ISTITUTO TECNICO INDUSTRIALE
G. M. ANGIOY
SASSARI

PROGETTAZIONE DI DATABASE

Definizione dei dati in SQL

Lezione 13

Versione del: 21/04/2017
Revisione numero: 3

Prof. Andrea Zoccheddu
Dipartimento di Informatica

Immagine di copertina da: http://www.iamsterdam.com/en-GB/living/education/Dutch-Education-System

DIPARTIMENTO
INFORMATICA E TELECOMUNICAZIONI

DEFINIZIONE DEI DATI IN SQL

LE DICHIARAZIONI DI STRUTTURE E DI OGGETTI

DUE GRUPPI DI COMANDI: DDL E DML

I comandi del linguaggio SQL sono divisi in tre grandi gruppi che formano due sottospecie di linguaggio, che sono:

DATA DESCRIPTION LANGUAGE	**LINGUAGGIO DI DESCRIZIONE DEI DATI.** È l'insieme dei comandi SQL con cui definire (costruire, eliminare e modificare) le strutture del database, ovvero il database stesso e gli elementi che lo compongono, per esempio le tabelle. I suoi comandi ricordano le dichiarazioni dei dati come in Pascal (const, type, var) o in C++ (typedef, dichiarazioni di variabile o funzione).
DATA MANIPULATION LANGUAGE	**LINGUAGGIO DI MANIPOLAZIONE DEI DATI.** È l'insieme dei comandi SQL con cui agire (costruire, eliminare e modificare ed interrogare) sui dati interni alle strutture del database, ovvero i record delle tabelle. I suoi comandi ricordano le istruzioni di linguaggi come Pascal (if then else) o in C++ (while, for).
DATA ACCESS LANGUAGE	**LINGUAGGIO DI ACCESSO AI DATI** Talvolta considerato un sottoinsieme del DDL. È l'insieme dei comandi SQL con cui gestire (creare, modificare, eliminare e associare) utenti, gruppi e diritti. I suoi comandi ricordano le direttive dei sistemi operativi con cui proteggere il sistema creando account, privilegi e gruppi di utenti.

DATA DESCRIPTION LANGUAGE

Il DDL permette diversi comandi ciascuno rivolto a adempiere ad uno specifico compito. Sebbene alcuni comandi non siano permessi in tutti i database, possiamo ricordare:

COMANDO		DESCRIZIONE
CREATE	DATABASE	Serve per creare un nuovo database
CREATE OR ALTER	DOMAIN	Serve per definire un nuovo tipo di dato
CREATE OR ALTER	TABLE	Serve per definire una tabella dentro il database
CREATE OR ALTER	INDEX	Serve per definire un indice di ricerca sui dati
CREATE OR ALTER	VIEW	Serve per definire una vista (query con diritti)
CREATE OR ALTER	TRIGGER	Serve per definire un meccanismo automatico sui dati
CREATE OR ALTER	ASSERTION	Serve per creare un vincolo globale sul database
COMANDO		**DESCRIZIONE**
CREATE OR ALTER	USER	Serve per definire un utente
CREATE OR ALTER	GROUP	Serve per definire un gruppo di utenti
CREATE OR ALTER	GRANT	Serve per definire i diritti di accesso ai dati

Note:

- Il comando CREATE DOMAIN è sostituito dal comando CREATE TYPE in MS SQL Server
- Il comando CREATE ASSERTION è sostituito dal comando CREATE RULE in MS SQL Server
- I comandi CREATE USER/GROUP/GRANT costituiscono i comandi del DAL (Data Access)

DATA MANIPULATION LANGUAGE

SELECT	Serve per interrogare il database e ottenere informazioni calcolate
INSERT INTO	Serve per inserire record nelle tabelle
DELETE	Serve per cancellare record dalle tabelle
UPDATE	Serve per modificare alcuni campi di record esistenti nelle tabelle

CREAZIONE DEL DATABASE

All'inizio c'era il nulla … e il database non esiste ancora. Esso va costruito prima di poter definire le sue strutture interne (es. le tabelle) e di poter operare coi dati. La sintassi solitamente è:

SQL

spazio base

```
CREATE DATABASE nome_database;
```

Esempio

Questo comando crea un nuovo database con un suo nome.

```
CREATE DATABASE Scuola;
```

In alcuni ambienti si possono usare comandi del sistema operativo invece di questo comando (es. Access richiede un salvataggio, in Unix si può usare un comando shell, ecc…).

Quando è creato questo database è vuoto… ma a volte il sistema inganna ed il motore del database server costruisce un insieme di tabelle di sistema (non visibili ad utenti generici) per predisporre un ambiente agevole: ad esempio alcuni preparano le tabelle per i contatori (i campi auto-incrementanti) o per gli utenti e i diritti… ma questo possiamo trascurarlo.

