Corso SQL e Database relazionali

Giuseppe ing. Grosso

Aeffegroup SRL

Via Dante Alighieri, 72

Indice generale

Introduzione

L’obiettivo del corso è quello di fornire agli allievi le competenze necessarie per poter progettare, implementare ed interrogare basi di dati relazionali utilizzando il linguaggio SQL; i concetti trattati sono applicabili ai più diffusi DBMS relazionali come Oracle, SQL Server, MySQL e PostgreSQL

Sintesi argomenti trattati:

- Introduzione ai database relazionali
- Approfondimento dei concetti principali: tabelle, campi, chiavi, relazioni, indici, vincoli, trigger ed integrità dei dati.
- Il disegno e l’implementazione di una base dati relazionale: regole e best practice.
- La sintassi del linguaggio SQL per la creazione di una base dati.
- La sintassi del linguaggio SQL per l’interrogazione di una base dati: particolare attenzione alle query con inner join, left join, full join.
- Tecniche avanzate per l’interrogazione di una base dati: ordinamento, raggruppamento, join tra tabelle.
- L’indicizzazione del database; analisi e miglioramento delle performance delle interrogazioni SQL.
- Introduzione alle transazioni.
- Introduzione all’SQL procedurale (PL/SQL, TSQL).

Requisiti ingresso

E’ richiesta la conoscenze delle nozioni di informatica di base.

Introduzione ai database relazionali.

Argomenti

- I Data base relazionali
- Tabelle e campi di un database
- Relazioni fra le tabelle

Base dati

- Una **base di dati** (BD) o *Data Base* (DB) è un sistema di archivi (file) organizzati per consentirne una facile consultazione
- DBMS (DB Management System) è il programma per gestire il DB
- Linguaggi per i DB
    - Data Definition Language (DDL)
    - Data Manipulation Language (DML)
    - Query Language (QL)
- Diversi tipi di organizzazione di DB:
    - Antichi: gerarchico, reticolare, a liste invertite, …
    - Attuali: relazionale

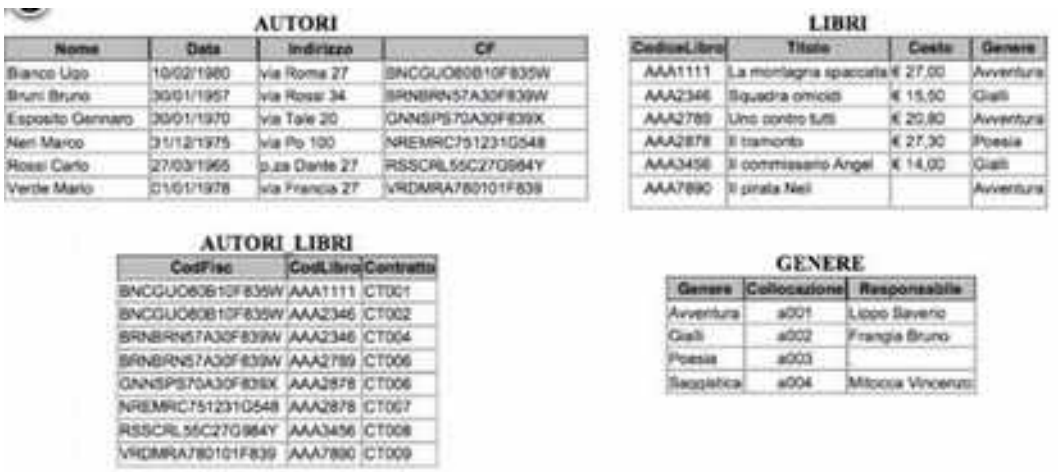
Relazioni e tabelle

- DB relazionale: l'organizzazione del DB basa sul concetto matematico di **"relazione"** che concretamente significa: **i dati sono organizzati in tabelle**

Ciclo di vita

- Progettazione
- Popolamento
- Interrogazione
- Aggiornamento dei dati
- Manutenzione

Es. di Database:



AUTORI

Nome	Data	Indirizzo	CF
Bianco Ugo	10/02/1980	via Roma 27	BNCGUO80B10F835W
Bruni Bruno	30/01/1957	via Rossi 34	BRNBRN57A30F839W
Esposito Gennaro	30/01/1970	via Tale 20	GNNSPS70A30F839X
Neri Marco	31/12/1975	via Po 100	NREMRC751231G548
Rossi Carlo	27/03/1965	p.za Dante 27	RSSCRL55C27G984Y
Verde Mario	01/01/1978	via Francia 27	VRDMRA780101F839

