LEZIONE-TSQL-01 DDL (VER01)

T-SQL PROGRAMMAZIONE PARTE PRIMA

MICROSOFT SQL SERVER

IL PRODOTTO

MICROSOFT SQL SERVER

Le dispense SQL descrivono un generico SQL utilizzato nella maggior parte dei database, noto come standard SQL92; tuttavia esistono delle varianti a questo linguaggio, generalmente indicate come «dialetti».

Microsoft Sql Server è un ambiente di gestione di basi di dati (RDBMS) che ha proposto diverse versioni ed utilizza uno specifico dialetto non come T-SQL, ovvero Transact-SQL. In questa dispensa analizzeremo la sintassi del linguaggio T-SQL per creare, gestire e analizzare i database.

Per saperne di più sul RDBMS e sul T-SQL è possibile consultare i seguenti link:

technet.microsoft.com	http://technet.microsoft.com/it-it/library/bb500469.aspx
Guida di riferimento a Transact-SQL	http://technet.microsoft.com/it-it/library/bb510741(v=sql.110).aspx
Versioni e distribuzioni di SQL Server	http://it.wikipedia.org/wiki/Microsoft_SQL_Server
Installazione per SQL Server 2012	http://technet.microsoft.com/it-it/library/bb500469.aspx

Le versioni di SQL Server sono state molte; traggo le principali da Wikipedia:

anno	*Nome*	*nome in codice*	*versione gratuita*
1993	SQL Server 4.2	-	-
1995	SQL Server 6.0	SQL95	No
1996	SQL Server 6.5	Hydra	No
1998	SQL Server 7.0	Sphinx	No
1999	SQL Server 7.0 OLAP	Plato	No
2000	SQL Server 2000 32-bit	Shiloh	No
2003	SQL Server 2000 64-bit	Liberty	No
2005	SQL Server 2005 (sia 32-bit che 64-bit)	Yukon	Si
2008	SQL Server 2008 (sia 32-bit che 64-bit)	Katmai	Express
2010	SQL Server 2008 R2 (sia 32-bit che 64-bit)	-	Express
2012	SQL Server 2012 (sia 32-bit che 64-bit) in fase beta	Denali	Express

Il linguaggio TSQL che è affrontato in questa dispensa:

- Si riferisce alla versione SQL Server 2008 (sia 32-bit che 64-bit);
- È trattato in modo parziale e sintetico, tralasciando alcuni elementi troppo tecnici.

IL DIALETTO TRANSACT SQL

DATA DESCRIPTION LANGUAGE

Il DDL di TSQL permette i consueti comandi di gestione delle strutture del database (creazione, modifica, eliminazione). La creazione di oggetti si esegue mediante un comando che ha la forma:

```
CREATE OGGETTO <DETTAGLI>
```

La modifica di oggetti esistenti si esegue mediante un comando che ha la forma:

```
ALTER OGGETTO <DETTAGLI>
```

La distruzione di oggetti si esegue mediante un comando che ha la forma:

```
DROP OGGETTO <DETTAGLI>
```

Vedremo alcuni dettagli della creazione (modifica) di oggetti, tra cui i seguenti:

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	PROCEDURE	Serve per definire una procedura strutturata e articolata
CREATE OR ALTER	GROUP	Serve per definire un gruppo di utenti
CREATE OR ALTER	USER	Serve per definire un utente
CREATE OR ALTER	GRANT	Serve per definire i diritti di accesso ai dati

DATA MANIPULATION LANGUAGE

Il DML di TSQL permette i consueti comandi di gestione dei dati contenuti nelle strutture del database, tra cui analizzeremo i seguenti:

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

TSQL PER PROCEDURE E TRIGGER

Oltre ai precedenti comandi, il DML di TSQL permette di scrivere il corpo delle procedure create con il consueto comando e prevede una sintassi specifica per elaborare i dati; analizzeremo la sintassi relativa a:

VARIABILI	Dichiarazione di variabili e assegnazione di valori, con operazioni
IF	Serve per decidere l'esecuzione di blocchi in funzione di un controllo
WHILE	Serve per iterare l'esecuzione di blocchi in funzione di un controllo
PARAMETRI	Servono per far comunicare la procedura con il chiamante

TSQL PER GESTIRE STRINGHE E DATE

Infine, oltre alle consuete funzioni previste dal SQL92, il DML di TSQL permette di usare funzioni per gestire dati che spesso sfuggono allo standard; analizzeremo la sintassi relativa a:

STRINGHE	Funzioni per manipolare stringhe, compiere ricerche e produrre nuove stringhe
DATE	Funzioni per ottenere date e orari dal sistema operativo, per gestire i formati di date/orari

CREAZIONE DEL DATABASE

All'inizio c'era il nulla … e il database non esisteva ancora. Sebbene la sintassi della creazione di database sia abbastanza articolata, ci limiteremo a vedere la sola sintassi semplificata seguente:

```
CREATE DATABASE NOME_DATABASE
```

Questo comando crea un nuovo database con un suo nome.

