

CREATE

Il comando CREATE in SQL viene utilizzato per gestire la creazione delle strutture del database.

COMANDO
CREATE DATABASE
CREATE TYPE
CREATE TABLE
CREATE INDEX
CREATE VIEW
CREATE TRIGGER
CREATE PROCEDURE
CREATE USER

CREATE DATABASE

Sintassi completa di
creazione del database



```
CREATE DATABASE database_name
[ CONTAINMENT = { NONE | PARTIAL } ]
[ ON
    [ PRIMARY ] <filespec> [ ,...n ]
    [ , <filegroup> [ ,...n ] ]
    [ LOG ON <filespec> [ ,...n ] ]
]
[ COLLATE collation_name ]
[ WITH <option> [ ,...n ] ]
[;]

<option> ::=
{
    FILESTREAM ( <filestream_option> [ ,...n ] )
    | DEFAULT_FULLTEXT_LANGUAGE = { lcid | language_name | language_alias }
    | DEFAULT_LANGUAGE = { lcid | language_name | language_alias }
    | NESTED_TRIGGERS = { OFF | ON }
    | TRANSFORM_NOISE_WORDS = { OFF | ON }
    | TWO_DIGIT_YEAR_CUTOFF = <two_digit_year_cutoff>
    | DB_CHAINING { OFF | ON }
    | TRUSTWORTHY { OFF | ON }
    | PERSISTENT_LOG_BUFFER=ON ( DIRECTORY_NAME='<Filepath to folder on DAX formatted volume>' )
}

<filestream_option> ::=
{
    NON_TRANSACTED_ACCESS = { OFF | READ_ONLY | FULL }
    | DIRECTORY_NAME = 'directory_name'
}

<filespec> ::=
{
    (
        NAME = logical_file_name ,
        FILENAME = { 'os_file_name' | 'filestream_path' }
        [ , SIZE = size [ KB | MB | GB | TB ] ]
        [ , MAXSIZE = { max_size [ KB | MB | GB | TB ] | UNLIMITED } ]
        [ , FILEGROWTH = growth_increment [ KB | MB | GB | TB | % ] ]
    )
}

<filegroup> ::=
{
    FILEGROUP filegroup_name [ [ CONTAINS FILESTREAM ] [ DEFAULT ] | CONTAINS MEMORY_OPTIMIZED_DATA ]
    <filespec> [ ,...n ]
}
```

CREATE

Questo comando si riduce



`CREATE DATABASE nome_database`

CREATE TABLE

CREATE TABLE

```
[ database_name . [ schema_name ] . | schema_name . ] table_name
[ AS FileTable ]
( { <column_definition> | <computed_column_definition>
  | <column_set_definition> | [ <table_constraint> ] [ ,...n ] } )
[ ON { partition_scheme_name ( partition_column_name ) | filegroup
  | "default" } ]
[ { TEXTIMAGE_ON { filegroup | "default" } } ]
[ FILESTREAM_ON { partition_scheme_name | filegroup
  | "default" } ]
[ WITH ( <table_option> [ ,...n ] ) ]
[ ; ]
```

```
CREATE TABLE table_name
(
  column_name1 data_type(size),
  column_name2 data_type(size),
  column_name3 data_type(size),
  ....
);
```

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.

CREATE TABLE

```
CREATE TABLE dbo.Esempio
(
    Id            INT PRIMARY KEY IDENTITY(1, 1),
    Campo1        NVARCHAR(30) NULL,
    Campo2        NVARCHAR(30) NOT NULL,
    Campo3        NVARCHAR(30) NOT NULL DEFAULT 'my_default',
    Campo4        INT NULL,
    Campo5        INT NOT NULL,
    Campo6        INT NOT NULL DEFAULT 0
)
```

Come si vede, dopo il nome del campo bisogna indicare il **tipo** e opzionalmente altre **proprietà/vincoli**.

Se non ne sono indicate il campo è NULL (cioè non obbligatorio).

Le **proprietà/vincoli** principali sono:

- **NULL**: il campo può assumere il valore NULL.
- **NOT NULL**: il campo non può assumere il valore NULL.
- **NOT NULL DEFAULT X**: il campo non può assumere il valore NULL; se non impostato nella query di inserimento allora assume il valore indicato dopo la parola 'DEFAULT'.
- **PRIMARY KEY**: il campo è la chiave primaria della tabella.
- **IDENTITY(N, M)**: il campo è generato dal database a partire dal valore N e incrementato ogni volta del valore M.

ALTER TABLE

Il comando ALTER in SQL viene utilizzato per modificare la struttura di una tabella esistente nel database.

Viene utilizzato quindi per aggiungere, modificare o eliminare colonne o vincoli in una tabella esistente.

