
DATA DEFINITION LANGUAGE (DDL) SQL

Materiale adattato dal libro Albano et al e
dal libro Atzeni-et al., Basi di dati

Definizione degli oggetti in SQL

Introduciamo il **Data Definition Language (DDL)** SQL, che consiste nell'insieme delle istruzioni SQL che permettono la creazione, modifica e cancellazione delle tabelle, dei domini e degli altri oggetti del database, al fine di definire il suo schema logico.

Definizione delle tabelle

- Le tabelle (corrispondenti alle relazioni dell'algebra relazionale) vengono definite in
- SQL mediante l'Istruzione **CREATE TABLE**.
- Questa istruzione
 - definisce uno schema di relazione e ne crea un'istanza vuota
 - specifica attributi, domini e vincoli

CREATE TABLE, sintassi

CREATE TABLE <nome_tabella>

```
(  nome_colonna_1 tipo_colonna_1 clausola_default_1 vincolo_di_colonna_1,  
   nome_colonna_2 tipo_colonna_2 clausola_default_2 vincolo_di_colonna_2,  
   .....  
   nome_colonna_k tipo_colonna_k clausola_default_k vincolo_di_colonna_k,  
   vincoli di tabella  
)
```

L'istruzione che crea la tabella è **CREATE TABLE**, seguito da un nome che la caratterizza, e dalla lista delle colonne (attributi), di cui si specificano le caratteristiche. Alla fine si possono anche specificare eventuali vincoli di tabella, di cui parleremo in seguito

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)  
)
```

CREATE TABLE, effetto

L'effetto del comando **CREATE TABLE** definisce uno schema di relazione e ne crea un'istanza vuota, specificandone attributi, domini e vincoli.

Una volta creata, la tabella è pronta per l'inserimento dei dati che dovranno soddisfare i vincoli imposti.

Nel caso dell'esempio dato nella slide precedente, si ottiene

IMPIEGATO

Matricola	Nome	Cognome	Dipart	Stipendio
-----------	------	---------	--------	-----------

La visualizzazione dello schema di una tabella, dopo che è stata creata, può essere ottenuta mediante il comando SQL **DESCRIBE**. Nel caso specifico:

DESCRIBE impiegato

SQL PER LA DEFINIZIONE DI BASI DI DATI

- SQL non è solo un linguaggio di interrogazione (Query Language), ma
- Un linguaggio per la definizione di basi di dati (Data-definition language (DDL))
 - CREATE SCHEMA Nome AUTHORIZATION Utente
 - CREATE TABLE o VIEW, con vincoli
 - CREATE INDEX
 - CREATE PROCEDURE
 - CREATE TRIGGER
- Un linguaggio per stabilire controlli sull'uso dei dati: GRANT
- Un linguaggio per modificare i dati.

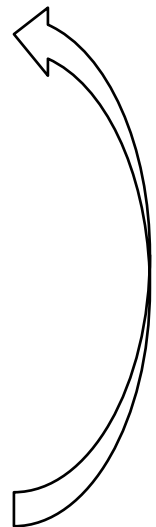
I tipi

I tipi più comuni per i valori degli attributi sono:

- **CHAR(n)** per stringhe di caratteri di lunghezza fissa n;
- **VARCHAR(n)** per stringhe di caratteri di lunghezza variabile di al massimo n caratteri;
- **INTEGER** per interi con la dimensione uguale alla parola di memoria standard dell'elaboratore;
- **REAL** per numeri reali con dimensione uguale alla parola di memoria standard dell'elaboratore;
- **NUMBER(p,s)** per numeri con p cifre, di cui s decimali;
- **FLOAT(p)** per numeri binari in virgola mobile, con almeno p cifre significative;
- **DATE** per valori che rappresentano istanti di tempo (in alcuni sistemi, come Oracle), oppure solo date (e quindi insieme ad un tipo **TIME** per indicare ora, minuti e secondi).

DEFINIZIONE DI TABELLE: ESEMPIO

```
CREATE TABLE Impiegati
(
    Codice CHAR(8) NOT NULL,
    Nome CHAR(20),
    AnnoNascita INTEGER CHECK (AnnoNascita < 2000),
    Qualifica CHAR(20) DEFAULT 'Impiegato',
    Supervisore CHAR(8),
    PRIMARY KEY pk_impiegato (Codice),
    FOREIGN KEY fk_Impiegati (Supervisore)
    REFERENCES Impiegati )
```



```
CREATE TABLE FamiliariACarico
(
    Nome CHAR(20),
    AnnoNascita INTEGER,
    GradoParentela CHAR(10),
    CapoFamiglia CHAR(8)
    FOREIGN KEY fk_FamiliariACarico (CapoFamiglia)
    REFERENCES Impiegati )
```

DEFINIZIONE DI TABELLE

- Ciò che si crea con un `CREATE` si può eliminare con il comando `DROP` o cambiare con il comando `ALTER`.

`CREATE TABLE Nome`

`(Attributo Tipo [ValoreDefault] [VincoloAttributo]
{, Attributo Tipo [Default] [VincoloAttributo]}
{, VincoloTabella})`

`Default := DEFAULT {valore | null | username}`

- Nuovi attributi si possono aggiungere con:

`ALTER TABLE Nome ADD COLUMN NuovoAttr Tipo`

Modificare una tabella

Con il comando **ALTER TABLE** è possibile (standard SQL):

1. Aggiungere una colonna (**ADD [COLUMN]**)
2. Eliminare una colonna (**DROP [COLUMN]**)
3. Modificare la colonna (**MODIFY**)
4. Aggiungere l'assegnazione di valori di default (**SET DEFAULT**)
5. Eliminare l'assegnazione di valori di default (**DROP DEFAULT**)
6. Aggiungere vincoli di tabella (**ADD CONSTRAINT**)
7. Eliminare vincoli di tabella (**DROP CONSTRAINT**)
8. Altre opzioni sono possibili nei linguaggi specifici (vedi manuali)

Aggiungere una colonna

Sintassi:

ALTER TABLE nome_tabella

ADD [COLUMN] nome_col tipo_col default_col vincolo_col

In mancanza di altre specifiche, la nuova colonna viene inserita come ultima colonna della tabella.

Altrimenti è possibile dare questa specifica:

ADD COLUMN <CREA_DEFINIZIONE> [FIRST/AFTER <nome_colonna>]

FIRST permette di aggiungerla come prima colonna/AFTER colonna subito dopo la colonna indicata

ESEMPIO: Aggiungere alla tabella Impiegato la colonna nomecapo.

ALTER TABLE impiegato

ADD COLUMN nomecapo varchar(20) default 'Rossi' not null

Regole per aggiungere una colonna

- Si può aggiungere una colonna in qualsiasi momento se **non** viene specificato **NOT NULL**.
- Come si può aggiungere una colonna **NOT NULL** con tre passaggi?

Eliminare una colonna

ALTER TABLE nome_tabella

DROP COLUMN nome_colonna {**RESTRICT/CASCADE**}

In SQL standard le opzioni **RESTRICT/CASCADE** sono alternative ed è obbligatorio specificare l'una o l'altra

RESTRICT: se un'altra tabella si ha un vincolo di integrità referenziale con questa colonna, l'esecuzione del comando drop fallisce.

CASCADE: eliminando la colonna, vengono eliminate tutte le dipendenze logiche di altre colonne dello schema da questa.

ALTER TABLE Impiegato
Drop column dipart **restrict**

ALTER TABLE impiegato
Drop column dipart **cascade**

Modificare una colonna

Se si vogliono modificare le caratteristiche di una colonna dopo averla definita, occorre eseguire l'istruzione:

```
ALTER TABLE nome_tabella MODIFY  
nome colonna tipo_col default_col vincoli_col
```

ESEMPIO: Supponendo che nella tabella Impiegato ci sia una colonna 'nome' definita come varchar(20), modificarla in modo che diventi un varchar(30) e sia definito su di essa il vincolo not null.