Quando viene creato, il database è vuoto ma, in realtà, 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 …; ai nostri fini è trascurabile sia l'analisi di queste tabelle, sia tutte le opzioni della sintassi TSQL: possiamo perciò limitarci alla sintassi più elementare (corretta) seguente:

ESEMPIO 1.

```
CREATE DATABASE SCUOLA
```

COMANDI CREATE

Per avere l'elenco completo dei comandi CREATE di T-SQL è possibile consultare qui: http://technet.microsoft.com/it-it/library/cc879262.aspx

COMANDI DROP

Per ogni comando Create di T-SQL esiste anche il suo antagonista DROP che serve per eliminare una struttura dal database. Per esempio:

```
DROP TABLE BIBLIOTECA
```

TIPI DI DATO PREESISTENTI

Come tutti i linguaggi, anche TSQL dispone di tipi di dato pronti. Questi tipi servono per sapere cosa poter archiviare dentro le tabelle (numeri, testo, ecc…). I tipi sono:

Tipo di dato	Categoria	Ampiezza	Descrizione	Esempio
bigint	Numerici esatti	8 byte	± 9 mld di mld circa	*accessi al sito web* ♥
numeric	Numerici esatti			
bit	Numerici esatti		usare per vero/falso, on/off	*si/no* ♥
smallint	Numerici esatti	2 byte	± 32.000 circa	*pagine di un libro* ♥
decimal	Numerici esatti			*temperatura febbre* ♥
smallmoney	Numerici esatti			*prezzo di un oggetto* ♥
int	Numerici esatti	4 byte	± 2 mld circa	♥
tinyint	Numerici esatti	1 byte	da 0 a 255	*età umana* ♥
money	Numerici esatti			*bilancio di una società* ♥
float	Numerici approssimati	K byte		*programma scientifico*
real	Numerici approssimati	4 byte		*programma scientifico*
date	Tempo	da 01-01-0001 a 31-12-9999		*giorno storico* ♥
datetimeoffset	Tempo	Da 1 gennaio 1 D.C. al 31 dicembre 9999 D.C., 23:59:59.9999999		
datetime2	Tempo	Data come **Date**, Orari come **Time**		
smalldatetime	Tempo	da 1 gennaio 1900 a 6 giugno 2079,		*scadenza prestito* ♥
datetime	Tempo	da 1/1/1753 a 31/12/ 9999, orari da 00:00:00.000 a 23:59:59.997		*momento ingresso docente* ♥
time	Tempo	da 00.00.00.0000000 a 23.59.59.9999999 (decimilionesimo sec)		*orario ingresso* ♥
char (n)	Stringhe di caratteri	n byte	**n ≤ 1800**	*codice fiscale* ♥
varchar (n)	Stringhe di caratteri	**max** n byte	**n ≤ 1800**	*nome di persona* ♥
text	Stringhe di caratteri	2.147.483.647 byte	Non unicode	*descrizione di un film* ♥
nchar (n)	Stringhe Unicode	n byte	**n ≤ 1800**	*codice fiscale*
nvarchar (n)	Stringhe Unicode	**max** n byte	**n ≤ 1800**	*nome di persona*
ntext	Stringhe Unicode	2.147.483.647 byte	Non unicode	*descrizione di un film*
binary	Stringhe binarie			*Video*
varbinary	Stringhe binarie			*video* ♥
image	Stringhe binarie			*foto* ♥
cursor	Altri tipi di dati			*speciale*
timestamp	Altri tipi di dati			*momento molto preciso*
hierarchyid	Altri tipi di dati			
uniqueidentifier	Altri tipi di dati			
sql_variant	Altri tipi di dati			
xml	Altri tipi di dati			*file xml*
table	Altri tipi di dati			

Con il simbolo ♥ si mettono in evidenza i tipi usati più di frequente.