- **Aggiungere una nuova colonna in una tabella esistente**

Sintassi `ALTER TABLE Table_Name ADD New_Column_Name Data_Type (Size);`

- **Modificare il tipo di dati e le dimensioni di una colonna esistente**

Sintassi `ALTER TABLE Table_Name ALTER COLUMN Column_Name New_Data_Type (New_Size)`

- **Eliminare la colonna esistente da una tabella:**

Sintassi `ALTER TABLE Table_Name DROP COLUMN Column_Name`

Esempio

Il seguente comando Crea creerà una nuova tabella 'Impiegato' nel database.



```
CREATE TABLE Employee  
(  
    Id INT PRIMARY KEY,  
    Name VARCHAR(50),  
    Salary DECIMAL(18, 2)  
);
```

La seguente istruzione Alter **aggiungerà** una nuova colonna nella tabella "Impiegato".



```
ALTER TABLE Employee ADD City VARCHAR(20)
```

La seguente istruzione Alter viene utilizzata per **modificare** il tipo di dati e la dimensione di una colonna esistente "Città".



```
ALTER TABLE Employee ALTER COLUMN City NVARCHAR (100);
```

La seguente istruzione Alter viene utilizzata per **eliminare** la colonna esistente "Città" dalla tabella.



```
ALTER TABLE Employee DROP COLUMN City;
```

Vincoli SQL

I vincoli SQL vengono utilizzati per specificare le **regole** per i dati in una tabella quindi per limitare il tipo di dati che possono essere inseriti in quella tabella.

Ciò garantisce **l'accuratezza e l'affidabilità dei dati** nella tabella.

In caso di violazione tra il vincolo e l'azione sui dati, l'azione viene interrotta.

I vincoli possono essere:

- a livello di colonna → si applicano a una colonna e i vincoli
- a livello di tabella → si applicano all'intera tabella.

I vincoli più usati in SQL:

- [NOT NULL](#) - Assicura che una colonna non possa avere un valore NULL
- [UNIQUE](#) - Assicura che tutti i valori in una colonna siano diversi
- [PRIMARY KEY](#) - Combinazione di a NOT NULL e UNIQUE. Identifica in modo univoco ogni riga in una tabella
- [FOREIGN KEY](#) - Impedisce azioni che distruggerebbero i collegamenti tra le tabelle
- [CHECK](#) - Assicura che i valori in una colonna soddisfino una condizione specifica
- [DEFAULT](#) - Imposta un valore predefinito per una colonna se non viene specificato alcun valore
- [CREATE INDEX](#) - Utilizzato per creare e recuperare dati dal database molto rapidamente

Primary Key

Sintassi

```
-- Column level Primary key
CREATE TABLE TableName
(
  Column1 data_type [NOT NULL ] [ PRIMARY KEY ],
  Column2 data_type [ NULL | NOT NULL ],
  Column3 ...
);

-----

-- Table level Primary key
CREATE TABLE TableName
(
  Column1 data_type [ NULL | NOT NULL ],
  Column2 datatype [ NULL | NOT NULL ],
  Column3 ...
  CONSTRAINT ConstraintName PRIMARY KEY (Column1, Column2)
);
```

Esempio pratico:

```
-- Column level Primary key
CREATE TABLE Employee
(
  ID INT CONSTRAINT PK_ID PRIMARY KEY,
  NAME VARCHAR (50),
  EMAIL VARCHAR(60)
)

-- OR

-- Table level Primary key
CREATE TABLE Employee
(
  ID INT NOT NULL,
  NAME VARCHAR (50),
  EMAIL VARCHAR(60)
  CONSTRAINT PK_ID PRIMARY KEY(ID)
)
```

Primary Key

Da interfaccia:

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'SampleDB' database is expanded, showing the 'Tables' folder. The 'dbo.Employee' table is selected, and its 'Columns' folder is expanded. The 'ID' column is highlighted with a red rectangle, and a blue callout bubble points to it with the text 'Primary key Created on ID Column'. Below the columns, the 'Keys' folder shows a new primary key named 'PK_ID'. On the right, the SQL command window shows the following code:

```
CREATE TABLE Employee  
(  
    ID INT CONSTRAINT PK_ID PRIMARY KEY,  
    NAME VARCHAR (50),  
    EMAIL VARCHAR(60)  
)
```

Below the code window, the 'Messages' pane shows the message: 'Command(s) completed successfully.'