```
ALTER TABLE impiegato MODIFY  
nome varchar(30) not null
```

Assegnare un valore di default

Nell'SQL standard è possibile imporre un valore di default col comando specifico SET DEFAULT, con la seguente sintassi

```
ALTER TABLE nome_tabella  
ALTER [COLUMN] nome_colonna  
SET DEFAULT valore_default
```

ESEMPIO: Imporre il valore di default 'Direzione Generale' ai valori della colonna Dipart in cui tale valore non è assegnato esplicitamente

```
ALTER TABLE Impiegato  
Alter [column] Dipart  
SET DEFAULT 'Direzione Generale'
```


Eliminare un valore di default

In SQL standard è possibile eliminare un vincolo di default da una colonna mediante l'istruzione

```
ALTER TABLE nome_tabella  
ALTER [COLUMN] nome_colonna  
DROP DEFAULT
```

Eseguendo questa istruzione il valore di default diventa automaticamente NULL

Esempio: Eliminare il default introdotto nell'esercizio precedente

```
ALTER TABLE Impiegato  
ALTER [COLUMN] Dipart  
DROP DEFAULT
```

Aggiungere vincoli di tabella (che vedremo in dettaglio dopo)

Se si vuole aggiungere un vincolo di tabella, si esegue il comando

```
ALTER TABLE nome_tabella
```

```
ADD CONSTRAINT nome_vincolo vincolo_di_tabella
```

ESEMPIO: Nella tabella Impiegato, aggiungere un vincolo di unicità alla coppia (nome, cognome)

```
ALTER TABLE impiegato
```

```
ADD CONSTRAINT unique_const unique(nome, cognome)
```

N.B.: Occorre assegnare un nome al vincolo

Esempi

Aggiungere un vincolo di chiave primaria

```
ALTER TABLE Info_Personali ADD CONSTRAINT  
Pkey PRIMARY KEY (id_impiegato);
```

Aggiungere un vincolo di chiave esterna

```
ALTER TABLE Info_Personali ADD CONSTRAINT  
Fkey FOREIGN KEY (id_Impiegato) REFERENCES Impiegati (id_impiegato)
```

Aggiungere un vincolo di unicità

```
ALTER TABLE Info_Personali ADD CONSTRAINT unique_con UNIQUE  
(codice_fiscale)
```

Aggiungere un vincolo CHECK

```
ALTER TABLE Info_Personali ADD CONSTRAINT check_con CHECK  
(stipendio > 0)
```

Eliminare vincoli di tabella

Nello standard SQL, se si vuole eliminare un vincolo di tabella si esegue l'istruzione

ALTER TABLE nome_tabella

DROP CONSTRAINT nome_vincolo{**RESTRICT/CASCADE**}

L'opzione **RESTRICT** non permette di eliminare vincoli di unicità e di chiave primaria su una colonna se esistono vincoli di chiave esterna che si riferiscono a tale colonna.

L'opzione **CASCADE** non opera questa restrizione.

Da notare che per eliminare un vincolo, esso deve essere definito mediante un identificatore

Drop Table

Si può eliminare una tabella mediante l'istruzione DROP TABLE

Nello standard SQL si possono anche specificare le opzioni

RESTRICT/CASCADE

RESTRICT: se la tabella è utilizzata nella definizione di altri oggetti dello schema, la sua eliminazione viene impedita.

CASCADE: vengono eliminate tutte le dipendenze degli altri oggetti dello schema da questa tabella

Esercizio

1. Creare una tabella studenti che contiene matricola, nome cognome data di nascita e numero di esami effettuati, senza specificare alcun vincolo.
2. Dopo aver creato la tabella, aggiungere una colonna con la media dei voti.
3. Aggiungere quindi le colonne telefono ed email.
4. Quindi modificare la tabella in modo tale da rendere il numero di matricola chiave primaria.
5. Aggiungere un vincolo di tabella, specificando che la tripla nome cognome e data di nascita non può essere uguale per diversi studenti.
6. Cancellare la colonna relativa al numero di esami effettuati
7. Eliminare il vincolo creato al punto 5.
8. Eliminare le colonne email e numero di telefono.

Domanda 1

- Creare una tabella studenti che contiene matricola, nome cognome data di nascita e numero di esami effettuati, senza specificare alcun vincolo.

```
CREATE TABLE studenti  
(  
    matricola char(6),  
    nome varchar(20),  
    cognome varchar(20),  
    nascita date,  
    n_esami number(3)  
)
```

Domanda 2 e 3

2. Dopo aver creato la tabella,aggiungere una colonna con la media dei voti.

```
Alter table studenti add  
media_voti number(5,2) check (media_voti>=0)
```

3. Aggiungere quindi le colonne telefono ed email.

```
Alter table studenti add  
(telefono varchar(15),  
Email varchar(20))
```

4. Quindi modificare la tabella in modo tale da rendere il numero di matricola chiave primaria.

Alter table studenti modify
Matricola char(6) primary key

5. Aggiungere un vincolo di tabella, specificando che la tripla nome cognome e data di nascita non può essere uguale per diversi studenti.

Alter table studenti add constraint
ncn_unique unique(nome,cognome, nascita)

6. Cancellare la colonna relativa al numero di esami effettuati

Alter table studenti
Drop column n_esami

7. Eliminare il vincolo creato al punto 5.

Alter Table studenti drop constraint ncn_unique

8. Eliminare le colonne email e numero di telefono.

Alter table studenti
Drop email, drop telefono

I vincoli

Vincoli intrarelazionali

- I **vincoli di integrità** consentono di limitare i valori ammissibili per una determinata colonna della tabella in base a specifici criteri.
- I vincoli di integrità intrarelazionali (ossia che non fanno riferimento ad altre relazioni) sono:
 - NOT NULL
 - UNIQUE definisce chiavi
 - PRIMARY KEY: chiave primaria (una sola, implica NOT NULL)
 - CHECK, vedremo più avanti

UNIQUE

- Può essere espresso in due forme:
 - nella definizione di un attributo, se forma da solo la chiave
 - come elemento separato
- Il vincolo **unique** utilizzato nella definizione dell'attributo indica che non ci possono essere due valori uguali in quella colonna.
- E' una chiave della relazione, ma non una chiave primaria.

Esempio di vincolo unique