Fonte: http://msdn.microsoft.com/it-it/library/ms187752.aspx

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 TYPE

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. La sintassi è:

```
CREATE TYPE NOMEDB.NOMETIPO
FROM TIPOBASE NOT NULL
AS TABLE (CONSTRAINT <ELENCO VINCOLI COME TABELLA> )
```

ESEMPIO 2.

```
CREATE TYPE SCUOLA.PIANO
FROM CHAR (1) NOT NULL
AS TABLE (CONSTRAINT CHECK (VALUE = "T" OR VALUE BETWEEN "1" AND "5" ) )
```

il tipo Piano è una stringa di un carattere obbligatorio che coincide con la lettera T oppure con il carattere che rappresenta una cifra compresa tra 1 e 5.

```
CREATE TYPE SCUOLA.INDIRIZZO
FROM CHAR (11) NOT NULL
AS TABLE (CONSTRAINT CHECK (VALUE IN ("BIENNIO", "CHIMICA", "ELETTRONICA",
                                      "INFORMATICA", "MECCANICA") ) )
```

il tipo Indirizzo è una stringa di 11 caratteri obbligatori che sia incluso tra quelli specificati.

```
CREATE TYPE SCUOLA.SESSO
FROM CHAR(1)
AS TABLE (CONSTRAINT CHECK (VALUE = "F" OR VALUE = "M") )
```

il tipo Sesso è una stringa di 1 carattere facoltativo, che sia F oppure M (oppure NULL).

```
CREATE TYPE SCUOLA.TIPOPREZZO
FROM SMALLMONEY
AS TABLE (CONSTRAINT CHECK (VALUE > 0) )
```

il tipo TipoPrezzo è una moneta i cui elementi devono soddisfare la condizione di essere maggiori di zero (zero escluso, in questo caso).

DEFINIZIONE DI TABELLE

CREATE TABLE

La sintassi semplificata del comando che useremo sarà la seguente:

```
CREATE TABLE    NOMEDB.NOMETABELLA (
      NOMECAMPO1 TIPOCAMPO1 VINCOLI1,
      . . .
      NOMECAMPOK TIPOCAMPOK VINCOLI2,
      CONSTRAINT VINCOLITABELLA
)
```

La creazione delle tabelle è fondamentale per un database. Solitamente si devono costruire specificando anche chiavi primarie, esterne, vincoli interni ed esterni; ma è anche possibile modificare la struttura della tabella in seguito, variando campi e vincoli.

L'opzione **CONSTRAINT** serve per specificare i vincoli sul campo oppure sull'intera tabella. Se la parola segue un campo significa che i vincoli sono riferiti al campo; se la parola è posta dopo la dichiarazione dei campi significa che i vincoli sono riferiti alla tabella. In realtà per imporre dei

vincoli su un campo la si può omettere. Invece è obbligatoria se si aggiungono vincoli dopo l'elenco dei campi.

Il vincolo **PRIMARY KEY** è ammesso una sola volta in una tabella, ma può essere riferito a più attributi insieme (si sconsiglia).

Il vincolo **FOREIGN KEY** è riferito ad un campo, ma può essere riferito a più attributi insieme (si sconsiglia) ed è sempre seguito dal riferimento alla tabella ed al campo correlato (solitamente una chiave primaria).

Il vincolo **CHECK** serve per compiere un controllo come la verifica se il valore è uguale ad un certo valore, oppure è compreso in un intervallo o in un elenco.

ESEMPIO 3. (CASO ELEMENTARE)

Il caso più semplice di tabella impone di indicare il suo nome e almeno un campo; non è obbligatorio indicare una chiave primaria (ovvero è sintatticamente corretto) ma è fortemente sconsigliato avere tabelle prive. L'esempio seguente definisce una tabelle per le aule della scuola, con una K, il corpo dove si trova (es. nel corpo H), il piano (es. piano 2), il numero dell'aula (es. 7) e i metri quadri. Il vincolo NOT NULL impone l'obbligatorietà di un valore.

```
CREATE TABLE    SCUOLA.AULE (
      IDAULA CHAR (4) NOT NULL CONSTRAINT PRIMARY KEY ,
      CORPO CHAR (1) NOT NULL ,
      PIANO CHAR (1) NOT NULL ,
      NUMERO CHAR (2) NOT NULL ,
      MQ DECIMAL (4,2)
)
```

ESEMPIO 4. (IDENTITY)