LIBRI

CodiceLibro	Titolo	Costo	Genere
AAA1111	La montagna spaccata	€ 27,00	Avventura
AAA2346	Squadra omicidi	€ 15,50	Gialli
AAA2789	Uno contro tutti	€ 20,80	Avventura
AAA2878	Il tramonto	€ 27,30	Poesia
AAA3456	Il commissario Angel	€ 14,00	Gialli
AAA7890	Il pirata Neil		Avventura

AUTORI_LIBRI

CodFisc	CodLibro	Contratto
BNCGUO80B10F835W	AAA1111	CT001
BNCGUO80B10F835W	AAA2346	CT002
BRNBRN57A30F839W	AAA2346	CT004
BRNBRN57A30F839W	AAA2789	CT006
GNNSPS70A30F839X	AAA2878	CT006
NREMRC751231G548	AAA2878	CT007
RSSCRL55C27G984Y	AAA3456	CT008
VRDMRA780101F839	AAA7890	CT009

GENERE

Genere	Collocazione	Responsabile
Avventura	a001	Lippo Saverio
Gialli	a002	Frangia Bruno
Poesia	a003	
Saggistica	a004	Milocca Vincenzo

Tabelle e campi di un Data Base

- Caratteristiche fondamentali del dato
  - Tipo: insieme di valori possibili e loro proprietà
  - Valore: il valore assunto
  - Attributo (nome): dà un significato al dato
- Esempi: tipo del “buon senso” e tipi del DBMS

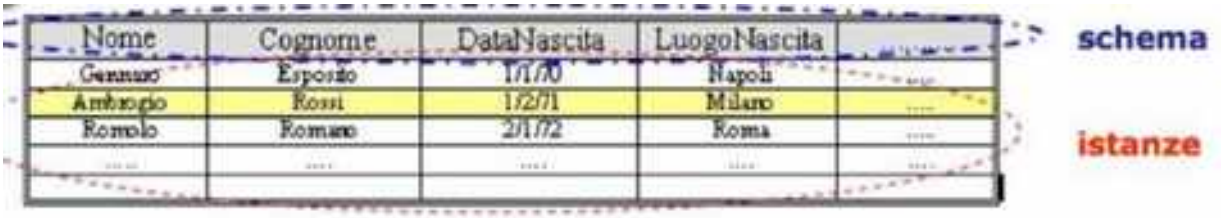
ATTRIBUTO	VALORE	TIPO	Nome
Radice dell'equazione	3.8	Numero reale	Radice
Numero di clienti	780	Numero intero	NumeroClienti
Cliente	Carlo Rossi	Nome di persona	Cliente
Prezzo in Euro	2970,00	Numero con 2 decimali	Prezzo
Codice del materiale	HP 29	Codice di magazzino	CODICE
Data di scadenza	31/12/2006	Data	SCADENZA
Sesso dell'impiegato	F	(M,F)	SESSO

Campi e record

La tabella come rappresentazione del concetto matematico di relazione fra insiemi

R ⊆ D1 × D2 × , … × Dn (Prodotto cartesiano)

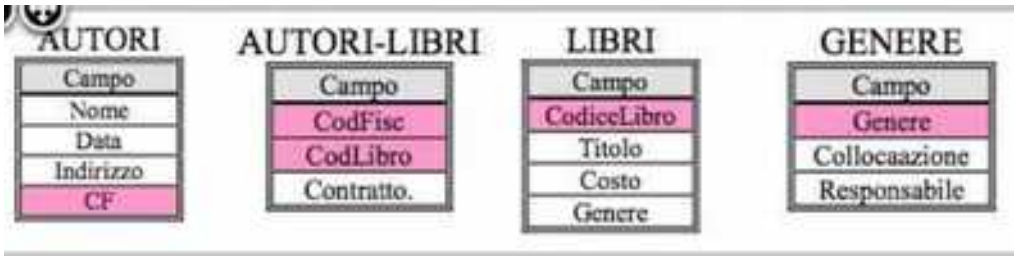
- Schema e istanza di una tabella
- Schema e istanze di una Base dati relazionale



Chiave primaria

- Valore Nullo. Un valore non definito di un campo
- Chiave primaria: Un valore che identifica univocamente un record
- Un valore-chiave

- Non può essere nullo
- Non può essere ripetuto
- Può essere “semplice” o “multiplo”