```
CREATE TABLE Impiegato(  
    Matricola CHAR(6) PRIMARY KEY,  
    Codice_fiscale CHAR(16) UNIQUE,  
    Nome VARCHAR(20) NOT NULL,  
    Cognome VARCHAR(20) NOT NULL,  
    Dipart VARCHAR(15),  
    Stipendio NUMBER(9) DEFAULT 0,  
    FOREIGN KEY(Dipart) REFERENCES Dipartimento(NomeDip),  
    UNIQUE (Cognome, Nome)  
)
```

Vincolo unique per insiemi di attributi

- Il **vincolo di unicità** può anche essere riferito a coppie o insiemi di attributi. Ciò non significa che per gli attributi dell'insieme considerato ogni singolo valore deve apparire una sola volta, ma che non ci siano due dati (righe) per cui l'insieme dei valori corrispondenti a quegli attributi siano uguali.
- In questo caso il vincolo viene dichiarato dopo aver dichiarato tutte le colonne mediante un vincolo di tabella, utilizzando il comando

Unique (lista attributi)

Esempio vincolo unique per insiemi di attributi

```
CREATE TABLE Impiegato(  
    Matricola CHAR(6) PRIMARY KEY,  
    codice_fiscale CHAR(16) UNIQUE,  
    Nome VARCHAR(20) NOT NULL,  
    Cognome VARCHAR(20) NOT NULL,  
    Dipart VARCHAR(15),  
    Stipendio NUMBER(9) DEFAULT 0,  
    FOREIGN KEY(Dipart) REFERENCES Dipartimento(NomeDip),  
    UNIQUE (Cognome, Nome)  
)
```

Scrivere:

**Nome VARCHAR(20) NOT NULL UNIQUE,
Cognome VARCHAR(20) NOT NULL UNIQUE,
Sarebbe stato equivalente?**

Primary key

Due forme:

- nella definizione di un attributo, se formato da solo la chiave
- come elemento separato

Il vincolo **PRIMARY KEY** è simile a unique, ma definisce la chiave primaria della relazione, ossia un attributo che individua univocamente un dato.

Implica sia il vincolo **UNIQUE** che il vincolo **NOT NULL** (non è ammesso che per uno degli elementi della tabella questo valore sia non definito). Serve ad indentificare univocamente i soggetti del dominio. Questo vincolo permette spesso il collegamento fra due tabelle (vedremo in seguito)

Esempio chiave primaria

```
CREATE TABLE Impiegato(  
  Matricola CHAR(6) PRIMARY KEY,  
  Codice_fiscale CHAR(16) UNIQUE,  
  Nome VARCHAR(20) NOT NULL,  
  Cognome VARCHAR(20) NOT NULL,  
  Dipart VARCHAR(15),  
  Stipendio NUMBER(9) DEFAULT 0,  
  FOREIGN KEY(Dipart) REFERENCES Dipartimento(NomeDip),  
  UNIQUE (Cognome, Nome)  
)
```

Chiave primaria con insiemi di attributi

Analogamente al vincolo unique, anche il vincolo di chiave primaria può essere definito su un insieme di elementi.

In tal caso la sintassi è simile a quella di unique

Primary key (lista di attributi)

```
CREATE TABLE Studente(  
  Nome VARCHAR(20),  
  Cognome VARCHAR(20),  
  nascita DATE,  
  Corso_Laurea VARCHAR(15),  
  Facolta VARCHAR (20)  
  PRIMARY KEY(Cognome, Nome, Nascita)  
)
```

Vincoli interrelazionali

- I vincoli interrelazionali sono quei vincoli che vengono imposti quando gli attributi di due diverse tabelle devono essere messi in relazione.
- Questo è fatto per soddisfare l'esigenza di un database di **non essere ridondante** e di avere i **dati sincronizzati**.
- Se due tabelle gestiscono gli stessi dati, è bene che di essi non ce ne siano più copie, sia allo scopo di non occupare troppa memoria, sia affinché le modifiche fatte su dati uguali utilizzati da due tabelle siano coerenti.

- **REFERENCES** e **FOREIGN KEY** permettono di definire vincoli di integrità referenziale
- di nuovo **due sintassi**
 - per singoli attributi (come vincolo di colonna)
 - su più attributi (come vincolo di tabella)
- E' possibile definire politiche di **reazione alla violazione** (ossia stabilire l'azione che il DBMS deve compiere quando si viola il vincolo)

Vincolo di chiave esterna come vincolo di colonna

```
CREATE <nome tabella>  
(attributo_1...,  
  attributo_2...,  
  ...  
  attributo_k REFERENCES tabella_riferita(colonna_riferita)  
  ...  
)
```

Vincolo di chiave esterna come vincolo di tabella

```
CREATE <nome_tabella>  
(attributo_1 ...,  
  attributo_2 ...,  
  ...  
  attributo_n ...,  
  FOREIGN KEY (col_referenti) REFERENCES  
    tab_riferita(col_riferite)  
  ...  
)
```

Vincolo referenziale, esempio

Auto

<u>Stato</u>	<u>Numero</u>	Cognome	Nome
IT	39548K	Rossi	Mario
UK	E39548	Rossi	Mario
FR	839548	Neri	Luca

Come vincolo
di tabella

Infrazioni

<u>Codice</u>	Data	Vigile	<u>Stato</u>	<u>Numero</u>
34321	1/2/95	3987	IT	39548K
53524	4/3/95	3295	UK	E39548
64521	5/4/96	3295	FR	839548
73321	5/2/98	9345	FR	839548

Come vincolo
di colonna

Vigili

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

Vincolo references, foreign key, esempio

```
Create TABLE Vigili (  
  Matricola INTEGER PRIMARY KEY,  
  Nome VARCHAR (15),  
  Cognome VARCHAR (15))
```

```
CREATE TABLE Auto (  
  Stato CHAR(2),  
  Numero CHAR (6),  
  Cognome VARCHAR(15),  
  Nome VARCHAR(15),  
  Primary key (stato, numero))
```

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

Vincolo: CHECK

Un vincolo di **CHECK** richiede che una colonna, o una combinazione di colonne, soddisfi una condizione per ogni riga della tabella.

Il vincolo **CHECK** deve essere una espressione booleana che è valutata usando i valori della colonna che vengono inseriti o aggiornati nella riga.

Può essere espresso sia come **vincolo di riga** che come **vincolo di tabella**.

Se è espresso come **vincolo di riga**, può coinvolgere solo l'attributo su cui è definito, mentre se serve eseguire un check che coinvolge due o più attributi, si deve definire come **vincolo di tabella**

Nessuno stipendio degli impiegati può avere valore minore di 0 (espresso come vincolo di riga).

```
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 CHECK (Stipendio >= 0),  
    FOREIGN KEY(Dipart) REFERENCES  
        Dipartimento(NomeDip),  
    UNIQUE (Cognome, Nome)  
)
```

CHECK, esempio

Nessuno stipendio degli impiegati può avere valore minore di 0 (espresso come vincolo di tabella).

```
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) ,  
    CHECK (Stipendio>=0)  
)
```

Check, esempio

I dipartimenti si possono trovare solo nelle locazioni di Boston, New York e Dallas.