TIPI DI DATO PREESISTENTI

Come tutti i linguaggi, anche SQL dispone di tipi di dato pronti. Questi tipi servono per sapere cosa poter archiviare dentro le tabelle (numeri, testo, ecc…). Purtroppo non c'è molta uniformità tra i diversi prodotti disponibili (commerciali o liberi) e lo standard.

Dato	Significato	Standard	MySql	Access	SQL-Server
CHAR	Un carattere singolo	SQL92	CHAR	TESTO (1)	NCHAR (1)
CHAR(N)	Frase di N caratteri	SQL92	CHAR(N)	TESTO (N)	NCHAR (1)
VARCHAR(N)	Una stringa di lunghezza variabile con un massimo di n caratteri	SQL92	CHAR(N)	TESTO (N)	NVARCHAR (N)
INTEGER	Intero (al massimo nove cifre numeriche)	SQL92	INTEGER	INTERO	INT
SMALLINT	Intero più piccolo di Integer	SQL92	SMALLINT	INTERO	SMALLINT
FLOAT	Numero a virgola mobile	SQL92	FLOAT	DECIMALE	FLOAT
FLOAT(N)	Numero a virgola mobile di N bit	SQL92	FLOAT(N)	DECIMALE	DECIMAL(A,B)
REAL	Numero a virgola mobile (in teoria è più preciso di FLOAT)	SQL92	REAL	DECIMALE	
DOUBLE PRECISION	Numero a virgola mobile (più o meno equivalente a REAL).	SQL92	DOUBLE PRECISION	DECIMALE	
DATE	Data, di solito nella forma *mm/gg/aaaa*	SQL92	DATE	DATA	DATE
TIME	Orario, nella forma *hh:mm:ss*, oppure solo *hh:mm*.	SQL92	TIME	DATA/ORA	TIME
TIMESTAMP	Informazione completa data-orario.	SQL92	TIMESTAMP	DATA/ORA	TIMESTAMP
INTERVAL	Intervallo di tempo.	SQL92	INTERVAL	---	
BOOLEAN	Dato Vero o Falso	NO	BOOLEAN	LOGICO	
BIT	Singolo bit (usato come boolean)	NO	---	---	BIT
BIT(N)	Lista di bit	NO	---	---	
OBJECT BLOB // OLE	Oggetto esterno di grande dimensione (immagine, audio, filmato)	NO	BLOB	OLE	OLE
	Denaro valuta	NO	VALUTA	MONEY	MONEY
	Date e orario	NO	DATA	DATETIME	DATETIME
	Interi molto grandi (8 byte)	NO		BIGINT	BIGINT
	Testo formato UNICODE	NO			NTEXT
	Immagini	NO	BLOB	OLE	IMAGE

NUOVI TIPI DI DATO

In un sistema per database compatibile con lo standard è possibile definire nuovi tipi di dato idonei per informazioni particolari. Nel modello logico relazionale le tabelle si chiamano Relazioni e i valori ammissibili nei campi sono detti Domini.

CREATE DOMAIN

Per definire un nuovo tipo di dato occorre specificare il suo Nome, il tipo di partenza da cui ottenerlo e una condizione per verificare il requisito a cui il tipo deve rispondere.

SQL

spazio base

```
CREATE DOMAIN NomeTipo AS TipoBase
  CHECK Condizione;
```

Esempio

Questo comando crea un nuovo database con un suo nome.

```
CREATE DOMAIN POSITIVO AS INT
  CHECK (VALUE > 0);
```

costruisce un tipo detto Positivo che di fatto è un intero i cui elementi devono soddisfare la condizione di essere maggiori di zero (zero escluso, in questo caso).

STRUTTURA DELLE TABELLE

La trattazione delle Tabelle è spiegata nel manuale SQL (pag. 147 – 154) e quindi non verrà riproposto in queste dispense. Di seguito si propongono solo alcuni esempi. Tutte le righe sono terminate da virgole, tranne l'ultima che è seguita dalla parentesi chiusa.

CHIAVE PRIMARIA SU UN CAMPO

Esempio

Questo comando crea una nuova tabella con un suo nome

```
CREATE TABLE Lavoratori (
   Matricola INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
   Cognome NCHAR(50) NOT NULL ,
   Nome NCHAR(75) NOT NULL ,
   Data_Nascita DATE ,
   Paga_Mensile MONEY ,
   Sesso NCHAR(1) NOT NULL
   )
```

PK su singolo campo

crea una nuova tabella denominata Studenti con i campi Matricola (PK), Cognome, Nome, Data_Nascita, Sesso. Si osservi che:

- La PK è un autoincrementante (parte da 1 e avanza a passi +1)
- Cognome è un testo fino a 50 caratteri unicode
- Nome è un testo fino a 75 caratteri unicode
- Data_Nascita è una data
- Paga_Mensile è denaro
- Sesso è un unico carattere unicode

CHIAVE PRIMARIA SU PIÙ CAMPI

Esempio

Questo comando crea una nuova tabella con un suo nome

```
CREATE TABLE Classi (
   Anno CHAR(1)  NOT NULL,
   Sezione CHAR(1)  NOT NULL,
   Indirizzo CHAR(3)  NOT NULL,
   Aula INT NOT NULL ,
 CONSTRAINT
   PRIMARY KEY (Anno, Sezione, Indirizzo)
);
```

PK tra i vincoli generali

crea una nuova tabella denominata Classi con PK sui campi Anno, Sezione, Indirizzo.

CONTROLLI (CHECK)

Esempio

Questo comando crea una nuova tabella con vincolo di controllo sul campo Anno e sul campo Indirizzo

```
CREATE TABLE Classi (
   Anno CHAR(1)  NOT NULL,
   Sezione CHAR(1)  NOT NULL,
   Indirizzo CHAR(3)  NOT NULL,
   Aula INT NOT NULL ,
 CONSTRAINT
   CHECK Anno IN ('1' , '2', '3' , '4' , '5')
   CHECK Indirizzo IN ('CHI' , 'ELE', 'MEC' , 'INF' , 'TEL')
   PRIMARY KEY (Anno, Sezione, Indirizzo)
);
```

CHECK tra i vincoli

crea una nuova tabella denominata Classi con i campi Anno, Sezione, Indirizzo e Aula.

Esempio

Questo comando crea una nuova tabella con dei vincoli interni su alcuni campi

```
CREATE TABLE Prestito (
   Richiedente CHAR(25) NOT NULL,
   Libro    CHAR(25) NOT NULL,
   Data DATE DEFAULT Day(Now()) NOT NULL,
   Scadenza DATE NOT NULL,
   Restituito DATE,
   Cauzione DECIMAL(5,2) DEFAULT 10.75,
CONSTRAINT
   PRIMARY KEY (Richiedente , Libro),
   CHECK Scadenza >= Data,
   CHECK Restituito >= Data,
   CHECK Cauzione >= 0
);
```

CHECK tra i vincoli

si richiede che la Scadenza (se non nulla) sia successiva alla data del prestito
si richiede che la data di restituzione (se non nulla) sia successiva alla data del prestito
si richiede che la Cauzione (se non nulla) sia positiva

CHIAVI ESTERNE SU UN SOLO CAMPO

Esempio

Questo comando crea una nuova tabella con una chiave esterna su un campo

```
CREATE TABLE Studenti (
   Matricola INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
   Cognome CHAR(25) NOT NULL ,
   Nome CHAR(30) NOT NULL ,
   ID_Classe_FK INT FOREIGN KEY ID_Classe_FK
                REFERENCES Classi (ID_Classe)
);
```

FK su singolo campo

si presume che esista una tabella Classi con chiave primaria ID_Classe

Esempio

Questo comando crea una nuova tabella con una chiave esterna su un campo

```
CREATE TABLE Studenti (
   Matricola INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
   Cognome CHAR(25) NOT NULL ,
   Nome CHAR(25) NOT NULL ,
   ID_Classe_FK INT,
   CONSTRAINT
   FOREIGN KEY ID_Classe_FK
            REFERENCES Classi (ID_Classe)
);
```

FK tra i vincoli generali

si presume che esista una tabella Classi con chiave primaria ID_Classe

CHIAVI ESTERNE SU PIÙ CAMPI

Esempio

Questo comando crea una nuova tabella con una chiave esterna su tre campi

```
CREATE TABLE Studenti (
   Matricola INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
   Cognome CHAR(1) NOT NULL ,
   Nome CHAR(1) NOT NULL ,
   Data_Nascita DATE NOT NULL,
   Sesso CHAR(1) NOT NULL,
   AnnoCorso CHAR(1) NOT NULL,
   SezioneCorso CHAR(1) NOT NULL,
   IndirizzoCorso CHAR(3) NOT NULL,
CONSTRAINT
   FOREIGN KEY (AnnoCorso, SezioneCorso, IndirizzoCorso)
                        REFERENCES Classi (Anno, Sezione, Indirizzo),
);
```

FK tra i vincoli generali

si presume che esista una tabella Classi con tre campi Anno, Sezione, Indirizzo

TABELLE ED INTEGRITÀ REFERENZIALE

Chi ha studiato l'integrità referenziale ricorderà che essa obbliga i campi di una tabella (detta secondaria) ai campi di un'altra tabella (detta primaria) e spesso alle sue chiavi primarie. Questo vincolo può essere violato quando si cancellano dei record nella tabella primaria o si modificano i valori dei campi a cui fanno riferimento altre tabelle.

Nella dichiarazione di tabella è possibile allora esplicitare le contromisure da adottare in questi casi. Vediamo le possibili sintassi:

SQL

Sintassi della creazione di tabella con gestione dei tentativi di violazione della integrità referenziale:

```
CREATE TABLE Tabella (
   Campo Tipo vincolo,
CONSTRAINT
   FOREIGN KEY (Campo) REFERENCES Tabella2 (Chiave)
     ON DELETE NO ACTION | CASCADE | SET NULL | SET DEFAULT
     ON UPDATE NO ACTION | CASCADE | SET NULL | SET DEFAULT
);
```

FK con reazione VVIR

per il caso ON DELETE occorre **indicare solo una** delle possibili opzioni
per il caso ON UPDATE occorre **indicare solo una** delle possibili opzioni
Le due opzioni sono indipendenti

dopo aver specificato la chiave esterna è possibile indicare uno o due clausole di reazione:

- **ON DELETE**, che viene attivata nel caso sia cancellata una riga dalla tabella primaria
- **ON UPDATE**, che viene attivata nel caso sia modificato il valore della chiave primaria in una riga della tabella primaria

Per ciascuna di queste due clausole è possibile scegliere uno tra tre possibili eventi:

- **NO ACTION**, significa che il comando è vietato e quindi la cancellazione o la modifica nella tabella primaria non deve avere effetti. È l'evento di default.
- **CASCADE**, significa che le righe della tabella secondaria subiscono la stessa sorte di quelle della tabella primaria (ovvero sono a loro volta cancellate o modificate)
- **SET NULL**, significa che nel campo chiave esterna delle righe correlate si impone il valore nullo. Questa opzione è ammissibile solo se la chiave esterna non sia obbligatoria (NOT NULL), altrimenti equivale a NO ACTION.
- **SET DEFAULT**, significa che nel campo chiave esterna delle righe correlate si impone il valore di base, indicato dalla CREATE TABLE.

Esempio

Esempio con cui si crea una tabella con chiavi esterne e gestione di cancellazione e di aggiornamento

```
CREATE TABLE Verifiche (
   Studente CHAR(7) NOT NULL,
   Materia  CHAR(12) DEFAULT Cinese,
   Data DATE NOT NULL,
CONSTRAINT
   FOREIGN KEY (Studente) REFERENCES Studenti (Matricola)
     ON DELETE NO ACTION
     ON UPDATE CASCADE
   FOREIGN KEY (Materia) REFERENCES Discipline (Codice)
     ON DELETE SET DEFAULT
     ON UPDATE SET NULL
);
```

FK con reazione VVIR

Questo esempio impone che:

- nel caso si tenti di cancellare uno studente ma siano presenti delle verifiche in questa tabella, allora viene impedita la cancellazione dello studente;
- nel caso si tenti di modificare la matricola di uno studente e siano presenti delle verifiche, allora queste ultime modificano la chiave esterna ed assumono la nuova indicata per lo studente;
- nel caso si tenti di cancellare una materia ma siano presenti delle rispettive verifiche in questa tabella, allora si sostituisce la materia con Cinese;
- nel caso si tenti di modificare il codice della materia e siano presenti delle verifiche, allora queste ultime modificano la chiave esterna col valore nullo.

CREATE INDEX

Un indice è un supporto per compiere ricerche più veloci su una tabella ed eventualmente effettuare controlli associati. In generale quando si definisce una tabella sarebbe opportuno predisporre degli indici sui campi: **chiave primaria** (ma è implicito e non si crea a parte), **chiavi esterne** (tutte), **campi con valori unici** (sempre), e **campi di ricerca** (es. cognome, targa, ecc),
La sintassi generale (che tuttavia NON è prevista dallo standard SQL92) è la seguente:

SQL

Sintassi generale di creazione di un indice

```
CREATE UNIQUE INDEX Nome-Indice
ON Nome-Tabella (campo1 ASC/DESC , campo2 ASC/DESC , ...)
```

Con le seguenti regole:

- La parola UNIQUE è opzionale. Se è indicata allora i valori dei campi non possono essere duplicati, altrimenti (se omessa) è possibile duplicarli. Si noti che in caso di molti campi allora la duplicazione si intende per la coppia o terna o n-pla.
- Il nome dell'indice è necessario perché serve per identificare l'indice nel database. Di solito si sceglie un nome distinto da altri oggetti del database.
- L'indice è associato ad una sola tabella. Alla stessa tabella è possibile associare molti indici diversi. I vincoli CONSTRAINT della creazione di tabella creano automaticamente degli indici.
- L'indice può essere creato su un solo nome di campo oppure su molti nomi di campo. In un caso si valuta il singolo valore, nell'altro si crea un indice su coppie, triple, ennuple. Ogni campo dell'indice è ordinato in modo crescente (ASC) o decrescente (DESC).

INDICE NON UNIVOCO (CHIAVI SECONDARIE)

Esempio

Esempio di creazione di un indice sui campi Cognome e Nome della tabella Studenti

```
CREATE INDEX Studenti_NDX
ON Studenti (Cognome ASC , Nome ASC)
```

Con ordinamento crescente su entrambi (prima il Cognome e poi il Nome)

Questo indice permette di effettuare ricerche più veloci sui campi cognome e nome degli alunni, come in effetti accade spesso …

INDICE SU PIÙ CAMPI UNIVOCO (N-PLE VIETATE)

Esempio

Esempio di creazione di un indice sui campi **Alunno, Materia, Data** della tabella **Interrogazioni**

```
CREATE UNIQUE INDEX Verifiche_NDX
ON Interrogazioni (Alunno, Materia, Data)
```

Questo indice permette di evitare che un alunno sia interrogato due volte nella stessa materia nello stesso giorno. Poiché non è specificato l'ordinamento è sottinteso che sia crescente (ASCendente).

INDICE SU UNA CHIAVE ESTERNA

Esempio

Esempio con cui si crea una tabella con chiavi esterne e gestione di cancellazione e di aggiornamento

```
CREATE TABLE Studenti (
   Matricola INT IDENTITY(1,1) NOT NULL PRIMARY KEY ,
   Cognome CHAR(25) NOT NULL ,
   Nome CHAR(30) NOT NULL ,
   ID_Classe_FK INT FOREIGN KEY ID_Classe_FK
                  REFERENCES Classi (ID_Classe)
)

CREATE UNIQUE INDEX ID_Classe_FK_NDX
  ON Studenti (ID_Classe_FK)

CREATE UNIQUE INDEX CognomeStudenti_NDX
  ON Studenti (Cognome)
```

INDICI opportuni

Questo esempio riporta gli indici opportuni per una tabella campione:

- La Matricola ha già un indice ma non è necessario dichiararlo separatamente perché implicito
- La chiave esterna deve avere un indice perché viene usata di frequente sia per assicurare il mantenimento dell'integrità referenziale sia perché usata frequentemente nelle operazioni di JOIN nelle query
- Il campo Cognome è un campo di frequente ricerca di uno studente e quindi è opportuno creare l'indice

OPZIONI SQL

In alcuni dialetti SQL è possibile aggiungere all'indice alcuni vincoli per mezzo della clausola WITH, secondo la seguente sintassi:

SQL

Sintassi della creazione di indice con opzioni:

```
CREATE UNIQUE INDEX Nome-Indice
ON Nome-Tabella (campo1 ASC/DESC , campo2 campo1 ASC/DESC , ...)
WITH
PRIMARY KEY || DISALLOW NULL || IGNORE NULL
```

- La parola chiave WITH può essere omessa con tutto quello che segue. Se specificata serve per impostare un vincolo sull'indice. I vincoli possibili sono tre ma se ne può scegliere solo uno.
- PRIMARY KEY vieta l'esistenza di valori nulli nei campi della tabella e sottintende l'opzione UNIQUE
- DISALLOW NULL vieta i valori nulli, ma può essere usato senza UNIQUE e rende il campo della tabella obbligatorio
- IGNORE NULL non vieta i valori nulli, e permette di costruire un indice generico

Esempio

Esempio di creazione di un indice sui campi **Cognome, Nome, Data_Nasxcita** della tabella **Studenti**

```
CREATE UNIQUE INDEX Indice_Studenti
ON Studenti (Cognome ASC , Nome ASC , Data_Nascita DESC)
WITH DISALLOW NULL
```

Questo indice crea un indice che non ammette valori nulli per alcuno dei tre campi coinvolti (li rende obbligatori) e vieta duplicati della tripla (non ammette due alunni omonimi con identica data di nascita).

ELIMINARE UN INDICE

In SQL è possibile eliminare l'indice dal database quando non serve più. Il comando ha seguente sintassi:

SQL

Sintassi della eliminazione di indice:

```
DROP INDEX Nome-Indice
```

Che elimina l'indice specificato dal suo nome.

CREATE VIEW

Una Vista è una tabella dinamica analoga ad una query ma che costituisce un elemento autonomo con propri diritti, protezioni e vantaggi.
In effetti una vista è calcolata attraverso l'esecuzione di una query ed il risultato costituisce però una vera e propria tabella.
La sintassi generale della vista è la seguente:

SQL

Sintassi della eliminazione di indice:

```
CREATE VIEW NomeVista (col1 , col2, ...) AS
(
SELECT ...
);
```

dove il numero di colonne della vista dovrebbe coincidere col numero di colonne della query.

Esempio

Esempio di creazione di un indice sui campi **Cognome, Nome, Data_Nasxcita** della tabella **Studenti**

```
CREATE VIEW Maggiorenni (Matricola, Cognome, Nome, Età) AS
(     SELECT Matricola, Cognome, Nome, Now() - Data_Nascita
      FROM STUDENTI
      WHERE Età >= 18  );
```

questa vista crea una tabella coi dati degli studenti maggiorenni.

PERCHÉ USARE UNA VISTA?

Il primo motivo per creare una vista è per mettere a disposizione di determinati utenti solo alcuni dati e non una intera tabella di dati. Per esempio un docente potrebbe avere accesso ai dati dei soli studenti delle classi in cui insegna ma non a quelli dell'intera scuola. In effetti alla vista possono essere associati dei diritti che permettono o impediscono l'accesso a determinati utenti o gruppi di utenti.

In tali dati inoltre è possibile effettuare anche operazioni (inserimenti, modifiche e cancellazioni) purché in particolari condizioni e disponendo dei necessari diritti. Per esempio le query di Access sono in effetti delle viste. Altro motivo è che la vista può essere impiegata in una clausola FROM di una ulteriore query, come se fosse una tabella a tutti gli effetti. Questo è utile se il linguaggio non ammette subquery nella clausola FROM.

Esempio

1

```
CREATE VIEW Query_01 (IDS, Quante)
SELECT IDS, COUNT(*)
FROM Verifiche
GROUP BY IDS;
```

questa vista crea una tabella che calcola per ogni studente quante verifiche ha svolto (verrà usata dopo)

Esempio

2

```
CREATE VIEW Query_02 (IDS, Cognome, Nome, NumeroVerifiche)
SELECT IDS, Cognome, Nome, Quante
FROM Studenti INNER JOIN Query_01
  ON Studenti.IDS = Query_01.IDS
WHERE Quante = (  SELECT MAX (Quante)
                  FROM Query_01 ) ;
```

questa vista rende i dati dello studente con più verifiche e fa uso della vista precedente

CREATE TRIGGER

Il Trigger (letteralmente grilletto, innesco) serve per definire un meccanismo automatico sui dati. Quando una determinata operazione viene effettuata sui dati il sistema verifica se esiste un trigger associato e lo esegue.
Questo meccanismo serve per garantire che le operazioni sui dati siano eseguite correttamente, magari correggendo o completando l'operazione richiesta. Purtroppo anche questo comando non è nello standard SQL92, sebbene sia ampiamente implementato in numerosi motori per database.

TRIGGER GENERICO

LINK

Fonte: http://it.wikipedia.org/wiki/Trigger_%28basi_di_dati%29

Il trigger, nelle basi di dati, è una procedura che viene eseguita in maniera automatica in coincidenza di un determinato evento, come ad esempio la cancellazione di un record di una tabella. In questo modo si ha a disposizione una tecnica per specificare e mantenere vincoli di integrità anche complessi.
I trigger permettono agli utenti di specificare vincoli di integrità più complessi dato che un trigger è essenzialmente una procedura PL/SQL (Oracle), Transact-SQL (Microsoft), PL/pgSQL (PostgreSQL), ecc.
La sintassi del comando è la seguente:

SQL

Sintassi della creazione di tabella con gestione dei tentativi di violazione della integrità referenziale:

```
CREATE TRIGGER NomeTrigger ( BEFORE | AFTER )
ON Tabella
FOR DELETE | INSERT | UPDATE
EXECUTE
Comandi //vedi procedure
END;
```

Le due opzioni BEFORE e AFTER sono alternative (se ne usa una e solo una) :

- AFTER, si usa nel caso di voler effettuare delle azioni prima che i controlli sui vincoli della tabella siano stati controllati. È utile per impostare valori di chiave primaria calcolati o per riempire dei campi lasciati vuoti prima che siano effettuati i normali controlli;
- BEFORE, si usa per effettuare azioni dopo che i controlli sono stati superati (e nel caso che l'operazione abbia avuto successo), magari per modificare altre tabelle collegate o per modificare alcuni valori di alcuni campi.

mentre le opzioni DELETE | INSERT | UPDATE sono cumulative (se ne usa una o due o tre) ed indicano il tipo di operazione che invoca l'esecuzione dei comandi.
All'interno del trigger è possibile usare delle parole chiave che rappresentano il record (l'ultimo eventualmente) oggetto dell'azione sia prima (OLD) che dopo (NEW) il tentativo.

Tale procedura è quindi associata ad una tabella e viene automaticamente richiamata dal motore del database quando una certa modifica (o evento) avviene all'interno della tabella. Le modifiche sulla tabella possono includere operazioni INSERT, UPDATE, e DELETE.
La definizione di un trigger consiste nei seguenti componenti:

nome trigger	`CREATE (OR REPLACE) TRIGGER <nome trigger>`
collocazione temporale del trigger	`BEFORE | AFTER | INSTEAD OF`
azione(i) del trigger	`INSERT OR UPDATE (OF <colonna(e)>) OR DELETE ON <tabella>`
tipo di trigger (opzionale)	`FOR EACH ROW`
restrizioni trigger (solo per triggers for each row)	`WHEN (<condizione>)`
corpo del trigger	`<blocco PL/SQL>`

TRIGGER A LIVELLO DI RIGA E TRIGGER A LIVELLO DI ISTRUZIONE

Al fine di programmare i trigger efficientemente (e correttamente), è essenziale capire la differenza tra trigger a livello di riga e trigger a livello di istruzione. Un trigger a livello di riga viene definito utilizzando la clausola for each row. Se questa clausola viene omessa, si assume che il trigger sia un trigger a livello di istruzione.
I trigger di riga hanno alcune speciali caratteristiche che non sono fornite con i trigger di istruzione: solo con un trigger di riga è possibile accedere ai valori degli attributi di una tupla (riga) prima e dopo la modifica, perché il trigger viene eseguito una volta per ogni tupla (riga).

SQL

Per un `UPDATE TRIGGER`, si può accedere al vecchio valore di attributo utilizzando:

```
:OLD.<colonna>
```

e si può accedere al nuovo attributo utilizzando

```
:NEW.<colonna>
```

La sintassi dipende dal dialetto SQL impiegato nel DBMS, ad esempio TSQL di Microsoft.

SQL

Per un `INSERT TRIGGER`, si può accedere solo al nuovo valore di attributo oggetto dell'inserimento utilizzando:

```
:NEW.<colonna>
```

La sintassi dipende dal dialetto SQL impiegato nel DBMS, ad esempio TSQL di Microsoft.

SQL

Per un `DELETE TRIGGER`, si può accedere solo al vecchio valore di attributo oggetto della cancellazione con:

```
:OLD.<colonna>
```

La sintassi dipende dal dialetto SQL impiegato nel DBMS, ad esempio TSQL di Microsoft.

Esempio

Questo comando crea un nuovo trigger con dei vincoli interni sui suoi campi

```
CREATE TRIGGER NumeroVerifiche BEFORE
ON Verifiche
FOR INSERT
EXECUTE
UPDATE Docenti
WHERE Docenti.ID_Docente = :NEW.ID_Docente
SET Docenti.NumeroVerifiche = Docenti.NumeroVerifiche + 1;
END;
```

Questo trigger si innesca ogni volta che viene inserito un record nella tabella Verifiche.
Il trigger incrementa di +1 il numero delle verifiche del docente coinvolto nella Verifica inserita.
La parola chiave NEW indica il nuovo record oggetto dell'inserimento in Verifiche e si suppone che la tupla inserita abbia un campo ID_Docente; il suo valore è usato per filtrare la ricerca del comando UPDATE.

TRIGGER CON CONTROLLO DI AZIONE

Un trigger può prevedere una o più opzioni di riferimento (INSERT/DELETE/UPDATE) e selezionare al suo interno il caso più opportuno con la seguente sintassi:

SQL

Sintassi generica di un trigger su più comandi

```
CREATE OR REPLACE TRIGGER Nome_Trigger
AFTER INSERT OR DELETE OR UPDATE
ON MiaTabella
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        <blocco PL/SQL>
    END IF;
    IF UPDATING THEN
        <blocco PL/SQL>
    END IF;
    IF DELETING THEN
        <blocco PL/SQL>
    END IF;
END;
```

Il trigger si innesca sia nel caso di inserimenti, aggiornamenti o cancellazioni di record sulla tabella MiaTabella.
Quando innescato controlla quale azione lo ha invocato e agisce di conseguenza.

CREATE ASSERTION

L'asserzione è un vincolo articolato che può coinvolgere una, due o più tabelle e serve per garantire che le operazioni sui dati rispettino dei vincoli complessi.
La sintassi del comando è la seguente:

SQL

spazio base

```
CREATE ASSERTION NomeAsserzione
CHECK
(espressione);
```

dove l'espressione deve essere booleana (vera o falsa) ed assume una forma analoga alle query annidate ovvero simili alle seguenti:

- `EXISTS (SELECT . . .  . . . )`
- `Valore IN (SELECT  . . . . . )`
- `(SELECT . . . ) OP (SELECT . . . )`

Esempio

La seguente asserzione:

```
CREATE ASSERTION CentoPerCento
CHECK
(NOT EXISTS (     SELECT SUM(Percentuale)
                  FROM Composizione
                  GROUP BY Prodotto
                  HAVING SUM(Percentuale) > 100 )
);
```

esprime il fatto che non esiste alcun prodotto la cui composizione è formata da oltre il 100 % di ingredienti.

ESERCIZI

ESERCIZI SUI DOMINI

1. Dichiarare un dominio che accetta stringhe per i giorni della settimana
2. Dichiarare un dominio che accetta numeri compresi tra 1 e 31 (Giorno)
3. Dichiarare un dominio che accetta stringhe per i mesi dell'anno
4. Dichiarare un dominio che accetta numeri decimali ma positivi (es. Quantità)
5. Dichiarare un dominio che accetta denaro ma positivo (es. Prezzo)
6. Dichiarare un dominio che accetta numeri interi compresi tra 50 e 100

ESERCIZI SULLE TABELLE

1. Dichiarare le CREATE TABLE per il seguente schema relazionale

```
Studenti (IDS, Nome, Cognome);
  PK IDG;
Verifiche (IDV, ID_Studente_FK, Data, Materia, Voto);
  PK IDV;
  FK ID_Studente_FK REF Studenti (IDS);
  CHECK Voto BETWEEN 1 AND 10;
```

2. Dichiarare le CREATE TABLE per il seguente schema relazionale

```
Giocatori (IDG, Nome, Cognome);
  PK IDG;
Partite (IDP, Data, ID_Bianco, ID_Nero, ID_Vincitore, Terminata);
  PK IDP;
  FK ID_Bianco REF Giocatori (IDG);
  FK ID_Nero REF Giocatori (IDG);
  FK ID_Vincitore REF Giocatori (IDG);
  CHECK (ID_Vincitore= ID_Bianco) OR (ID_Vincitore= ID_Nero);
```

3. Dichiarare le CREATE TABLE per il seguente schema relazionale

```
Piloti (IDP, Nome, Cognome);
  PK IDP;
Auto (IDA, Marca, Modello);
  PK IDA;
Squadra (IDS, Pilota_FK, Navigatore_FK, Auto_FK);
  PK IDS;
  FK Pilota_FK REF Piloti (IDP);
  FK Navigatore_FK REF Piloti (IDP);
  FK Auto_FK REF Auto (IDA);
Gare (IDG, Data, Note);
  PK IDG;
Dettagli (IDD, Gara_FK, Squadra_FK);
  PK IDD;
  FK Gara_FK REF Gare (IDG);
  FK Squadra_FK REF Squadra (IDS);
```

4. Dichiarare le CREATE TABLE per il seguente schema relazionale

```
Uomini (IDU, Nome, Cognome, Moglie_FK);
  PK IDU;
  FK Moglie_FK REF Donne (IDD);
Donne (IDD, Nome, Cognome);
  PK IDD;
Amicizie (IDA, Uomo_FK, Donna_FK);
  PK IDA;
  FK Uomo_FK REF Uomini (IDU);
  FK Donna_FK REF Donne (IDD);
```

ESERCIZI SUGLI INDICI

1. Dichiarare gli INDICI per il seguente schema relazionale

```
Studenti (IDS, Nome, Cognome);
  PK IDG;
Verifiche (IDV, ID_Studente_FK, Data, Materia, Voto);
  PK IDV;
  FK ID_Studente_FK REF Studenti (IDS);
  CHECK Voto BETWEEN 1 AND 10;
```

2. Dichiarare gli INDICI per il seguente schema relazionale

```
Giocatori (IDG, Nome, Cognome);
  PK IDG;
Partite (IDP, Data, ID_Bianco, ID_Nero, ID_Vincitore, Terminata);
  PK IDP;
  FK ID_Bianco REF Giocatori (IDG);
  FK ID_Nero REF Giocatori (IDG);
  FK ID_Vincitore REF Giocatori (IDG);
  CHECK (ID_Vincitore= ID_Bianco) OR (ID_Vincitore= ID_Nero);
```

3. Dichiarare gli INDICI per il seguente schema relazionale

```
Piloti (IDP, Nome, Cognome);
  PK IDP;
Auto (IDA, Marca, Modello);
  PK IDA;
Squadra (IDS, Pilota_FK, Navigatore_FK, Auto_FK);
  PK IDS;
  FK Pilota_FK REF Piloti (IDP);
  FK Navigatore_FK REF Piloti (IDP);
  FK Auto_FK REF Auto (IDA);
Gare (IDG, Data, Note);
  PK IDG;
Dettagli (IDD, Gara_FK, Squadra_FK);
  PK IDD;
  FK Gara_FK REF Gare (IDG);
  FK Squadra_FK REF Squadra (IDS);
```

ESERCIZI SULLE VISTE

1. Risolvere con le viste il seguente problema:
   Mostrare per ogni materia il voto medio attribuito
2. Risolvere con le viste il seguente problema:
   Mostrare gli studenti che hanno una sola interrogazione in Storia
3. Risolvere con le viste il seguente problema:
   Mostrare gli studenti che hanno un voto in Storia superiore alla media
4. Risolvere con le viste il seguente problema:
   Mostrare i giocatori che hanno vinto il maggior numero di partite a scacchi
5. Risolvere con le viste il seguente problema:
   Mostrare i dati dei giocatori dell'ultima partita
6. Risolvere con le viste il seguente problema:
   Mostrare i dati dei piloti che non hanno fatto gare
7. Risolvere con le viste il seguente problema:
   Mostrare i dati delle auto che non hanno fatto gare
8. Risolvere con le viste il seguente problema:
   Eliminare le squadre in cui il pilota coincide col navigatore

SOMMARIO