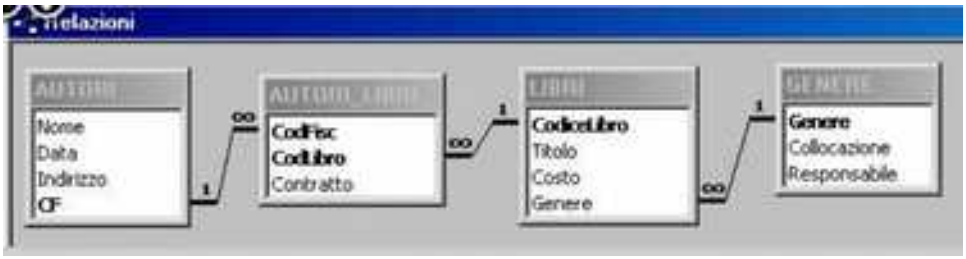
Viene colorato il campo chiave primario.

Campi-indice

- Un **indice** ordina “logicamente” i record di una tabella rispetto ad un campo, detto **campo-indice** o indicizzato.
- La tabella mantiene il suo ordinamento fisico, ma è associata ad un “indice” che mette in corrispondenza gli indirizzi dei record della tabella con i valori del campo-indice e viene ordinato rispetto a questo.
- L’indice velocizza la ricerca dei valori del campo.
- La chiave primaria è un indice, ma a questo se ne possono aggiungere altri.
- Possono essere indicizzati anche campi “non chiave”, cioè con valori duplicati ammessi (al contrario delle chiavi)
- Il vantaggio di velocità nella ricerca di un valore del campo-indice si paga con il tempo necessario per la generazione dell’indice. Indicizzare o no un campo è dunque una delicata scelta di progetto

Relazioni fra tabelle

- Relazioni 1 a molti: un record di una tabella contiene dati che interessano molti record di un’altra
- Relazioni 1 a 1: un record di una tabella contiene dati che interessano 1 record di un’altra (l’una è la “continuazione” dell’altra)
- Il lato “1″ della relazione è una chiave primaria
- Il lato “molti” (indicato con ∞ da ACCESS) si dice **Chiave esterna**



Integrità referenziale

- Le relazioni fra tabelle si “dichiarano” nel DBMS
  - o direttamente
  - o dichiarando le chiavi esterne

- Così, il DBMS può controllare il rispetto delle **Integrità referenziale**: **La chiave primaria di una relazione fra tabelle non si può cancellare**

Approfondimento dei concetti principali.

Tabelle.

Una **tabella** è una collezione di dati correlati fra loro, consiste in una serie di colonne e di righe. Ogni colonna rappresenta i campi della tabella, questi sono definiti in sede di progettazione della tabella. Mentre le righe rappresentano i valori associati ai campi, questi ultimi vengono inseriti in numero variabile e rappresentano le varie righe di contenuto della tabella.
Creazione di un database in mySQL:

```
CREATE DATABASE nome_database
```

Definizione di una tabella in MySQL.

```
CREATE table nome_tabella (
    nome_colonna tipo_colonna [ clausola_default ] [ vincoli_di_colonna ]
    [ , nome_colonna tipo_colonna [ clausola_default ] [ vincoli_di_colonna ] … ]
    [ , [ vincolo_di tabella] … ] )
```

Campi.

nome_colonna: è il nome della colonna che compone la tabella. Sarebbe meglio non esagerare con la lunghezza degli identificatori di colonna, dal momento che l'SQL Entry Level prevede nomi non piu' lunghi di 18 caratteri. Si consulti comunque la documentazione dello specifico database. I nomi devono iniziare con un carattere alfabetico.
tipo_colonna: è l'indicazione del tipo di dato che la colonna potra' contenere. I principali tipi previsti dallo standard SQL sono:

- CHARACTER(n)
  Una stringa a lunghezza fissa di esattamente n caratteri. CHARACTER puo' essere abbreviato con CHAR
- CHARACTER VARYING(n)
  Una stringa a lunghezza variabile di al massimo n caratteri. CHARACTER VARYING puo' essere abbreviato con VARCHAR o CHAR VARYING.
- INTEGER
  Un numero intero con segno. Puo' essere abbreviato con INT. La precisione, cioe' la grandezza del numero intero che puo' essere memorizzato in una colonna di questo tipo, dipende dall'implementazione del particolare DBMS.
- SMALLINT
  Un numero intero con segno con precisione non superiore a INTEGER.
- FLOAT(p)
  Un numero a virgola mobile, con precisione p. Il valore massimo di p dipende dall'implementazione del DBMS. E' possibile usare FLOAT senza indicazione della

precisione, utilizzando quindi la precisione di default, anch'essa dipendente dall'implementazione. REAL e DOUBLE PRECISION sono dei sinonimi per un FLOAT con una particolare precisione. Anche in questo caso le precisioni dipendono dall'implementazione, con il vincolo che la precisione del primo non sia superiore a quella del secondo.