```
CREATE TABLE Dipartimenti  
(dip_cod char(4) primary key,  
dip_nome varchar2(20) not null,  
dip_citta varchar2(15) not null,  
CHECK (dip_citta = 'Boston'  
       or dip_citta = 'New York'  
       or dip_citta = 'Dallas')  
)
```

Esempio Check

Un vincolo check sullo stipendio e la commissione per evitare che la commissione sia più alta del salario.

```
Create table pagamenti(  
pag_cod char(6),  
pag_codicef char(16) REFERENCES dipendenti(dipe_codicef),  
pag_stipendio number(8,2),  
pag_commissione number (8,2),  
check (pag_stipendio > pag_commissione) )
```

VINCOLI D'INTEGRITA': CHIAVI E GENERALI

- Vincoli su attributi
 - VincoloAttributo :=
[NOT NULL [UNIQUE]] | [CHECK (Condizione)]
[REFERENCES Tabella [(Attributo {, Attributo})]]
- Vincoli su tabella
 - VincoloTabella := UNIQUE (Attributo {, Attributo})
| CHECK (Condizione) |
| PRIMARY KEY [Nome] (Attributo {, Attributo})
| FOREIGN KEY [Nome] (Attributo {, Attributo})
REFERENCES Tabella [(Attributo {, Attributo})]
[ON DELETE {NO ACTION| CASCADE | SET NULL}]

Reazione alla violazione

Quando si crea un vincolo foreign key in una tabella, in SQL standard si può specificare l'azione da intraprendere quando delle righe nella tabella riferita vengono cancellate o modificate.

Tali reazioni alla violazione vengono dichiarate al momento della definizione dei vincoli di foreign key rispettivamente mediante i comandi

ON DELETE
ON UPDATE

Sintassi

Per vincoli foreign key **di colonna**:

```
CREATE TABLE nome_tabella  
( attr_1...,  
  attr_2...,  
  ...  
  attr_k ... REFERENCES tab_riferita(attr_riferito)  
                        ON DELETE/ON UPDATE reazione  
  ...  
  attr_n ...  
  vincoli tabella)
```

Per vincoli foreign key **di tabella**:

```
CREATE TABLE nome_tabella  
( attr_1...,  
  attr_2...,  
  ...  
  attr_n ...,  
  FOREIGN KEY tab_referente(attr_referenti) REFERENCES  
    tab_riferita(attr_riferito) ON DELETE/ON UPDATE reazione
```

Reazioni alla violazione on delete

Impedire il delete (NO ACTION): Blocca il delete delle righe dalla tabella riferita quando ci sono righe che dipendono da essa.

Questa è l'azione che viene attivata per **default**.

Generare un delete a catena (CASCADE): Cancella tutte le righe dipendenti dalla tabella quando la corrispondente riga è cancellata dalla tabella riferita.

Assegnare valore NULL (SET NULL): Assegna NULL ai valori della colonna che ha il vincolo foreign key nella tabella quando la riga corrispondente viene cancellata dalla tabella riferita.

Assegnare il valore di default (SET DEFAULT): Assegna il valore di default ai valori della colonna che ha il vincolo foreign key nella tabella quando la riga corrispondente viene cancellata dalla tabella riferita.

Reazioni alla violazione on update

Nello standard SQL la reazione alla violazione può anche essere attivata quando i dati della tabella riferita vengono aggiornati.

Viene attivato mediante il comando **ON UPDATE** seguito da:

CASCADE : Le righe della tabella referente vengono impostati ai valori della tabella riferita

SET NULL : i valori della tabella referente vengono impostati a NULL

SET DEFAULT : i valori della tabella referente vengono impostati al valore di default

NO ACTION : rifiuta gli aggiornamenti che violino l'integrità referenziale

ESEMPIO

```
CREATE TABLE Impiegati
( Codice CHAR(8) NOT NULL,
  Nome CHAR(20) NOT NULL,
  AnnoNascita INTEGER NOT NULL,
  Dipartimento CHAR(20),
  Stipendio FLOAT NOT NULL,
  Supervisore CHAR(8),
  PRIMARY KEY pk_impiegato (Codice),
  FOREIGN KEY fk_Impiegati (Supervisore)
    REFERENCES Impiegati
    ON DELETE SET NULL
)
```

ESEMPIO

```
CREATE TABLE FamiliariACarico  
( Nome CHAR(20) NOT NULL,  
  AnnoNascita INTEGER NOT NULL,  
  GradoParentela CHAR(10) NOT NULL,  
  CapoFamiglia CHAR(8) NOT NULL,  
  PRIMARY KEY pk_ FamiliariACarico (CapoFamiglia, Nome)  
  FOREIGN KEY fk_ FamiliariACarico (CapoFamiglia)  
  REFERENCES Impiegati  
  ON DELETE CASCADE )
```

viste

Viste

Le **Viste Logiche** o **Viste** o **View** possono essere definite come delle tabelle virtuali, i cui dati sono riaggregazioni dei dati contenuti nelle tabelle "fisiche" presenti nel database.

Le tabelle fisiche sono gli unici veri contenitori di dati.

Le viste non contengono dati fisicamente diversi dai dati presenti nelle tabelle, ma forniscono una diversa visione, dinamicamente aggiornata, di quegli stessi dati.

La vista appare all'utente come una normale tabella, in cui può effettuare **interrogazioni** e, limitatamente ai suoi privilegi, anche **modifiche dei dati**.

Viste, Vantaggi

- Le viste **semplificano la rappresentazione dei dati**. Oltre ad assegnare un nome alla vista, la sintassi dell'istruzione CREATE VIEW consente di cambiare i nomi delle colonne
- Le viste possono essere anche estremamente **convenienti per svolgere una serie di query molto complesse**
- Le viste consentono di **proteggere i database**: le view ad accesso limitato possono essere utilizzate per controllare le informazioni alle quali accede un certo utente del database
- Le viste consentono inoltre di **convertire le unità di misura e creare nuovi formati**

- non è possibile utilizzare gli operatori booleani **UNION, INTERSECT ED EXCEPT**;
- Gli operatori intersect ed except possono essere realizzati mediante una select semplice. La stessa cosa non si può dire dell'operatore Union
- Non è possibile utilizzare la clausola **ORDER BY**

Viste, sintassi

Il comando DDL che consente di definire una vista ha la seguente sintassi

```
CREATE VIEW NomeVista [ ( ListaAttributi ) ] AS SelectSQL  
[ with [ local | cascaded ] check option ]
```

I nomi delle colonne indicati nella lista attributi sono i nomi assegnati alle colonne della vista, che corrispondono ordinatamente alle colonne elencate nella select.

Se questi non sono specificati, le colonne della vista assumono gli stessi nomi di quelli della/e tabella/e a cui si riferisce.

Create View, Esempio

- Creare una vista contenente la matricola, il nome, il cognome e lo stipendio degli impiegati del dipartimento di Amministrazione il cui stipendio è maggiore di 1000 euro

Create view ImpiegatiAmmin

(Matricola, Nome, Cognome, Stipendio) AS

Select Matr, Nome, Cognome, Stip

From Impiegato

Where Dipart = 'Amministrazione' and

Stipendio > 1000

Create View, Esempio

Veicoli

<u>Targa</u>	Cod_mod	Cod_cat	Cilindrata	Cod_comb.	cav.Fisc	Velocita	Posti	Imm
--------------	---------	---------	------------	-----------	----------	----------	-------	-----

- Creare una vista che contiene la targa e la cilindrata delle macchine con cilindrata <1500

