid = P.vid AND P.bid = 103
```

(vid)	vnome	espe-rienza	età	(vid)	bid	giorno
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

Nota sulle variabili di range

- Se ne ha veramente bisogno solo se la stessa relazione appare due volte nella clausola FROM. L'interrogazione precedente può anche essere scritta come:

```
SELECT V.vnome
FROM   Velisti V, Prenota P
WHERE V.vid = R.vid AND bid = 103
```

OPPURE

```
SELECT vnome
FROM   Velisti, Prenota
WHERE Velisti.vid = Prenota.vid
                AND bid = 103
```


Trovare gli id dei velisti che hanno prenotato almeno una barca

```
SELECT      V.vid
FROM        Velisti V, Prenota P
WHERE       V.vid = P.vid
```

- Farebbe differenza aggiungere DISTINCT a questa interrogazione?
- Qual è l'effetto della sostituzione di V.vid con V.vnome nella clausola SELECT? Farebbe differenza aggiungere DISTINCT a questa variante dell'interrogazione?

Espressioni e stringhe

```
SELECT      V.età, età1 = V.età -5, 2 * V.età AS eta2
FROM        Velisti V
WHERE                 V.vnome LIKE 'B_%B'
```

- Illustra l'uso delle espressioni aritmetiche e del pattern matching di stringhe: trovare le triple (le età dei velisti e due campi definiti da espressioni) per quei velisti il cui nome inizia e termina con B e contiene almeno tre caratteri)
- AS e = sono due modi di dare un nome ai campi del risultato
- LIKE è usato per il matching di stringhe. "_" indica qualunque carattere singolo e "%" sta per 0 o più caratteri arbitrari

Trovare i vid dei velisti che hanno prenotato una barca rossa o una barca verde

- UNION: può essere usato per calcolare l'unione di qualunque coppia di insiemi di tuple (essi stessi risultati di interrogazioni SQL) *compatibili rispetto all'unione*
- Se nella prima versione sostituiamo OR con AND, cosa otteniamo?
- Disponibile anche: EXCEPT (cosa otteniamo se sostituiamo UNION con EXCEPT?)

```
SELECT  P.sid
FROM  Barche B, Prenota R
WHERE  P.bid=B.bid
  AND (B.colore='rosso' OR B.colore='verde')
```

```
SELECT   P.vid
FROM     Barche B, Prenota P
WHERE    P.bid = B.bid
      AND (B.colore = 'rosso' OR B.colore =
'verde')
UNION
SELECT   V.vid
FROM     Velisti V, Barche B, Prenota P
WHERE    V.vid = P.vid AND P.bid = B.bid
                   AND B.colore = 'verde'
```


Trovare i vid dei velisti che hanno prenotato una barca rossa e una barca verde

- INTERSECT: può essere usato per calcolare l'intersezione di qualunque coppia di insiemi di tuple *compatibili rispetto all'unione*
- Incluso nello standard SQL/92, ma alcuni sistemi non lo supportano
- Confrontate la simmetria delle interrogazioni con UNION e INTERSECT con la diversità delle altre versioni

```
SELECT   V.vid
FROM     Velisti V, Barche B1, Prenota P1,
                   Barche B2, Prenota P2
WHERE    V.vid = P1.vid AND P1.bid = B1.bid
         AND V.vid = P2.vid AND P2.bid = B2.bid
         AND (B1.colore = 'rosso' AND B2.colore = 'verde')
```

Campo chiave!

```
SELECT   V.vid
FROM     Velisti V, Barche B, Prenota P
WHERE    V.vid = P.vid AND P.bid = B.bid
                   AND B.colore = 'rosso'
INTERSECT
SELECT   V.vid
FROM     Velisti V, Barche B, Prenota P
WHERE    V.vid = P.vid AND P.bid = B.bid
                   AND B.colore = 'verde'
```


Interrogazioni annidate

Trovare i nomi dei velisti che hanno prenotato la barca #103:

```
SELECT V.vnome
FROM   Velisti V
WHERE V.vid IN (SELECT P.vid
                FROM Prenota P
                WHERE P.bid = 103)
```

- Una funzione molto potente di SQL: una clausola WHERE può essa stessa contenere una interrogazione SQL! (In realtà, lo stesso vale per le clausole FROM e HAVING)
- Per trovare i velisti che *non* hanno prenotato la barca #103, usare NOT IN
- Per comprendere la semantica delle interrogazioni annidate, pensate alla valutazione dei *cicli annidati*: *per ciascuna tupla di Velisti, controllare la qualificazione calcolando la sottointerrogazione*.

Ancora sugli operatori di confronto tra insiemi

- Abbiamo già visto IN, EXISTS e UNIQUE. Possiamo anche usare NOT IN, NOT EXISTS e NOT UNIQUE
- Disponibili anche: op ANY, op ALL, op IN, >, <, =, ≥, ≤, ≠
- Trovare i velisti la cui esperienza è maggiore di quella di qualche velista chiamato Horatio:

```
SELECT *
FROM  Velisti V
WHERE          V.esperienza  > ANY (SELECT   V2.esperienza
                                    FROM       Velisti V2
                                    WHERE      V2.vnome = 'Horatio')
```

Se non ci sono velisti chiamati Horatio il confronto restituisce FALSE

Riscrittura delle interrogazioni INTERSECT usando IN

Trovare i vid dei velisti che hanno prenotato sia una barca rossa che una barca verde:

```
SELECT V.vid
FROM   Velisti V, Barche B, Prenota P
WHERE V.vid = P.vid AND P.bid = B.bid AND B.colore = 'rosso'
              AND V.vid IN (SELECT    V2.vid
                            FROM    Velisti V2, Barche B2, Prenota P2
                            WHERE V2.vid = P2.vid AND P2.bid = B2.bid
                                           AND B2.colore = 'verde')
```

- Analogamente, le interrogazioni EXCEPT si riscrivono usando NOT IN
- Per trovare i nomi (non i *vid*) dei velisti che hanno prenotato sia barche rosse che barche verdi, basta sostituire V.vid con V.vnome nella clausola SELECT (che si può dire dell'interrogazione con INTERSECT?)

Operatori di aggregazione

- Importante estensione dell'algebra relazionale.

```
COUNT (*)
COUNT ([DISTINCT] A)
SUM          ([DISTINCT] A)
AVG          ([DISTINCT] A)
MAX          (A)
MIN          (A)
```

Colonna singola

```
SELECT COUNT(*)
FROM   Velisti V
```

```
SELECT V.vnome
FROM   Velisti V
WHERE V.esperienza = (SELECT MAX(V2.esperienza)
                      FROM Velisti V2)
```

```
SELECT  AVG(V.età)
FROM    Velisti V
WHERE  V.esperienza=10
```

```
SELECT  COUNT (DISTINCT V.esperienza)
FROM    Velisti V
WHERE  V.vnome = 'Bob'
```

```
SELECT  AVG(DISTINCT V.età)
FROM    Velisti V
WHERE  V.esperienza = 10
```

Trovare nome ed età del/i velista/i più anziano/i

- La prima interrogazione è illegale! (Ne vedremo le ragioni un po' più tardi, quando discuteremo GROUP BY)
- La terza interrogazione è equivalente alla seconda, ed è permessa nello standard SQL/92, ma non è supportata in alcuni sistemi

```
SELECT V.vnome, MAX(V.età)
FROM   Velisti V
```

```
SELECT V.vnome, V.età
FROM   Velisti V
WHERE V.età =
          (SELECT MAX(V2.età)
          FROM   Velisti V2)
```

```
SELECT V.vnome, V.età
FROM   Velisti V
WHERE (SELECT MAX(V2.età)
          FROM   Velisti V2) = V.età
```


GROUP BY e HAVING

- Finora abbiamo applicato operatori di aggregazione a tutte le tuple (qualificanti). A volte vogliamo applicarli a ciascuno tra diversi *gruppi* di tuple
- Consideriamo: Trovare l'età del velista più giovane per ciascun grado di esperienza
  - In generale, non sappiamo quanti gradi di esperienza esistano, e quali siano i loro valori!
  - Supponiamo di sapere che i valori di esperienza variano da 1 a 10: possiamo scrivere 10 interrogazioni che somigliano a queste (!)

For $i = 1, 2, \ldots, 10$:

```
SELECT      MIN(V.età)
FROM        Velisti V
WHERE       V.esperienza = i
```


Interrogazioni con GROUP BY e HAVING

```
SELECT [DISTINCT] lista-target
FROM              lista-relazioni
WHERE             qualificazioni
GROUP BY          lista-gruppi
HAVING            qualificazione-gruppi
```

- La *lista-target* contiene
  1. nomi di attributi
  2. termini con operazioni di aggregazione (ad es., MIN(V.età))
  - La lista di attributi 1 deve essere un sottoinsieme della lista-gruppi. Intuitivamente, ciascuna tupla nella risposta corrisponde a un *gruppo*, e questi attributi devono avere un singolo valore per gruppo (un *gruppo* è un insieme di tuple con lo stesso valore per tutti gli attributi in *lista-attributi*)

Valutazione concettuale

- Viene calcolato il prodotto cartesiano della *lista-relazioni*, vengono scartate le tuple che non passano la *qualificazione*, i campi "non necessari" vengono cancellati, e le tuple rimanenti sono partizionate in gruppi dai valori degli attributi in *lista-gruppi*.
- La *qualificazione-gruppi* viene poi applicata per eliminare alcuni gruppi. Le espressioni in *qualificazione-gruppi* devono avere un *singolo valore per gruppo*!
  - In effetti, un attributo in *qualificazione-gruppi* che non è un argomento di un operatore di aggregazione appare anche in *lista-gruppi* (SQL qui non sfrutta la semantica delle chiavi primarie!)
- Viene generata una tupla di risposta per ogni gruppo qualificante

Trovare l'età del velista più giovane con età ≥18, per ciascun livello di esperienza con almeno 2 velisti.

```
SELECT      V.esperienza, MIN(V.età)
FROM        Velisti V
WHERE       V.età >= 18
GROUP BY    V.esperienza
HAVING      COUNT(*) >= 2
```

vid	vnome	espe-rienza	età
22	dustin	7	45.0
31	lubber	8	55.5
71	zorba	10	16.0
64	horatio	7	35.0
29	brutus	1	33.0
58	rusty	10	35.0

- Solo V.esperienza e V.età sono menzionati nelle clausole SELECT, GROUP BY e HAVING; altri attributi sono "non necessari"
- La seconda colonna del risultato non ha un nome (usare AS per darle un nome)

esperienza	*età*
1	33
7	45
7	35
8	55.5
10	35

Relazione risultato

espe-rienza	*età*
7	35

Per ciascuna barca rossa, trovare il numero di prenotazioni per tale barca.

```
SELECT B.bid, COUNT(*) AS vconta
FROM          Velisti V, Barche B, Prenota P
WHERE         V.vid = P.vid AND P.bid = B.bid AND B.colore = 'rosso'
GROUP BY      B.bid
```

- Raggruppamento su un join di tre relazioni
- Cosa otteniamo se rimuoviamo B.colore = 'rosso' dalla clausola WHERE e aggiungiamo una clausola HAVING con tale condizione?
- Che succede se eliminiamo Velisti e la condizione che coinvolge V.vid?

Trovare l'età del velista più giovane con età >18 per ciascun livello di esperienza con almeno 2 velisti (di qualunque età)

```
SELECT    V.esperienza, MIN(V.età)
FROM      Velisti V
WHERE     V.età > 18
GROUP BY  V.esperienza
HAVING    1 < (SELECT   COUNT(*)
               FROM   Velisti V2
               WHERE V.esperienza = V2.esperienza)
```

- Mostra come la clausola HAVING possa anche contenere una sottointerrogazione
- Confrontate questa interrogazione con quella in cui consideravamo solo i livelli di esperienza con due velisti con più di 18 anni!
- Che succede se la clausola HAVING viene sostituita con
  - HAVING COUNT(*) > 1

Trovare quei livelli di esperienza per cui l'età media è minima su tutti i livelli

- Gli operatori di aggregazione non possono essere annidati! SBAGLIATO:

```
SELECT    V.esperienza
FROM      Velisti V
WHERE     V.età = (SELECT MIN(AVG(V2.età)) FROM Velisti V2)
```

❖ Soluzione corretta (in SQL/92):

```
SELECT    Temp.esperienza. Temp.etamedia
FROM      (SELECT     V.esperienza, AVG(V.età) AS etamedia
           FROM       Velisti V
           GROUP BY   V.esperienza) AS Temp
WHERE     Temp.etamedia = (SELECT  MIN(Temp.etamedia)
                           FROM  Temp)
```


Valori Null

- I valori dei campi di una tupla sono a volte *sconosciuti* (ad esempio, non è ancora stato stabilito un livello di esperienza) oppure *inapplicabili* (ad esempio, nessuna moglie)
  - SQL fornisce uno speciale valore *null* per tali situazioni
- La presenza di *null* complica parecchie cose. Ad esempio:
  - operatori speciali sono necessari per controllare se un valore è/non è *null*
  - *esperienza* > *8* è vera o falsa quando *esperienza* è *null*? Che si può dire dei connettori AND, OR e NOT?
  - Abbiamo bisogno di una logica a 3 valori (*vero*, *falso* e *sconosciuto*)
  - Il significato dei costrutti deve essere attentamente definito (ad esempio la clausola WHERE elimina le righe che non vengono valutate come *vero*)
  - Nuovi operatori (in particolare di join esterno) sono possibili/necessari

Sintassi, riassumiamo

SelectSQL ::=

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


Sommario su SELECT

- Un fattore importante nel rapido sviluppo del modello relazionale; più naturale che in precedenza, linguaggi di interrogazione procedurali
- Relazionalmente completo; di fatto, potere espressivo significativamente superiore all'algebra relazionale
- Persino le interrogazioni che possono essere espresse in AR possono spesso essere espresse in maniera più naturale con SQL
- Molti modi alternativi di scrivere una interrogazione; l'ottimizzatore dovrebbe cercare il piano di valutazione più efficiente
  - Nella pratica, gli utenti devono essere consci di come le interrogazioni sono ottimizzate e valutate per ottenere risultati migliori

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 ...


```
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]

```
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]

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

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

Vincoli di integrità generici: check

- Specifica di vincoli di ennupla (e anche vincoli più complessi)

  check (Condizione)

```
create table Impiegato (
Matricola character(6),
Cognome character(20),
Nome character(20),
Sesso character not null check (sesso in ('M','F'))
Stipendio integer,
Superiore character(6),
check (Stipendio <= (select Stipendio
                     from Impiegato J
                     where Superiore = J.Matricola)
)
```


Vincoli di integrità generici: asserzioni

- Specifica vincoli a livello di schema

```
create assertion NomeAss check ( Condizione )

create assertion AlmenoUnImpiegato
    check (1 <= ( select count(*)
                  from Impiegato ))
```


Viste

```
create view NomeVista [ ( ListaAttributi ) ] as SelectSQL
[ with [ local | cascaded ] check option ]

create view ImpiegatiAmmin
      (Matricola, Nome, Cognome, Stipendio) as
    select Matricola, Nome, Cognome, Stipendio
    from Impiegato
    where Dipart = 'Amministrazione'
```


Aggiornamenti sulle viste

- Ammessi (di solito) solo su viste definite su una sola relazione
- Alcune verifiche possono essere imposte

Controllo dell'accesso

- In SQL è possibile specificare chi (utente) e come (lettura, scrittura, …) può utilizzare la base di dati (o parte di essa)
- Oggetto dei privilegi (diritti di accesso) sono di solito le tabelle, ma anche altri tipi di risorse, quali singoli attributi, viste o domini
- Un utente predefinito _system (amministratore della base di dati) ha tutti i privilegi
- Il creatore di una risorsa ha tutti i privilegi su di essa

Privilegi

- Un privilegio è caratterizzato da:
  - la risorsa cui si riferisce
  - l'utente che concede il privilegio
  - l'utente che riceve il privilegio
  - l'azione che viene permessa
  - la trasmissibilità del privilegio

Tipi di privilegi offerti da SQL

- insert: permette di inserire nuovi oggetti (ennuple)
- update: permette di modificare il contenuto
- delete: permette di eliminare oggetti
- select: permette di leggere la risorsa
- references: permette la definizione di vincoli di integrità referenziale verso la risorsa (può limitare la possibilità di modificare la risorsa)
- usage: permette l'utilizzo in una definizione (per esempio, di un dominio)

grant e revoke

- Concessione di privilegi:

  grant < Privileges | all privileges > on Resource
  to Users [with grant option]

  – grant option specifica se il privilegio può essere trasmesso ad altri utenti

  grant select on Department to Stefano

- Revoca di privilegi

  revoke Privileges on Resource from Users
  [restrict | cascade]