- DECIMAL(p,q)
  Un numero a virgola fissa di almeno p cifre e segno, con q cifre dopo la virgola. DEC e' un'abbreviazione per DECIMAL. DECIMAL(p) e' un'abbreviazione per DECIMAL(p,0). Il valore massimo di p dipende dall'implementazione.
- INTERVAL
  Un periodo di tempo (anni, mesi, giorni, ore, minuti, secondi e frazioni di secondo).
- DATE, TIME e TIMESTAMP
  Un preciso istante temporale. DATE permette di indicare l'anno, il mese e il giorno. Con TIME si possono specificare l'ora, i minuti e i secondi. TIMESTAMP e' la combinazione dei due precedenti. I secondi sono un numero con la virgola, permettendo cosi' di specificare anche frazioni di secondo.

clausola_default: indica il valore di default che assumerà la colonna se non gliene viene assegnato uno esplicitamente nel momento della creazione della riga. La sintassi da utilizzare e' la seguente:

DEFAULT { valore | NULL }

dove, valore e' un valore valido per il tipo con cui la colonna e' stata definita.

vincoli_di_colonna: sono vincoli di integrita' che vengono applicati al singolo attributo. Sono:

- NOT NULL, che indica che la colonna non puo' assumere il valore NULL.
- PRIMARY KEY, che indica che la colonna e' la chiave primaria della tabella.
- una definizione di riferimento, con cui si indica che la colonna e' una chiave esterna verso la tabella e i campi indicati nella definizione. La sintasi e' la seguente:

  REFERENCES nome_tabella [(colonna1 [, colonna2 …])]
  [ON DELETE { CASCADE | SET DEFAULT | SET NULL }]
  [ON UPDATE { CASCADE | SET DEFAULT | SET NULL }]

  Le clausole ON DELETE e ON UPDATE indicano quale azione deve essere compiuta nel caso in cui una tupla nella tabella referenziata venga eliminata o aggiornata. Infatti in tali casi nella colonna referenziante (che e' quella che si sta definendo) potrebbero esserci dei valori inconsistenti. Le azioni possono essere:

  - CASCADE: eliminare la tupla contenente la colonna referenziante (nel caso di ON DELETE) o aggiornare anche la colonna referenziante (nel caso di ON UPDATE).
  - SET DEFAULT: assegnare alla colonna referenziante il suo valore di default.
  - SET NULL: assegnare alla colonna referenziante il valore NULL.

un controllo di valore, con il quale si permette o meno l'assegnazione di un valore alla colonna, in base al risultato di un espressione. La sintassi da usare e':
CHECK (espressione_condizionale)

dove espressione_condizionale e' un'espressione che restituisce vero o falso.
Ad esempio, se stiamo definendo la colonna COLONNA1, definendo il seguente controllo:

CHECK (COLONNA1 < 1000)

in tale colonna potranno essere inseriti solo valori inferiori a 1000.

Chiavi.

vincolo_di_tabella: sono vincoli di integrita' che possono riferirsi a piu' colonne della tabella. Sono:

- la definizione della chiave primaria:

  PRIMARY KEY (colonna1 [, colonna2 …])

  Si noti che in questo caso, a differenza della definizione della chiave primaria come vincolo di colonna, essa puo' essere formata da piu' di un attributo.

Relazioni.

Mysql come altri database permette la definizioni delle chiavi esterne:

FOREIGN KEY (colonna1 [, colonna2 …]) definizione_di_riferimento

La definizione_di_riferimento ha la stessa sintassi e significato di quella che puo' comparire come vincolo di colonna.

- un controllo di valore, con la stessa sintassi e significato di quello che puo' essere usato come vincolo di colonna.

Indici.