```
Create view PiccolaCilindrata  
(PC_Targa, PC_cilindrata)  
As Select targa, cilindrata  
From Veicoli  
Where cilindrata<1500
```

- La vista ottenuta è composta da due colonne denominate PC_targa e PC_cilindrata corrispondenti a Targa e Cilindrata di Veicoli e alle righe della stessa tabella in cui la cilindrata è minore di 1500

Modifica di una vista

Sebbene il **contenuto** di una vista sia **dinamico**, la sua **struttura non** lo è.

Se una vista è definita su una subquery

`Select * From T1`

E in seguito alla tabella T1 viene aggiunta una colonna, questa nuova definizione non si estende alla vista. Ossia la vista conterrà sempre le stesse colonne che aveva prima dell'inserimento della nuova colonna in T1.

Vista basata su due tabelle

Categorie

<u>Cod_cat</u>	Nome_cat
----------------	----------

Veicoli

<u>Targa</u>	Cod_mod	Categoria*	Cilindrata	Cod_comb	cav.Fisc	Velocita	Posti	Imm
--------------	---------	------------	------------	----------	----------	----------	-------	-----

Creare una vista che descrive la targa, il codice del modello e il nome della categoria dei veicoli.

```
Create view A2 as  
Select targa, Cod_Modello, Nome_Categoria  
From Veicoli, Categorie  
Where Categorie.cod_cat=Veicoli.Cod_cat
```

Si noti che manca la specifica dei nomi delle colonne della vista. In tal caso vengono acquisiti i nomi delle colonne della tabella madre.

Viste di gruppo

Una **vista di gruppo** è una vista in cui una delle colonne è una funzione di gruppo.


In questo caso è obbligatorio assegnare un nome alla colonna della vista corrispondente alla funzione di gruppo

Modelli

<u>Cod_Mod</u>	Nome_Mod	Cod_Fab	Cilind_Media	num_versioni
----------------	----------	---------	--------------	--------------

Creare una vista che per ciascuna fabbrica riporti il numero globale delle versioni dei modelli prodotti

```
Create view A3 (cod_Fabbrica, numero_versioni) AS  
Select cod_Fabbrica, sum(num_versioni)  
From Modelli  
Group by Cod_Fabbrica
```



Oss: nelle viste di gruppo è necessario ridenominare la colonna relativa all'operatore aggregato

Viste di gruppo

E' una vista di gruppo anche una vista che è definita in base ad una vista di gruppo

Esempio:

```
Create view A3 (cod_Fabbrica, numero_versioni) AS  
Select cod_Fabbrica, sum(num_versioni)  
From Modelli  
Group by Cod_Fabbrica
```

```
Create view A4 AS  
Select num_versioni  
From A3
```


Eliminazione delle viste

- Le viste si eliminano col comando Drop View.
- Sintassi:

Drop View nome_view {Restrict/Cascade}

Restrict: la vista viene eliminata solo se non è riferita nella definizione di altri oggetti

Cascade: oltre che essere eliminata la vista, vengono eliminate tutte le dipendenza da tale vista di altre definizioni dello schema

Esempio

```
Create view A3 (cod_Fabbrica, num_versioni) AS  
Select cod_Fabbrica, sum(numero_versioni)  
From Modelli  
Group by Cod_Fabbrica
```

```
Create view A4 AS  
Select num_versioni  
From A3
```

L'istruzione **Drop View A3 Cascade** elimina oltre che la vista A3, anche la vista A4 che dipende da essa.

Invece **Drop View A3 Restrict** impedisce la cancellazione di A3, finchè è presente A4, che dipende da essa

TABELLE INIZIALIZZATE E TABELLE CALCOLATE

- Tabelle inizializzate:

```
CREATE TABLE Nome EspressioneSELECT
```

```
CREATE TABLE Supervisor
```

```
SELECT Codice, Nome, Qualifica, Stipendio
```

```
FROM Impiegati
```

```
WHERE Supervisore IS NULL
```

- Tabelle calcolate (viste):

```
CREATE VIEW Nome [(Attributo {, Attributo})]
```

```
AS EspressioneSELECT [WITH CHECK OPTION];
```

```
CREATE VIEW Supervisor
```

```
AS SELECT Codice, Nome, Qual., Stip.
```

```
FROM Impiegati
```

```
WHERE Supervisore IS NULL
```

VISTE MODIFICABILI

- Le tabelle delle viste si interrogano come le altre, ma in generale non si possono modificare.
- Deve esistere una corrispondenza biunivoca fra le righe della vista e le righe di una tabella di base, ovvero:
 - 1) `SELECT` senza `DISTINCT` e solo di attributi
 - 2) `FROM` una sola tabella modificabile
 - 3) `WHERE` senza `SottoSelect`
 - 4) `GROUP BY` e `HAVING` non sono presenti nella definizione.
- Possono esistere anche delle restrizioni su `SELECT` su viste definite usando `GROUP BY`.

Aggiornamento delle VIEW

Le operazioni INSERT/UPDATE/DELETE sulle VIEW non erano permesse nelle prime edizioni di SQL

- I nuovi DBMS permettono di farlo con certe limitazioni dovute alla definizione della VIEW stessa
- Ha senso aggiornare una VIEW?

Dopotutto si potrebbe aggiornare la tabella di base direttamente ...

Aggiornamento delle VIEW

- ... utile nel caso di accesso dati controllato
- Esempio:
Impiegato(Nome, Cognome, Dipart, Ufficio, Stipendio)
- Il personale della segreteria non può accedere ai dati sullo stipendio ma può modificare gli altri campi della tabella, aggiungere e/o cancellare tuple
- Si può controllare l'accesso tramite la definizione della VIEW:
CREATE VIEW Impiegato2 AS
SELECT Nome, Cognome, Dipart, Ufficio
FROM Impiegato
INSERT INTO Impiegato2 VALUES (...)
- Stipendio verrà inizializzato a Null
- Se Null non è permesso per Stipendio l'operazione fallisce

Aggiornamento VIEW (2)

- Immaginiamo la seguente VIEW:

```
CREATE VIEW ImpiegatoRossi  
  
AS  
  
SELECT *  
  
FROM Impiegato  
  
WHERE Cognome='Rossi'
```

- La seguente operazione ha senso:

```
INSERT INTO ImpiegatoRossi (... 'Rossi', ...)
```

Aggiornamento VIEW (2)

CREATE VIEW ImpiegatoRossi

AS

SELECT *

FROM Impiegato

WHERE Cognome='Rossi'

- Ma che succede nel caso di:

INSERT INTO ImpiegatoRossi (... 'Bianchi', ...)

- In genere è permesso, finisce nella tabella base ma non è visibile dalla VIEW

With check option 1

- L'opzione **With Check Option** messa alla fine della definizione della vista assicura che le operazioni di inserimento e di modifica dei dati effettuate utilizzando la vista soddisfino la clausola Where della subquery.