La chiave primaria può essere di qualsiasi tipo ma, spesso, si preferisce usare il tipo intero col vincolo **IDENTITY** che indica un contatore (auto incremento) che può avere anche due argomenti: il primo è il valore di partenza; il secondo è il passo di incremento. L'esempio seguente impone una chiave primaria numerica contatore, che parte da 1 ed avanza di +1 ad ogni nuovo inserimento.

```
CREATE TABLE    SCUOLA.AULE (
      IDAULA INT IDENTITY (1, 1) NOT NULL CONSTRAINT PRIMARY KEY ,
      CORPO CHAR (1) NOT NULL ,
      PIANO CHAR (1) NOT NULL ,
      NUMERO CHAR (2) NOT NULL ,
      MQ DECIMAL (4,2)
)
```

una versione equivalente è la seguente:

```
CREATE TABLE    SCUOLA.AULE (
      IDAULA INT IDENTITY (1, 1) NOT NULL ,
      CORPO CHAR (1) NOT NULL ,
      PIANO CHAR (1) NOT NULL ,
      NUMERO CHAR (2) NOT NULL ,
      MQ DECIMAL (4,2) ,
CONSTRAINT
      IDAULA PRIMARY KEY ,
)
```

ESEMPIO 5. (UNIQUE)

Se occorre imporre che altri campi siano unici (e magari anche obbligatori diventando così delle chiavi candidate) è possibile usare il vincolo UNIQUE. Il vincolo si può usare su un solo campo o su un gruppo di campi. Per esempio:

```
CREATE TABLE    SCUOLA.AULE (
      IDAULA INT IDENTITY (1, 1) NOT NULL CONSTRAINT PRIMARY KEY ,
```

```
      CORPO CHAR (1) NOT NULL UNIQUE ,
      PIANO CHAR (1) NOT NULL UNIQUE ,
      NUMERO CHAR (2) NOT NULL UNIQUE ,
      MQ DECIMAL (4,2)
)
```

ma non funzionerebbe bene, poiché questo vincolo impedisce di avere più aule nello stesso corpo, impedisce di avere più aule con lo stesso piano, e più aule con stesso numero; tuttavia è possibile più sensatamente scrivere così:

```
CREATE TABLE    SCUOLA.AULE (
      IDAULA INT IDENTITY (1, 1) NOT NULL CONSTRAINT PRIMARY KEY ,
      CORPO CHAR (1) NOT NULL ,
      PIANO CHAR (1) NOT NULL ,
      NUMERO CHAR (2) NOT NULL ,
      MQ DECIMAL (4,2) ,
CONSTRAINT
      UNIQUE (CORPO, PIANO, NUMERO)
)
```

che invece impedisce di duplicare la tripla (Corpo, Piano, Numero) e quindi impedisce che esistano due aule nello stesso corpo, allo stesso piano e con lo stesso numero; permette comunque di avere due aule allo stesso corpo e piano ma con diverso numero, o stesso piano e numero, ma diverso corpo e così via.

ESEMPIO 6. (CHECK)

Se occorre imporre che altri campi siano unici (e magari anche obbligatori diventando così delle chiavi candidate) è possibile usare il vincolo UNIQUE. Il vincolo si può usare su un solo campo o su un gruppo di campi. Per esempio:

```
CREATE TABLE    SCUOLA.AULE (
      IDAULA INT IDENTITY (1, 1) NOT NULL CONSTRAINT PRIMARY KEY ,
      PIANO CHAR (1) NOT NULL
            CONSTRAINT CHECK (VALUE IN ("T","1","2","3","4","5"),
      MQ DECIMAL (4,2) NOT NULL CONSTRAINT CHECK (VALUE > 0) DEFAULT 16.0
)
CREATE TABLE    SCUOLA.CLASSI (
      IDCLASSE INT IDENTITY (1, 1) NOT NULL PRIMARY KEY ,
      IDAULA INT NOT NULL
            CONSTRAINT FOREIGN KEY REFERENCES SCUOLA.AULE(IDAULA)
      ANNO CHAR (1) NOT NULL
            CONSTRAINT CHECK (VALUE IN ("1","2","3","4","5"),
      SEZIONE CHAR (1) NOT NULL
            CONSTRAINT CHECK (VALUE BETWEEN "A" AND "Z") ,
      INDIRIZZO CHAR (1) NOT NULL
            CONSTRAINT CHECK (VALUE IN ("BIE","CHI","ELE","INF","MEC"),
CONSTRAINT
      UNIQUE (ANNO, SEZIONE, INDIRIZZO)
)
CREATE TABLE     SCUOLA.STUDENTI (
      IDSTUDENTE INT IDENTITY (1, 1) NOT NULL PRIMARY KEY ,
      COGNOME VARCHAR (250) NOT NULL ,
      NOME VARCHAR (250) NOT NULL ,
      SESSO VARCHAR (5) NOT NULL ,
      CAP VARCHAR (5) NOT NULL ,
      CITTÀ VARCHAR (250) NOT NULL ,
      INDIRIZZO VARCHAR (250) NOT NULL ,
      TELEFONO VARCHAR (15) ,
      EMAIL VARCHAR (150),
      CLASSE INT NOT NULL
            CONSTRAINT FOREIGN KEY REFERENCES SCUOLA.CLASSI(IDCLASSE)
)
```

```
CREATE TABLE SCUOLA.PRESTITI (
      IDPRESTITI INT IDENTITY (1, 1) NOT NULL PRIMARY KEY ,
      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
      CHECK SCADENZA > DATA,
      CHECK RESTITUITO > DATA,
      CHECK CAUZIONE >= 0
)
```