La creazione di indici in un database MySQL permette di evitare che ogni ricerca sia preceduta da una scansione completa delle tabelle utilizzate (*full table scan*). L'indicizzazione è stata ideata appositamente per velocizzare l'esecuzione delle query di selezione e viene introdotta semplicemente utilizzando l'apposito comando, `CREATE INDEX`, seguito dal nome del campo o dei campi interessati alla generazione degli indici:

```
mysql> CREATE INDEX nome_cognome ON tbl (nome,cognome);
```

In alternativa è possibile creare un indice anche per alterazione della tabella:

```
mysql> ALTER TABLE tbl ADD INDEX nome_cognome (nome,cognome);
```

MySQL consente di creare fino a 16 indici all'interno di una stessa tabella, sono inoltre supportati indici su più colonne, indici multipli relativi a più colonne e indici per ricerche full-text. In pratica, con gli indici effettueremo le stesse operazioni che si svolgono comunemente in una biblioteca quando si ordinano i vari testi per titolo, autore, argomento ecc., evitando così di dover passare in rassegna tutti i libri ogni volta che si presenta la necessità di consultarne uno solo o una parte di essi.

Vincoli.

I **constraint** (o vincoli), sono delle regole che vengono stabilite all'interno di una tabella che servono ad evitare che l'utente possa inserire dati non conformi ad alcune regole. Queste regole possono preservare l'inserimento dei dati o l'integrità referenziale tra le tabelle.

Possono essere dichiarati sia in fase di creazione tabella (`CREATE TABLE`), sia in fase di modifica della tabella (`ALTER TABLE`). Se il constraint è applicato ad una singola colonna allora è detto constraint di colonna, altrimenti se è applicato a più di una colonna è detto constraint di tabella.

Durante la dichiarazione del vincolo possiamo associargli un nome (usando la clausola `CONSTRAINT`), in caso contrario sarà Oracle a definirne uno di sistema (inizierà sempre con il prefisso `SYS_`). I vincoli vengono memorizzati da Oracle nel dizionario dei dati.

Nota: In tutti gli esempi riportati in questo capitolo usiamo l'utente "myself" creato nelle lezioni precedenti.

Constraint NOT NULL

Può essere definito **esclusivamente su una singola colonna** della tabella. La sua funzione è quella di evitare che l'utente possa inserire valori nulli (`NULL`) nella colonna associata, pertanto questa dovrà sempre contenere un valore diverso da `NULL`.

Sinstassi del vincolo NOT NULL

<nome colonna> <tipo> CONSTRAINT <nome vincolo> NOT NULL

Sinstassi del vincolo NOT NULL (anonimo)

<nome colonna> <tipo> NOT NULL

Per esempio creiamo una tabella denominata "myTableNull" avente due colonne, entrambe "`NOT NULL`" e ne visualizziamo la struttura.

```
CREATE TABLE myTableNull
(
   Cognome VARCHAR2(15) CONSTRAINT nn_myTableNull_Cognome NOT NULL,
   Nome VARCHAR2(15) CONSTRAINT nn_myTableNull_Nome NOT NULL
);
```

trigger ed integrità dei dati.

Regole aziendali. Ulteriori vincoli, detti **vincoli di integrità generici** in quanto non legati al modello relazionale, sono quelli imposti dalle regole aziendali. Tali vincoli possono essere rappresentati in SQL in più modi: mediante il costrutto `check` nella definizione di una tabella, mediante le asserzioni, oppure attraverso l'uso di regole attive (*trigger*).

E' bene chiarire in anticipo che i vincoli di integrità generici rappresentano un argomento contrastato. A differenza dei vincoli relazionali, gli strumenti per specificare vincoli generici non sono stabilmente inseriti nello standard SQL (ad esempio i trigger sono stati aggiunti sono nell'ultima versione di SQL dopo essere stati disponibili per molto tempo nei DBMS). Di conseguenza, mentre i vincoli tipici del modello relazionale sono supportati efficientemente da tutti i DBMS relazionali, gli strumenti per specificare vincoli generici variano notevolmente tra i vari

DBMS disponibili e non sempre garantiscono l'efficienza del sistema risulante. E' quindi fortemente consigliato accertarsi di quali siano e di come funzionino gli strumenti per vincoli generici supportati dal DBMS prescelto.

Il costrutto `check` permette di specificare, mediante una condizione come quella che può apparire nella clausola `where` di una interrogazione SQL, vincoli generici a livello di di tabella o, mediante le asserzioni, a livello di schema di base di dati.

Si noti che un uso indiscriminato di questi vincoli appesantisce il carico del sistema in quanto, solitamente, i DBMS garantiscono una implementazione efficiente solo per i vincoli propri del modello relazionale.