Primary Key composta

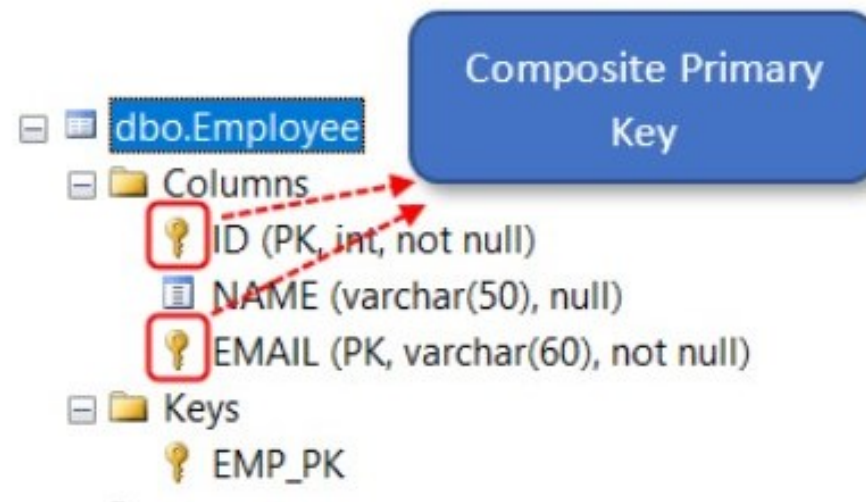
La chiave primaria costituita da più colonne o campi è nota come **chiave primaria composta**.
Nell'esempio seguente, creeremo una chiave primaria composta su più colonne come ID ed EMAIL.

Nota: Le 2 colonne devono essere entrambe NOT NULL altrimenti si avrà un errore durante la creazione della chiave primaria composta.

```
-- Create composite primary key
```

```
ALTER TABLE Employee ADD CONSTRAINT EMP_PK PRIMARY KEY (ID, EMAIL);
```

Da Interfaccia:



Foreign Key

La **FK (Foreign Key)** è un campo (o una raccolta di campi) di una tabella, che fa riferimento a una PK (Primary Key) di un'altra tabella.

- La tabella con la chiave esterna FK è chiamata tabella **figlio**
- La tabella con la chiave primaria PK è chiamata tabella **referenziata o padre**.

Sintassi creazione di una FK
in fase di creazione di una
nuova tabella:

```
CREATE TABLE Orders (  
    OrderID int NOT NULL PRIMARY KEY,  
    OrderNumber int NOT NULL,  
    PersonID int FOREIGN KEY REFERENCES Persons(PersonID)  
);
```

per definire una FK
eventualmente anche su più
colonne

```
CREATE TABLE Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)  
    REFERENCES Persons(PersonID)  
);
```

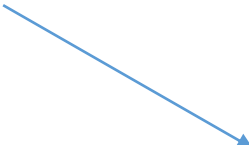
Foreign Key

Sintassi creazione di una FK
attraverso la modifica di una
tabella già esistente:

per definire una FK
eventualmente anche su più
colonne



```
ALTER TABLE Orders  
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```



```
ALTER TABLE Orders  
ADD CONSTRAINT FK_PersonOrder  
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

Violazione Vincoli di Integrità

```
CREATE TABLE Tabella (  
    . . . . . ,  
    Campo_FK Tipo Vincolo,  
    CONSTRAINT [Nome_Vincolo]  
    FOREIGN KEY (Campo_FK) REFERENCES AltraTabella (AltroCampo)  
        ON DELETE NO ACTION | CASCADE | SET NULL | SET DEFAULT  
        ON UPDATE NO ACTION | CASCADE | SET NULL | SET DEFAULT  
    . . . . .  
);
```

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

Violazione Vincoli di Integrità

Per ciascuna delle 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 Non ACTION.

SET DEFAULT, significa che nel campo chiave esterna delle righe correlate si impone il valore di base, indicato dalla CREATE TABLE.

UNIQUE

Il vincolo UNIQUE garantisce che tutti i valori in una colonna siano diversi.

Entrambi i vincoli UNIQUE e PRIMARY KEY forniscono una garanzia di unicità per una colonna o un insieme di colonne.

Un vincolo PRIMARY KEY ha automaticamente un vincolo UNIQUE.

Tuttavia, puoi avere molti vincoli UNIQUE per tabella, ma solo una PRIMARY KEY per tabella.

```
CREATE TABLE Persons (  
    ID int NOT NULL UNIQUE,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int  
);
```

```
ALTER TABLE Persons  
ADD UNIQUE (ID);
```

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    CONSTRAINT UC_Person UNIQUE (ID,LastName)  
);
```

```
ALTER TABLE Persons  
ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
```


CONTROLLI (CHECK)

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.

```
CREATE TABLE NomeTabella (  
    . . . . . ,  
    [Campo1] Tipo Vincolo,  
    [Campo2] Tipo Vincolo,  
    [Campo3] Tipo Vincolo,  
    CONSTRAINT [Nome_Vincolo_1]  
        CHECK ( condizione)  
)
```

I vincoli CHECK possono anche essere imposti dopo la creazione della tabella usando (l'alter table add constraint).

Demo

Accedere a SQL Server
Design of a Relational DB