CHIAVI PRIMARIE SU PIÙ CAMPI

È possibile anche specificare il vicolo CONSTRAINT alla fine, dopo la dichiarazione dei campi; in tal caso è possibile esprimere vincoli su più campi insieme e, per esempio, imporre una chiave primaria su più campi oppure una chiave esterna su più campi.

```
CREATE TABLE    SCUOLA.CLASSI (
      ANNO CHAR (1) NOT NULL ,
      SEZIONE CHAR (1) NOT NULL
            CONSTRAINT CHECK (VALUE BETWEEN "A" AND "Z") ,
      INDIRIZZO CHAR (1) NOT NULL
            CONSTRAINT CHECK (VALUE IN ("BIE","CHI","ELE","INF","MEC") ,
CONSTRAINT
      CHECK ANNO IN ('1' , '2', '3' , '4' , '5') ,
      PRIMARY KEY (ANNO, SEZIONE, INDIRIZZO) ,
      FOREIGN KEY AULA REFERENCES AULE (CODICE)
)
```

CHIAVI ESTERNE SU PIÙ CAMPI

```
CREATE TABLE    SCUOLA.AULE (
      IDAULA INT IDENTITY (1, 1) ,
      ANNOCORSO CHAR(1) NOT NULL,
      SEZIONECORSO CHAR(1) NOT NULL,
      INDIRIZZOCORSO CHAR(3) NOT NULL,
CONSTRAINT
      PRIMARY KEY (IDAULA) ,
      FOREIGN KEY (ANNOCORSO, SEZIONECORSO, INDIRIZZOCORSO)
                              REFERENCES CLASSI (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.

Proposta 1:

```
CREATE TABLE TABELLA (
      CAMPO TIPO <VINCOLI> FOREIGN KEY (CAMPO) REFERENCES TABELLA2 (CHIAVE)
                  ON DELETE NO ACTION | CASCADE | SET NULL | SET DEFAULT
                  ON UPDATE NO ACTION | CASCADE | SET NULL | SET DEFAULT
      , <ALTRI CAMPI>
)
```

Proposta 2:

```
CREATE TABLE TABELLA (
      CAMPO TIPO <VINCOLI>,
      , <ALTRI CAMPI>
      ,
      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
)
```

Nella proposta 1 il vincolo è specificato immediatamente dopo gli altri vincoli del campo se ce ne sono (es. NOT NULL); nella proposta 2 il vincolo è specificato dopo aver dichiarato tutti i campi e formulato nella sezione CONSTRAINT.

Dopo aver specificato la chiave esterna è possibile (ma non è obbligatorio) 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 7. (FOREIGN KEY, ON DELETE, ON UPDATE)

```
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
)
```

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. La sintassi semplificata in T-SQL è la seguente:

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] 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).

ESEMPIO 8. (INDICE SU UN CAMPO)

```
CREATE INDEX PRESTITI_NDX
ON PRESTITI (DATA)
```

Questo indice permette di effettuare ricerche più veloci sul campo data del prestito.

ESEMPIO 9. (INDICE SU PIÙ CAMPI)

```
CREATE INDEX STUDENTI_NDX
ON STUDENTI (COGNOME ASC , NOME ASC)
```

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

ESEMPIO 10. (INDICE UNIVOCO)

```
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.