Vediamo alcuni esempi. Supponiamo di avere le tabelle dipendente, teatro e lavoro. La tabella lavoro associa i dipendenti ai relativi teatri. Un teatro può avere più dipendenti e un dipendente può lavorare per più teatri. Supponiamo di voler esprimere un **vincolo massimo di partecipazione** per un dipendente rispetto alla relazione lavoro: un dipendente non può lavorare per più di due teatri. Questo vincolo può essere specificato sulla tabella lavoro nel seguente modo:

```
create table teatro
(
  nome             varchar(20) primary key,
  indirizzo        varchar(40) not null
)

create table dipendente
(
  cf                    char(16) primary key,
  nome                  varchar(20) not null,
  cognome               varchar(20) not null,
  dataDiNascita         date,
  luogoDiNascita        varchar(30),
  capo                  char(16),
  foreign key capo references dipendente(cf)
)

create table lavoro
(
  teatro              varchar(20),
  dipendente          char(16),
  primary key(teatro, dipendente),
  foreign key teatro references teatro(nome),
  foreign key dipendente references dipendente(cf),
  check(2 >= (select count(*)
              from lavoro L
              where dipendente = L.dipendente))
)
```

Il vincolo afferma che per ogni dipendente non ci possono essere più di due righe nella tabella lavoro, quindi più di due teatri per cui il dipendente lavoro. Nella query di definizione del vincolo si può usare il nome degli attributi sui quali si sta definenendo il vincolo (dipendente in questo caso).

Lo stesso vincolo può essere espresso mediante una **asserzione**. Solitamente viene scritta una interrogazione SQL che seleziona le righe della base di dati che violano il vincolo. La condizione dell'asserzione viene fomata mettendo tale interrogazione come argomento del predicano `not exists`. Dunque il vincolo di integrità specificato dall'asserzione è verificato che il risultato della interrogazione è vuoto, cioè se non esistono righe che violano il vincolo. Vediamo un esempio:

```
create assertion limitaImpieghi check (not exists(

  select dipendente
  from lavoro
  group by dipendente
  having count(*) > 2

))
```

Supponiamo ora di voler affermare il seguente **vincolo minimo di partecipazione** per un dipendente rispetto alla relazione lavoro: ogni dipendente deve essere assunto presso almeno un teatro. Possiamo imporre questo vincolo sull'attributo `cf` della tabella dipendente:

```
create table dipendente
(
  cf                    char(16) primary key,
  nome                  varchar(20) not null,
  cognome               varchar(20) not null,
  dataDiNascita         date,
  luogoDiNascita        varchar(30),
  capo                  varchar(20),
  foreign key capo references dipendente(cf),
  check (cf in (select dipendente from lavoro))
)
```

Lo stesso vincolo può essere espresso mediante una asserzione come segue:

```
create assertion disoccupato check (not exists (

  select cf
  from dipendente
  where cf not in (select dipendente from lavoro)

))
```

Vediamo un vincolo che coinvolge più attributi della stessa tabella. Ad esempio, supponiamo di voler affermare che i dipendenti nati a Milano devono essere nati prima del 1970. Possiamo riscrivere la definizione della tabella dipendente come segue:

```
create table dipendente
(
  cf                    char(16) primary key,
  nome                  varchar(20) not null,
  cognome               varchar(20) not null,
  dataDiNascita         date,
  luogoDiNascita        varchar(30),
  capo                  varchar(20),
  foreign key capo references dipendente(cf),
  check (luogoDiNascita <> "Milano" or
         dataDiNascita < '1970-01-01')
)
```

Vediamo un esempio di vincolo che coinvolge più tabelle. Supponiamo di voler specificare il seguente vincolo minimo di partecipazione: un teatro deve avere almeno 5 dipendenti. Possiamo scrivere la seguente asserzione:

```
create assertion vincoloDipendentiTeatro check (

  not exists (select nome
              from teatro
```

```
              where nome not in (select teatro from lavoro))

  and

  not exists (select count(*)
              from lavoro
              group by teatro
              having count(*) < 5)

)
```

Il vincolo asserisce che non esistono teatri privi di dipendenti e, tra quelli che hanno almeno un dipendente, non esistono teatri con meno di 5 dipendenti. Dunque tutti i teatri hanno almeno 5 dipendenti.

Inoltre, vediamo un vincolo sulla cardinalità di una tabella. La seguente asserzione afferma che ci devono essere almeno 3 teatri nella rete:

```
create assertion vincoloTeatriRete check (

  3 <= (select count(*) from teatro)

)
```

Le **regole attive** (*trigger*) permettono di *gestire* i vincoli di integrità. La differenza rispetto agli strumenti fin ora introdotti per specificare vincoli di integrità (relazionali o generici) è la seguente: un trigger specifica una azione da intraprendere qualora in vincolo non sia soddisfatto, solitamente una azione riparatrice della integrità violata.

Un trigger segue il **paradigma evento-condizione-azione**: se un certo evento si verifica, la relativa condizione viene controllata e, se soddisfatta, l'azione viene intrapresa. Un evento è solitamento un aggiornamento della base di dati (insert, update, delete). Una condizione è un predicato espresso in SQL. Una azione è una interrogazione SQL (solitamente di aggiornamento della base di dati) oppure una eccezione che annulla gli effetti dell'operazione che ha attivato il trigger riportando la base di dati allo stato precedente a tale operazione (*rollback*). Il trigger può essere attivato prima o dopo l'evento.

Si noti che ci possono essere più trigger associati ad un evento. L'**ordine di esecuzione** dei trigger in tal caso è gestito dal sistema e generalmente tiene conto dell'ordine di creazione dei trigger. Un trigger che come azione aggiorna lo stato della base di dati può a sua volta innescare altri trigger, che a loro volta possono attivare altri trigger, con la possibilità di avere **reazioni a catena infinite**. Inoltre, l'azione di un trigger può violare vincoli di integrità. La violazione di un vincolo di integrità di chiave esterna può causare, come conseguenza delle politiche di gestione di tali vincoli, ulteriori modifiche alla base di dati che al loro volta possono scatenare altri trigger, oppure violare altri vincoli di integrità. Si badi bene che la violazione di un vincolo di integrità non gestito, a qualsiasi livello della catena di attivazione, produce un **annullamento degli effetti** di tutte le operazioni innescate dalla primitiva madre che ha generato la catena di trigger, compresi gli effetti della primitiva madre stessa. I trigger sono dunque strumenti semplici da scrivere in modo indipendente ma difficili da gestire in modo integrato.

Supponiamo di voler specificare un trigger per il vincolo che afferma che lo stipendio di un dipendente non può essere incrementato più del 20%:

```
create trigger LimitaIncrementoStipendio
```

```
after update of stipendio on dipendente
for each row
when (New.stipendio > Old.Stipendio * 1.2)
update dipendente
set New.stipendio = Old.Stipendio * 1.2
where cf = New.cf
```

Il trigger LimitaIncrementoStipendio viene attivato dall'evento modifica (update) dello stipendio di un dipendende. Per ogni riga modificata, se il nuovo stipendio è stato incrementato più del 20% rispetto al vecchio (condizione when), allora lo stipendio viene incrementato del massimo possibile senza violare il vincolo di integrità. Si noti che è possibile usare le variabili di tupla `New` e `Old` per riferirsi, rispettivamente, alla tupla dopo e prima la modifica. Per gli eventi di inserimento, solo New è accessibile, per gli eventi di cancellazione, solo Old è accessibile.

Vediamo un altro esempio. Vogliamo modellare la regola che dice che una prenotazione per uno spettacolo può essere effettuata solo se vi sono ancora posti a disposizione in sala. Usiamo i seguenti quattro trigger:

```
create trigger disponibilità-1
after insert on messaInScena
for each row
update messaInScena
set postiDisponibili = (select capienza
                        from spazio
                        where nome = New.spazio)
where (data = New.data and
       ora = New.ora
       spazio = New.spazio)

create trigger disponibilità-2
after insert on prenotazione
for each row
update messaInScena
set postiDisponibili = postiDisponibili - 1
where (data = New.dataSpettacolo and
       ora = New.oraSpettacolo
       spazio = New.spazioSpettacolo)


create trigger disponibilità-3
after delete on prenotazione
for each row
update messaInScena
set postiDisponibili = postiDisponibili + 1
where (data = Old.dataSpettacolo and
       ora = Old.oraSpettacolo
       spazio = Old.spazioSpettacolo)

create trigger disponibilità-4
before insert on prenotazione
for each row
when (0 = (select postiDisponibili
           from messaInScena
           where (data = New.dataSpettacolo and
                  ora = New.oraSpettacolo
                  spazio = New.spazioSpettacolo)))
rollback("Posti esauriti")
```

Per specificare la regola aziendale sui posti disponibili abbiamo usato i seguenti trigger:

1. disponibilità-1, che imposta il numero di posti disponibili alla capienza dello spazio teatrale quando uno spettacolo viene inserito;
2. disponibilità-2, che decrementa di uno i posti disponibili quando una prenotazione viene inserita;
3. disponibilità-3, che incrementa di uno i posti disponibili quando una prenotazione viene cancellata;
4. disponibilità-4, che controlla, *prima* dell'inserimento della prenotazione nella base di dati, se esistono posti disponibili. Se non ne esistono, esso annulla l'operazione di inserimento e avvisa l'utente che i posti sono esauriti.

Si noti che la soluzione funziona assumendo che disponibilità-4 venga eseguito prima di disponibilità-2 (di solito è così in quanto disponibilità-4 è di tipo before e disponibilità-2 è di tipo after).

I trigger sono anche utili per specificare le regole di calcolo degli **attributi calcolati**. Supponiamo che il prezzo ridotto di uno spettacolo debba essere scontato del 20% rispetto a quello intero. Dunque l'attributo prezzo ridotto è calcolato rispetto al prezzo intero. I trigger per gestire questo vincolo seguono:

```
create trigger CalcolaPrezzoRidottoInsert
after insert on messaInScena
for each row
update messaInScena
set New.prezzoRidotto = New.prezzoIntero * 0.8
where codice = New.codice

create trigger CalcolaPrezzoRidottoUpdate
after update of prezzoIntero on messaInScena
for each row
update messaInScena
set New.prezzoRidotto = New.prezzoIntero * 0.8
where codice = New.codice
```

Non tutti i vincoli di integrità possono essere descritti a livello di schema in SQL. Solitamente, quando un vincolo non è descrivibile in SQL, esso viene catturato a livello di applicazione implementandolo in qualche linguaggio di programmazione. E' bene che tutti i vincoli esprimibili in SQL vengano definiti a livello di schema in modo da renderli condivisi da tutte le applicazioni invece che replicare il vincolo per ogni applicazione. In tal modo le modifiche di un vincolo sono gestite a livello di schema senza modificare le applicazioni. Si parla in tal caso di **indipendenza dalla conoscenza**, dove per conoscenza si intende l'insieme delle regole codificate nei vincoli che regolano l'integrità della base.

E' possibile aggiungere e rimuovere vincoli di integrità definiti su una tabella mediante il comando `alter`. Per rimuovere un vincolo occorre averlo definito per nome mediante il costrutto `constraint`. Ad esempio:

```
create table dipendente
(
  cf                    char(16) primary key,
  nome                  varchar(20) not null,
  cognome               varchar(20) not null,
  indirizzo             varchar(30),
  constraint chiaveCandidata unique(nome, cognome)
)
```

```
alter table dipendente drop constraint chiaveCandidata
alter table dipendente add constraint chiaveCandidata unique(indirizzo)
```

Per rimuovere una asserzione o un trigger occorre usare il comando `drop` seguito dal nome del costrutto.

Concludiamo la parte sulla definizione dei dati in SQL parlando brevemente del **catalogo dei dati**. Il catalogo dei dati è una base relazionale per archiviare lo schema fisico di una base di dati; tale base contiene una descrizione dei dati e non i dati veri e propri. Ad esempio, il catalogo dei dati contiene una tabella per gli attributi delle tabelle di uno schema fisico. Ogni riga della tabella specifica, tra l'altro, il nome dell'attributo, la tabella di appartenenza, il suo valore di default e l'obligatorietà.

Il catalogo dei dati viene solitamente mantenuto dal DBMS e non deve essere creato o modificato dall'utente. Il catalogo dei dati può però essere interrogato dall'utente. Questo offre la possibilità interessante di costruire interrogazioni che accedano sia ai dati che ai metadati. E' bene che i dati e i metadati vengano organizzati nel medesimo modello dei dati (relazionale, ad oggetti, XML). In questo modo è possibile archiviare dati e metadati con le stesse strutture e interrogarli con lo stesso linguaggio. Questa caratteristica prende il nome di **riflessività**.

Il disegno e l’implementazione di una base dati relazionale: regole e best practice.

La sintassi del linguaggio SQL per la creazione di una base dati.

La sintassi del linguaggio SQL per l’interrogazione di una base dati.

Tecniche avanzate per l’interrogazione di una base dati: ordinamento, raggruppamento, join tra tabelle.

L’indicizzazione del database.

Analisi e miglioramento delle performance delle interrogazioni SQL.

Introduzione alle transazioni.

Introduzione all’SQL procedurale (PL/SQL, TSQL).