```
CREATE VIEW ImpiegatoRossi AS  
    SELECT * FROM Impiegato  
    WHERE Cognome='Rossi'  
    WITH CHECK OPTION
```

- Adesso l'insert con 'Bianchi' fallisce, quella con 'Rossi' viene invece eseguita.

Local, Cascaded

Supponiamo che una **vista V1** sia definita in termini di **un'altra vista V2**. Se si crea V1 specificando la clausola **WITH CHECK OPTION**, il DBMS verifica che la nuova tupla t inserita soddisfi **sia la definizione di V1 che quella di V2** (e di tutte le altre eventuali viste da cui V1 dipende), **indipendentemente** dal fatto che V2 sia stata a sua volta definita **WITH CHECK OPTION**

Questo comportamento di default è equivalente a definire V1
WITH CASCADED CHECK OPTION

Lo si può alterare definendo V1
WITH LOCAL CHECK OPTION

Ora il DBMS verifica solo che t soddisfi la specifica di V1 e quelle di tutte e **sole le viste da cui V1 dipende per cui è stata specificata** la clausola **WITH CHECK OPTION**

Esempio

Veicoli

Targa	Cod_mod	Categoria	Cilindrata	Cod_comb.	cav.Fisc	Velocita	Posti	Imm
-------	---------	-----------	------------	-----------	----------	----------	-------	-----

Categorie

Cod_cat	Nome_cat
---------	----------

La seguente vista è aggiornabile

```
Create view A1  
(A1_Targa, A1_cilindrata)  
As Select targa, cilindrata  
From Veicoli  
Where cilindrata<1500
```

Quest'altra (in generale) invece non lo è

```
Create view A2 as  
Select targa, Cod_Modello, Nome_Categoria  
From Veicoli, Categorie  
Where Categorie.cod_cat=Veicoli.Cod_cat
```

—————→ Due tabelle

Vantaggi delle viste: facilitazione nell'accesso ai dati

- In generale uno dei requisiti per la progettazione di un database relazionale è la **normalizzazione dei dati**.
- Sebbene la forma normalizzata del database permette una corretta modellazione della realtà che il DB rappresenta, a volte dal punto di vista dell'utente comporta una maggiore difficoltà di comprensione rispetto a una rappresentazione non normalizzata.
- Le viste permettono di fornire all'utente i dati in una forma più intuitiva.

Vantaggi delle viste: diverse visioni dei dati

Esistono dei dati che sono presenti nelle tabelle del database, che sono **poco significativi per l'utente**, e altri che **devono essere nascosti all'utente** (esempio: lo stipendio di un dipendente, la password di un account etc.).

L'uso delle viste da parte dell'utente permette di **limitare il suo accesso ai dati del database**, eliminando quelli non interessanti per lui e quelli che devono essere tenuti nascosti.

L'uso delle viste può essere considerato come una **tecnica per assicurare la sicurezza dei dati**

Vantaggi delle Viste: Indipendenza Logica

Un vantaggio delle viste riguarda **l'indipendenza logica** delle applicazioni e delle operazioni eseguite dagli utenti rispetto alla struttura logica dei dati.

Ciò significa che è possibile poter operare modifiche allo schema senza dover apportare modifiche alle applicazioni che utilizzano il database.

UTILITÀ DELLE VISTE

- Per nascondere certe modifiche all'organizzazione logica dei dati (indipendenza logica)
- Per offrire visioni diverse degli stessi dati senza ricorrere a duplicazioni
- Per rendere più semplici, o per rendere possibili, alcune interrogazioni

Un'interrogazione non standard

Il dipartimento che impiega il massimo budget in stipendi dei dipendenti

Select Dipart

from Impiegato

group by Dipart

having sum(Stipendio) >= all

(select sum(Stipendio)

from Impiegato

group by Dipart)

Soluzione con le viste

```
CREATE VIEW BudgetStipendi(Dipartimento,TotaleStipendi) AS  
SELECT Dipart, sum(Stipendio)  
FROM Impiegato  
GROUP BY Dipart
```

```
SELECT Dipartimento  
FROM BudgetStipendi  
WHERE  
    TotaleStipendi =(SELECT max(TotaleStipendi)  
                      FROM BudgetStipendi)
```

La vista è utilizzata come una normale tabella. Di solito vengono utilizzate in questo modo quando si devono usare a catena due diversi operatori aggregati (la massima somma, il minimo numero...)

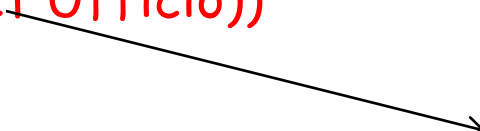
Ancora sulle viste

Calcolare la media del numero degli uffici distinti presenti in ogni dipartimento

Interrogazione scorretta

```
select avg(count(distinct Ufficio))  
from Impiegato  
group by Dipart
```

Due operatori
aggregati annidati



Con una vista

```
create view UfficiDipart (NomeDip,NroUffici) as  
select Dipart, count(distinct Ufficio)  
from Impiegato  
group by Dipart
```

```
select avg(NroUffici)  
from UfficiDipart
```

Procedure e trigger

CREATE PROCEDURE/FUNCTION

```
CREATE FUNCTION contaStudenti IS  
  DECLARE  
    tot INTEGER;  
  BEGIN  
    SELECT COUNT(*) INTO tot FROM STUDENTI;  
    RETURN (tot);  
  END
```

Trigger

- Un **trigger** definisce un'azione che il database deve attivare automaticamente quando si verifica (nel database) un determinato evento.
- Possono essere utilizzati:
 - per **migliorare l'integrità** referenziale dichiarativa
 - per **imporre regole complesse** legate all'attività del database
 - per **effettuare revisioni** sulle modifiche dei dati.

Trigger

- L'esecuzione dei trigger è quindi trasparente all'utente.
- I trigger vengono eseguiti automaticamente dal database quando specifici tipi di comandi (**Eventi**) di manipolazione dei dati vengono eseguiti su specifiche tabelle.
- Tali comandi comprendono i comandi DML **insert**, **update** e **delete**, ma gli ultimi DBMS prevedono anche trigger su istruzioni DDL come **Create View** ecc.
- Anche gli aggiornamenti di specifiche colonne possono essere utilizzati come trigger di eventi.

Trigger a livello di riga

- I trigger a livello di riga vengono eseguiti una volta per ciascuna riga modificata in una transazione; vengono spesso utilizzati in applicazioni di revisione dei dati e si rivelano utili per **operazioni di audit dei dati** e per **mantenere sincronizzati i dati distribuiti**.
- Per creare un trigger a livello di riga occorre specificare la clausola

FOR EACH ROW

- nell'istruzione create trigger.

Trigger a livello di istruzione

- I trigger a livello di istruzione vengono eseguiti **una sola volta per ciascuna transazione**, indipendentemente dal numero di righe che vengono modificate (quindi anche se, ad esempio, in una tabella vengono inserite 100 righe, il trigger verrà eseguito solo una volta).
- Vengono pertanto utilizzati per **attività correlate ai dati**; vengono utilizzati di solito per **imporre misure aggiuntive di sicurezza sui tipi di transazione che possono essere eseguiti su una tabella**.
- E' il tipo di trigger predefinito nel comando create trigger (ossia non occorre specificare che è un trigger al livello di istruzione).

Trigger, struttura

- I trigger si basano sul paradigma evento-condizione-azione (ECA).
- L'istruzione **Create Trigger** seguita dal **nome** assegnato al trigger
- **Tipo di trigger**, **Before/After**
- **Evento** che scatena il trigger **Insert/Delete/Update**
- **[For each row]**, Se si vuole specificare trigger al livello di riga (altrimenti nulla per trigger al livello di istruzione)
- Specificare a quale **tabella** si applica
- **Condizione** che si deve verificare perché il trigger sia eseguito
- **Azione**, definita dal codice da eseguire se si verifica la condizione

Trigger: sintassi

```
create trigger <NomeTrigger>  
Tipo di trigger Evento {, Evento}  
ON <TabellaTarget>  
for each row  
[when <Predicato SQL>]  
Blocco PL/SQL
```

Tipo di trigger Evento: before o after
Evento: insert, update, delete
for each row specifica la granularità.

In assenza di questa clausola si intende per ogni istruzione

ESEMPIO DI TRIGGER

```
CREATE TRIGGER ContolloStipendio
  BEFORE INSERT ON Impiegati
  DECLARE
    StipendioMedio FLOAT
  BEGIN
    SELECT avg(Stipendio) INTO StipendioMedio
      FROM   Impiegati
      WHERE  Dipartimento = :new.Dipartimento;
    IF :new.Stipendio > 2 * StipendioMedio
      THEN RAISE_APPL._ERR.(-2061, 'Stipendio alto')
    END IF;
  END;
```

Tipi di Trigger

- **BEFORE** e **AFTER**: i trigger possono essere eseguiti prima o dopo l'utilizzo dei comandi **insert**, **update** e **delete**; all'interno del trigger è possibile fare riferimento ai vecchi e nuovi valori coinvolti nella transazione.
- Occorre utilizzare la clausola
BEFORE/AFTER <tipo di evento> (insert, delete, update).
- Se si tratta di un trigger **BEFORE UPDATE**:
 - per valori **vecchi** intendiamo i valori che sono nella tabella e che vogliamo modificare
 - per **nuovi** quelli che vogliamo inserire al posto dei vecchi.
- Se si tratta di un trigger **AFTER UPDATE**:
 - per **vecchi** intendiamo quelli che c'erano prima dell'update
 - Per **nuovi** quelli presenti nella tabella alla fine della modifica.

Trigger Attivi e Passivi

- Un **trigger** è **attivo** quando, in corrispondenza di certi eventi, modifica lo stato della base di dati.
- Un **trigger** è **passivo** se serve a provocare il fallimento della transazione corrente sotto certe condizioni.

Tipi di Trigger

INSTEAD OF: per specificare che cosa fare invece di eseguire le azioni che hanno attivato il trigger.

- Ad esempio, è possibile utilizzare un trigger **INSTEAD OF** per reindirizzare le INSERT in una tabella verso una tabella differente o per aggiornare con update più tabelle che siano parte di una vista.
- I trigger **instead-of** possono essere definiti su viste (relazionali od oggetto).
- I trigger instead-of devono essere a livello di riga.

I TRIGGER

- Proprietà essenziale dei trigger: terminazione
- Utilità dei trigger
 - Trattare vincoli non esprimibili nello schema
 - Attivare automaticamente azioni sulla base di dati quando si verificano certe condizioni

Controllo degli accessi

Controllo degli accessi

- Ogni componente dello schema (risorsa) può essere protetta (tabelle, attributi, viste, domini, ecc.)
- Il possessore della risorsa (colui che la crea) assegna dei privilegi agli altri utenti
- Un utente predefinito (`_system`) rappresenta l'amministratore della base di dati ed ha completo accesso alle risorse
- Ogni privilegio è caratterizzato da:
 - la risorsa a cui si riferisce
 - l'utente che concede il privilegio
 - l'utente che riceve il privilegio
 - l'azione che viene permessa sulla risorsa
 - se il privilegio può esser trasmesso o meno ad altri utenti

CONTROLLO DEGLI ACCESSI - Tipi di privilegi

- Tipi di privilegi:
 - **SELECT**: lettura di dati
 - **INSERT [(Attributi)]**: inserire record (con valori non nulli per gli attributi)
 - **DELETE**: cancellazione di record
 - **UPDATE [(Attributi)]**: modificare record (o solo gli attributi)
 - **REFERENCES [(Attributi)]**: definire chiavi esterne in altre tabelle che riferiscono gli attributi.
- **WITH GRANT OPTION**: si possono trasferire i privilegi ad altri utenti.

CONTROLLO DEGLI ACCESSI

- Chi crea lo schema della BD è l'unico che può fare CREATE, ALTER e DROP
- Chi crea una tabella stabilisce i modi in cui altri possono farne uso:
 - GRANT Privilegi ON Oggetto TO Utenti [WITH GRANT OPTION]

grant e revoke

- Per concedere un privilegio ad un utente:

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

grant option specifica se ha il privilegio di propagare il privilegio ad altri utenti

Es.: grant select on Department to Stefano

Per revocare il privilegio:

```
revoke Privileges on Resource from Users [ restrict | cascade ]
```

Opzioni di grant e revoke

- La revoca deve essere fatta dall'utente che aveva concesso i privilegi
 - restrict (di default) specifica che il comando non deve essere eseguito qualora la revoca dei privilegi all'utente comporti qualche altra revoca (dovuta ad un precedente grant option)
 - cascade invece forza l'esecuzione del comando
- Attenzione alle reazioni a catena

CONTROLLO DEGLI ACCESSI (cont.)

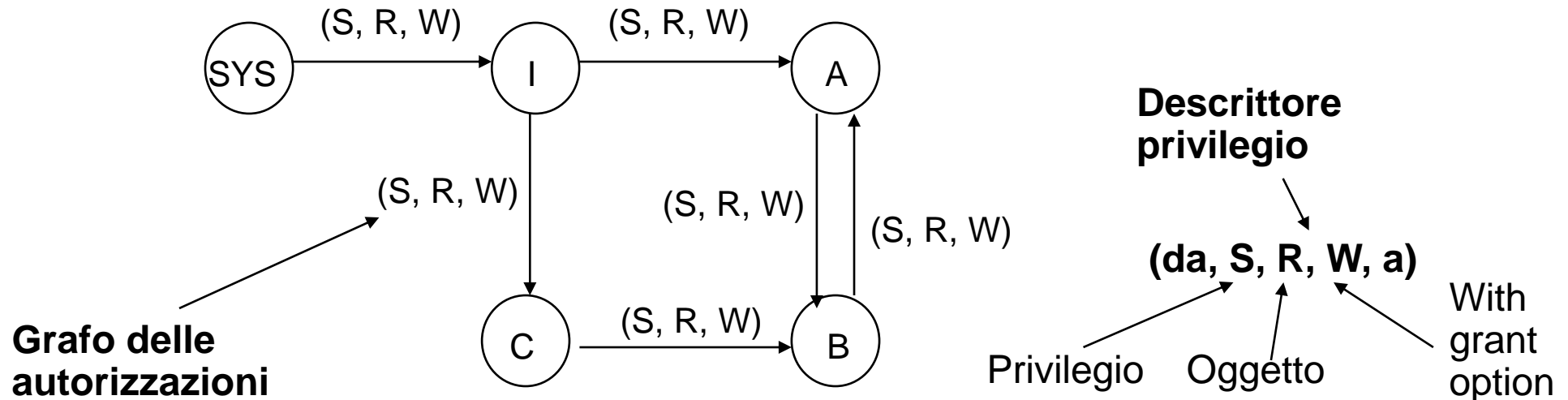
- Chi definisce una tabella o una VIEW ottiene automaticamente tutti i privilegi su di esse, ed è l'unico che può fare un DROP e può autorizzare altri ad usarla con GRANT.
- Nel caso di viste, il "creatore" ha i privilegi che ha sulle tabelle usate nella definizione.
- Le autorizzazioni si annullano con il comando:
 - REVOKE [GRANT OPTION FOR] Privilegi ON Oggetto FROM Utenti [CASCADE]
- Quando si toglie un privilegio a U, lo si toglie anche a tutti coloro che lo hanno avuto solo da U.

ESEMPI DI GRANT

- GRANT INSERT, SELECT ON Esami TO Tizio .
- GRANT DELETE ON On Esami TO Capo WITH GRANT OPTION
 - Capo può cancellare record e autorizzare altri a farlo.
- GRANT UPDATE (voto) ON Esami TO Sicuro
 - Sicuro può modificare solo il voto degli esami.
- GRANT SELECT, INSERT ON VistaEsamiBD1 TO Albano
 - Albano può interrogare e modificare solo i suoi esami.

GRAFO DELLE AUTORIZZAZIONI

- L'utente I ha creato la tabella R e innesca la seguente successione di eventi:
 - I: GRANT SELECT ON R TO A WITH GRANT OPTION
 - A: GRANT SELECT ON R TO B WITH GRANT OPTION
 - B: GRANT SELECT ON R TO A WITH GRANT OPTION
 - I: GRANT SELECT ON R TO C WITH GRANT OPTION
 - C: GRANT SELECT ON R TO B WITH GRANT OPTION



GRAFO DELLE AUTORIZZAZIONI: PROPRIETA'

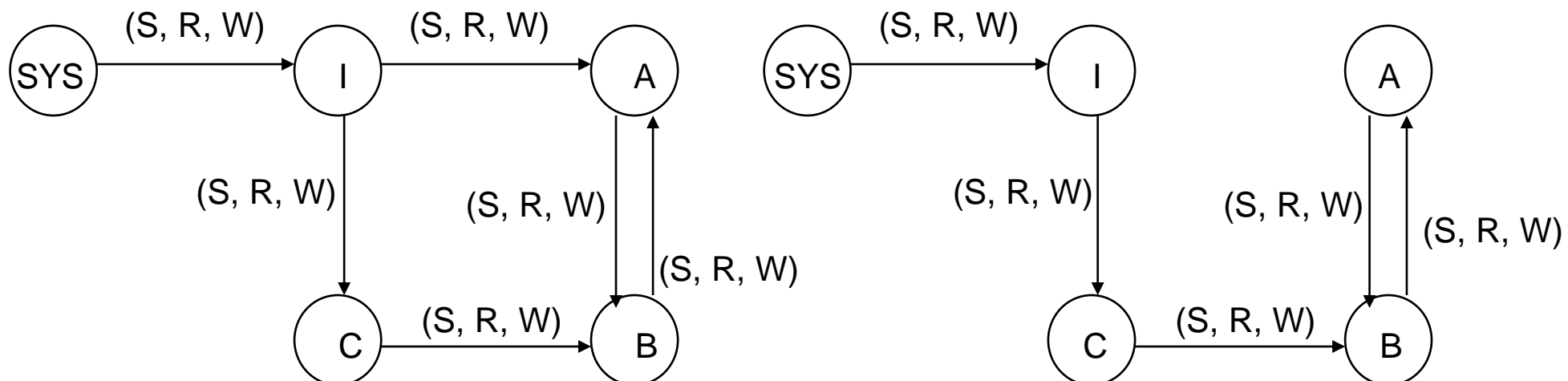
- Se un nodo N ha un arco uscente con un privilegio, allora esiste un cammino da *SYSTEM* a N con ogni arco etichettato dallo stesso privilegio + *WGO*.

- Effetto del *REVOKE*, ad es.

I: *REVOKE SELECT ON R FROM A CASCADE*

- e poi

I: *REVOKE SELECT ON R FROM C CASCADE*



Indice e catalogo

CREAZIONE DI INDICI

- Cosa sono e a cosa servono
- Non è un comando standard dell' SQL e quindi ci sono differenze nei vari sistemi
 - `CREATE INDEX NomeIdx ON Tabella(Attributi)`
 - `CREATE INDEX NomeIdx ON Tabella
WITH STRUCTURE = BTREE, KEY = (Attributi)`
 - `DROP INDEX NomeIdx`

CATALOGO (DEI METADATI)

- Alcuni esempi di tabelle, delle quali si mostrano solo alcuni attributi, sono:
 - Tabella delle password:
 - `PASSWORD(username, password)`
 - Tabella delle basi di dati:
 - `SYSDB(dbname, creator, dbpath, remarks)`
 - Tabella delle tabelle (type = view or table):
 - `SYSTABLES(name, creator, type, colcount, filename, remarks)`

CATALOGO (cont.)

- Alcuni esempi di tabelle, delle quali si mostrano solo alcuni attributi, sono:
 - Tabella degli attributi:
 - `SYSCOLUMNS(name, tbname, tbcreator, colno, coltype, lenght, default, remarks)`
 - Tabella degli indici:
 - `SYSINDEXES(name, tbname, creator, uniquerule, colcount)`
 - e altre ancora sulle viste, vincoli, autorizzazioni, etc. (una decina).

RIEPILOGO

- DDL consente la definizione di tabelle, viste e indici. Le tabelle si possono modificare aggiungendo o togliendo attributi e vincoli.
- Le viste si possono interrogare come ogni altra tabella, ma in generale non consentono modifiche dei dati.
- I comandi *GRANT* / *REVOKE* + viste offrono ampie possibilità di controllo degli usi dei dati.

RIEPILOGO

- SQL consente di dichiarare molti tipi di vincoli, oltre a quelli fondamentali di chiave e referenziale.
- Oltre alle tabelle fanno parte dello schema le procedure e i trigger.
- La padronanza di tutti questi meccanismi -- e di altri che riguardano aspetti fisici, affidabilità, sicurezza -- richiede una professionalità specifica (DBA).