ELIMINARE UN INDICE

In T-SQL è possibile eliminare l'indice dal database quando non serve più. Il comando ha seguente sintassi:

```
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.

In –SQL crea una tabella virtuale il cui contenuto (colonne e righe) è definito da una query. È possibile utilizzare questa istruzione per creare una tabella virtuale dei dati contenuti in una o più tabelle nel database, filtrate e aggregate come si preferisce.

Ad esempio, è possibile utilizzare una vista per gli scopi seguenti:

- Per analizzare, semplificare e personalizzare la visualizzazione del database per ogni utente.
- Come meccanismo di sicurezza grazie al quale è possibile consentire agli utenti di accedere ai dati tramite una vista, senza concedere loro le autorizzazioni di accesso alle tabelle di base sottostanti.
- Per fornire un'interfaccia compatibile con le versioni precedenti tramite la quale è possibile emulare una tabella precedente il cui schema è stato modificato.

La sintassi semplificata della vista è la seguente:

```
CREATE VIEW DB_NOME.NOME_VISTA (COL1 , COL2, ..., COLX) AS
SELECT ...
. . .
```

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

Se nella SELECT si usa un clausola ORDER BY, questa è utilizzata esclusivamente per determinare le righe restituite dalla clausola TOP oppure OFFSET nella definizione della vista. La clausola ORDER BY non garantisce risultati ordinati in caso di query sulla vista, a meno che tale clausola non venga specificata anche nella query.

ESEMPIO 11. (CREARE UNA VISTA)

```
CREATE VIEW MAGGIORENNI (MATRICOLA, COGNOME, NOME, ETÀ) AS
SELECT MATRICOLA, COGNOME, NOME, ETÀ
FROM STUDENTI
WHERE ETÀ >= 18
```

questa vista crea una tabella coi dati degli studenti maggiorenni.

ESEMPIO 12. (CREARE UNA VISTA)

```
CREATE VIEW QUANTIPERFASCIA (SESSO, ETÀ, QUANTI) AS
SELECT SESSO, ETÀ, COUNT(*)
FROM STUDENTI
GROUP BY SESSO, ETÀ
```

questa vista crea una tabella che conta quanti studenti ci sono per sesso e per ogni fascia d'età.

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 13. (GIUNZIONE TRA UNA TABELLA E UNA VISTA)

```
CREATE VIEW VOTI_MAGGIORENNI (MATRICOLA, COGNOME, NOME, ETÀ, VOTO) AS
SELECT MATRICOLA, COGNOME, NOME, ETÀ, VOTO
FROM MAGGIORENNI , VERIFICHE
WHERE MAGGIORENNI.MATRICOLA = VERIFICHE.MATRICOLA
```

questa vista crea una tabella coi voti degli studenti maggiorenni. Nella clausola FROM si usa il nome MAGGIORENNI per fare riferimento alla vista creata prima e non è una tabella.

VISTE AGGIORNABILI

È possibile modificare, mediante la modifica dei dati di una vista, i dati di una tabella concreta sottostante ma solo se vengono soddisfatti i requisiti seguenti:

- Tutte le modifiche, incluse le istruzioni UPDATE, INSERT e DELETE, devono fare riferimento a colonne di una sola tabella di base.
- Le colonne modificate nella vista devono fare riferimento direttamente ai dati sottostanti nelle colonne della tabella. Le colonne non possono essere derivate in altro modo, ad esempio tramite:
  - Una funzione di aggregazione: AVG, COUNT, SUM, MIN, MAX, GROUPING, STDEV, STDEVP, VAR e VARP.
  - Un calcolo. La colonna non può essere calcolata tramite un'espressione che utilizza altre colonne. Le colonne create mediante gli operatori sui set UNION, UNION ALL, CROSSJOIN, EXCEPT e INTERSECT sono considerate un calcolo e non sono aggiornabili.
- Le colonne da modificare non sono interessate da clausole GROUP BY, HAVING o DISTINCT.
- La clausola TOP non può essere utilizzata in tutte le posizioni dell'argomento select_statement della vista in combinazione con la clausola WITH CHECK OPTION.

Le restrizioni sopra indicate sono valide sia per qualsiasi sottoquery nella clausola FROM della vista sia per la vista stessa. In genere, il motore di database deve essere in grado di tracciare senza ambiguità le modifiche dalla definizione della vista a una tabella di base.

Per ulteriori informazioni, vedere http://msdn.microsoft.com/it-it/library/ms187956.aspx .

Argomenti Trattati: