**Guida di riferimento a Microsoft Jet SQL**

http://office.microsoft.com/it-it/access/CH062526881040.aspx

1. **Panoramica**

* Tipi di dati SQL
* Tipi di dati SQL ANSI equivalenti
* Calcolo dei campi in funzioni SQL

1. **Definizione dei dati (DDL)**

* Istruzione CREATE TABLE
* Proposizione CONSTRAINT
* Istruzione ALTER TABLE
* Istruzione CREATE INDEX
* Istruzione CREATE VIEW
* Istruzione CREATE USER o GROUP
* Istruzione ADD USER
* Istruzione DROP USER o GROUP
* Istruzione DROP
* Istruzione GRANT
* Istruzione REVOKE

1. **Interrogazione dei dati**

* Istruzione SELECT
* Proposizione FROM
* Proposizione IN
* Predicati ALL, DISTINCT, DISTINCTROW, TOP
* Proposizione WHERE
* Operatore Between...And
* Operatore In
* Operatore Like
* Proposizione ORDER BY
* Proposizione GROUP BY
* Proposizione HAVING
* JOIN
  + Operazione INNER JOIN
  + Operazioni LEFT JOIN, RIGHT JOIN
* Operazione UNION

1. **Modifica dei dati (DML)**

* Istruzione SELECT...INTO
* Istruzione INSERT INTO
* Istruzione UPDATE
* Istruzione DELETE
* Dichiarazione PARAMETERS
* Dichiarazione WITH OWNERACCESS OPTION

1. **Procedure**

* Istruzione CREATE PROCEDURE
* Istruzione EXECUTE

1. **Funzioni di aggregazione SQL**

* Funzione COUNT (Conteggio)
* Funzione SUM (Somma)
* Funzione AVG (Media)
* Funzioni MIN, MAX
* Funzioni FIRST, LAST (Primo, Ultimo)
* Funzioni DevSt, StDevP
* Funzioni Var, VarP

1. **Appendici**

* Integrità Referenziale
* Join
* Perform Joins Using Access 2013 SQL
* Subquery

1. **Glossario**

**Panoramica**

Generalità

In informatica SQL (Structured Query Language) è un linguaggio standardizzato per database basati sul modello relazionale (RDBMS) progettato per:

creare e modificare schemi di database (DDL - Data Definition Language);

inserire, modificare e gestire dati memorizzati (DML - Data Manipulation Language);

interrogare i dati memorizzati (DQL - Data Query Language);

creare e gestire strumenti di controllo ed accesso ai dati (DCL - Data Control Language).

Nonostante il nome, non si tratta dunque solo di un semplice linguaggio di interrogazione, ma alcuni suoi sottoinsiemi si occupano della creazione, della gestione e dell'amministrazione del database.

Operatori

Gli operatori, messi a disposizione dal SQL standard si dividono in sette categorie:

Operatori di assegnazione

Operatori di confronto

Operatori stringa

Operatori aritmetici

Operatori condizionali

Operatori logici

Operatori tra bit

* Operatori di assegnazione

Gli operatori di assegnazione assegnano un valore a una variabile o a un campo.

= Esprime un'assegnazione e non restituisce alcun valore.

:= Esprime un'assegnazione di un valore ad una variabile non ancora istanziata e non restituisce alcun valore.

* Operatori di confronto

Gli operatori di confronto servono a determinare uguaglianze e disuguaglianze tra valori e ad effettuare ricerche all'interno dei dati. Di seguito uno schema tabellare:

|  |  |
| --- | --- |
| = | Esprime uguaglianza tra due valori numerici o stringhe di caratteri (dove non è usato come operatore di assegnazione) |
| IS | Si usa per verificare se un valore è NULL, oppure se corrisponde a un valore booleano (TRUE, FALSE, UNKNOWN). |
| LIKE | Esprime somiglianza tra due valori letterali: con l'operatore LIKE è possibile usare, per i confronti, i caratteri speciali % (sostituisce un arbitrario numero di lettere) e \_ (sostituisce una lettera arbitraria ) |
| < | Stabilisce se un valore è minore di un altro |
| > | Stabilisce se un valore è maggiore di un altro |
| <= | Stabilisce se un valore è minore o uguale di un altro |
| >= | Stabilisce se un valore è maggiore o uguale di un altro |
| <> | Stabilisce se due valori sono diversi tra loro |
| != | equivale a <> |
| BETWEEN ... AND | Recupera un valore compreso tra due valori |
| IN | Stabilisce se un valore è contenuto in una lista di valori possibili |
| EXISTS | Stabilisce se una determinata subquery restituisce un valore |
| ANY o SOME | Stabilisce se una determinata subquery restituisce almeno uno dei valori specificati |
| ALL | Stabilisce se una determinata subquery restituisce tutti i valori desiderati |

Ad alcuni di questi operatori corrisponde un operatore contrario che fa uso del termine NOT:

IS NOT

NOT LIKE

NOT BETWEEN

NOT IN

NOT EXISTS

* **Tipi di dati SQL**

I tipi di dati [SQL](javascript:go('/search/redir.aspx?AssetID=HP010441931040&CTT=5&Origin=HP010322481040#idh_dadefansisql92mode')) del modulo di gestione di database Microsoft Jet includono 13 tipi di dati primari definiti dal modulo di gestione di database Microsoft® Jet e diversi sinonimi validi e riconosciuti per questo tipo di dati.

|  |  |  |
| --- | --- | --- |
| **Tipo di dati**  **primari** | **Capacità di memoria** | **Descrizione** |
| BINARY | 1 byte per carattere | In un campo di questo tipo è possibile memorizzare qualsiasi tipo di dati. Non viene eseguita alcuna conversione dei dati, ad esempio in testo. I dati vengono visualizzati nell'output così come vengono inseriti in un campo di tipo Binary. |
| BIT | 1 byte | Valori di tipo Sì/No e campi che includono solo uno dei due valori. |
| TINYINT | 1 byte | Valore intero compreso tra 0 e 255. |
| MONEY | 8 byte | Un valore intero scalare compreso tra  – 922.337.203.685.477,5808 e 922.337.203.685.477,5807. |
| DATETIME (Vedere DOUBLE) | 8 byte | Valore relativo a una data o a un'ora compresa tra l'anno 100 e l'anno 9999. |
| UNIQUEIDENTIFIER | 128 bit | Numero identificativo univoco utilizzato con le chiamate di procedure remote. |
| REAL | 4 byte | Valore a virgola mobile e precisione singola in un intervallo compreso tra – 3,402823E38 e – 1,401298E-45 per i numeri negativi e tra 1,401298E-45 e 3,402823E38 per i numeri positivi e 0. |
| FLOAT | 8 byte | Valore a virgola mobile e precisione doppia in un intervallo compreso tra – 1,79769313486232E308 e – 4,94065645841247E-324 per i valori negativi e tra 4,94065645841247E-324 e 1,79769313486232E308 per i valori positivi e 0. |
| INT | 2 byte | Valore intero breve compreso tra – 32.768 e 32.767 (vedere note). |
| INTEGER | 4 byte | Valore intero lungo compreso tra – 2.147.483.648 e 2.147.483.647 (vedere note). |
| DECIMAL | 17 byte | Tipo di dati numerico esatto per la memorizzazione dei valori compresi nell'intervallo da 1028 - 1 a -1028 - 1. È possibile specificare sia la precisione (1 - 28) che la scala (0 - precisione definita). I valori predefiniti per la precisione e la scala sono, rispettivamente, 18 e 0. |
| TEXT | 2 byte per carattere (vedere note) | Da 0 a un massimo di 2,14 gigabyte. |
| IMAGE | Come necessario | Da 0 a un massimo di 2,14 gigabyte. Utilizzato per oggetti OLE. |
| CHARACTER | 2 byte per carattere (vedere note) | Da 0 a 255 caratteri. |

**Note**

* Un'istruzione ALTER TABLE consente di modificare sia la [base](javascript:go('/search/redir.aspx?AssetID=HP010441931040&CTT=5&Origin=HP010322481040#idh_dadefseed')) che l'[incremento](javascript:go('/search/redir.aspx?AssetID=HP010441931040&CTT=5&Origin=HP010322481040#idh_dadefincrement')). Le nuove righe inserite nella tabella includeranno valori che si basano sui nuovi valori di base e incremento, generati automaticamente per la colonna. Vengono generati duplicati nel caso in cui i nuovi valori di base e incremento includano valori corrispondenti a valori generati dalla base e dall'incremento precedenti. Se la colonna è una [chiave primaria](javascript:go('/search/redir.aspx?AssetID=HP010441931040&CTT=5&Origin=HP010322481040#idh_dadefprimarykey')), l'inserimento di nuove righe può provocare errori durante la creazione di valori duplicati.
* Per individuare l'ultimo valore utilizzato in una colonna a incremento automatico, è possibile utilizzare l'istruzione seguente: SELECT @@IDENTITY. Non è possibile specificare un nome per la tabella. Il valore restituito fa riferimento all'ultima tabella aggiornata contenente una colonna a incremento automatico.
* I caratteri dei campi definiti come tipo di dati [TEXT](javascript:go('/search/redir.aspx?AssetID=HP010441931040&CTT=5&Origin=HP010322481040#idh_dadeftextdatatype')) (anche noti come MEMO) o [CHAR](javascript:go('/search/redir.aspx?AssetID=HP010441931040&CTT=5&Origin=HP010322481040#idh_dadefchardatatype')) (anche noti come TEXT(n) con una lunghezza specificata) vengono memorizzati nel [formato di rappresentazione Unicode](javascript:go('/search/redir.aspx?AssetID=HP010441931040&CTT=5&Origin=HP010322481040#idh_dadefunicoderepresentationformat')). Per la memorizzazione di ogni carattere Unicode sono necessari 2 byte. Per i database Microsoft Jet esistenti che contengono prevalentemente caratteri, le dimensioni del file del database risulteranno raddoppiate, dopo la conversione del file nel formato di Microsoft Jet versione 4.0. Tuttavia, la rappresentazione Unicode di molti set di caratteri, in particolare quelli noti in precedenza come SBCS (Single-Byte Character Sets), può essere facilmente compressa nel formato a byte singolo. Definendo l'attributo COMPRESSION in una colonna CHAR, i dati verranno automaticamente compressi durante la memorizzazione, per essere quindi decompressi quando vengono recuperati dalla colonna.
* **Tipi di dati SQL ANSI equivalenti**

Nella seguente tabella vengono elencati i tipi di dati SQL ANSI, i tipi di dati SQL equivalenti del modulo di gestione di database Microsoft Jet e i relativi sinonimi. Vengono inoltre elencati i tipi di dati Microsoft® SQL Server™ equivalenti.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo di dati  SQL ANSI** | **Tipo di dati SQL di Microsoft Jet** | **Sinonimo** | **Tipo di dati Microsoft SQL Server** |
| BIT, BIT VARYING | BINARY (vedere note) | VARBINARY, BINARY VARYING BIT VARYING | BINARY, VARBINARY |
| Non supportato | BIT (vedere note) | BOOLEAN, LOGICAL, LOGICAL1, YESNO | BIT |
| Non supportato | TINYINT | INTEGER1, BYTE | TINYINT |
| Non supportato | COUNTER (vedere note) | AUTOINCREMENT | (vedere note) |
| Non supportato | MONEY | CURRENCY | MONEY |
| DATE, TIME, TIMESTAMP | DATETIME | DATE, TIME (vedere note) | DATETIME |
| Non supportato | UNIQUEIDENTIFIER | GUID | UNIQUEIDENTIFIER |
| DECIMAL | DECIMAL | NUMERIC, DEC | DECIMAL |
| REAL | REAL | SINGLE, FLOAT4, IEEESINGLE | REAL |
| DOUBLE PRECISION, FLOAT | FLOAT | DOUBLE, FLOAT8, IEEEDOUBLE, NUMBER (vedere note) | FLOAT |
| SMALLINT | SMALLINT | SHORT, INTEGER2 | SMALLINT |
| INTEGER | INTEGER | LONG, INT, INTEGER4 | INTEGER |
| INTERVAL | Non supportato |  | Non supportato |
| Non supportato | IMAGE | LONGBINARY, GENERAL, OLEOBJECT | IMAGE |
| Non supportato | TEXT (vedere note) | LONGTEXT, LONGCHAR, MEMO, NOTE, NTEXT (vedere note) | TEXT |
| CHARACTER, CHARACTER VARYING, NATIONAL CHARACTER, NATIONAL CHARACTER VARYING | CHAR (vedere note) | TEXT(n), ALPHANUMERIC, CHARACTER, STRING, VARCHAR, CHARACTER VARYING, NCHAR, NATIONAL CHARACTER, NATIONAL CHAR, NATIONAL CHARACTER VARYING, NATIONAL CHAR VARYING (vedere note) | CHAR, VARCHAR, NCHAR, NVARCHAR |

**Note**

* Il tipo di dati BIT SQL ANSI non corrisponde al tipo di dati BIT SQL di Microsoft Jet ma al tipo di dati Binary. Non esiste un equivalente SQL ANSI per il tipo di dati BIT SQL di Microsoft Jet.
* TIMESTAMP non è più supportato come sinonimo di DATETIME.
* NUMERIC non è più supportato come sinonimo di FLOAT o DOUBLE, viene invece utilizzato come sinonimo di DECIMAL.
* Nella rappresentazione Unicode viene sempre memorizzato un campo LONGTEXT.
* Se si utilizza TEXT come nome del tipo di dati senza specificare la lunghezza facoltativa, ad esempio TEXT(25), verrà creato un campo LONGTEXT. In questo modo è possibile scrivere l'istruzione CREATE TABLE che consente di rendere i tipi di dati compatibili con Microsoft SQL Server.
* Nella rappresentazione Unicode viene sempre memorizzato un campo CHAR, che corrisponde al tipo di dati SQL ANSI CHAR NATIONAL.
* Se si utilizza TEXT come nome del tipo di dati, specificando la lunghezza facoltativa, ad esempio TEXT(25), il tipo di dati del campo sarà equivalente al tipo di dati CHAR. In questo modo viene mantenuta la compatibilità con le versioni precedenti della maggior parte delle applicazioni Microsoft Jet, rendendo possibile l'allineamento del il tipo di dati TEXT (senza specificare una lunghezza) con Microsoft SQL Server.
* **Calcolo dei campi in funzioni SQL**

È possibile utilizzare l'argomento dell'espressione stringa in una funzione di aggregazione SQL per eseguire un calcolo sui valori inclusi in un campo. È possibile, ad esempio, calcolare una percentuale, quale una sovrattassa o l'IVA, moltiplicando il valore di un campo per una frazione.

La tabella seguente riporta alcuni esempi di calcoli eseguiti nei campi delle tabelle Ordini e Dettagli ordini.

|  |  |
| --- | --- |
| **Calcolo** | **Esempio** |
| Aggiunta di un numero a un campo | Trasporto + 5 |
| Sottrazione di un numero da un campo | Trasporto - 5 |
| Moltiplicazione di un campo per un numero | PrezzoUnitario \* 2 |
| Divisione di un campo per un numero | Trasporto / 2 |
| Aggiunta di un campo a un altro | Scorte + QuantitàOrdinata |
| Sottrazione di un campo da un altro | LivelloDiRiordino - Scorte |

Nell'esempio seguente viene calcolato lo sconto medio per tutti gli ordini del database. I valori contenuti nei campi PrezzoUnitario e Sconto vengono moltiplicati per determinare il valore dello sconto di ciascun ordine e ottenere quindi la media.

SELECT Media(PrezzoUnitario \* Sconto) AS [Sconto medio] FROM [Dettagli sugli ordini];

**Definizione dei dati (DDL)**

* **Istruzione CREATE TABLE**

Crea una nuova tabella.

##### Sintassi

CREATE [TEMPORARY] TABLE *tabella* (*tipo campo1* [(*dimensione*)] [NOT NULL] [WITH COMPRESSION | WITH COMP] [*indice1*] [, *tipo* *campo2* [(*dimensione*)] [NOT NULL] [*indice2*] [, ...]] [, CONSTRAINT *indicecampomultiplo* [, ...]])

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *tabella* | Nome della tabella da creare. |
| *campo1*, *campo2* | Nome del campo o dei campi da creare nella nuova tabella. È necessario creare almeno un campo. |
| *tipo* | Tipo di dati del *campo* nella nuova tabella. |
| *dimensione* | Dimensione del campo in caratteri (solo per campi di tipo testo o binario). |
| *indice1*, *indice2* | Indice di campo singolo definito da una proposizione CONSTRAINT. |
| *indicecampomultiplo* | Indice di un campo a più campi definito da una proposizione CONSTRAINT. |

##### Osservazioni

Utilizzando l'istruzione CREATE TABLE è possibile definire una nuova tabella, nonché i relativi campi e vincoli di campo. Se per un campo viene specificato un valore NOT NULL, sono necessari nuovi record per ottenere dati validi nel campo.

La proposizione CONSTRAINT consente di stabilire varie limitazioni per un campo e può essere utilizzata per definire la chiave primaria. La chiave primaria può inoltre essere definita mediante l'istruzione CREATE INDEX che consente di aggiungere indici a tabelle esistenti.

È possibile utilizzare un valore NOT NULL in un campo singolo o in una proposizione CONSTRAINT denominata valida per un campo singolo o per un campo a più campi denominato CONSTRAINT. La limitazione NOT NULL può tuttavia essere applicata solo una volta al campo; se si applica tale limitazione più di una volta, verrà generato un errore di run-time.

La tabella TEMPORARY è visibile solo durante la sessione nella quale viene creata. Al termine della sessione, viene eliminata automaticamente. Più utenti possono accedere contemporaneamente alle tabelle temporanee.

L'attributo WITH COMPRESSION può essere utilizzato solo con i tipi di dati CHARACTER e MEMO (anche detti TEXT) e i relativi sinonimi.

L'attributo WITH COMPRESSION è stato aggiunto alle colonne CHARACTER a causa della modifica alla rappresentazione dei caratteri nel sistema Unicode, per il quale sono sempre necessari due byte per ogni carattere. Dal momento che i database Microsoft® Jet includono principalmente caratteri, le dimensioni del file del database risultano raddoppiate quando il file viene convertito nel formato di Microsoft Jet versione 4.0. Tuttavia, la rappresentazione Unicode di molti set di caratteri, in particolare quelli precedentemente denominati SBCS (set di caratteri a un byte, Single-Byte Character Sets), può essere facilmente compressa nel formato a byte singolo. Definendo una colonna CHARACTER con questo attributo, i dati verranno automaticamente compressi durante la memorizzazione, per poi essere decompressi quando vengono recuperati dalla colonna.

È inoltre possibile definire colonne di tipo MEMO per memorizzare dati in formato compresso, con un limite: possono essere compresse solo le istanze di colonne MEMO che, compresse, non superino i 4096 byte. Tutte le altre istanze di colonne MEMO non verranno compresse e quindi può accadere che nella stessa colonna MEMO di una tabella, possono essere presenti dati compressi e dati non compressi.

Nel seguente esempio si crea una nuova tabella denominata Questa tabella con due campi di tipo testo.

CREATE TABLE QuestaTabella (Nome CHAR, Cognome CHAR);

La seguente istruzione di creazione tabella crea una tabella con un campo contatore:

CREATE TABLE t\_test(id COUNTER PRIMARY KEY, name TEXT(10));

La seguente istruzione di creazione tabella crea una tabella con tutti i tipi di dati dispobili in Microsoft Access:

CREATE TABLE TestAllTypes

(

MyText TEXT(50),

MyMemo MEMO,

MyByte BYTE,

MyInteger INTEGER,

MyLong LONG,

MyAutoNumber COUNTER,

MySingle SINGLE,

MyDouble DOUBLE,

MyCurrency CURRENCY,

MyReplicaID GUID,

MyDateTime DATETIME,

MyYesNo YESNO,

MyOleObject LONGBINARY,

MyBinary BINARY(50)

)

Esempi con vincoli

Create Table GruppiConcerti

(

IDGruppo int,

IDConcerto int,

nome char(20),

luogo char (20),

Data date,

PRIMARY KEY (IDGruppo, IDConcerto),

FOREIGN KEY (IDGruppo) references Gruppi(IDGruppo),

FOREIGN KEY (IDConcerto) references Concerti(IDConcerto)

)

Create Table GruppiConcerti

(

IDGruppo int constraint FKGruppi references Gruppi(IDGruppo),

IDConcerto int constraint FKConcerti references Concerti(IDConcerto),

nome char(20),

luogo char (20),

Data date,

constraint PKGruppiConcerti PRIMARY KEY (IDGruppo, IDConcerto)

)

Create Table GruppiConcerti

(

IDGruppo int,

IDConcerto int,

nome char(20),

luogo char (20),

Data date,

constraint PKGruppiConcerti PRIMARY KEY (IDGruppo, IDConcerto),

constraint FKGruppi FOREIGN KEY (IDGruppo) references Gruppi(IDGruppo),

FOREIGN KEY (IDConcerto) references Concerti(IDConcerto)

)

* **Proposizione CONSTRAINT**

Un vincolo è simile a un indice, sebbene possa essere utilizzato anche per stabilire una relazione con un'altra tabella.

La proposizione CONSTRAINT viene utilizzata nelle istruzioni ALTER TABLE e CREATE TABLE per creare o eliminare vincoli. Esistono due tipi di proposizioni CONSTRAINT: il primo tipo consente di creare un vincolo in un singolo campo, mentre il secondo tipo consente di creare vincoli in più campi.

##### Sintassi

Vincolo in un singolo campo:

CONSTRAINT *nome* {PRIMARY KEY | UNIQUE | NOT NULL |  
REFERENCES *tabellaesterna* [(*campoesterno1, campoesterno2*)]  
[ON UPDATE CASCADE | SET NULL]  
[ON DELETE CASCADE | SET NULL]}

Vincolo in più campi:

CONSTRAINT *nome*  
{PRIMARY KEY (*primaria1*[, *primaria2* [, ...]]) |  
UNIQUE (*univoca1*[, *univoca2* [, ...]]) |  
NOT NULL (*nonnull1*[, *nonnull2* [, ...]]) |  
FOREIGN KEY [NO INDEX] (*rif1*[, *rif2* [, ...]]) REFERENCES *tabellaesterna* [(*campoesterno1* [, *campoesterno2* [, ...]])]  
[ON UPDATE CASCADE | SET NULL]  
[ON DELETE CASCADE | SET NULL]}

La proposizione CONSTRAINT è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *nome* | Nome del vincolo da creare. |
| *primaria1*, *primaria2* | Nome del campo o dei campi da definire come chiave primaria. |
| *univoca1*, *univoca2* | Nome del campo o dei campi da definire come chiave univoca. |
| *nonnull1, nonnull2* | Nome del campo o dei campi che vengono limitati a valori diversi da Null. |
| *rif1*, *rif2* | Nome del campo o dei campi di una chiave esterna che fanno riferimento ai campi di un'altra tabella. |
| *tabellaesterna* | Nome della tabella esterna contenente il campo o i campi specificati da *campoesterno*. |
| *campoesterno1*, *campoesterno2* | Nome del campo o dei campi della *tabellaesterna* indicati da *rif1* e *rif2*. È possibile omettere questa proposizione se il campo di riferimento corrisponde alla chiave primaria della *tabellaesterna*. |

##### Osservazioni

La sintassi per la creazione di un vincolo in un singolo campo viene utilizzata nella proposizione di definizione del campo di un'istruzione ALTER TABLE o CREATE TABLE, subito dopo la specifica del tipo di dati del campo.

La sintassi per la creazione di un vincolo in più campi deve essere utilizzata tutte le volte che si utilizza la parola riservata CONSTRAINT al di fuori della proposizione di definizione di un campo in un'istruzione ALTER TABLE o CREATE TABLE.

La proposizione CONSTRAINT consente di assegnare a un campo uno dei seguenti tipi di vincoli:

* Utilizzando la parola riservata UNIQUE è possibile definire un campo come chiave univoca, in modo che la tabella non possa contenere due record con lo stesso valore del campo specificato. Qualsiasi campo o elenco di campi può essere definito come univoco. Se come chiave univoca viene definito un vincolo in più campi, la combinazione dei valori di tutti i campi dell'indice dovrà essere univoca, anche se due o più record includono lo stesso valore in uno solo dei campi.
* Utilizzando la parola riservata PRIMARY KEY è possibile definire un campo o un insieme di campi di una tabella come chiave primaria. Tutti i valori della chiave primaria devono essere univoci e diversi da **Null**. Ogni tabella, inoltre, può includere una sola chiave primaria.

**Nota**  Se si imposta un vincolo PRIMARY KEY in una tabella che già include una chiave primaria, si verificherà un errore.

* Utilizzando la parola riservata FOREIGN KEY è possibile definire un campo come chiave esterna. Se la chiave primaria della tabella esterna include più campi, è necessario utilizzare una definizione di vincolo in più campi, elencando tutti i campi di riferimento, il nome della tabella esterna e i nomi dei campi della tabella esterna a cui viene fatto riferimento, nello stesso ordine in cui vengono elencati i campi di riferimento. Non è necessario specificare i campi a cui viene fatto riferimento se questi sono stati definiti come chiave primaria della tabella esterna. Per impostazione predefinita, il modulo di gestione di database presume che la chiave primaria della tabella esterna corrisponda ai campi a cui viene fatto riferimento.

I vincoli di chiave esterna consentono di specificare l'esecuzione di determinate operazioni quando viene modificato un valore della chiave primaria corrispondente.

* È possibile associare l'esecuzione di un'azione nella tabella esterna all'esecuzione di un'azione corrispondente in una chiave primaria della tabella per cui è stata definita la proposizione CONSTRAINT. Si consideri ad esempio la definizione seguente della tabella Clienti:

CREATE TABLE Clienti (IdCliente INTEGER PRIMARY KEY, CLstNm NCHAR VARYING (50))

Nella definizione seguente della tabella Ordini viene creata una relazione di chiave esterna che fa riferimento alla chiave primaria della tabella Clienti:

CREATE TABLE Ordini (IdOrdine INTEGER PRIMARY KEY, IdCliente INTEGER, NoteOrdine NCHAR VARYING (255), CONSTRAINT FKOrdiniIdCliente FOREIGN KEY (IdCliente) REFERENCES Clienti ON UPDATE CASCADE ON DELETE CASCADE

Sia la proposizione ON UPDATE CASCADE che la proposizione ON DELETE CASCADE vengono definite in relazione alla chiave esterna. La prima proposizione consente di replicare nella tabella Ordini l'aggiornamento di un valore identificativo del cliente (IdCliente) eseguito nella tabella Clienti. Ogni ordine contenente il valore identificativo del cliente corrispondente verrà automaticamente aggiornato in base al nuovo valore. La seconda proposizione consente di estendere l'eliminazione di un cliente dalla tabella Clienti a tutte le righe della tabella Ordini che contengono lo stesso valore identificativo del cliente.

Si consideri la diversa definizione della tabella Ordini riportata di seguito, in cui viene utilizzata l'azione SET NULL invece dell'azione CASCADE:

CREATE TABLE Ordini (IdOrdine INTEGER PRIMARY KEY, IdCliente INTEGER, NoteOrdine NCHAR VARYING (255), CONSTRAINT FKOrdiniIdCliente FOREIGN KEY (IdCliente) REFERENCES Clienti ON UPDATE SET NULL ON DELETE SET NULL

La proposizione ON UPDATE SET NULL consente di stabilire che, in caso di aggiornamento del valore identificativo di un cliente (IdCliente) nella tabella Clienti, i valori della chiave esterna corrispondenti nella tabella Ordini verranno automaticamente impostati su NULL. Analogamente, la proposizione ON DELETE SET NULL consente di stabilire che, se viene eliminato un cliente dalla tabella Clienti, tutte le chiavi esterne corrispondenti nella tabella Ordini verranno automaticamente impostate su NULL.

**Nota**: È possibile non specificare se "Aggiornare CASCADE" o "Eliminare CASCADE" si desiderano con una relazione creata utilizzando DDL. Queste funzionalità sono disponibili quando utilizzano esclusivamente le interfacce Microsoft DAO (oggetto DAO) tramite il codice o utilizzano l'interfaccia utente di Microsoft Access. (http://support.microsoft.com/?id=180841)

Per impedire la creazione automatica di indici di chiave esterne, è possibile utilizzare il modificatore NO INDEX. È consigliabile utilizzare questa definizione di chiave esterna solo nel caso in cui i valori dell'indice ottenuto vengano duplicati frequentemente. In questo caso, infatti, l'utilizzo di un indice è meno efficace rispetto alla semplice analisi della tabella. La gestione di questo tipo di indice, con continui inserimenti ed eliminazioni di righe, influisce negativamente sulla prestazioni senza offrire alcun vantaggio.

* + **Istruzione ALTER TABLE**

Modifica la struttura di una tabella creata tramite l'istruzione CREATE TABLE.

##### Sintassi

ALTER TABLE *tabella*

{

ADD {COLUMN *campo tipo*[(*dimensione*)] [NOT NULL] [CONSTRAINT *indice*] |  
ALTER COLUMN *campo tipo*[(*dimensione*)] |  
CONSTRAINT *indicemulticampo*} |  
DROP {COLUMN *campo* I CONSTRAINT *nomeindice*}

}

L'istruzione ALTER TABLE è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *tabella* | Il nome della tabella da modificare. |
| *campo* | Il nome del campo da aggiungere o da eliminare dalla *tabella* oppure il nome del campo che deve essere modificato all'interno della *tabella*. |
| *tipo* | Il tipo di dati del *campo*. |
| *dimensione* | La dimensione del campo in caratteri (solo per campi di tipo testo o binario). |
| *indice* | L'indice del *campo*. |
| *indicemulticampo* | La definizione di un indice multicampo da aggiungere alla *tabella*. |
| *nomeindice* | Il nome dell'indice multicampo da rimuovere. |

##### Osservazioni

Esistono diversi modi per modificare una tabella esistente mediante l'istruzione ALTER TABLE. Ad esempio:

* È possibile utilizzare ADD COLUMN per aggiungere un nuovo campo alla tabella, specificando il nome del campo, il tipo di dati e, facoltativamente (per i campi di tipo testo o binario), la dimensione. Ad esempio, l'istruzione seguente aggiunge alla tabella Impiegati un campo di tipo testo di 25 caratteri denominato Note:

ALTER TABLE Impiegati ADD COLUMN Note TEXT(25)

È inoltre possibile definire un indice relativo a tale campo.

Se per un campo si specifica NOT NULL, sono necessari nuovi record per includere dati validi nel campo.

* È possibile utilizzare ALTER COLUMN per modificare il tipo di dati di un campo esistente, specificando il nome del campo, il nuovo tipo di dati e, facoltativamente (per i campi di tipo testo o binaro), la dimensione.

L'istruzione seguente modifica il tipo di dati di un campo della tabella Impiegati denominato CAP (originariamente definito come tipo Integer) in un campo di tipo testo di 10 caratteri:

ALTER TABLE Impiegati ALTER COLUMN CAP TEXT(10)

L’istruzione seguente aggiunge il vincolo di Non Nullità al campo MioCampo

ALTER TABLE MiaTabella ALTER COLUMN MioCampo CHAR NOT NULL;

* È possibile utilizzare ADD CONSTRAINT per aggiungere un indice multicampo.
* È possibile utilizzare DROP COLUMN per eliminare un campo, specificando solo il nome del campo.
* È possibile utilizzare DROP CONSTRAINT per eliminare un indice multicampo, specificando solo il nome dell'indice seguito dalla parola riservata CONSTRAINT.

**Esempi vari:**

' Delete the Salary field from the Employees table.

ALTER TABLE Employees DROP COLUMN Salary

' Add a foreign key to the Orders table.

ALTER TABLE Orders ADD CONSTRAINT OrdersRelationship FOREIGN KEY (EmployeeID) REFERENCES Employees (EmployeeID)

' Remove the OrdersRelationship foreign key from the Orders table.

ALTER TABLE Orders DROP CONSTRAINT OrdersRelationship

**Note**

* Non è possibile aggiungere o eliminare più di un campo o indice alla volta.
* È possibile utilizzare l'istruzione CREATE INDEX per aggiungere un indice a campo singolo o multicampo a una tabella. Mediante le istruzioni ALTER TABLE e DROP è possibile eliminare un indice creato con le istruzioni ALTER TABLE o CREATE INDEX.
* È possibile utilizzare NOT NULL in un campo singolo o in una proposizione denominata CONSTRAINT riferita a un campo singolo o a un campo multiplo denominato CONSTRAINT. La limitazione NOT NULL può tuttavia essere applicata una sola volta a un campo. Se si applica tale limitazione più di una volta, verrà generato un errore di run-time.
* **Istruzione CREATE INDEX**

Crea un nuovo indice in una tabella esistente.

##### Sintassi

CREATE [ UNIQUE ] INDEX *indice*  
ON *tabella* (*campo* [ASC|DESC][, *campo* [ASC|DESC], ...])  
[WITH { PRIMARY | DISALLOW NULL | IGNORE NULL }]

L'istruzione CREATE INDEX è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *indice* | Nome dell'indice da creare. |
| *tabella* | Nome della tabella esistente che includerà l'indice. |
| *campo* | Nome del campo o dei campi da indicizzare. Per creare un indice relativo a un singolo campo, indicare il nome del campo racchiuso tra parentesi dopo il nome della tabella. Per creare un indice relativo a più campi, indicare il nome di tutti i campi da includere nell'indice. Per creare indici in ordine decrescente, utilizzare la parola riservata DESC. In caso contrario gli indici verranno creati in ordine crescente. |

##### Osservazioni

Per impedire l'immissione di valori duplicati nel campo o nei campi indicizzati di record diversi, utilizzare la parola riservata UNIQUE.

Nella proposizione facoltativa WITH è possibile imporre regole di convalida dei dati. Ad esempio:

* L'opzione DISALLOW NULL consente di impedire l'immissione di voci [Null](javascript:go('/search/redir.aspx?AssetID=HP010441931040&CTT=5&Origin=HP010322171040#idh_dadefnull')) relative a nuovi record nel campo o nei campi indicizzati.
* L'opzione IGNORE NULL consente di evitare l'inclusione nell'indice di record con valori **Null** nel campo o nei campi indicizzati.
* La parola riservata PRIMARY consente di definire il campo o i campi indicizzati come chiave primaria. In questo caso, poiché la chiave è unica, la parola riservata UNIQUE può essere omessa.

Per aggiungere a una tabella un indice a campo singolo o a più campi, è possibile utilizzare inoltre l'istruzione ALTER TABLE. Tale istruzione, insieme con l'istruzione DROP, consente di eliminare un indice creato mediante ALTER TABLE o CREATE INDEX.

**Nota**  Non utilizzare la parola riservata PRIMARY durante la creazione di un nuovo indice in una tabella che già include una chiave primaria. In caso contrario, si verificherà un errore.

La seguente istruzioneQuery crea un single-field univoco per decrescere l'indice denominato MySalaryIndex sul campo Stipendio della tabella HRInfo:

CREATE UNIQUE INDEX MySalaryIndex

ON HRInfo (Salary DESC) WITH DISALLOW NULL

Questo indice applica che ogni record deve avere un valore per il campo Salariale.

* **Istruzione CREATE VIEW**

Crea una nuova vista.

##### Sintassi

CREATE VIEW *vista* [(*campo1*[, *campo2*[, ...]])] AS *istruzioneselect*

L'istruzione CREATE VIEW è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *vista* | Nome della vista da creare. |
| *campo1*, *campo2* | Nome del campo o dei campi corrispondenti ai campi specificati in *istruzioneselect*. |
| *istruzioneselect* | Istruzione SQL SELECT. |

##### Osservazioni

L'istruzione SELECT con cui viene definita la vista non può essere un'istruzione SELECT...INTO.

L'istruzione SELECT con cui viene definita la vista non può includere alcun parametro.

Il nome della vista non può essere lo stesso nome utilizzato per una tabella esistente. È possibile aggiornare la visualizzazione solo se è possibile aggiornare la query definita dall'istruzione SELECT. In caso contrario, la vista sarà di sola lettura.

Se i nomi di due campi della query definita dall'istruzione SELECT sono identici, la definizione della vista dovrà includere un elenco di campi in cui vengono specificati nomi univoci per ognuno dei campi della query.

Nel seguente esempio viene creata ed utilizzata da un’altra query la vista Europa

CREATE VIEW Europa AS SELECT name, population AS pop

FROM world WHERE region='Europe';

SELECT \* FROM v\_europe

* **Istruzione DROP**

Elimina una tabella, una procedura o una vista da un database, oppure un indice da una tabella.

##### Sintassi

DROP {TABLE *tabella* | INDEX *indice* ON *tabella* | PROCEDURE *procedura* | VIEW *vista*}

L'istruzione DROP è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *tabella* | Nome della tabella da eliminare o della tabella da cui deve essere eliminato un indice. |
| *procedura* | Nome della procedura da eliminare. |
| *vista* | Nome della vista da eliminare. |
| *indice* | Nome dell'indice da eliminare dalla *tabella*. |

##### Osservazioni

* È necessario chiudere la tabella prima di procedere all'eliminazione della stessa o dell'indice.
* Per eliminare un indice da una tabella è inoltre possibile utilizzare l'istruzione ALTER TABLE.
* È possibile utilizzare l'istruzione CREATE TABLE per creare una tabella e le istruzioni CREATE INDEX o ALTER TABLE per creare un indice. Per modificare una tabella utilizzare ALTER TABLE.

**Controllo ed accesso ai dati (DCL - Data Control Language)**

* **Istruzione GRANT**

Concede privilegi speciali a un utente o a un gruppo esistente.

##### Sintassi

GRANT {*privilegio*[, *privilegio*, …]} ON{TABLE *tabella* |   
OBJECT *oggetto*|

CONTAINER *contenitore* } TO {*nomeautorizzazione*[, *nomeautorizzazione*, …]}

L'istruzione GRANT è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *privilegio* | Privilegio o privilegi da concedere. I privilegi vengono specificati mediante le parole chiave seguenti:  SELECT, DELETE, INSERT, UPDATE, DROP, SELECTSECURITY, UPDATESECURITY, DBPASSWORD, UPDATEIDENTITY, CREATE, SELECTSCHEMA, SCHEMA e UPDATEOWNER. |
| *nometabella* | Nome di tabella valido. |
| *oggetto* | Può trattarsi di qualsiasi oggetto diverso da una tabella, ad esempio una query archiviata (vista o procedura). |
| *contenitore* | Nome valido di un contenitore. |
| *nomeautorizzazione* | Nome di utente o di gruppo. |

GRANT SELECT, INSERT, UPDATE (voto, condotta) ON studenti TO professore;

GRANT ALL ON studenti TO segreteria;

* **Istruzione REVOKE**

Revoca specifici privilegi a un utente o a un gruppo esistenti.

##### Sintassi

REVOKE {*privilegio*[, *privilegio*, …]} ON  
{TABLE *tabella* |  
OBJECT *oggetto*|

CONTAINTER *contenitore*}  
FROM {*nomeautorizzazione*[, *nomeautorizzazione*, …]}

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *privilegio* | Il privilegio o i privilegi da revocare. I privilegi vengono specificati mediante le seguenti parole chiave: SELECT, DELETE, INSERT, UPDATE, DROP, SELECTSECURITY, UPDATESECURITY, DBPASSWORD, UPDATEIDENTITY, CREATE, SELECTSCHEMA, SCHEMA e UPDATEOWNER. |
| *tabella* | Un nome valido di tabella. |
| *oggetto* | Un oggetto diverso da una tabella. Un esempio può essere una query memorizzata (vista o procedura). |
| *contenitore* | Il nome di un contenitore valido. |
| *nomeautorizzazione* | Un nome di utente o di gruppo. |

**Interrogazione dei dati**

* **Istruzione SELECT**

Indica al modulo di gestione di database Microsoft Jet di restituire le informazioni del database come set di record.

##### Sintassi

SELECT [*predicato*] { \* | *tabella*.\* | [*tabella*.]*campo1* [AS *alias1*] [, [*tabella*.]*campo2* [AS *alias2*] [, ...]]}  
FROM *espressionetabella* [, ...] [IN *databaseesterno*]  
[WHERE... ]  
[GROUP BY... ]  
[HAVING... ]  
[ORDER BY... ]  
[WITH OWNERACCESS OPTION]

L'istruzione SELECT è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *predicato* | Uno dei predicati seguenti: ALL, DISTINCT, DISTINCTROW o TOP. I predicati consentono di limitare il numero di record restituito. Se non viene specificato alcun predicato, l'impostazione predefinita è ALL. |
| *\** | Specifica che tutti i campi delle tabelle specificate sono selezionati. |
| *tabella* | Il nome della tabella che contiene i campi di cui vengono selezionati i record. |
| *campo1*, *campo2* | I nomi dei campi che contengono i dati da recuperare. Se si includono più campi, tali campi vengono recuperati secondo l'ordine dell'elenco. |
| *alias1*, *alias2* | I nomi da utilizzare nelle intestazioni di colonna al posto dei nomi originali delle colonne della *tabella*. |
| *espressionetabella* | Il nome della tabella o delle tabelle che contengono i dati da recuperare. |
| *databaseesterno* | Il nome del database contenente le tabelle indicate in *espressionetabella*, se queste sono esterne al database corrente. |

##### Osservazioni

Per eseguire questa operazione, il modulo di gestione di database Microsoft® Jet esegue la ricerca nelle tabelle specificate, estrae le colonne scelte, seleziona le righe che corrispondono al criterio prescelto e ordina o raggruppa le righe risultanti nell'ordine specificato.

Le istruzioni SELECT non modificano i dati nel database.

In genere SELECT è la prima parola di un'istruzione SQL. La maggior parte delle istruzioni SQL è rappresentata da istruzioni SELECT o SELECT...INTO.

La sintassi minima di un'istruzione SELECT è la seguente:

SELECT *campi* FROM *tabella*

È possibile utilizzare un asterisco (\*) per selezionare tutti i campi di una tabella. Nell'esempio seguente vengono selezionati tutti i campi della tabella Impiegati:

SELECT \* FROM Impiegati;

Se il nome di un campo è incluso in più tabelle indicate nella proposizione FROM, è necessario anteporre al nome del campo il nome della tabella seguito dall'operatore **.** (punto). Nell'esempio che segue, il campo Reparto è incluso sia nella tabella Impiegati sia nella tabella Superiori. L'istruzione SQL seleziona i reparti dalla tabella Impiegati i nomi dei superiori nella tabella Superiori:

SELECT Impiegati.Reparto, Superiori.NomeSuperiore

FROM Impiegati INNER JOIN Superiori

WHERE Impiegati.Reparto = Superiori.Reparto;

Durante la creazione di un oggetto Recordset, il nome del campo della tabella viene utilizzato dal modulo di gestione di database Microsoft Jet come nome dell'oggetto **Recordset**. Se si desidera utilizzare un nome di campo diverso o un nome non incluso nell'espressione utilizzata per generare il campo, utilizzare la parola riservata AS. Nell'esempio seguente viene utilizzata l'intestazione Nascita per denominare l'oggetto **Field** restituito nell'oggetto **Recordset** risultante:

SELECT DataDiNascita AS Nascita FROM Impiegati;

Se si utilizzano funzioni di aggregazione o query che restituiscono nomi di oggetti **Field** duplicati o non univoci, è necessario utilizzare la proposizione AS per indicare un nome alternativo per l'oggetto **Field**. Nell'esempio seguente viene utilizzata l'intestazione ContatoreIniziale per denominare l'oggetto **Field** restituito nell'oggetto **Recordset** risultante:

SELECT COUNT(IDImpiegato) AS ContatoreIniziale FROM Impiegati;

È possibile integrare l'istruzione SELECT con altre proposizioni per limitare e organizzare in modo più efficace i dati restituiti. Per ulteriori informazioni, vedere l'argomento della Guida relativo alla proposizione in uso.

##### Esempi

In questo esempio si selezionano i campi Cognome e Nome contenuti in tutti i record della tabella Impiegati.

SELECT Cognome, Nome FROM Impiegati;

Nel seguente esempio si conta il numero di record che contengono dati nel campo CAP e denomina il campo restituito Conteggio.

SELECT Count(CAP) AS Conteggio FROM Clienti;

Nel seguente esempio si visualizza il numero di impiegati, lo stipendio medio e lo stipendio massimo.

SELECT Count (\*) AS TotaleImpiegati, Avg(Stipendio) AS StipendioMedio, AS StipendioMassimo FROM Impiegati;

* **Proposizione FROM**

Specifica le tabelle o le query che includono i campi elencati nell'istruzione SELECT.

##### Sintassi

SELECT *elencocampi*  
FROM *espressionetabella* [IN *databaseesterno*]

Un'istruzione SELECT che include una proposizione FROM è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *elencocampi* | Nome del campo o dei campi da recuperare insieme agli alias dei nomi di campo, alle funzioni di aggregazione SQL, ai predicati di selezione (ALL, DISTINCT, DISTINCTROW o TOP) e altre opzioni dell'istruzione SELECT. |
| *espressionetabella* | Espressione che identifica una o più tabelle da cui recuperare i dati. L'espressione può essere un singolo nome di tabella, un nome di una query salvata o il risultato restituito da operazioni INNER JOIN, LEFT JOIN o RIGHT JOIN. |
| *databaseesterno* | Percorso completo di un database esterno che include tutte le tabelle indicate in *espressionetabella*. |

##### Osservazioni

La proposizione FROM è obbligatoria e deve seguire l'istruzione SELECT.

L'ordine dei nomi delle tabelle in *espressionetabella* non ha alcuna rilevanza.

Per ottenere prestazioni ottimali e maggiore facilità d'utilizzo, è consigliabile recuperare dati da un database esterno utilizzando una tabella collegata invece di una proposizione IN.

Nell'esempio seguente vengono recuperati dati dalla tabella Impiegati:

SELECT Cognome, Nome FROM Impiegati;

* **Proposizione IN**

Identifica le tabelle di un database esterno a cui connettere il modulo di gestione di database Microsoft Jet, ad esempio un database dBASE o Paradox oppure un database Microsoft® Jet esterno.

##### Sintassi

Per identificare una tabella di destinazione:

[SELECT | INSERT] INTO *destinazione* IN  
{*percorso* | ["*percorso*" "*tipo*"] | ["" [*tipo*; DATABASE = *percorso*]]}

Per identificare una tabella di origine:

FROM *espressionetabella* IN  
{*percorso* | ["*percorso*" "*tipo*"] | ["" [*tipo*; DATABASE = *percorso*]]}

Un'istruzione SELECT che contiene una proposizione IN è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *destinazione* | Nome della tabella esterna nella quale vengono inseriti i dati. |
| *espressionetabella* | Nome della tabella o delle tabelle da cui recuperare i dati. Questo argomento può essere il nome di una singola tabella, nel nome di una query salvata o dal risultato derivante dalle operazioni INNER JOIN, LEFT JOIN o RIGHT JOIN. |
| *percorso* | Percorso completo della directory o del file che include la *tabella*. |
| *tipo* | Nome del tipo di database utilizzato per creare la *tabella*, se diverso da un database Microsoft Jet (ad esempio, dBASE III, dBASE IV, Paradox 3.x o Paradox 4.x). |

##### Osservazioni

La proposizione IN consente di connettere un solo database esterno alla volta.

In alcuni casi, l'argomento *percorso* fa riferimento alla directory che contiene i file del database. Ad esempio, utilizzando le tabelle dei database dBASE, Microsoft FoxPro® o Paradox, l'argomento *percorso* specifica la directory contenente i file con estensione dbf o db. Il nome del file della tabella deriva dall'argomento *destinazione* o *espressionetabella*.

Per specificare un database diverso da Microsoft Jet, aggiungere al nome un punto e virgola (;) e includerlo tra le virgolette singole (' ') o doppie (" "). È possibile, ad esempio, utilizzare sia 'dBASE IV;' che "dBASE IV;". È inoltre possibile utilizzare la parola riservata DATABASE per specificare il database esterno. Ad esempio, nelle righe seguenti viene specificata la stessa tabella:

... FROM Tabella IN "" [dBASE IV; DATABASE=C:\DBASE\DATA\SALES;];

... FROM Tabella IN "C:\DBASE\DATA\SALES" "dBASE IV;"

**Note**

Per ottenere prestazioni ottimali e maggiore facilità d'utilizzo, è consigliabile recuperare dati utilizzando una tabella collegata invece di una proposizione IN.

È inoltre possibile utilizzare la parola riservata IN come operatore di confronto in un'espressione.

* **Predicati ALL, DISTINCT, DISTINCTROW, TOP**

Specificano i record selezionati con query SQL.

##### Sintassi

SELECT [ALL | DISTINCT | DISTINCTROW | [TOP *n* [PERCENT]]]  
FROM *tabella*

Un'istruzione SELECT che include questi predicati è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| ALL | Viene presupposto questo predicato qualora non ne venga incluso un altro. Il modulo di gestione di database Microsoft Jet seleziona tutti i record che soddisfano le condizioni dell'istruzione SQL. Gli esempi che seguono sono equivalenti e in essi vengono restituiti tutti i record della tabella Impiegati:  SELECT ALL \* FROM Impiegati ORDER BY IDImpiegato;  SELECT \* FROM Impiegati ORDER BY IDImpiegato; |
| DISTINCT | Omette i record che includono dati duplicati nei campi selezionati. Nei risultati della query vengono inclusi solo i record che presentano valori univoci per ogni campo elencato nell'istruzione SELECT. Ad esempio, può verificarsi il caso in cui diversi impiegati elencati nella tabella Impiegati abbiano lo stesso cognome. Se due record contengono Zaffaroni nel campo Cognome, mediante la seguente istruzione SQL viene restituito solo uno dei record che include il cognome Zaffaroni:  SELECT DISTINCT Cognome FROM Impiegati;  Se si omette DISTINCT, vengono restituiti entrambi i record Zaffaroni.  Se la proposizione SELECT include più campi, è necessario che la combinazione dei valori di tutti i campi di un record sia univoca perché tale record possa essere incluso nei risultati.  L'output di una query in cui viene utilizzato DISTINCT non è aggiornabile e non riflette le modifiche apportate successivamente da altri utenti. |
| DISTINCTROW | Omette i dati in base agli interi record duplicati e non solo in base ai campi duplicati. Ad esempio, è possibile creare una query che esegua il join delle tabelle Clienti e Ordini in base al campo IDCliente. Nella tabella Clienti non sono inclusi campi IDCliente duplicati, diversamente dalla tabella Ordini, dove invece sono presenti in quanto a ogni cliente possono essere associati più ordini. Nell'istruzione SQL seguente viene illustrato l'utilizzo di DISTINCTROW per generare l'elenco delle società che hanno eseguito almeno un ordine, ignorando i dettagli relativi agli ordini:  SELECT DISTINCTROW NomeSocietà FROM Clienti INNER JOIN Ordini ON Clienti.IDCliente = Ordini.IDCliente ORDER BY NomeSocietà;  Se si omette DISTINCTROW, la query restituisce più righe per ogni società che ha eseguito più di un ordine. Affinché il predicato DISTINCTROW produca un risultato, è necessario selezionare i campi di alcune ma non tutte le tabelle utilizzate dalla query. DISTINCTROW viene ignorato se la query include solo una tabella o se vengono restituiti campi di tutte le tabelle. |
| TOP *n* [PERCENT] | Restituisce un determinato numero di record che si trovano all'inizio o alla fine di un intervallo specificato mediante una proposizione ORDER BY. Nell'esempio che segue viene illustrato come ottenere i nomi dei migliori 25 studenti della classe 1994:  SELECT TOP 25 Nome, Cognome FROM Studenti WHERE AnnoLaurea = 1994 ORDER BY VotoLaurea DESC;  Se si omette la proposizione ORDER BY, la query restituirà un insieme casuale di 25 record della tabella Studenti che soddisfa la proposizione WHERE.  Il predicato TOP non consente di scegliere tra valori uguali. Facendo riferimento all'esempio precedente, se il venticinquesimo e il ventiseiesimo voto di laurea coincidono, la query restituirà 26 record.  È inoltre possibile utilizzare la parola riservata PERCENT per restituire una determinata percentuale di record che si trovano all'inizio o alla fine di un intervallo specificato mediante una proposizione ORDER BY. Nell'esempio che segue viene illustrato come ottenere il nome del 10 % dei peggiori studenti della classe:  SELECT TOP 10 PERCENT Nome, Cognome FROM Studenti WHERE AnnoLaurea = 1994  ORDER BY VotoLaurea ASC;  Mediante il predicato ASC è possibile specificare la restituzione dei valori che si trovano alla fine. Dopo TOP deve essere indicato un valore **Integer** senza segno.  L'utilizzo del predicato TOP non influenza l'aggiornabilità della query. |
| *tabella* | Il nome della tabella da cui vengono recuperati i record. |

Nel seguente esempio si crea una query che crea un join tra le tabelle Clienti e Ordini sul campo ID cliente. La tabella Clienti non contiene campi ID cliente duplicati, mentre la tabella Ordini li contiene in quanto a ciascun cliente possono corrispondere molti ordini. L'utilizzo di DISTINCTROW produce un elenco di società a cui corrisponde almeno un ordine che non include però i dettagli relativi a tali ordini.

SELECT DISTINCTROW NomeSocietà FROM Clienti

INNER JOIN Ordini ON Clienti.IDCliente = Ordini.IDCliente ORDER BY NomeSocietà;

* **Proposizione WHERE**

Specifica quali record delle tabelle elencate nella proposizione FROM sono influenzati dalle istruzioni SELECT, UPDATE o DELETE.

##### Sintassi

SELECT *elencocampi*  
FROM *espressionetabella*  
WHERE *criteri*

Un'istruzione SELECT contenente una proposizione WHERE è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *elencocampi* | Il nome del campo o dei campi da recuperare insieme agli alias dei nomi di campo, ai predicati di selezione (ALL, DISTINCT, DISTINCTROW o TOP) o altre opzioni dell'istruzione SELECT. |
| *espressionetabella* | Il nome della tabella o delle tabelle di cui recuperare i dati. |
| *criteri* | Un'espressione che i record devono soddisfare per essere inclusi nei risultati della query. |

##### Osservazioni

Il modulo di gestione di database Microsoft Jet seleziona i record che soddisfano le condizioni elencate nella proposizione WHERE. Se non viene specificata una proposizione WHERE, la query restituirà tutte le righe della tabella. Se invece nella query vengono specificate più tabelle senza includere una proposizione WHERE o JOIN, la query genera un prodotto cartesiano delle tabelle.

La proposizione WHERE è facoltativa ma, se inclusa, segue la proposizione FROM. Ad esempio, è possibile selezionare tutti gli impiegati del reparto vendite (WHERE Reparto = 'Vendite') o tutti i clienti tra i 18 e i 30 anni (WHERE Età Between 18 And 30).

La proposizione WHERE è simile alla proposizione HAVING. La proposizione WHERE determina i record da selezionare. Analogamente, la proposizione HAVING determina i record da visualizzare dopo che questi sono stati raggruppati mediante la proposizione GROUP BY.

Mediante la proposizione WHERE è possibile eliminare i record che si desidera escludere dal raggruppamento eseguito tramite la proposizione GROUP BY.

Utilizzare le diverse espressioni per determinare i record che l'istruzione SQL deve restituire. Ad esempio, l'istruzione SQL seguente seleziona tutti i dipendenti il cui stipendio è superiore a 21.000 euro:

SELECT Cognome, Stipendio FROM Impiegati WHERE Stipendio > 21000;

In una proposizione WHERE è possibile includere fino a 40 espressioni collegate mediante operatori logici, quali **And** e **Or**.

Se si immette il nome di un campo che contiene spazi o punteggiatura, è necessario includere il nome tra parentesi quadre ([ ]). Ad esempio, una tabella informativa sui clienti include informazioni su determinati clienti:

SELECT [Ristorante preferito del cliente]

Se si specifica l'argomento *criteri*, i valori letterali di data devono rispettare il formato statunitense, anche se si utilizza una versione non statunitense del modulo di gestione di database Microsoft® Jet. Ad esempio, la data 10 maggio 1996, che in Italia viene scritta 10/5/96, negli Stati Uniti viene invece scritta 5/10/96. I valori letterali di data devono essere inclusi tra simboli di cancelletto (#), come illustrato negli esempi che seguono.

Per individuare i record datati 10 maggio 1996 in un database italiano, utilizzare l'istruzione SQL seguente:

SELECT \* FROM Ordini WHERE DataSpedizione = #5/10/96#;

È inoltre possibile utilizzare la funzione **DateValue** che rispetta le impostazioni internazionali stabilite da Microsoft Windows®. Ad esempio, utilizzare il codice seguente per gli Stati Uniti:

SELECT \* FROM Ordini WHERE DataSpedizione = DateValue('5/10/96');

Utilizzare invece questo codice per l'Italia:

SELECT \* FROM Ordini WHERE DataSpedizione = DateValue('10/5/96');

* **Predicati Between...And**

Verifica se il valore di un'espressione è compreso in un determinato intervallo di valori. È possibile utilizzare l'operatore Between...And con le istruzioni SQL.

##### Sintassi

*espr* [**Not**] **Between** *valore1* **And** *valore2*

La sintassi dell'operatore **Between...And** è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *espr* | Espressione che identifica il campo contenente i dati che si desidera valutare. |
| *valore1*, *valore2* | Espressioni con le quali viene confrontata l'espressione *espr*. |

##### Osservazioni

Se il valore di *espr* è compreso tra *valore1* e *valore2* (inclusi), l'operatore **Between...And** restituisce **True**. In caso contrario, restituisce **False**. È possibile includere l'operatore logico **Not** per valutare la condizione opposta ovvero il caso in cui il valore di *espr* non sia compreso nell'intervallo specificato tra *valore1* e *valore2*.

È possibile utilizzare l'operatore **Between...And** per stabilire se il valore di un campo è compreso in un determinato intervallo numerico. L'esempio riportato di seguito consente di determinare se un ordine specificato è stato spedito in una località compresa in un intervallo di codici postali. Se il codice postale è compreso tra 40100 e 40199, la funzione **IIf** restituisce “Locale”. In caso contrario, restituirà “Nonlocale”.

SELECT IIf(CodicePostale Between 40100 And 40199, “Locale”, “Nonlocale”) FROM Editori

Se il valore di *espr*, *valore1* o *valore2* è **Null**, l'operatore **Between...And** restituirà il valore **Null**.

Dal momento che i caratteri jolly, ad esempio l'asterisco (\*), vengono considerati come caratteri letterali, non possono essere utilizzati con gli operatori **Between...And**. Ad esempio, non è possibile utilizzare 4010\* e 4019\* per individuare un codice postale che inizia con un cifre comprese tra 4010 e 4019. È invece possibile aggiungere un'espressione alla query che utilizza i tre caratteri a sinistra del campo di testo e correlare l'operatore **Between...And** a tali caratteri. In alternativa, è possibile aggiungere caratteri integrativi ai valori massimo e minimo, in questo caso da 40100 a 40199.

* **Predicati In**

Determina se il valore di un'espressione equivale a uno dei valori di un elenco specificato.

##### Sintassi

*espr* [**Not**] **In(***valore1, valore2, . . .***)**

##### Osservazioni

La sintassi dell'operatore **In** è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *espr* | Espressione che identifica il campo che include i dati che si desidera valutare. |
| *valore1*, *valore2* | Espressione o elenco di espressioni con le quali deve essere confrontata *espr*. |

Se nell'elenco dei valori viene rilevata *espr*, l'operatore *In* restituisce **True**. In caso contrario viene restituito **False**. È possibile includere l'operatore logico **Not** per valutare la condizione opposta, ovvero l'assenza di *espr* nell'elenco dei valori.

Ad esempio, mediante l'operatore **In** è possibile individuare gli ordini inviati a un determinato gruppo di zone di destinazione:

SELECT \* FROM Ordini WHERE Zona In (‘RC’,’SP’,’PC’)

* **Predicati Like**

Confronta un'espressione stringa con un modello presente in un'espressione SQL.

##### Sintassi: *espressione* Like “*modello*”

La sintassi dell'operatore **Like** include le parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *espressione* | Espressione SQL utilizzata in una proposizione WHERE. |
| *modello* | Stringa o valore letterale di stringa di caratteri con cui l'*espressione* viene confrontata. |

* **Utilizzo di caratteri jolly nei confronti fra stringhe**

I criteri di ricerca incorporati offrono uno strumento versatile per eseguire confronti fra stringhe. Nella tabella seguente vengono illustrati i caratteri jolly che possono essere utilizzati con l'operatore **Like** e il numero di cifre o stringhe a cui corrispondono.

|  |  |
| --- | --- |
| **Caratteri inclusi nel *criterio*** | **Corrispondenza nell'*espressione*** |
| ? o \_ (carattere di sottolineatura) | Qualsiasi carattere singolo |
| \* o % | Zero o più caratteri |
| # | Qualsiasi cifra singola (0 - 9) |
| [*elencocaratteri*] | Qualsiasi carattere singolo incluso in *elencocaratteri* |
| [!*elencocaratteri*] | Qualsiasi carattere singolo non incluso in *elencocaratteri* |

È possibile utilizzare un gruppo di uno o più caratteri (*elencocaratteri*) inclusi tra parentesi ([ ]) per individuare la corrispondenza con qualsiasi carattere singolo dell'*espressione*. Inoltre, *elencocaratteri* può includere quasi tutti i caratteri del set di caratteri ANSI, incluse le cifre. Per poter utilizzare i caratteri speciali quali la parentesi quadra di apertura ([ ), il punto interrogativo (?), il simbolo cancelletto (#) e l'asterisco (\*) per individuare corrispondenze dirette con caratteri speciali corrispondenti, è necessario includerli tra parentesi quadre. Non è possibile utilizzare la parentesi quadra di chiusura ( ]) all'interno di un gruppo per individuare corrispondenze con caratteri corrispondenti, tuttavia è possibile utilizzarla all'esterno di un gruppo come carattere individuale.

Oltre a un elenco semplice di caratteri racchiusi tra parentesi quadre, in *elencocaratteri* è possibile specificare un intervallo di caratteri utilizzando un trattino (-) per separare l'inizio e la fine dell'intervallo. Ad esempio, utilizzando [A-Z] nel *criterio*, si avrà un riscontro se la posizione del carattere corrispondente nell'*espressione* contiene una delle lettere maiuscole comprese nell'intervallo A-Z. È possibile includere più intervalli nelle parentesi quadre senza necessità di delimitarli. Ad esempio, con [a-zA-Z0-9] vengono individuate le corrispondenze con qualsiasi carattere alfanumerico.

I caratteri jolly SQL ANSI (%) e (\_) sono supportati solo nella versione 4.x di Microsoft® Jet e nel provider Microsoft OLE DB per Jet. Tali caratteri verranno quindi considerati come variabili letterali se utilizzati tramite Microsoft Access o DAO.

Di seguito vengono illustrate altre importanti regole per i criteri di ricerca.

1. Un punto esclamativo (!) all'inizio di *elencocaratteri* indica che la corrispondenza si verifica solo se nell'*espressione* sono presenti caratteri diversi da quelli inclusi in *elencocaratteri*. Se utilizzato fuori delle parentesi quadre, il punto esclamativo individua una corrispondenza con un carattere punto esclamativo.
2. Per fare in modo che il trattino (-) individui una corrispondenza con un trattino, è necessario utilizzarlo all'inizio (dopo l'eventuale punto esclamativo) o alla fine di *elencocaratteri*. Se viene utilizzato in un'altra posizione, il trattino indicherà un intervallo di caratteri ANSI.
3. Quando si specifica un intervallo di caratteri, questi ultimi devono essere indicati in ordine ascendente (A-Z o 0-100). [A-Z] è un criterio valido, mentre [Z-A] non lo è.
4. La sequenza di caratteri [ ] viene ignorata, perché considerata come una stringa a lunghezza zero (“”).

##### Osservazioni

È possibile utilizzare l'operatore **Like** per trovare i valori di un campo corrispondenti al modello specificato. Come *modello* può essere specificato il valore completo (ad esempio, Like “Zaffaroni”) oppure è possibile utilizzare i caratteri jolly per individuare un intervallo di valori (ad esempio, Like “Za\*”).

In un'espressione, mediante l'operatore **Like** è possibile confrontare un valore di campo con un'espressione stringa. Se ad esempio si immette Like “C\*” in una query SQL, verranno restituiti tutti i valori di campo che iniziano con la lettera C. In una query con parametri è possibile richiedere all'utente di specificare il modello per la ricerca.

Nell'esempio seguente vengono restituiti i dati che iniziano con la lettera P seguita da una lettera compresa tra A e F e da tre cifre:

Like “P[A-F]###”

Nella tabella riportata di seguito viene illustrato l'utilizzo dell'operatore **Like** per provare le espressioni per modelli diversi.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo di corrispondenza** | **Modello** | **Corrispondenza (viene restituito True)** | **Nessuna corrispondenza (viene restituito False)** |
| Più caratteri | a\*a | aa, aBa, aBBBa | aBC |
|  | \*ab\* | abc, AABB, Xab | aZb, bac |
| Carattere speciale | a[\*]a | a\*a | aaa |
| Più caratteri | ab\* | abcdefg, abc | cab, aab |
| Carattere singolo | a?a | aaa, a3a, aBa | aBBBa |
| Cifra singola | a#a | a0a, a1a, a2a | aaa, a10a |
| Intervallo di caratteri | [a-z] | f, p, j | 2, & |
| All'esterno di un intervallo | [!a-z] | 9, &, % | b, a |
| Nessuna cifra | [!0-9] | A, a, &, ~ | 0, 1, 9 |
| Combinazione | a[!b-m]# | An9, az0, a99 | abc, aj0 |

##### Esempio

Nel seguente esempio viene restituito un elenco degli impiegati il cui nome inizia con le lettere da A a D.

SELECT Cognome, Nome FROM Impiegati WHERE Cognome Like '[A-D]\*';

# Informazioni sull'utilizzo dei caratteri jolly con Microsoft Jet e SQL ANSI

I caratteri jolly vengono utilizzati come segnaposto per altri caratteri quando si specifica un valore che si desidera trovare e:

* Si conosce solo parte del valore oppure
* Si desidera trovare dei valori che iniziano con una lettera specifica o che corrispondono a una determinata struttura.

I caratteri jolly vengono utilizzati in genere per i campi con tipo di dati Testo.

|  |  |  |
| --- | --- | --- |
| **Carattere** | **Descrizione** | **Esempio** |
| \* | Corrisponde a un numero qualsiasi di caratteri. Può essere utilizzato come primo o ultimo carattere nella stringa dei caratteri. | **no\*** trova nome, note e notare |
| ? | Corrisponde a qualsiasi singolo carattere alfabetico. | **B?llo** trova ballo, bello e bollo |
| [ ] | Corrisponde a qualsiasi singolo carattere all'interno delle parentesi quadre. | **B[ae]llo** trova ballo e bello, ma non bollo |
| ! | Corrisponde a qualsiasi carattere non incluso nelle parentesi quadre. | **B[!ae]llo** trova bollo e bullo, ma non ballo e bello |
| - | Corrisponde a uno qualsiasi dei caratteri di un intervallo. È necessario specificare l'intervallo in ordine crescente (dalla A alla Z, non dalla Z alla A). | **b[a-c]d** trova bad, bbd e bcd |
| # | Corrisponde a qualsiasi singolo carattere numerico. | **1#3** trova 103, 113, 123 |

Per trovare valori di campo, record o nomi di file in un progetto o in un database di Microsoft Access che utilizza una sintassi compatibile con Microsoft SQL Server, utilizzare i seguenti caratteri jolly ANSI-92.

|  |  |  |
| --- | --- | --- |
| **Carattere** | **Descrizione** | **Esempio** |
| % | Corrisponde a un numero qualsiasi di caratteri. Può essere utilizzato come primo o ultimo carattere nella stringa dei caratteri. | **no%** trova nome, note e notare |
| \_ | Corrisponde a qualsiasi singolo carattere alfabetico. | **B\_llo** trova ballo, bello e bollo |
| [ ] | Corrisponde a qualsiasi singolo carattere all'interno delle parentesi quadre. | **B[ae]llo** trova ballo e bello, ma non bollo |
| ^ | Corrisponde a qualsiasi carattere non incluso nelle parentesi quadre. | **B[^ae]llo** trova bollo e bullo, ma non ballo e bello |
| - | Corrisponde a uno qualsiasi dei caratteri di un intervallo. È necessario specificare l'intervallo in ordine crescente (dalla A alla Z, non dalla Z alla A). | **b[a-c]d** trova bad, bbd e bcd |

**Note**

* Quando si utilizzano i caratteri jolly per altri caratteri jolly come un asterisco (\*), un punto interrogativo (?), un cancelletto (#), una parentesi quadra di apertura ([) o un trattino (-), è necessario racchiudere tra parentesi quadre l'elemento che si sta cercando. Per la ricerca di un punto esclamativo (!) o una parentesi quadra di chiusura (]), non è necessario racchiudere il carattere tra parentesi quadre.

Per cercare ad esempio un punto interrogativo, digitare **[?]** nella finestra di dialogo **Trova**. Se si sta cercando contemporaneamente un trattino e altri caratteri, inserire il trattino prima o dopo tutti gli altri caratteri all'interno delle parentesi quadre. Se tuttavia dopo la parentesi quadra di apertura c'è un punto esclamativo (!), inserire il trattino dopo di esso.

* Non è possibile ricercare contemporaneamente le parentesi quadre di apertura e chiusura ([ ]) perché in Microsoft Access questa combinazione viene interpretata come una stringa a lunghezza zero. È necessario racchiudere a loro volta le parentesi quadre di apertura e chiusura tra parentesi ([[ ]]).
* **Proposizione ORDER BY**

Ordina i record risultanti di una query in base a uno o più campi specificati in ordine crescente o decrescente.

##### Sintassi

SELECT *elencocampi*  
FROM *tabella*  
WHERE *criteriselezione*  
[ORDER BY *campo1* [ASC | DESC ][, *campo2* [ASC | DESC ]][, ...]]]

Un'istruzione SELECT che contiene una proposizione ORDER BY è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *elencocampi* | Il nome del campo o dei campi da recuperare insieme agli alias dei nomi di campo, alle funzioni di aggregazione SQL, ai predicati di selezione (ALL, DISTINCT, DISTINCTROW o TOP) e altre opzioni dell'istruzione SELECT. |
| *tabella* | Il nome della tabella di cui vengono recuperati i record. |
| *criteriselezione* | Criteri di selezione. |
| *campo1*, *campo2* | I nomi dei campi in base ai quali ordinare i record. |

##### Osservazioni

La proposizione ORDER BY è facoltativa, tuttavia se si desidera che i dati vengano visualizzati in base a un determinato ordine, è necessario utilizzarla.

Il criterio di ordinamento predefinito è quello crescente (da A a Z, da 0 a 9). In entrambi gli esempi seguenti vengono ordinati i nomi degli impiegati in base al cognome:

SELECT Cognome, Nome FROM Impiegati ORDER BY Cognome;

SELECT Cognome, Nome FROM Impiegati ORDER BY Cognome ASC;

Per eseguire l'ordinamento in senso decrescente (da Z a A, da 9 a 0), aggiungere la parola riservata DESC alla fine dei campi che si desidera ordinare in senso decrescente. Nell'esempio seguente, gli stipendi vengono selezionati e ordinati in senso decrescente:

SELECT Cognome, Stipendio FROM Impiegati ORDER BY Stipendio DESC, Cognome;

Se nella proposizione ORDER BY viene specificato un campo contenente dati Memo o Oggetto OLE, si verifica un errore.

La proposizione ORDER BY in genere è l'ultimo elemento di un'istruzione SQL.

È possibile includere ulteriori campi nella proposizione ORDER BY. I record vengono ordinati prima in base al campo elencato dopo ORDER BY; quindi, i record che contengono valori identici in tale campo vengono ordinati in base al valore del secondo campo elencato, e così via.

* **Proposizione GROUP BY**

Riunisce in un singolo record tutti i record che presentano valori uguali nell'elenco di campi specificato. Se nell'istruzione SELECT viene inclusa una funzione di aggregazione SQL, ad esempio **Somma** o **Conteggio**, per ogni record viene creato un valore di riepilogo.

##### Sintassi

SELECT *elencocampi*  
FROM *tabella*  
WHERE *criteri*  
[GROUP BY *elencocampigruppo*]

Un'istruzione SELECT che include una proposizione GROUP BY è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *elencocampi* | Nome del campo o dei campi da recuperare insieme agli alias dei nomi di campo, alle funzioni di aggregazione SQL, ai predicati di selezione (ALL, DISTINCT, DISTINCTROW o TOP) e altre opzioni dell'istruzione SELECT. |
| *tabella* | Nome della tabella da cui vengono recuperati i record. |
| *criteri* | Criteri di selezione. |
| *elencocampigruppo* | Nomi di un numero massimo di 10 campi utilizzati per raggruppare i record. L'ordine dei nomi dei campi in *elencocampigruppo* determina i livelli di raggruppamento, dal livello più alto a quello più basso. |

##### Osservazioni

La proposizione GROUP BY è facoltativa.

I valori di riepilogo vengono omessi se l'istruzione SELECT non include funzioni di aggregazione SQL.

I valori **Null** inclusi nei campi GROUP BY vengono raggruppati e non sono omessi. Tuttavia, i valori **Null** non vengono valutati dalle funzioni di aggregazione SQL.

La proposizione WHERE consente di ignorare le righe che si desidera escludere dall'operazione di raggruppamento, mentre la proposizione HAVING consente di filtrare i record dopo il raggruppamento.

Se non include dati di tipo Memo o Oggetto OLE, un campo dell'elenco di campi GROUP BY può far riferimento a qualsiasi campo incluso in una tabella elencata nella proposizione FROM, anche se il campo non è incluso nell'istruzione SELECT, a patto che tale istruzione includa almeno una funzione di aggregazione SQL. Il modulo di gestione di database Microsoft® Jet non consente di raggruppare campi con dati di tipo Memo o Oggetto OLE.

Tutti i campi nell'elenco di campi SELECT devono essere inclusi nella proposizione GROUP BY oppure devono essere inclusi come argomenti della funzione di aggregazione SQL.

* **Proposizione HAVING**

Specifica quale gruppo di record viene visualizzato in un'istruzione SELECT con la proposizione GROUP BY. Dopo il raggruppamento tramite GROUP BY, la proposizione HAVING consente di visualizzare i record raggruppati dalla proposizione GROUP BY che soddisfano le condizioni della proposizione HAVING.

##### Sintassi

SELECT *elencocampi*  
FROM *tabella*  
WHERE *criteriselezione*  
GROUP BY *elencocampigruppo*  
[HAVING *criterigruppo*]

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *elencocampi* | Nome del campo o dei campi da recuperare insieme agli alias dei nomi di campo, alle funzioni di aggregazione SQL, ai predicati di selezione (ALL, DISTINCT, DISTINCTROW o TOP) e altre opzioni dell'istruzione SELECT. |
| *tabella* | Nome della tabella da cui vengono recuperati i record. |
| *criteriselezione* | Criteri di selezione |
| *elencocampigruppo* | Nomi di un numero massimo di 10 campi utilizzati per raggruppare i record. L'ordine dei nomi dei campi in *elencocampigruppo* determina i livelli di raggruppamento, dal livello più alto a quello più basso. |
| *criterigruppo* | Espressione che determina i record raggruppati da visualizzare. |

##### Osservazioni

La proposizione HAVING è facoltativa.

La proposizione HAVING è simile alla proposizione WHERE, che consente di stabilire i record selezionati. La proposizione HAVING consente di specificare i record visualizzati dopo che questi sono stati raggruppati dalla proposizione GROUP BY:

SELECT IDCategoria, Somma(Scorte) FROM Prodotti GROUP BY IDCategoria HAVING Somma(Scorte) > 100 And Like "BOS\*";

In una proposizione HAVING è possibile includere fino a 40 espressioni collegate mediante operatori logici, quali **And** e **Or**.

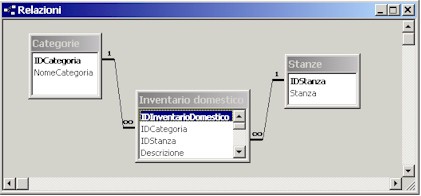
* **Join**

# Query "Join" da più tabelle con Microsoft Access

<http://www.mc2.it/query-join-da-piu-tabelle-access.html>

Due tabelle possono essere collegate tra loro tramite un oggetto, detto "Relazione". Tale relazione lega un campo chiave primaria di una delle 2 tabelle con un campo corrispondente presente all'interno dell'altra, detto anche chiave esterna.  
  
A seconda del modo in cui le tabelle vengono unite, è possibile creare i seguenti 3 tipi di JOIN:

Il primo tipo è **INNER JOIN (Join interno)**. Questo tipo consente la selezione di un record (per essere incluso in una Query di selezione), soltanto quando nelle 2 tabelle esiste l'esatta corrispondenza tra i campi posti in relazione.  
  
Esempio: Database Inventario Domestico. Quella che segue è la finestra delle Relazioni. Le tabelle sono 3: Inventario domestico, Categorie e Stanze.



Costruiamo una Query di selezione semplice sulle tabelle 'Inventario domestico' e 'Stanze', le quali sono in relazione "uno a molti" tra di loro e collegate dal campo 'IdStanza'. La query include tutti i campi della prima tabella e il campo 'Stanza' (la descrizione della stanza) della seconda.

****

Se ora proviamo ad aprire la query in modalità 'Visualizzazione Foglio Dati' notiamo che i records estratti sono 11:

****

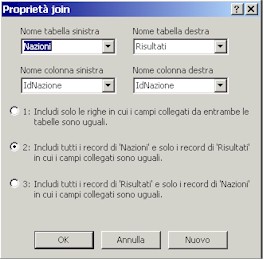
mentre se apriamo la tabella 'Stanze' ci accorgiamo che i records sono 12, come mai ? In effetti la Cucina non ha al momento alcuna registrazione di inventario, per cui il relativo record non è stato estratto perchè non soddisfa la condizione di INNER JOIN (tutti i records con esatta corrispondenza nelle 2 tabelle...)

****

Il secondo tipo è **LEFT OUTER JOIN (join a sinistra)**, vediamo l'esempio:  
  
Database Olimpiadi. Vogliamo ottenere un prospetto contenente tutte le Nazioni del mondo e a fianco di ciascuna di esse l'indicazione del numero di medaglie vinte nel corso di tutte le edizioni.  
  
Faremo quindi ancora una operazione di JOIN, e più precisamente tra la tabella Nazioni e la tabella dei Risultati, il campo che le lega è il campo IdNazione.    
Se però eseguissimo una INNER JOIN otterremmo un elenco che escluderebbe tutte le Nazioni che non hanno mai vinto una medaglia d'oro, perchè la INNER JOIN prevede la presenza del valore del campo relazionato in entrembe le tabelle; eseguiremo invece una OUTER JOIN di tipo LEFT, ponendo "sulla sinistra" la tabella delle Nazioni per ottenere l'elenco completo delle Nazioni e naturalmente otteremo molti valori uguali a zero.

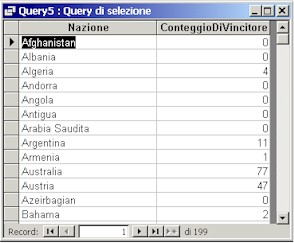


In modalità visualizzazione struttura selezioniamo 'Totali' quindi la Formula 'Raggruppamento' per il campo Nazione e la Formula 'Conteggio' per il campo Vincitore (vedi anche tutorial '[Query di raggruppamento'](http://www.mc2.it/query-raggruppamento-access.html)).  
Ora facciamo doppio click sulla linea che congiunge le 2 tabelle, nella parte alta della finestra: appare la finestra di dialogo 'Proprietà join'.  Notiamo che la tabella di sinistra è Nazioni, mentre quella di destra è 'Risultati'.   
  
Definire una JOIN di tipo LEFT OUTER JOIN equivale a selezionare l'opzione 2 e cioè 'Includi tutti i record di 'Nazioni' e solo i record di 'Risultati' in cui i campi collegati sono uguali.

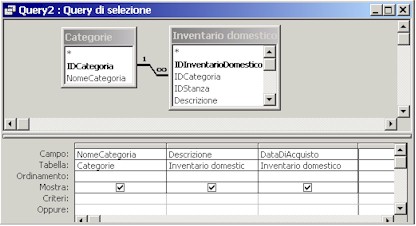


Facciamo click su 'OK' e notiamo ancora che la linea che collega le 2 tabelle presenta una freccia rivolta verso destra: questo sta ad indicare è stato impostato un LEFT JOIN (la regola è inversa). Se avessimo impostato '3' la freccia sarebbe rivolta verso sinistra ma in quel caso si tratterebbe di RIGTH JOIN (vedi esempio più avanti).

Ecco comunque il risultato della nostra query di raggruppamento (che è pur sempre una query di selezione) con definizione di criterio LEFT OUTER JOIN : le nazioni sono 199 e sono elencate indipendentemente dal fatto che abbiano mai avuto medaglie oppure no.



Il terzo tipo è **RIGHT OUTER JOIN (join a destra)**, vediamo l'esempio:  
  
Torniamo al database 'Inventario Domestico'. Costruiamo una nuova query di selezione questa volta sulle tabella 'Categoria' e sulla tabella 'Inventario domestico', selezionando il campo 'NomeCategoria' (da Categorie), quindi il campo 'Descrizione' (da tabella 'Inventario domestico') e infine 'DataDiAcquisto' (da tabella 'Inventario domestico')



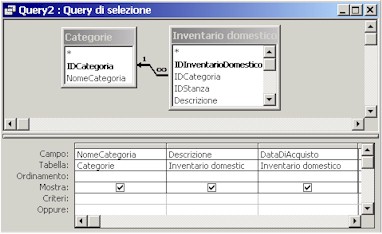
Non impostiamo per ora alcun criterio di JOIN, quello predefinito è sempre 'INNER JOIN' o join interno, corrispondente alla selezione 1 della finestra di dialogo 'Proprietà join': "Includi solo le righe in cui i campi collegati da entrambe le tabelle sono uguali".  
Visualizziamo i dati della query: i records estratti sono 11. Quello che vogliamo porre in evidenza in questo caso è che  questo tipo di query non visualizzerebbe  eventuali registrazioni (records) nelle quali ci si è dimenticati di impostare la categoria!   
Quindi vorremmo una query che ci mostrasse invece proprio quali records di inventario non trovano una corrispondenza nella tabella delle Categorie.



Per giungere al nostro obiettivo non dobbiamo far altro che tornare nell'ambiente di definizione struttura della nostra query, fare doppio click sulla linea che congiunge le 2 tabelle e far comparire la finestra di dialogo 'Proprietà join': impostiamo la selezione questa volta sul numero '3', cioè: "Includi tutti i record di 'Inventario domestico' e solo i record di 'Categorie' in cui i campi collegati sono uguali".  
Abbiamo appena impostato come criterio di Join, il criterio RIGHT OUTER JOIN  (join a destra).



Facciamo click su 'OK'. Notiamo che adesso la freccia è rivolta verso sinistra **(join a destra)**.



Non ci resta ora che **visualizzare nuovamente i risultati della query** ed ecco che **"spiccano" i records** che **non sono abbinati ad alcuna categoria**: in questo caso la registrazione 'Lavastoviglie', acquistata il 1/12/1995, che non appariva prima!



* **Operazione INNER JOIN**

Riunisce record da due tabelle in cui vi siano valori corrispondenti in un campo comune.

##### Sintassi

FROM *tabella1* INNER JOIN *tabella2* ON *tabella1*.*campo1* *operatorediconfronto tabella2*.*campo2*

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *tabella1*, *tabella2* | Nomi delle tabelle dalle quali vengono raggruppati i record. |
| *campo1*, *campo2* | Nomi dei campi che vengono collegati. Se non includono dati numerici, i campi devono includere lo stesso tipo di dati e contenere lo stesso genere di dati, tuttavia non è necessario che utilizzino lo stesso nome. |
| *operatorediconfronto* | Operatore di confronto relazionale: "=," "<," ">," "<=," ">=," o "<>." |

##### Osservazioni

È possibile utilizzare un'operazione INNER JOIN in qualsiasi proposizione FROM. Si tratta del tipo di join più comune. In un join interno vengono collegati i record di due tabelle che contengono valori corrispondenti in un campo comune a entrambe le tabelle.

È possibile utilizzare INNER JOIN con le tabelle Dipartimenti e Impiegati per selezionare tutti gli impiegati di un dipartimento. Al contrario, per selezionare tutti i dipartimenti (anche se ad alcuni non sono stati assegnati impiegati) o tutti gli impiegati (anche se alcuni non sono assegnati a un dipartimento), è possibile utilizzare un operazione LEFT JOIN o RIGHT JOIN per creare un join esterno.

Viene generato un errore tutte le volte che si tenta di collegare campi che contengono dati di tipo Oggetto OLE o Memo. È possibile collegare due campi numerici di tipo simile. Ad esempio, è possibile collegare i campi Contatore e Long dal momento che si tratta di tipi simili. Tuttavia, non è possibile collegare campi di tipo Single e Double.

Nell'esempio seguente viene mostrato come collegare le tabelle Categorie e Prodotti in base al campo IDCategoria:

SELECT NomeCategoria, NomeProdotto FROM Categorie

INNER JOIN Prodotti ON Categorie.IDCategoria = Prodotti.IDCategoria;

In questo esempio, IDCategoria indica il campo collegato che tuttavia non viene incluso nella query di output dal momento che non è stato incluso nell'istruzione SELECT. Per includere il campo collegato, inserire il nome del campo nell'istruzione SELECT, in questo caso Categorie.IDCategoria.

Tramite la sintassi seguente è possibile collegare diverse proposizioni ON in un'istruzione JOIN:

SELECT *campi*  
FROM *tabella1* INNER JOIN *tabella2*  
ON *tabella1*.*campo1* *operatorediconfronto* *tabella2*.*campo1* AND  
ON *tabella1*.*campo2* *operatorediconfronto* *tabella2*.*campo2*) OR  
ON *tabella1*.*campo3* *operatorediconfronto* *tabella2*.*campo3*)];

È inoltre possibile nidificare un'istruzione JOIN utilizzando la seguente sintassi:

SELECT *campi*  
FROM *tabella1* INNER JOIN  
(*tabella2* INNER JOIN [( ]*tabella3*  
[INNER JOIN [( ]*tabellax* [INNER JOIN ...)]   
ON *tabella3*.*campo3* *operatorediconfronto* *tabellax*.*campox*)]  
ON *tabella2*.*campo2* *operatorediconfronto* *tabella3*.*campo3*)   
ON *tabella1*.*campo1* *operatorediconfronto* *tabella2*.*campo2*;

**È possibile** nidificare un'istruzione LEFT JOIN o un'istruzione RIGHT JOIN in un'istruzione INNER JOIN, tuttavia **non è possibile** nidificare un'istruzione INNER JOIN in istruzioni LEFT JOIN e RIGHT JOIN.

##### Esempio

##### Nell’ esempio si creano due equi-join(Join in cui i record di due tabelle vengono combinati e selezionati solo quando ci sono valori uguali nei campi tra cui è stato creato il join. Viene anche definito join interno): uno tra la tabella Dettagli ordini e la tabella Ordini e un altro tra la tabella Ordini e la tabella Impiegati. Ciò è necessario perché la tabella Impiegati non contiene dati relativi alle vendite e la tabella Dettagli ordini non contiene dati relativi agli impiegati. La query produce un elenco degli impiegati e il totale delle vendite da essi realizzate.

SELECT DISTINCTROW Sum(PrezzoUnitario \* Quantità) AS Vendite, (Nome & Chr(32) & Cognome) AS Nome

FROM Impiegati INNER JOIN(Ordini INNER JOIN [Dettagli ordini] ON [Dettagli ordini].IDOrdine = Ordini.IDOrdine) ON Ordini.IDImpiegato = Impiegati.IDImpiegato

GROUP BY (Nome & Chr(32) & Cognome);

* **Istruzione LEFT JOIN, RIGHT JOIN**

Combinano record della tabella di origine se utilizzate in una proposizione FROM.

##### Sintassi

FROM *tabella1* [ LEFT | RIGHT ] JOIN *tabella2*  
ON *tabella1.campo1* *operatorediconfronto tabella2.campo2*

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *tabella1*, *tabella2* | I nomi delle tabelle di cui vengono combinati i record. |
| *campo1*, *campo2* | I nomi dei campi che vengono collegati. I campi devono essere dello stesso tipo di dati e includere lo stesso genere di dati, tuttavia non è necessario che utilizzino lo stesso nome. |
| *operatorediconfronto* | Un operatore di confronto relazionale: "=," "<," ">," "<=," ">=," o "<>." |

##### Osservazioni

Mediante un'operazione LEFT JOIN è possibile creare un join esterno sinistro, che include tutti record della prima tabella (quella di sinistra) di due tabelle, anche se non vi sono valori corrispondenti ai record della seconda tabella (quella di destra).

Definizione di join esterno sinistro: Join esterno in cui tutti i record del lato sinistro dell'operazione LEFT JOIN nell'istruzione SQL della query vengono selezionati, anche se non ci sono valori corrispondenti nel campo con cui è stato creato il join nella tabella di destra. I record della tabella di destra vengono combinati con quelli della tabella di sinistra solo quando nei campi tra cui è stato creato il join ci sono valori corrispondenti. Quando un record di sinistra non ha alcun record corrispondente, sul lato destro viene creato un join con una riga di valori Null.

Mediante un'operazione RIGHT JOIN è possibile creare un join esterno destro, che include tutti i record della seconda tabella (quella di destra) di due tabelle, anche se non vi sono valori corrispondenti ai record della prima tabella (quella di sinistra).

Definizione di join esterno destro: Join esterno in cui tutti i record del lato destro dell'operazione RIGHT JOIN nell'istruzione SQL della query vengono aggiunti all'oggetto Recordset, anche se non ci sono valori corrispondenti nel campo della tabella di sinistra con cui è stato creato il join. I record della tabella di sinistra vengono combinati con quelli della tabella di destra solo quando ci sono valori corrispondenti nei campi tra cui è stato creato il join.

È ad esempio possibile utilizzare l'operazione LEFT JOIN con le tabelle Reparti (sinistra) e Impiegati (destra) per selezionare tutti i reparti, inclusi quelli a cui non sono stati assegnati impiegati. Per selezionare tutti gli impiegati, inclusi quelli non assegnati ad alcun reparto, dovrà essere utilizzata l'operazione RIGHT JOIN.

Nell'esempio seguente viene illustrato il collegamento delle tabelle Categorie e Prodotti in base al campo IDCategoria. La query produce l'elenco di tutte le categorie, incluse quelle che non contengono prodotti:

SELECT NomeCategoria, NomeProdotto FROM Categorie

LEFT JOIN Prodotti ON Categorie.IDCategoria = Prodotti.IDCategoria;

In questo esempio, il campo collegato è IDCategoria, che tuttavia non viene incluso nei risultati della query perché non è stato incluso nell'istruzione SELECT. Se si desidera includere nei risultati il campo collegato, immettere il nome del campo nell'istruzione SELECT, in questo caso Categorie.IDCategoria.

**Note**

Per creare una query che includa solo i record in cui i dati dei campi collegati sono uguali, utilizzare un'operazione INNER JOIN.

* È possibile nidificare un'operazione LEFT JOIN o RIGHT JOIN in un'operazione INNER JOIN, ma non è possibile nidificare un'operazione INNER JOIN in un'operazione LEFT JOIN o RIGHT JOIN. Per ulteriori informazioni sulla nidificazione tra diverse operazioni di join, vedere la trattazione della nidificazione nell'argomento relativo all'operazione INNER JOIN.
* È possibile collegare più proposizioni ON. Per ulteriori informazioni, vedere la trattazione del collegamento di proposizioni nell'argomento relativo all'operazione INNER JOIN.

Quando si tenta di collegare campi che contengono dati Oggetto OLE o Memo, si verifica un errore.

**Esempio**

Nell’ esempio si presuppone l'esistenza in una tabella Impiegati dei campi ipotetici Nome reparto e ID reparto e sono selezionati tutti i reparti, compresi quelli senza impiegati.

SELECT [Nome reparto], Nome & Chr(32) & Cognome AS Nome FROM Reparti

LEFT JOIN Impiegati ON Reparti.[ID reparto] = Impiegati.[ID reparto]

ORDER BY [Nome reparto];

* **Istruzione UNION**

Crea una query di unione, che combina i risultati di due o più tabelle o query indipendenti.

##### Sintassi

[TABLE] *query1* UNION [ALL] [TABLE] *query2* [UNION [ALL] [TABLE] *queryn* [ ... ]]

O in altra forma,

enunciato-di -selezione

UNION [ALL]

enunciato-di -selezione

[ORDER BY {nome-colonna|nome-colonna}]

[ASC | DESC]

L'operazione UNION è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *query1-n* | Un'istruzione SELECT, il nome di una query memorizzata o il nome di una tabella memorizzata preceduto dalla parola chiave TABLE. |

##### Osservazioni

È possibile unire i risultati di due o più query, tabelle e istruzioni SELECT, in qualsiasi combinazione, con una singola operazione UNION. Nell'esempio che segue vengono unite una tabella esistente denominata Nuovi conti e un'istruzione SELECT:

TABLE [Nuovi conti] UNION ALL SELECT \* FROM Clienti WHERE ImportoOrdini > 1000;

Per impostazione predefinita, quando si utilizza l'operazione UNION, non vengono restituiti record duplicati. Tuttavia, se si include il predicato ALL, è possibile ottenere la restituzione di tutti i record, rendendo più rapida l'esecuzione della query.

Tutte le query di un'operazione UNION devono richiedere lo stesso numero di campi ma non è necessario che i campi abbiano la stessa dimensione o lo stesso tipo di dati.

Utilizzare gli alias solo nella prima istruzione SELECT, dal momento che vengono ignorati nelle altre. Nella proposizione ORDER BY fare riferimento ai campi in base al nome specificato nella prima istruzione SELECT.

**Note**

* È possibile utilizzare una proposizione GROUP BY o HAVING in ogni argomento della *query* per raggruppare i dati restituiti.
* Mediante una proposizione ORDER BY inclusa alla fine dell'ultimo argomento *query* è possibile visualizzare i dati restituiti nell'ordine specificato.

**Esempi**

Costruzione di una lista contenente il nome e l’indirizzo e-mail da due tabelle Employees e Customers che hanno la stessa struttura:

SELECT [Last Name], [First Name], Email

FROM tblCustomers

UNION

SELECT [Last Name], [First Name], Email

FROM tblEmployees

Per recuperare tutti i dati dalle due tabelle utilizzare la parola chiave TABLE:

TABLE tblCustomers

UNION

TABLE tblEmployees

L’operatore UNION non visualizzerà i records duplicati; per includere tutti I records compresi quelli duplicati utilizzare il predicato ALL:

SELECT [Last Name], [First Name], Email

FROM tblCustomers

UNION ALL

SELECT [Last Name], [First Name], Email

FROM tblEmployees

**Modifica dei dati (DML)**

* **Istruzione SELECT...INTO**

Crea una query di creazione tabella.

##### Sintassi

SELECT *campo1*[, *campo2*[, ...]] INTO *nuovatabella* [IN *databaseesterno*]  
FROM *origine*

L'istruzione SELECT...INTO è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *campo1*, *campo2* | Il nome dei campi da copiare nella nuova tabella. |
| *nuovatabella* | Il nome della tabella da creare, che deve rispettare le convenzioni di denominazione standard. Se si utilizza per *nuovatabella* lo stesso nome già utilizzato per una tabella esistente, si verifica un errore intercettabile. |
| *databaseesterno* | Il percorso di un database esterno. Per una descrizione del percorso, vedere la proposizione IN. |
| *origine* | Il nome della tabella esistente in cui vengono selezionati i record. Può trattarsi di una o più tabelle o di una query. |

##### Osservazioni

Mediante le query di creazione tabella è possibile archiviare record, eseguire copie di backup delle tabelle oppure eseguire copie da esportare in un altro database o da utilizzare come base per report in cui vengono visualizzati i dati relativi a un determinato periodo di tempo. Ad esempio, è possibile generare un report sulle vendite mensili per zona, eseguendo la stessa query di creazione tabella ogni mese.

**Note**

* È consigliabile definire una chiave primaria per la nuova tabella. Durante la creazione della tabella, i campi della nuova tabella ereditano il tipo di dati e la dimensione di ogni campo delle tabelle sottostanti della query, tuttavia non vengono trasferiti gli altri campi o proprietà delle tabelle.
* Per aggiungere dati a una tabella esistente, utilizzare l'istruzione INSERT INTO invece di creare una query di accodamento.
* Per individuare i record che verranno selezionati prima di eseguire la query di creazione tabella, esaminare innanzitutto i risultati di un'istruzione SELECT che utilizza gli stessi criteri di selezione.

##### Esempi

Nel seguente esempio si selezionano tutti i record della tabella Impiegati e si copiano in una nuova tabella denominata Backup impiegati.

SELECT Impiegati.\* INTO [Backup Imp] FROM Impiegati;

* **Istruzione INSERT...INTO**

Aggiunge uno o più record a una tabella. Viene inoltre definita query di accodamento.

##### Sintassi

Query di accodamento a più record:

INSERT INTO *destinazione* [(*campo1*[, *campo2*[, ...]])] [IN *databaseesterno*]  
SELECT [*origine*.]*campo1*[, *campo2*[, ...]  
FROM *espressionetabella*

Query di accodamento a record singolo:

INSERT INTO *destinazione* [(*campo1*[, *campo2*[, ...]])]  
VALUES (*valore1*[, *valore2*[, ...])

L'istruzione INSERT INTO è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *destinazione* | Nome della tabella o della query a cui accodare i record. |
| *campo1*, *campo2* | Nomi dei campi a cui accodare i dati, se segue un argomento *destinazione*, oppure i nomi dei campi da cui ottenere i dati, se segue un argomento *origine*. |
| *databaseesterno* | Percorso di un database esterno. Per una descrizione del percorso, vedere la proposizione IN. |
| *origine* | Nome della tabella o della query da cui copiare i record. |
| *espressionetabella* | Nome della tabella o delle tabelle da cui vengono inseriti i dati. Questo argomento può essere il nome di una singola tabella, il nome di una query salvata o il risultato derivante dalle operazioni INNER JOIN, LEFT JOIN o RIGHT JOIN. |
| *valore1*, *valore2* | Valori da inserire nei campi specifici del nuovo record. Ogni valore viene inserito nel campo che corrisponde alla posizione del valore nell'elenco: *valore1* viene inserito nel *campo1* del nuovo record, *valore2* nel *campo2* e così via. È necessario separare i valori con una virgola e includere i campi di testo tra virgolette singole (' '). |

##### Osservazioni

È possibile utilizzare l'istruzione INSERT INTO per aggiungere un record singolo a una tabella utilizzando la sintassi della query di accodamento di una query a record singolo, come mostrato in precedenza. In questo caso, il codice specifica il nome e il valore di ogni campo del record. È necessario specificare i valori e i campi a cui assegnare tali valori. In caso contrario, nelle colonne omesse viene inserito il valore predefinito o un valore **Null**. I record vengono aggiunti alla fine della tabella.

È inoltre possibile utilizzare INSERT INTO per accodare un set di record da un'altra tabella o query, mediante la proposizione SELECT ... FROM così come mostrato in precedenza per la sintassi della query di accodamento a più record. In questo caso, la proposizione SELECT specifica i campi da accodare alla tabella di *destinazione* specificata.

Le tabelle *origine* e *destinazione* possono fare riferimento a una query o a una tabella. Nel primo caso, il modulo di gestione di database Microsoft Jet potrà accodare record a una o a tutte le tabelle specificate dalla query.

Se la tabella di destinazione include dei campi non presenti nella tabella di origine questi verranno assegnati a **Null**. Se la tabella di origine include dei campi non presenti nella tabella di destinazione si avrà una segnalazione di errore.

L'istruzione INSERT INTO è facoltativa, ma se inclusa, precede l'istruzione SELECT.

Se la tabella di destinazione include una chiave primaria, è necessario accodare valori univoci, diversi da **Null** al campo o ai campi della chiave primaria. In caso contrario, il modulo di gestione di database Microsoft Jet non accoderà i record.

Se i record vengono accodati a una tabella con un campo Contatore e si desidera rinumerare i record accodati, non includere il campo Contatore nella query. Includere il campo solo se si desidera conservare i valori originali del campo.

Utilizzare la proposizione IN per accodare record a una tabella di un altro database.

Per creare una nuova tabella, utilizzare l'istruzione SELECT... INTO invece di una query di creazione tabella.

Per individuare i record che verranno accodati prima di eseguire la query di accodamento, eseguire una query di selezione con gli stessi criteri di selezione e visualizzarne i risultati.

Una query di accodamento consente di copiare record da una o più tabelle in altre, senza alterare la tabella che contiene i record da copiare.

Invece di accodare record esistenti da un'altra tabella, è possibile specificare il valore di ogni campo in un nuovo record unico, utilizzando la proposizione VALUES. Se si omette l'elenco dei campi, è necessario includere nella proposizione VALUES un valore relativo a ogni campo della tabella; in caso contrario l'operazione INSERT non verrà eseguita correttamente. Utilizzare un'ulteriore istruzione INSERT INTO con una proposizione VALUES per ogni record aggiuntivo che si desidera creare.

##### Esempi

Nel seguente esempio si selezionano tutti i record in una tabella ipotetica denominata Nuovi clienti e li si aggiungono alla tabella Clienti. Se non sono designate singole colonne, i nomi di colonna della tabella relativa a SELECT devono corrispondere esattamente a quelli della tabella specificata da INSERT INTO.

INSERT INTO Clienti SELECT \* FROM [Nuovi clienti];

Nel seguente esempio si crea un nuovo record nella tabella Impiegati.

INSERT INTO Impiegati (Nome, Cognome, Posizione)

VALUES ('Marco', 'Fazio', 'In addestramento');

* **Istruzione UPDATE**

Crea una query di aggiornamento che modifica i valori dei campi di una tabella specificata in base ai criteri specificati.

##### Sintassi

UPDATE *tabella*  
SET *nuovovalore*  
WHERE *criteri*;

L'istruzione UPDATE è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *tabella* | Il nome della tabella che contiene i dati da modificare. |
| *nuovovalore* | Un'espressione che determina il valore da inserire in un determinato campo dei record aggiornati. |
| *criteri* | Un'espressione che determina i record da aggiornare. Vengono aggiornati solo i record che soddisfano l'espressione. |

##### Osservazioni

L'istruzione UPDATE è particolarmente utile quando si desidera modificare numerosi record o quando i record da modificare sono inclusi in più tabelle.

È possibile modificare molti campi contemporaneamente. Nell'esempio che segue, i valori della tabella Importo ordini vengono aumentati del 10 percento mentre quelli della tabella Trasporto vengono aumentati del 3 percento in relazione alle spedizioni al Regno Unito:

UPDATE Ordini

SET ImportoOrdini = ImportoOrdini \* 1.1, Trasporto = Trasporto \* 1.03

WHERE PaeseDestinatario = 'UK';

**Importante**

* L'istruzione UPDATE non genera un set di risultati. Inoltre, dopo aver aggiornato i record mediante una query di aggiornamento, non è possibile annullare l'operazione. Se si desidera sapere quali record sono stati aggiornati, esaminare i risultati di una query di selezione che utilizza gli stessi criteri e quindi eseguire la query di aggiornamento.
* Mantenere sempre le copie di backup dei dati. Se vengono aggiornati i record sbagliati, è possibile recuperarli da tali copie di backup.
* **Istruzione DELETE**

Crea una query di eliminazione che consente di rimuovere record da una o più tabelle elencate nella proposizione FROM che soddisfano i criteri indicati nella proposizione WHERE.

##### Sintassi

DELETE [*tabella*.\*]  
FROM *tabella*  
WHERE *criteri*

L'istruzione DELETE è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *tabella* | Nome facoltativo della tabella da cui vengono eliminati i record. |
| *tabella* | Nome della tabella da cui vengono eliminati i record. |
| *criteri* | Espressione che consente di definire i record da eliminare. |

##### Osservazioni

L'istruzione DELETE è particolarmente utile quando è necessario eliminare numerosi record.

Per eliminare un'intera tabella dal database è possibile utilizzare il metodo Execute con un'istruzione DROP. Se si elimina la tabella, tuttavia, verrà eliminata anche la relativa struttura. Se invece si utilizza l'istruzione DELETE vengono eliminati solo i dati e restano intatte la struttura e tutte le proprietà della tabella, ad esempio gli indici e gli attributi di campo.

È possibile utilizzare l'istruzione DELETE per eliminare record da tabelle che presentano una relazione uno-a-molti con altre tabelle. Le operazioni di eliminazione a catena consentono di eliminare record inclusi nelle tabelle correlate, contemporaneamente all'eliminazione del record corrispondente nel lato uno della relazione. Ad esempio, nella relazione tra le tabelle Clienti e Ordini, nella quale la tabella Clienti è sul lato uno della relazione e la tabella Ordini sul lato molti, specificando l'opzione di eliminazione a catena, l'eliminazione di un record dalla tabella Clienti determinerà la contemporanea eliminazione dei record corrispondenti nella tabella Ordini.

Tramite una query di eliminazione viene eliminato l'intero record e non solo i dati contenuti nei campi. Se si desidera eliminare i valori di un determinato campo, creare una query di aggiornamento per modificare i valori in **Null**.

**Importante**

* Non è possibile annullare l'operazione dopo aver eliminato i record mediante la query di eliminazione. Se si desidera conoscere in anticipo quali record verranno eliminati, controllare i risultati di una query di selezione che utilizza gli stessi criteri, quindi eseguire la query di eliminazione.
* Mantenere sempre copie di backup dei dati. Se vengono eliminati i record sbagliati, sarà possibile recuperarli da tali copie di backup.

**Procedure**

* **Istruzione CREATE PROCEDURE**

Crea una stored procedure.

##### Sintassi

CREATE PROCEDURE *procedura*  
[*param1 tipodidati*[, *param2 tipodidati*[, ...]] AS istruzionesql

L'istruzione CREATE PROCEDURE è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *procedura* | Nome della procedura. Deve rispettare le convenzioni di denominazione standard. |
| *param1*, *param2* | Da un minimo di 1 a un massimo di 225 nomi di campo o parametri. Ad esempio:  CREATE PROCEDURE Vendite\_Per\_Paese [Data Inizio] DataOra, [Data fine] DataOra; |
| *tipodidati* | Uno dei tipi di dati SQL primari di Microsoft Jet o relativi sinonimi. |
| *istruzionesql* | Istruzione SQL, ad esempio SELECT, UPDATE, DELETE, INSERT, CREATE TABLE, DROP TABLE e così via. |

##### Osservazioni

Una procedura SQL è rappresentata da una proposizione PROCEDURE che specifica il nome della procedura, da un elenco facoltativo di definizioni di parametro e da una singola istruzione SQL.

Il nome della procedura non può essere uguale al nome di una tabella esistente.

1. **Istruzione EXECUTE**

Utilizzata per richiamare l'esecuzione di una procedura.

##### Sintassi

EXECUTE *procedura* [*param1*[, *param2*[, ...]]

L'istruzione EXECUTE è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *procedura* | Nome della procedura da eseguire. |
| *param1, param2, …* | Valori per i parametri definiti dalla procedura. |

**Funzioni di aggregazione SQL**

* **Funzione COUNT (Conteggio)**

Calcola il numero di record restituiti da una query.

##### Sintassi

**Conteggio(***espr***)**

Il segnaposto *espr* indica un'espressione stringa con la quale viene identificato il campo contenente i dati di cui si desidera ottenere il conteggio ovvero un'espressione con la quale viene eseguito un calcolo utilizzando i dati inclusi nel campo. Negli operandi di *espr* possono essere inclusi il nome del campo di una tabella o una funzione, intrinseca o definita dall'utente, ma non un'altra funzione di aggregazione SQL. È possibile eseguire il conteggio di qualsiasi tipo di dati, incluso testo.

##### Osservazioni

È possibile utilizzare la funzione **Conteggio** per eseguire il conteggio del numero di record di una query sottostante. Ad esempio, la funzione**Conteggio** consente di eseguire il conteggio del numero di ordini inviati in un determinato paese.

Sebbene *espr* consenta di eseguire il calcolo in un campo, la funzione **Conteggio** considera solo il numero di record, prescindendo dai valori che essi includono.

La funzione **Conteggio** non si applica ai record che contengono campi **Null**, a meno che *espr* non corrisponda al carattere jolly asterisco (\*). Se si utilizza l'asterisco, la funzione **Conteggio** calcola il numero totale di record, inclusi anche quelli con campi **Null**. La funzione **Conteggio(**\***)** è notevolmente più rapida rispetto a **Conteggio(**[*Nome colonna*]**)**. Non inserire l'asterisco tra virgolette (' '). Nell'esempio che segue viene calcolato il numero di record presenti nella tabella Ordini:

SELECT Count(\*) AS TotalOrders FROM Orders;

Se *espr* identifica più campi, la funzione **Conteggio** conteggia un record soltanto se almeno uno dei campi è diverso da **Null**. Se tutti i campi specificati sono **Null**, il record non verrà incluso nel conteggio. Separare i nomi dei campi con la e commerciale (&). Nell'esempio che segue, il conteggio viene limitato ai record nei quali DataSpedizione o Trasporto non sono **Null**:

SELECT Count('ShippedDate & Freight') AS [Not Null] FROM Orders;

La funzione **Conteggio** può essere utilizzata in una espressione di query.

* **Funzione SUM (Somma)**

Restituisce la somma di un insieme di valori contenuti nel campo specificato di una query.

##### Sintassi

**Somma(***espr***)**

Il segnaposto *espr* rappresenta un'espressione stringa che identifica il campo contenente i dati numerici da aggiungere o un'espressione che esegue un calcolo utilizzando i dati presenti nel campo. Negli operandi di *espr* possono essere inclusi il nome di un campo di una tabella, una costante o una funzione (intrinseca o definita dall'utente ma non una delle altre funzioni di aggregazione SQL).

##### Osservazioni

La funzione **Somma** restituisce il totale dei valori di un campo. Ad esempio, è possibile utilizzare la funzione **Somma** per determinare il costo totale delle spese di trasporto.

La funzione **Somma** ignora i record che contengono campi **Null**. Nell'esempio che segue viene indicato come calcolare la somma dei prodotti dei campi Prezzo e Quantità:

SELECT Somma(Prezzo \* Quantità) AS [Ricavo totale] FROM [Dettagli ordini];

La funzione **Somma** può essere utilizzata in un'espressione di query.

* **Funzione AVG (Media)**

Calcola la media aritmetica di un insieme di valori inclusi in un campo specificato di una query.

##### Sintassi

**Media(***espr***)**

Il segnaposto *espr* indica un'espressione stringa con la quale viene identificato il campo contenente i dati numerici di cui si desidera ottenere la media o un'espressione con la quale viene eseguito un calcolo utilizzando i dati presenti nel campo. Negli operandi di *espr* possono essere inclusi il nome del campo di una tabella, una costante o una funzione (intrinseca o definita dall'utente, ma non una delle altre funzioni di aggregazione SQL).

##### Osservazioni

La funzione **Media** consente di calcolare la media aritmetica, ovvero la somma dei valori divisa per il numero dei valori. Tale funzione può essere ad esempio utilizzata per calcolare le spese medie di trasporto.

Nella funzione **Media** non può essere incluso alcun campo **Null** per il calcolo.

È possibile utilizzare la funzione **Media** in una espressione di query

* **Funzioni MIN, MAX**

Restituiscono il valore minimo o massimo di un insieme di valori incluso in un campo specificato di una query.

##### Sintassi

**Min(***espr***)**

**Max(***espr***)**

Il segnaposto *espr* rappresenta un'espressione stringa che identifica il campo contenente i dati da valutare ovvero un'espressione che esegue un calcolo utilizzando i dati presenti nel campo. Negli operandi di *espr* possono essere inclusi il nome di un campo di tabella, una costante o una funzione (intrinseca o definita dall'utente ma non una delle altre funzioni di aggregazione SQL).

##### Osservazioni

Mediante **Min** e **Max** è possibile determinare i valori minimo e massimo di un campo in base all'aggregazione o al raggruppamento specificato. Ad esempio, è possibile utilizzare tali funzioni per restituire i costi minimi e massimi di trasporto. Se non è specificata alcuna aggregazione, viene utilizzata l'intera tabella.

È possibile utilizzare la funzione **Media** in una espressione di query

* **Funzioni FIRST, LAST (Primo, Ultimo)**

Restituiscono un valore di campo per il primo o per l'ultimo record della serie di risultati restituita da una query.

##### Sintassi

**Primo(***espr***)**

**Ultimo(***espr***)**

Il segnaposto *espr* indica un'espressione stringa con la quale viene identificato il campo contenente i dati da utilizzare o un'espressione con cui viene eseguito un calcolo utilizzando i dati presenti nel campo. Negli operandi di *espr* possono essere inclusi il nome del campo di una tabella, una costante o una funzione intrinseca o definita dall'utente ma non una delle altre funzioni di aggregazione SQL.

##### Osservazioni

Tali funzioni consentono semplicemente di ottenere il valore di un campo specificato, rispettivamente, nel primo o nell'ultimo record della serie di risultati restituiti da una query. Dal momento che i record non vengono restituiti in un determinato ordine (a meno che la query includa una proposizione ORDER BY), i record restituiti da queste funzioni saranno casuali.

* **Funzioni DevSt, StDevP**

Restituiscono le stime della deviazione standard relativa a una popolazione o a un campione di popolazione rappresentate come un insieme di valori contenuto in un campo specificato di una query.

##### Sintassi

**DevSt(***espr***)**

**StDevP(***espr***)**

Il segnaposto *espr* rappresenta un'espressione stringa che identifica il campo contenente i dati numerici da valutare o un'espressione che esegue un calcolo utilizzando i dati presenti nel campo. Negli operandi di *espr* possono essere inclusi il nome di un campo di una tabella, una costante o una funzione (intrinseca o definita dall'utente ma non una delle altre funzioni di aggregazione SQL).

##### Osservazioni

La funzione **StDevP** valuta una popolazione, mentre la funzione **DevSt** valuta un campione di popolazione.

Se la query sottostante contiene meno di due record (o non contiene alcun record nel caso della funzione **StDevP**), tali funzioni restituiscono un valore **Null**, che indica l'impossibilità di calcolare una deviazione standard.

Le funzioni **DevSt** e **StDevP** possono essere utilizzate in un'espressione di query.

* **Funzioni Var, VarP**

Restituiscono le stime della varianza relativa a una popolazione o a un campione di popolazione rappresentate da un insieme di valori in un determinato campo di una query.

##### Sintassi

**Var(***espr***)**

**VarP(***espr***)**

Il segnaposto *espr* indica un'espressione stringa con la quale si identifica il campo contenente i dati numerici che si desidera analizzare o un'espressione con cui viene eseguito un calcolo utilizzando i dati presenti nel campo. Negli operandi dell'*espressione* possono essere inclusi il nome di un campo di una tabella, una costante o una funzione (intrinseca o definita dall'utente ma non una delle altre funzioni di aggregazione SQL).

##### Osservazioni

La funzione **VarP** valuta una popolazione, mentre la funzione **Var** valuta un campione di popolazione.

Se la query sottostante contiene meno di due record, le funzioni **Var** e **VarP** restituiscono un valore **Null**, che indica l'impossibilità di calcolare una varianza.

È possibile utilizzare le funzioni **Var** e **VarP** in un'espressione della query o in un'istruzione SQL.

**Appendici**

* **Integrità Referenziale**

L'integrità referenziale è un sistema di regole utilizzato in Microsoft Access per assicurare che le relazioni tra i record di tabelle correlate siano valide e che i dati correlati non vengano eliminati o modificati accidentalmente. È possibile impostare l'integrità referenziale quando vengono rispettate tutte le condizioni seguenti:

* Il campo corrispondente della tabella primaria è una chiave primaria o ha un indice univoco.
* I campi correlati presentano lo stesso tipo di dati. Vi sono due eccezioni. Un campo di tipo Contatore può essere correlato a un campo di tipo Numerico con la proprietà DimensioneCampo impostata su Intero lungo, e un campo di tipo Contatore la cui proprietà DimensioneCampo è impostata su ID replica può essere correlato a un campo di tipo Numerico con la proprietà DimensioneCampo impostata su ID replica.
* Entrambe le tabelle appartengono allo stesso database di Microsoft Access. Se le tabelle sono collegate, devono essere nel formato di Microsoft Access. Inoltre, per impostare l'integrità referenziale, deve essere aperto il database in cui sono memorizzate. L'integrità referenziale non può essere applicata alle tabelle collegate di database in altri formati.

Nell'utilizzo dell'integrità referenziale valgono le regole seguenti:

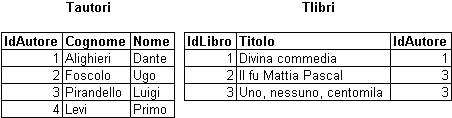
* Non è possibile immettere un valore nel campo della chiave primaria della tabella correlata inesistente nella chiave primaria della tabella primaria. Tuttavia è possibile immettere un valore Null nella chiave esterna, specificando che i record non sono correlati. Ad esempio, non è possibile che un ordine sia associato a un cliente inesistente, ma è possibile che sia presente un ordine non associato ad alcun cliente immettendo un valore Null nel campo IDCliente.
* Non è possibile eliminare un record da una tabella primaria se in una tabella correlata sono presenti record corrispondenti. Ad esempio non è possibile eliminare il record di un dipendente dalla tabella Dipendenti se a tale dipendente sono associati ordini nella tabella Ordini.
* Non è possibile modificare il valore di una chiave primaria della tabella primaria se tale record ha record correlati. Ad esempio non è possibile modificare l'ID di un dipendente nella tabella Dipendenti se a tale dipendente sono associati ordini nella tabella Ordini.

### Aggiornamenti ed eliminazioni a catena

Per le relazioni in cui viene applicata l'integrità referenziale, è possibile specificare se si desidera che Microsoft Access aggiorni o elimini a catena i record correlati. Se si impostano queste opzioni, vengono consentite le operazioni di eliminazione e aggiornamento normalmente impedite dalle regole di integrità referenziale. Quando si eliminano i record o si modificano i valori delle chiavi primarie di una tabella primaria, vengono apportate le necessarie modifiche alle tabelle correlate per mantenere l'integrità referenziale.  
  
Se, quando si definisce una relazione, si seleziona la casella di controllo Aggiorna campi correlati a catena, ogni volta che si modifica la chiave primaria di un record della tabella primaria la chiave primaria viene automaticamente aggiornata con il nuovo valore in tutti i record correlati. Se ad esempio si modifica l'ID di un cliente nella tabella Clienti, il campo IDCliente della tabella Ordini viene aggiornato automaticamente per tutti gli ordini del cliente, in modo che la relazione non venga infranta. In Microsoft Access gli aggiornamenti a catena vengono eseguiti senza che venga visualizzato alcun messaggio.   
  
NOTA: se la chiave primaria della tabella primaria è un campo di tipo Contatore, la selezione della casella di controllo Aggiorna campi correlati a catena non sortisce alcun effetto in quanto non è possibile modificare il valore di un campo di tipo Contatore.  
  
Se, quando si definisce una relazione, si seleziona la casella di controllo Elimina record correlati a catena, ogni volta che si eliminano record nella tabella primaria i record correlati vengono automaticamente eliminati nella tabella correlata. Se ad esempio si elimina il record di un cliente dalla tabella Clienti, tutti gli ordini di tale cliente vengono automaticamente eliminati dalla tabella Ordini (inclusi i record della tabella Dettagli sugli ordini correlati ai record della tabella Ordini). Se si eliminano record da un modulo o da un foglio dati quando è selezionata la casella di controllo Elimina record correlati a catena, l'utente viene avvertito che verranno eliminati anche i record correlati. Tuttavia, quando si eliminano i record mediante una query di eliminazione, i record delle tabelle correlate vengono eliminati automaticamente senza che venga visualizzato alcun avviso.

### Esempio

Consideriamo le tabelle rappresentate nella seguente figura



Nella tabella Tlibri si vuole inserire "I promessi sposi" di Alessandro Manzoni. Senza l'integrità referenziale tale operazione sarebbe possibile, in quanto il programma non controlla se i dati inseriti nella chiave esterna (campo IdAutore della tabella Tlibri) corrispondono a quelli della chiave primaria (campo IdAutore della tabella Tautori). Potrebbe, pertanto, essere inserito il record seguente:

IdLibro: 4

Titolo: Promessi Sposi

IdAutore: 5

dove il numero 5 identifica l'autore Manzoni, che però non è presente nella tabella Tautori.  
Con l'integrità referenziale, tentando di inserire il record precedente, il programma segnalerebbe il seguente errore:



L'integrità referenziale controlla, durante l'inserimento di un nuovo libro, che l'autore del libro sia presente nella tabella Tautori. Prima si deve inserire il record dell'autore, nella tabella Tautori, solo successivamente è possibile inserire il libro, dello stesso autore.

Quando si seleziona l'integrità referenziale, si rendono disponibili altre due proprietà:

- AGGIORNA CAMPI CORRELATI A CATENA: fa in modo che, quando si modifica il valore nel campo chiave primaria, venga automaticamente aggiornato il valore corrispondente nella chiave esterna. Per fare un esempio banale (non proponibile nella realtà), se l'autore Manzoni cambiasse IdAutore nella tabella Tautori, sarebbe modificato automaticamente anche il valore del campo IdAutore nella tabella Tlibri. In genere, questa proprietà è sempre attiva in quanto molto comoda nella maggior parte delle situazioni.  
- ELIMINA CAMPI CORRELATI A CATENA: quando si elimina uno (o più) record nel lato uno della relazione, automaticamente sono eliminati tutti i record del lato molti con la chiave esterna corrispondente. Per esempio, eliminando un autore nella tabella Tautori, verrebbero eliminati anche tutti i suoi libri. Prima di rendere attiva questa proprietà si deve analizzare in dettaglio la situazione, in quanto è molto pericolosa: potrebbe succedere che, partendo da pochi record di una tabella, siano eliminati migliaia di record in altre tabelle.



### Sintassi SQL

Vincolo su singolo campo:

CONSTRAINT name {PRIMARY KEY | UNIQUE | NOT NULL |     REFERENCES foreigntable [(foreignfield1, foreignfield2)]     [ON UPDATE CASCADE | SET NULL]     [ON DELETE CASCADE | SET NULL]}

Vincolo su più campi:

CONSTRAINT name     {PRIMARY KEY (primary1[, primary2 [, …]]) |     UNIQUE (unique1[, unique2 [, …]]) |     NOT NULL (notnull1[, notnull2 [, …]]) |     FOREIGN KEY [NO INDEX] (ref1[, ref2 [, …]]) REFERENCES foreigntable [(foreignfield1 [, foreignfield2 [, …]])]     [ON UPDATE CASCADE | SET NULL]     [ON DELETE CASCADE | SET NULL]}

Esempi:

CREATE TABLE Orders (OrderId INTEGER PRIMARY KEY, CustId INTEGER, OrderNotes NCHAR VARYING (255), CONSTRAINT FKOrdersCustId FOREIGN KEY (CustId) REFERENCES Customers ON UPDATE CASCADE ON DELETE CASCADE)

CREATE TABLE Orders (OrderId INTEGER PRIMARY KEY, CustId INTEGER, OrderNotes NCHAR VARYING (255), CONSTRAINT FKOrdersCustId FOREIGN KEY (CustId) REFERENCES Customers ON UPDATE CASCADE ON DELETE SET NULL)

### Integrità referenziale e SQL

In Access non è possibile implementare l’integrità referenziale mediante un comando SQL direttamente dall’interfaccia. Le seguenti query vengono eseguite correttamente:

CREATE TABLE T1 (Id AUTOINCREMENT PRIMARY KEY);

CREATE TABLE T2 (

Id AUTOINCREMENT PRIMARY KEY,

T1\_\_Id int NOT NULL,

CONSTRAINT FK\_T2\_1 FOREIGN KEY (T1\_\_Id) REFERENCES T1 (Id)

);

mentre le query seguenti :

CREATE TABLE T1 (Id AUTOINCREMENT PRIMARY KEY);

CREATE TABLE T2 (

Id AUTOINCREMENT PRIMARY KEY,

T1\_\_Id int NOT NULL,

CONSTRAINT FK\_T2\_1 FOREIGN KEY (T1\_\_Id) REFERENCES T1 (Id) ON DELETE CASCADE

);

darebbero il seguente messaggio di errore:

SQL Error: Syntax error in CONSTRAINT clause.

Per creare tabelle che gesticono l’integrità referenziale occorre utilizzare Jet OLE DB Provider e ADO.

* **Join - Concatenazione di tabelle**

Il vocabolo join significa unione e nel caso di SQL sta ad indicare unione tra tabelle. L’importanza principale del join risiede nella possibilità che ci offre per correlare e visualizzare dati appartenenti a tabelle diverse o alla medesima tabella, logicamente correlati tra di loro. I semplici dati, da noi uniti, possono assumere la forma di complesse informazioni così come noi li vogliamo. Esistono vari tipi di join:

* **CROSS JOIN (unione incrociata)**

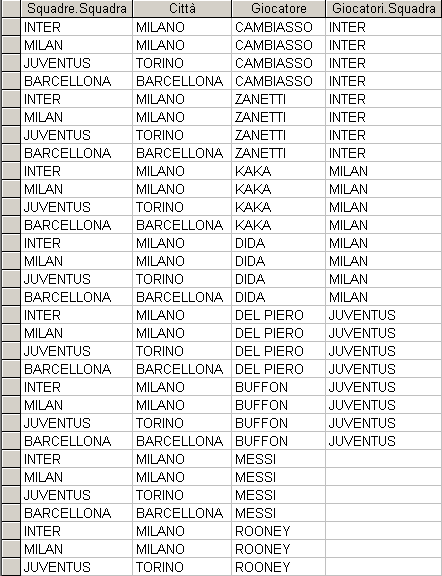
il prodotto cartesiano tra due o più tabelle si traduce in una istruzione chiamata CROSS JOIN. Il CROSS JOIN si ottiene in maniera molto semplice elencando dopo la FROM le tabelle che devono essere coinvolte. Vediamo un esempio di CROSS JOIN:

Per lo scopo usiamo due tabelle: Squadre e Giocatori





Query CrossJoin: SELECT \* FROM Squadre, Giocatori;



Questo è il risultato che si ottiene dal CROSS JOIN delle tabelle Squadre e Giocatori; come si può vedere non è altro che un prodotto cartesiano. Chiaramente avremmo potuto usare anche più di due tabelle.

Il CROSS JOIN non è particolarmente utile e viene usato raramente, ma se in una CROSS JOIN si utilizza la clausola WHERE potremmo ottenere join molto più interessanti.

* **NATURAL JOIN**

Il NATURAL JOIN è un tipo di operazione che ci permette di correlare due o più tabelle sulla base di valori uguali in attributi contenenti lo stesso tipo di dati. La selezione di un record avviene soltanto quando nelle due tabelle esiste l'esatta corrispondenza tra i campi posti in relazione.

Vogliamo ottenere un join delle righe delle due tabelle, Squadre e Giocatori, dove i valori dei campi Squadra sono uguali.

Query NaturalJoin: SELECT \* FROM Squadre, Giocatori WHERE Squadre.Squadra = Giocatori.Squadra;



* **INNER JOIN**

È un tipo di join in cui le righe delle tabelle vengono combinate solo se i campi collegati con join soddisfano una determinata condizione. La selezione di un record avviene soltanto quando nelle due tabelle esiste l'esatta corrispondenza tra i campi posti in relazione.

Vediamo un esempio:

Vogliamo ottenere un join delle righe delle due tabelle, Squadre e Giocatori, dove i valori dei campi Squadra sono uguali e dove il valore del campo Squadra è uguale ad ‘INTER’.

Query InnerJoin:

SELECT \*

FROM Squadre, Giocatori

WHERE Squadre.Squadra=Giocatori.Squadra AND Squadre.Squadra='INTER';



Esistono anche, delle parole chiavi specifiche per eseguire l’operazione di INNER JOIN.

Query InnerJoinSpecifico:

SELECT \*

FROM Squadre INNER JOIN Giocatori ON (Squadre.Squadra=Giocatori.Squadra AND Squadre.Squadra='INTER');



Usando le parole chiave specifiche dobbiamo indicare, per alcuni DBMS come nel caso di Access, a quale tabella appartengono i campi.

Vediamo altri esempi:

Usando le parole chiave specifiche, vogliamo ottenere lo stesso JOIN che abbiamo usato come esempio nell’esempio del NATURAL JOIN:

SELECT \* FROM Squadre INNER JOIN Giocatori ON Squadre.Squadra=Giocatori.Squadra;

Possiamo renderci conto che questo tipo di join è simile al natural join; infatti **il natural join è un particolare caso di inner join**.

* **OUTER JOIN**

Con l’OUTER JOIN è possibile estrapolare anche quei dati, appartenenti ad una delle tabelle, che non verrebbero estrapolati nei tipi di join visti fino a questo momento. Infatti OUTER significa esterno; dati esterni al normale tipo di join.

Dobbiamo specificare quale è la tabella di cui vogliamo estrapolare i dati anche se non soddisfano la condizione di join, questo lo facciamo indicando con LEFT o RIGHT se la tabella in questione è quella che appare a destra o a sinistra del comando JOIN.

SELECT . . .

FROM tabella1 [**LEFT** | **RIGHT**] JOIN tabella2

ON tabella1.campox condizione tabella2.campoy

Vediamo alcuni esempi:

Vogliamo visualizzare nel nostro JOIN oltre a tutte le squadre che possiedono almeno un giocatoree e i relativi giocatori, anche quelle che non possiedono nessun giocatore:

Query LeftJoin: SELECT \* FROM Squadre LEFT JOIN Giocatori ON Squadre.Squadra=Giocatori.Squadra;



Query RightJoin: SELECT \* FROM Squadre RIGHT JOIN Giocatori ON Squadre.Squadra=Giocatori.Squadra;



* **SELF JOIN**

Il SELF JOIN ci consente di unire una tabella con se stessa. La sintassi è simile a quella della query vista nel paragrafo che trattava il CROSS JOIN. Vediamo un esempio usando la tabella Squadre:

Query SelfJoin: SELECT S1.Squadra, S2.Squadra FROM Squadre as S1, Squadre as S2;



Come possiamo vedere dalla query otteniamo un prodotto cartesiano. Dopo la parola chiave SELECT siamo costretti a simulare l’esistenza di due tabelle mente invece ne abbiamo una soltanto. Dopo la parola chiave FROM faremo riferimento al nome delle colonne e alla tabella a cui appartengono:

SELECT nomeTabellaInesitente1.nomeColonna, nomeTabellaInesitente2.nomeColonna

FROM nomeColonna nomeTabellaInesitente1, nomeColonna nomeTabellaInesitente2

[WHERE condizioni];

* **JOIN tra tabelle usando operatori di confronto che non siano il segno di uguale (=)**

Possiamo usare dopo la clausola where anche tipi di operatori che non siano l’operatore di uguale (=). In casi del genere il join che si ottiene è abbastanza inusuale, ma può accadere che si renda necessario eseguire query di questo tipo.

La sintassi, dunque sarà uguale a quella di tutti i join visti in precedenza con la sola differenza che la dove appare il segno di uguale (=) possiamo usare, al suo posto, qualsiasi altro operatore di confronto.

* **JOIN su più di due tabelle**

Come il prodotto cartesiano può essere eseguito su più di due insiemi, anche i vari tipi di join possono essere applicati a più di due tabelle. Fa eccezione il self join, ma che comunque può simulare l’esistenza anche di più di due tabelle; vediamo un esempio senza però visualizzare il risultato della query:

SELECT T1.NOME, T2.NOME, T3.NOME

FROM PERSONE T1, PERSONE T2, PERSONE T3

WHERE . . .

Un altro caso particolare è quello dell’outer join che per essere applicato a più di due tabelle ha bisogno di una sintassi particolare; vediamo uno schema generale:

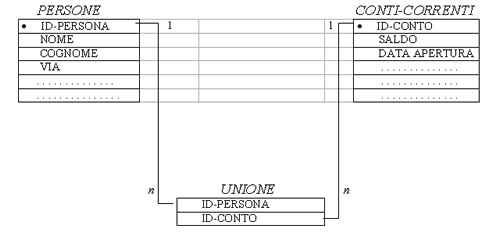
SELECT . . .

FROM tabella1 [LEFT | RIGHT] JOIN ( specifiche di join tra altre due o più tabelle)

ON . . .

I restanti tipi di join non presentano particolari sintassi o eccezioni ad essere applicati su più di due tabelle.

Vediamo ora un caso di join che si presenta frequentemente tra tre tabelle: chi conosce bene il modello relazionale sa che non è possibile ‘correlare’ due tabelle usando una relazione di tipo molti a molti. In questo caso si utilizza una terza tabella che contiene come chiavi esterne la chiave primaria della prima tabella e la chiave primaria della seconda tabella. Nell’esempio seguente abbiamo la tabella PERSONE correlata con la tabella CONTI-CORRENTI; il tipo di relazione è di tipo molti a molti. Infatti una persona può avere più conti correnti e un conto corrente può appartenere a più persone.



Se vogliamo visualizzare i dati così correlati usiamo la seguente query:

SELECT . . .

FROM PERSONE, UNIONE, CONTI-CORRENTI

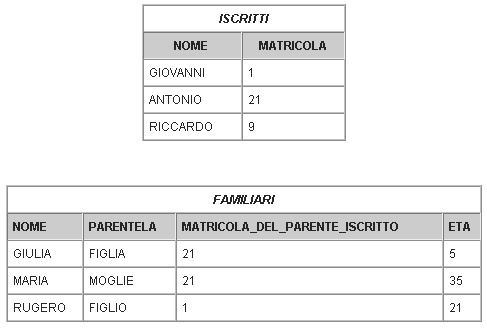
WHERE PERSONE.ID-PERSONA = UNIONE.ID-PERSONA

AND UNIONE.ID-CONTI = CONTI-CORRENTI.ID-CONTI;

Alla select precedente nulla ci impedisce di aggiungere altre condizioni, magari per poter visualizzare soltanto i conti appartenenti al Sig. Rossi Antonio. Lascio al lettore il compito di aggiungere all’espressione l’ulteriore condizione.

**Esempi:**

http://www.studiamo.it/dispense/informatica/sql/combinazioni-tabelle.html



Nella tabella ISCRITTI sono presenti gli associati ad un club, nella tabella FAMILIARI sono registrati gli appartenenti alla famiglia di ogni associato. Per alcuni esercizi faremo riferimento a queste due tabelle.

* Dalle tabelle ISCRITTI e FAMILIARI estrapolare un’unica tabella in cui compaiono per ogni iscritto i suoi familiari e il tipo di parentela. Gli iscritti senza familiari non dovranno apparire.

SELECT ISCRITTI.NOME, FAMILIARI.NOME, PARENTELA

FROM ISCRITTI, FAMILIARI

WHERE MATRICOLA = MATRICOLA\_DEL\_PARENTE\_ISCRITTO;

oppure

SELECT ISCRITTI.NOME, FAMILIARI.NOME, PARENTELA

FROM ISCRITTI INNER JOIN FAMILIARI

ON ISCRITTI.MATRICOLA = FAMILIARI.MATRICOLA\_DEL\_PARENTE\_ISCRITTO;

* Dalle tabelle ISCRITTI e FAMILIARI estrapolare un’unica tabella in cui compaiono per ogni iscritto i suoi familiari e il tipo di parentela. Gli iscritti senza familiari dovranno apparire.

ELECT ISCRITTI.NOME, FAMILIARI.NOME, PARENTELA

FROM ISCRITTI LEFT JOIN FAMILIARI

ON ISCRITTI.MATRICOLA = FAMILIARI.MATRICOLA\_DEL\_PARENTE\_ISCRITTO;

* Dalle tabelle ISCRITTI e FAMILIARI estrapolare un’unica tabella in cui compaiono per ogni iscritto i suoi familiari e il tipo di parentela, solo se tali familiari hanno un’età inferiore ai 20 anni. Gli iscritti con famigliari che non soddisfano la condizione non dovranno essere visualizzati.

SELECT ISCRITTI.NOME, FAMILIARI.NOME, PARENTELA

FROM ISCRITTI INNER JOIN FAMILIARI

ON (ISCRITTI.MATRICOLA = FAMILIARI.MATRICOLA\_DEL\_PARENTE\_ISCRITTO AND ETA < 20);

oppure

SELECT ISCRITTI.NOME, FAMILIARI.NOME, PARENTELA

FROM ISCRITTI, FAMILIARI

WHERE ISCRITTI.MATRICOLA = FAMILIARI.MATRICOLA\_DEL\_PARENTE\_ISCRITTO AND ETA < 20;

* Per ogni studente determina quanti esami ha sostenuto (ordina per num. decrescente di esami sostenuti)

SELECT Studenti.Nome, Studenti.Cognome, Count(\*) AS Esami\_sostenuti

FROM Studenti INNER JOIN Esami ON Studenti.Matricola = Esami.Matricola

GROUP BY Studenti.Nome, Studenti.Cognome

ORDER BY Count(\*) DESC;

Esempio di Join nidificati

* Seleziona gli studenti e gli esami che hanno sostenuto con i rispettivi titoli e voti

SELECT Studenti.Nome, Studenti.Cognome, Corsi.Titolo, Esami.Voto. Esami.Lode

FROM Studenti, Corsi, Esami

WHERE Corsi.CodiceCorso=Esami.CodiceCorso and Studenti.Matricola = Esami.Matricola;

oppure

SELECT Studenti.Nome, Studenti.Cognome, Corsi.Titolo, Esami.Voto,Esami.Lode

FROM

(Studenti INNER JOIN Esami ON Studenti.Matricola = Esami.Matricola)

INNER JOIN Corsi ON Corsi.CodiceCorso = Esami. CodiceCorso

oppure

SELECT Studenti.Nome, Studenti.Cognome, Corsi.Titolo, Esami.Voto, Esami.Lode

FROM Studenti INNER JOIN

(Corsi INNER JOIN Esami ON Corsi.CodiceCorso = Esami. CodiceCorso)

ON Studenti.Matricola = Esami.Matricola

Generalizzando

SELECT (Attributi)

FROM Tabella1, Tabella2, Tabella3

WHERE Tabella1.ATX = Tabella2.ATY and Tabella2.ATZ = Tabella3.ATW

diventa

SELECT (Attributi)

FROM (Tabella1 INNER JOIN Tabella2 ON Tabella1.At1 = Tabella2.At2)

INNER JOIN Tabella3 ON Tabella2.ATZ = Tabella3.ATW)

oppure

SELECT (Attributi)

FROM Tabella1 INNER JOIN

(Tabella2 INNER JOIN Tabella3 ON Tabella2.ATZ = Tabella3.ATW)

ON Tabella1.At1 = Tabella2.At2

* Libri presi in prestito da un utente il cui nome è un parametro

SELECT Utenti.Nome, Libri.Titolo FROM libri, prestiti, utenti

WHERE (Utenti.Nome like "\*"& dimmi l'Utente & "\*")

AND Prestiti.Cod\_utente=Utenti.Cod\_Utente

AND Prestiti.N\_inv=Libri.N\_inv;

* **Perform Joins Using Access 2013 SQL**

http://msdn.microsoft.com/en-us/library/office/ff845668.aspx

In a relational database system like Access 2013, you often need to extract information from more than one table at a time. This can be accomplished by using an SQL JOIN statement, which enables you to retrieve records from tables that have defined relationships, whether they are one-to-one, one-to-many, or many-to-many.

**INNER JOINs**

The INNER JOIN, also known as an equi-join, is the most commonly used type of join. This join is used to retrieve rows from two or more tables by matching a field value that is common between the tables. The fields you join on must have similar data types, and you cannot join on MEMO or OLEOBJECT data types. To build an INNER JOIN statement, use the INNER JOIN keywords in the FROM clause of a SELECT statement. This example uses the INNER JOIN to build a result set of all customers who have invoices, in addition to the dates and amounts of those invoices.

SELECT [Last Name], InvoiceDate, Amount

FROM tblCustomers INNER JOIN tblInvoices

ON tblCustomers.CustomerID=tblInvoices.CustomerID

ORDER BY InvoiceDate

Be aware that the table names are divided by the INNER JOIN keywords and that the relational comparison is after the ON keyword. For the relational comparisons, you can also use the <, >, <=, >=, or <> operators, and you can also use the BETWEEN keyword. Also note that the ID fields from both tables are used only in the relational comparison; they are not part of the final result set.

To further qualify the SELECT statement, you can use a WHERE clause after the join comparison in the ON clause. The following example narrows the result set to include only invoices dated after January 1, 1998.

SELECT [Last Name], InvoiceDate, Amount

FROM tblCustomers INNER JOIN tblInvoices

ON tblCustomers.CustomerID=tblInvoices.CustomerID

WHERE tblInvoices.InvoiceDate > #01/01/1998#

ORDER BY InvoiceDate

When you must join more than one table, you can nest the **INNER JOIN** clauses. The following example builds on a previous **SELECT** statement to create the result set, but also includes the city and state of each customer by adding the **INNER JOIN** for the tblShipping table.

SELECT [Last Name], InvoiceDate, Amount, City, State

FROM (tblCustomers INNER JOIN tblInvoices

ON tblCustomers.CustomerID=tblInvoices.CustomerID)

INNER JOIN tblShipping

ON tblCustomers.CustomerID=tblShipping.CustomerID

ORDER BY InvoiceDate

Be aware that the first **JOIN** clause is enclosed in parentheses to keep it logically separated from the second **JOIN** clause. It is also possible to join a table to itself by using an alias for the second table name in the **FROM** clause. Suppose that you want to find all customer records that have duplicate last names. You can do this by creating the alias "A" for the second table and checking for first names that are different.

SELECT tblCustomers.[Last Name],

tblCustomers.[First Name]

FROM tblCustomers INNER JOIN tblCustomers AS A

ON tblCustomers.[Last Name]=A.[Last Name]

WHERE tblCustomers.[First Name]<>A.[First Name]

ORDER BY tblCustomers.[Last Name]

**OUTER JOINs**

An OUTER JOIN is used to retrieve records from multiple tables while preserving records from one of the tables, even if there is no matching record in the other table. There are two types of OUTER JOINs that the Access database engine supports: LEFT OUTER JOINs and RIGHT OUTER JOINs. Think of two tables that are beside each other, a table on the left and a table on the right. The LEFT OUTER JOIN selects all rows in the right table that match the relational comparison criteria, and also selects all rows from the left table, even if no match exists in the right table. The RIGHT OUTER JOIN is simply the reverse of the LEFT OUTER JOIN; all rows in the right table are preserved instead.

As an example, suppose that you want to determine the total amount invoiced to each customer, but if a customer has no invoices, you want to show it by displaying the word "NONE."

SELECT [Last Name] & ', ' & [First Name] AS Name,

IIF(Sum(Amount) IS NULL,'NONE',Sum(Amount)) AS Total

FROM tblCustomers LEFT OUTER JOIN tblInvoices

ON tblCustomers.CustomerID=tblInvoices.CustomerID

GROUP BY [Last Name] & ', ' & [First Name]

Several things occur in the previous SQL statement. The first is the use of the string concatenation operator "&". This operator allows you to join two or more fields together as one string. The second is the immediate if (IIf) statement, which checks to see if the total is null. If it is, the statement returns the word "NONE." If the total is not null, the value is returned. The final thing is the OUTER JOIN clause. Using the LEFT OUTER JOIN preserves the rows in the left table so that you see all customers, even those who do not have invoices.

OUTER JOINs can be nested inside INNER JOINs in a multi-table join, but INNER JOINs cannot be nested inside OUTER JOINs.

**The Cartesian product**

A term that often comes up when discussing joins is the Cartesian product. A Cartesian product is defined as "all possible combinations of all rows in all tables." For example, if you were to join two tables without any kind of qualification or join type, you would get a Cartesian product.

SELECT \*

FROM tblCustomers, tblInvoices

This is not a good thing, especially with tables that contain hundreds or thousands of rows. You should avoid creating Cartesian products by always qualifying your joins.

* **Sottoquery SQL**

A volte si incontrano casi in cui non è possibile estrapolare i dati in maniera immediata e semplice. Per far fronte a situazione di questo tipo SQL ci mette a disposizione un altro potente strumento sintattico: la subquery.

Essa è una query che sta all’interno di un’altra interrogazione. La query interna passa i risultati alla query esterna che li verifica nella condizione che segue la clausola WHERE; vediamo i vari tipi di subquery.

Una sottoquery è un'istruzione SELECT nidificata in un'altra sottoquery o in un'istruzione SELECT, SELECT...INTO, INSERT...INTO, DELETE o UPDATE.

##### Sintassi

Per creare una sottoquery è possibile utilizzare tre forme di sintassi:

*confronto* [ANY | ALL | SOME] (*istruzione sql*)

*espressione* [NOT] IN (*istruzione sql*)

[NOT] EXISTS (*istruzione sql*)

Una sottoquery è composta dalle parti seguenti:

|  |  |
| --- | --- |
| **Parte** | **Descrizione** |
| *confronto* | Un'espressione e un operatore per eseguire il confronto tra l'espressione e i risultati della sottoquery. |
| *espressione* | L'espressione che viene ricercata in base al set di risultati della sottoquery. |
| *Istruzione sql* | Un'istruzione SELECT, formulata in base allo stesse regole e allo stesso formato di qualsiasi altra istruzione SELECT. È necessario racchiuderla tra parentesi. |

##### Osservazioni

È possibile utilizzare una sottoquery invece di un'espressione nell'elenco di campi di un'istruzione SELECT oppure in una proposizione HAVING o WHERE. In una sottoquery, l'istruzione SELECT fornisce un insieme di uno o più valori specifici da valutare nell'espressione delle proposizioni WHERE o HAVING.

Utilizzare i predicati ANY o SOME, che sono sinonimi, per recuperare i record della query principale che soddisfano il confronto con uno dei record recuperati nella sottoquery. Nell'esempio seguente vengono restituiti tutti i prodotti il cui prezzo unitario è maggiore di quello dei prodotti venduti con uno sconto pari o superiore al 25 percento:

SELECT \* FROM Prodotti WHERE PrezzoUnitario > ANY

(SELECT PrezzoUnitario FROM DettagliOrdini WHERE Sconto >= .25);

Utilizzare il predicato ALL per recuperare solo i record della query principale che soddisfano il confronto con tutti i record recuperati nella sottoquery. Se nell'esempio precedente si sostituisce ALL ad ANY, la query restituirà solo i prodotti il cui prezzo unitario è maggiore di quello di tutti i prodotti venduti con uno sconto pari o superiore al 25 percento. In tal modo viene ulteriormente ristretto l'ambito.

* **ALL**  Utilizzare ALL in una clausola WHERE per recuperare le righe che soddisfano la condizione quando viene eseguito il confronto con ogni riga restituita dalla sottoquery.

Si supponga, ad esempio, di analizzare i dati degli studenti di un'università. Gli studenti devono mantenere una media annuale (GPA) minima (Min\_GPA) che varia da disciplina a disciplina. Le discipline e le relative medie GPA minime sono archiviate in una tabella relativa alle discipline di specializzazione (Majors), mentre le informazioni sugli studenti sono archiviate in una tabella relativa ai record degli studenti (Student\_Records).

Per visualizzare un elenco di discipline e delle relative medie GPA minime per cui ogni studente che segue quella disciplina supera la media GPA minima, è possibile utilizzare la query riportata di seguito:

SELECT [Major], [Min\_GPA] FROM [Majors] WHERE [Min\_GPA] < ALL

(SELECT [GPA] FROM [Student\_Records] WHERE [Student\_Records].[Major]=[Majors].[Major]);

* **ANY**  Utilizzare ANY in una clausola WHERE per recuperare le righe che soddisfano la condizione quando viene eseguito il confronto con almeno una delle righe restituite dalla sottoquery.

Si supponga, ad esempio, di analizzare i dati degli studenti di un'università. Gli studenti devono mantenere una media annuale (GPA) minima (Min\_GPA) che varia da disciplina a disciplina. Le discipline e le relative medie GPA minime sono archiviate in una tabella relativa alle discipline di specializzazione (Majors), mentre le informazioni sugli studenti sono archiviate in una tabella relativa ai record degli studenti (Student\_Records).

Per visualizzare un elenco di discipline e delle relative medie GPA minime per cui qualsiasi studente che segue quella disciplina non soddisfa la media GPA minima, è possibile utilizzare la query riportata di seguito:

SELECT [Major], [Min\_GPA] FROM [Majors] WHERE [Min\_GPA] > ANY

(SELECT [GPA] FROM [Student\_Records] WHERE [Student\_Records].[Major]=[Majors].[Major]);

**Nota**  È inoltre possibile utilizzare la parola chiave SOME per lo stesso scopo, essendo SOME sinonimo di ANY.

Utilizzare il predicato IN per recuperare solo i record della query principale che contengono valori uguali a un record della sottoquery. Nell'esempio che segue vengono restituiti tutti i prodotti con uno sconto pari o superiore al 25 percento:

SELECT \* FROM Prodotti WHERE IDProdotto IN

(SELECT IDProdotto FROM DettagliOrdini WHERE Sconto >= .25);

D'altro canto, è possibile utilizzare NOT IN per recuperare solo i record della query principale che non contengono valori uguali a quelli di alcun record della sottoquery.

* **IN**  Utilizzare IN in una clausola WHERE per verificare che un valore nella riga corrente della query principale fa parte del set restituito dalla sottoquery. È inoltre possibile far precedere NOT a IN per verificare che un valore nella riga corrente della query principale non fa parte del set restituito dalla sottoquery.

Ad esempio, la query riportata di seguito restituisce una elenco di ordini (Orders), con le relative date, elaborati da dipendenti (Employees) la cui posizione (Job Title) non è quella di venditore (Sales Representative):

SELECT [Order ID], [Order Date] FROM [Orders]

WHERE [Employee ID] IN (SELECT [ID] FROM [Employees]

WHERE [Job Title]<>'Sales Representative');

Utilizzando NOT IN è possibile scrivere la stessa query come segue:

SELECT [Order ID], [Order Date] FROM [Orders]

WHERE [Employee ID] NOT IN (SELECT [ID] FROM [Employees]

WHERE [Job Title]='Sales Representative');

Utilizzare il predicato EXISTS, con la parola riservata facoltativa NOT, nei confronti di tipo vero/falso per determinare se la sottoquery restituisce record.

* **EXISTS**  Utilizzare EXISTS in una clausola WHERE per indicare che una sottoquery dovrà restituire almeno una riga. È inoltre possibile far precedere NOT a EXISTS per indicare che una sottoquery non dovrà restituire alcuna riga.

Ad esempio, la query riportata di seguito restituisce un elenco di prodotti presenti in almeno un ordine esistente:

SELECT \* FROM [Products] WHERE EXISTS

(SELECT \* FROM [Order Details]

WHERE [Order Details].[Product ID]=[Products].[ID]);

Se si utilizza NOT EXISTS, la query restituisce un elenco di prodotti non presenti in almeno un ordine esistente:

SELECT \* FROM [Products] WHERE NOT EXISTS

(SELECT \* FROM [Order Details]

WHERE [Order Details].[Product ID]=[Products].[ID]);

È inoltre possibile utilizzare gli alias dei nomi di tabella in una sottoquery per fare riferimento alle tabelle elencate in una proposizione FROM esterna alla sottoquery. Nell'esempio seguente vengono restituiti i nomi degli impiegati i cui stipendi sono uguali o superiori allo stipendio medio di tutti gli impiegati con la stessa posizione. Alla tabella Impiegati viene assegnato l'alias "T1":

SELECT Cognome, Nome, Posizione, Stipendio FROM Impiegati AS T1

WHERE Stipendio >= (SELECT Media(Stipendio)

FROM Impiegati WHERE T1.Posizione = Impiegati.Posizione) Order by Posizione;

Nell'esempio precedente, la parola riservata AS è facoltativa.

Nel seguente esempio si elencano la società e il contatto di tutti i clienti che hanno effettuato un ordine nel secondo trimestre del 1995.

SELECT NomeContatto, PosizioneContatto, Telefono FROM Clienti

WHERE IDCliente IN (SELECT IDCliente ID FROM Ordini WHERE DataOrdine Between #01/04/95# And #01/07/95#);

**Approfondimenti**

* **Subquery che ci restituiscono un solo valore**

Per gli esempi di questo paragrafo si farà riferimento alla tabella DIPENDENTI:



Vogliamo conoscere il nome dei dipendenti le cui ferie godute superino la media delle ferie godute da tutti. A qualcuno potrebbe venire in mente di scrivere una query di questo tipo:

SELECT NOME FROM DIPENDENTI

WHERE FERIE\_GODUTE > AVG(FERIE\_GODUTE);

ma il risultato che si otterrebbe è un messaggio di errore; infatti non è possibile far seguire la clausola WHERE da funzioni di gruppo. Facciamo finta di fare un altro esperimento:

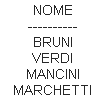
SELECT NOME FROM DIPENDENTI

HAVING FERIE\_GODUTE > AVG(FERIE\_GODUTE);

anche in questo caso quello che si otterrebbe è un messaggio di errore: non è possibile usare la clausola HAVING in espressioni dove non compare la clausola GROUP BY e non è possibile, nella nostra interrogazione, eseguire raggruppamenti, dunque dobbiamo per forza usare dopo la clausola WHERE una subquery.

SELECT NOME FROM DIPENDENTI

WHERE FERIE\_GODUTE > (SELECT AVG(FERIE\_GODUTE) FROM DIPENDENTI);



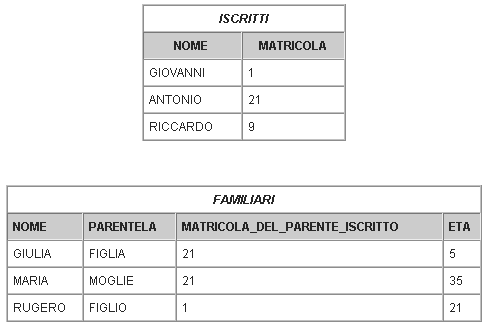
Sapendo a priori che il valore medio delle ferie godute da ogni dipendente è pari a 8,875 giorni possiamo verificare che l’espressione scritta è corretta e estrapola esattamente i dati che ci interessavano.

È evidente che il risultato della subquery è un unico valore; infatti non è possibile, con questo tipo di sintassi, estrapolare dalla subquery più di un valore e non è possibile usare le clausole GROUP BY e HAVING.

Ricapitolando elenchiamo delle regole valide per l’utilizzo di questo tipo di subquery:

* 1. La subquery deve restituire un unico valore
  2. Nella subquery non possono apparire le clausole GROUP BY e HAVING
  3. La subquery deve comparire alla destra dell’operatore di confronto
  4. Non si possono confrontare due subquery (conseguenza della regola precedente).
* **Subquery con IN**

Questo operatore ci consente di estrapolare dalla subquery non un solo valore, ma più valori da cui verrà verificata la corrispondenza. Cerchiamo di capire con un esempio questa sintassi:



Nella tabella ISCRITTI sono presenti gli associati ad un club di cacciatori, nella tabella FAMILIARI sono registrati gli appartenenti alla famiglia di ogni associato. Vogliamo visualizzare gli associati che hanno almeno un famigliare:

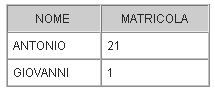
SELECT \*

FROM ISCRITTI

WHERE MATRICOLA

IN

(SELECT MATRICOLA\_DEL\_PARENTE\_ISCRITTO FROM FAMILIARI);



Come possiamo vedere sono stati estrapolati solo i nominativi Antonio e Giovanni, gli unici che hanno familiari, dunque la subquery (quella tra parentesi) estrapola una serie di matricole le quali la dove c’è corrispondenza con le matricole della prima select la condizione è verificata. Adesso vogliamo visualizzare gli associati che hanno uno o più figlie.

SELECT \*

FROM ISCRITTI

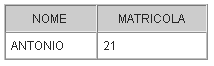
WHERE MATRICOLA

IN

(SELECT MATRICOLA\_DEL\_PARENTE\_ISCRITTO

FROM FAMILIARI

WHERE PARENTELA = 'FIGLIA');



Il ‘meccanismo’ che abbiamo usato è simile a quello della query precedente, con la differenza che abbiamo aggiunto una ulteriore condizione nella subquery. Possiamo aggiungere tante ulteriori condizioni quante ne servono; addirittura considerando una subquery come una query qualsiasi, nessuno ci impedisce di confrontare nella condizione i valori estrapolati da ‘subsubquery’. Vediamo nel prossimo paragrafo di comprendere meglio quanto detto.

* **Subquery annidate**

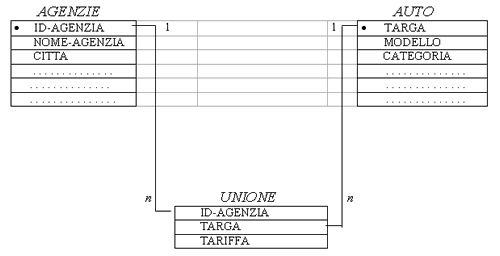
Con il termine annidate si identificano quelle query che si trovano all’interno di subquery:

SELECT nome campi

FROM nome tabella

WHERE (SUBQUERY condizione (SUBQUERY condizione (SUBQUERY )));

nello schema soprastante abbiamo inserito tre subquery nella prima select una dentro l’altra come se fossero scatole cinesi, ma avremo potuto inserirne anche più. La potenza di una simile struttura è notevole anche se la sua complessità richiede nella sua applicazione una particolare attenzione e tecnica di ‘costruzione’. Cercheremo con un esempio di comprendere questo tipo di struttura sintattica. La struttura sottostante rappresenta il database che conserva le informazioni inerenti all’attività che andremo ad analizzare:



Si ha una azienda che noleggia auto. L’azienda opera su tutto il territorio nazionale tramite agenzie presenti nelle maggiori città. Le automobili non possono essere associabili alle singole agenzie e il loro costo di noleggio dipende oltre che dalla categoria dell’auto (utilitaria, sport, gran turismo, ecc.) anche dall’agenzia da dove è stata noleggiata: una automobile noleggiata all’agenzia di Milano costa di più della stessa automobile noleggiata tramite l’agenzia di Catania.

Nello schema si vedono tre tabelle; la tabella Agenzie è associata alla tabella Auto tramite la tabella Unione. Questo tipo di struttura è necessaria per far fronte al tipo di relazione, molti a molti, che si ha tra una agenzia e un’auto. Infatti la stessa auto viene associata a tutte le agenzie e ad una agenzia associamo tutte le auto; in questo modo per mezzo dell’attributo tariffa sappiamo il costo di ogni auto noleggiata per ogni agenzia.

Supponiamo ora che vogliamo conoscere il nome delle agenzie che applicano una tariffa inferiore a 200 euro per automobili di categoria sport.

Scomponendo il problema in più moduli che poi risolveremo con delle singole select, arriveremo alla soluzione in modo semplice:

Iniziamo dalla selezione delle automobili di tipo sport:

SELECT TARGA FROM AUTO WHERE CATEGORIA = ‘sport’ ;

Questa query la chiameremo Q1.

Continuiamo selezionando le tariffe che soddisfano le condizioni del quesito:

SELECT ID\_AGENZIA FROM UNIONE

WHERE TARGA IN (Q1)

AND TARIFFA < 200;

Questa query la chiamiamo Q2.

A questo punto abbiamo a disposizione i codici delle agenzie che soddisfano il nostro quesito, ma non abbiamo ancora a disposizione i nomi di tali agenzie. Inoltre i codici delle agenzie sono inutilmente ripetuti più volte, dunque la successiva query che soddisfa completamente il quesito è:

SELECT NOME\_AGENZIA

FROM AGENZIE

WHERE ID\_AGENZIA IN (Q2);

Vediamo ora la query scritta in modo completo:

SELECT NOME\_AGENZIA

FROM AGENZIE

WHERE ID\_AGENZIA IN

(SELECT ID\_AGENZIA

FROM UNIONE

WHERE TARGA IN

(SELECT TARGA

FROM AUTO

WHERE CATEGORIA = 'SPORT';)

AND TARIFFA < 200;);

Considerando il modo in cui abbiamo costruito questa interrogazione, possiamo renderci conto che la maniera migliore per effettuare la lettura e comprensione di una query di questo tipo, è iniziare ad analizzare le subquery più interne e man mano passare a quelle più esterne. Questa considerazione è importante ed è valida anche per la scrittura della query. In quest’ultimo caso però, decidere quale sarà la subquery più interna è difficile, comunque sia dobbiamo affidarci non tanto a delle eventuali regole assiomatiche, ma alla nostra logica che se utilizzata con rigore non può tradirci.

* **Interrogazioni annidate con predicati any, all e exists**

Per usare i predicati di confronto con selezioni che possono restituire piú di una riga occorre usare le quantificazioni any o all. Il predicato any dà “vero” se il confronto è vero almeno per una riga; dà “falso” se tutti i confronti di riga sono “falsi”, o la subquery restituisce una tabella vuota; se non ci sono confronti veri e almeno uno è null, allora il risultato è null. Ad esempio, la seguente

interrogazione permette di ottenere le date delle gare il cui vincitore ha corso con Baio:

select giorno from Gara

where FantinoVincente = any (select Fantino from Coppia

where cavallo = ’Baio’);

Il predicato all dà “vero” se tutti i confronti di riga sono veri o la subquery restituisce la tabella vuota;1 dà “falso” se almeno uno dà falso; restituisce null se non ci sono confronti “falsi”, ma almeno uno è “indefinito”. Ad esempio, le date delle gare che precedono tutte le gare vinte da Baio si ottengono con:

select giorno from Gara

where giorno < all (select G.giorno from Gara G

inner join Coppia P ON G.Coppia = P.Coppia

where P.cavallo = ’Baio’);

Il predicato IN è equivalente a = ANY. Il predicato not in è equivalente a <> ALL.

Il predicato exists è vero se la subquery restituisce almeno una riga, altrimenti è falso. Ad esempio, i nomi dei fantini che hanno vinto qualche gara si possono ottenere con l’istruzione:

select P.fantino from Coppia P

where exists (select \* from Gara G

where G.FantinoVincente = P.Vincente);

Si noti l’uso, tipico, di una subquery del tipo select \* (non interessano particolari attributi) e il riferimento, all’interno della subquery, alla tabella nella clausola from della select piú esterna (la variabile P).

**Esempi:**

1. Informazioni dei dipendenti che non sono impiegati e che hanno lo stipendio superiore a quello di uno qualsiasi tra gli impiegati.

SELECT cognome, nome, funzione

FROM dipendenti

WHERE funzione <> ‘impiegato’

AND stipendioBase > ANY

(SELECT stipendioBase

FROM dipendenti

WHERE funzione = ‘impiegato’);

1. Informazioni dei dipendenti che hanno lo stipendio minore di 1200 e funzione uguale a uno qualsiasi tra i dipendenti con la stessa funzione ma stipendio maggiore di 1200.

SELECT cognome, nome, stipendioBase, funzione

FROM dipendenti

WHERE stipendioBase <1200

AND funzione = ANY

(SELECT funzione

FROM dipendenti

WHERE stipendioBase >=1200);

1. Informazioni dei dipendenti che non sono impiegati e che hanno lo stipendio superiore a quello di tutti gli impiegati.

SELECT cognome, nome, funzione

FROM dipendenti

WHERE funzione <> ‘impiegato’

AND stipendioBase > ALL

(SELECT stipendioBase

FROM dipendenti

WHERE funzione = ‘impiegato’);

1. Informazioni dei dipendenti che hanno lo stipendio superiore alla media della propria funzione.

SELECT cognome, nome, funzione

FROM dipendenti AS d

WHERE stipendioBase >

(SELECT AVG(stipendioBase)

FROM dipendenti

WHERE dipendenti.funzione =d.funzione);

1. Elenco dei dipendenti che lavorano in filiali che hanno più di 10 dipendenti.

SELECT cognome, nome

FROM dipendenti

WHERE filiale IN (SELECT filiale

FROM dipendenti

GROUP BY filiale

HAVING COUNT (\*)>10);

1. Elenco dei dipendenti solo se esistono dipendenti di sesto livello.

SELECT cognome, nome FROM dipendenti

WHERE EXISTS (SELECT \* FROM dipendenti WHERE livello = 6);

1. Ottenere i nomi dei giocatori della stessa nazionalità di Kramnik.

select nome from Giocatore

where nazionalita = (select nazionalita from Giocatore

where nome = ’Kramnik’);

1. Determinare i nomi e le varianti delle aperture il cui nome segue, nell’ordinamento lessico-graﬁco, il nome dell’apertura con codice E86.

select nome, variante from Apertura

where nome > (select nome from Apertura where eco = ’E86’);

1. Determinare le partite aperte con la variante ‘1 e4 c5; 2 c3’

select ∗ from Partita

where eco = (select eco from Apertura

where variante = ’1 e4 c5; 2 c3’);

1. Ottenere i dati dei fantini piú giovani di Pino Galoppo.

select ∗ from Fantino

where eta < (select eta from Fantino where nome = ’Pino Galoppo’);

1. Determinare i cavalli di scuderie diverse da quella di Baio.

select ∗ from Cavallo

where scuderia <> (select scuderia from Cavallo

where nome = ’Baio’);

1. Ottenere i nomi dei fantini vincitori di gare svoltesi nello stesso ippodromo della gara del 24 marzo 2006.

select P.fantino from Coppia as P join Gara as G

on P.Fantino = G.FantinoVincente

where ippodromo = (select ippodromo from Gara

where giorno = ’2006−03−24’);

1. Ottenere gli autori di libri letti da qualche socio.

select L.autore from Libro as L

where L.ISBN in (select ISBN from Ha\_letto);

select Libro.autore from Libro

where exists (select ISBN from Ha\_letto where Libro.ISBN=Ha\_letto.ISBN);

1. Titoli e autori di libri il cui ISBN à superiore a qualcuno degli ISBN dei libri letti da Pino Galoppo

select L.titolo, L.autore from Libro L

where L.ISBN > any (select H.ISBN from Ha\_letto H

where H.ci = (select ci from Socio where nome = ’ Pino Galoppo’));

1. Cavalli che non hanno vinto alcuna gara

select cavallo from Coppia

where Fantino <> all (select FantinoVincente from Gara);

1. Età dei fantini che hanno partecipato a qualche gara

select F.nome, F.eta from Fantino as F

where exists (select \* from Coppia as P

where F.nome = P.fantino);

1. Età dei fantini che hanno corso con qualche cavallo della scuderia Rioja

select F.eta from Fantino F

where exists (select \* from Coppia P join Cavallo C

on P.cavallo = C.nome

where C.scuderia = ’Rioja’ and F.nome = P.fantino);

**Glossario**

alias

In SQL (Structured Query Language), nome alternativo che viene assegnato a un campo o a un'espressione di un'istruzione SELECT, utilizzando la proposizione facoltativa AS. Nello standard ANSI l'alias viene definito nome di correlazione.

argomento

Costante, variabile o espressione che fornisce informazioni aggiuntive a un'azione, una routine o un metodo. Gli argomenti vengono utilizzati con operandi nelle espressioni per fornire informazioni aggiuntive necessarie per l'esecuzione di determinate azioni.

asterisco

Carattere (\*) utilizzato come carattere jolly nelle espressioni SQL LIKE per accettare qualsiasi stringa di caratteri. L'espressione LIKE "\*città\*" restituisce, ad esempio, record che includono la stringa "città" seguita da zero o più caratteri. Quando si crea una query o si immette un'istruzione SQL, utilizzare l'asterisco per includere tutti i campi presenti nella tabella o nella query sottostante. "SELECT \* FROM Tabella" restituirà, ad esempio, tutti i campi (colonne) della tabella. Nelle query eseguite mediante il provider Microsoft OLE DB per Jet l'asterisco non viene utilizzato come carattere jolly. È infatti il segno di percentuale (%) a rappresentare il carattere jolly ANSI. Per le query archiviate tutto ciò non ha alcuna importanza, ma nelle nuove query (visualizzazioni), create mediante il provider Microsoft OLE DB per Jet, per ottenere un comportamento simile a quello ottenuto dall'utilizzo dell'asterisco (\*) e del punto interrogativo (?), sarà necessario utilizzare i caratteri jolly ANSI, ossia il segno di percentuale (%) e il carattere di sottolineatura (\_). Nel provider Microsoft OLE DB per Jet l'asterisco e il punto interrogativo saranno interpretati in maniera letterale e il set di risultati restituito sarà diverso da quello che si sarebbe ottenuto utilizzando l'interfaccia di Microsoft Access o DAO.

aggiornamento a catena

Nelle relazioni che applicano l'integrità referenziale tra tabelle, l'aggiornamento a catena è un'opzione mediante la quale se si modifica la chiave primaria in un record della tabella primaria, viene automaticamente aggiornata la chiave esterna in tutti i record associati della tabella o delle tabelle esterne correlate. Se si stabilisce, ad esempio, una relazione tra una tabella (principale) Clienti e una tabella (esterna) Ordini con l'opzione di aggiornamento a catena attivata, quando viene modificata la chiave primaria in un record della tabella Clienti, vengono modificati anche tutti gli ordini associati al cliente in questione della tabella esterna Ordini. Tale processo è ora definibile anche mediante il linguaggio DDL SQL.

attributo Identity

Attributo che viene aggiunto per consentire l'equivalenza sintattica con il server SQL. È disponibile solo se si utilizza il linguaggio DDL SQL dal provider Microsoft OLE DB per Jet. Questo tipo di dati ha le stesse funzionalità dei tipi di dati Counter e AutoNumber. Si tratta, infatti, di un sinonimo senza funzionalità aggiuntive.

analisi

Processo che consente di identificare le parti di un'istruzione o di un'espressione e di convalidarle a fronte delle regole di linguaggio appropriate.

autorizzazione

Uno o più attributi che specificano il tipo di accesso di un utente a dati o oggetti di un database. Se, ad esempio, l'utente dispone dell'autorizzazione Lettura dati per una tabella o una query, può visualizzare o recuperare ma non modificare i dati della tabella o della query.

aggiornamento

Processo che consente di salvare le modifiche ai dati di un record. Fino a quando il record non viene salvato, le modifiche vengono memorizzate in un record temporaneo denominato buffer di copia. La proposizione UPDATE di un'istruzione SQL modifica i valori dei dati in uno o più record (righe) di una tabella di database.

campo Contatore

Tipo di dati di un campo che consente di memorizzare automaticamente un numero univoco per ogni record aggiunto a una tabella. In un campo Contatore viene sempre utilizzato il tipo di dati **Long**. I numeri generati da un campo Contatore non possono essere modificati, a meno che non sia stata concessa una speciale autorizzazione per la tabella mediante l'istruzione Grant o Revoke della sintassi DDL SQL. Il campo Contatore è noto anche come tipo di dati Contatore o Identità. Se si utilizza il linguaggio DDL SQL, questo tipo di dati supporta ora diversi valori di inizializzazione e di incremento, a differenza di quanto accade nelle precedenti versioni, in cui il valore di inizializzazione è 1 e il valore di incremento è 1.

collisione

Conflitto che può verificarsi durante un aggiornamento in blocco. Se un client modifica i dati originali del server mentre un altro client legge dati dal server e tenta di modificarli con un aggiornamento in blocco, si verifica una collisione. In una situazione di questo tipo, il secondo client tenta di modificare dati del server senza sapere quali dati siano realmente presenti nel server.

colonna

Rappresentazione visiva di un campo in una griglia. Una colonna definisce il tipo di dati, la dimensione e altri attributi di un campo di una riga (record) di dati. Tutte le colonne considerate come un insieme definiscono una riga (record) nel database. Una singola colonna include dati correlati per tipo e scopo nell'intera tabella; in altre parole, la definizione di una colonna non cambia da riga a riga.

commit

Operazione che consente di accettare una transazione in sospeso. Se si utilizza l'elaborazione delle transazioni e si avvia una transazione mediante DAO, ADO o SQL, nessuna delle modifiche eseguite nella transazione verrà scritta nel database fino a quando non si eseguirà il commit, ovvero non si accetterà la transazione.

criteri

Serie di condizioni di limitazione, quali **= "Danimarca"** (uguale a Danimarca) o **> 30000** , utilizzate per la creazione di una query o di un filtro per visualizzare uno specifico insieme di record.

campo di destinazione

Campo di una query in cui verranno inclusi i risultati della query.

campo

Categoria di informazioni memorizzate in una tabella di un database o in una colonna di dati. Elemento di una tabella di database che include un'informazione specifica, ad esempio un cognome.

chiave esterna

Uno o più campi di tabella che fanno riferimento al campo o ai campi chiave primaria di un'altra tabella. Una chiave esterna indica la correlazione tra le tabelle. I dati dei campi chiave esterna e chiave primaria devono corrispondere. Un elenco di numeri di parte validi conterrà, ad esempio, una chiave esterna per una tabella di inventario che include riferimenti a numeri di parte validi. Le chiavi esterne vengono utilizzate quando si stabilisce l'integrità referenziale per un database.

cursore forward-only

Set di risultati in cui la posizione corrente può essere spostata solo in avanti di un numero specificato di record o sull'ultimo record. La posizione corrente non può essere spostata all'inizio del set di risultati o sul record precedente. L'appartenenza, l'ordine e i valori del cursore sono in genere fissi, se il cursore è aperto. Se altri utenti aggiornano, eliminano o inseriscono righe, il cursore non riflette queste modifiche fino a quando non viene chiuso e riaperto.

costante intrinseca

Costante implementata da un'applicazione. Ad esempio, le costanti DAO (Data Access Objects) sono elencate nella libreria di oggetti DAO e possono essere visualizzate utilizzando il Visualizzatore oggetti.

cursore keyset

Set di righe che può essere utilizzato per aggiungere, modificare o eliminare righe da una tabella o più tabelle di database sottostanti. I movimenti all'interno del keyset sono illimitati. Un cursore keyset può contenere colonne di una o più tabelle di un database. L'appartenenza è fissa.

campo Null

Campo che non contiene caratteri né valori. Un campo Null è diverso da una stringa a lunghezza zero ("") o da un campo con un valore pari a 0. Un campo viene impostato su Null se il suo contenuto è sconosciuto. Ad esempio, un campo Data completamento di una tabella di attività sarà impostato su Null fino a quando non sarà stata completata un'attività.

chiave primaria

Uno o più campi il cui valore o i cui valori identificano in modo univoco ogni record di una tabella. In una relazione una chiave primaria viene utilizzata per fare riferimento a record specifici di una tabella da un'altra tabella. Una chiave primaria viene definita chiave esterna quando si fa riferimento a essa da un'altra tabella. È possibile avere una sola chiave primaria. In una tabella Dipendenti, ad esempio, è possibile utilizzare il codice fiscale come chiave primaria.

self-join

Join in cui i record di una tabella sono combinati con altri record della stessa tabella, se esistono valori corrispondenti nei campi collegati tramite join. Un self-join può essere un inner join o un join esterno. Un self-join è utile per interrogazioni gerarchiche, ad esempio per una query che visualizza tutti i dirigenti e le persone di cui sono responsabili.

sottoquery

Istruzione SELECT nidificata in un'istruzione SELECT, SELECT...INTO, INSERT...INTO, DELETE o UPDATE o in un'altra sottoquery.

caratteri jolly

L'asterisco (\*), il segno di percentuale (%), il punto interrogativo (?), il carattere di sottolineatura (\_), il cancelletto (#), il punto esclamativo (!), il segno meno (-) e le parentesi quadre ([ ]) rappresentano caratteri jolly. Tali caratteri possono essere utilizzati nelle query e nelle espressioni per includere tutti i record, i nomi di file o altri elementi che iniziano con caratteri specifici o corrispondono a un determinato schema. È inoltre possibile utilizzare caratteri jolly e caratteri corrispondenti per definire ulteriormente una ricerca, quando si utilizza un'istruzione SQL.

DDL (Data Definition Language)

Linguaggio utilizzato per descrivere attributi di un database, in particolare tabelle, campi, indici e strategia di archiviazione. In base allo standard ANSI i token di questo linguaggio sono CREATE, DROP e ALTER.

istruzione/stringa SQL

Espressione che definisce un comando SQL (Structured Query Language), quale SELECT, UPDATE o DELETE, e può includere proposizioni quali WHERE e ORDER BY. Le stringhe e le istruzioni SQL vengono utilizzate in genere in query ma possono essere utilizzate anche per creare o modificare una struttura di database.

eliminazione a catena

Nelle relazioni che applicano l'integrità referenziale tra tabelle, l'eliminazione a catena è un'opzione mediante la quale l'eliminazione di un record dalla tabella primaria provoca automaticamente l'eliminazione di tutti i record associati della tabella o delle tabelle esterne correlate. Se si stabilisce, ad esempio, una relazione tra una tabella (principale) Clienti e una tabella (esterna) Ordini con l'opzione di eliminazione a catena attivata, quando viene eliminato un record nella tabella Clienti, vengono eliminati anche tutti gli ordini associati al cliente in questione nella tabella esterna Ordini. Tale processo è ora definibile anche mediante il linguaggio DDL SQL.

equi-join

Join in cui i record di due tabelle vengono riuniti e aggiunti all'oggetto **Recordset** solo quando sono disponibili valori uguali nei campi collegati tramite join. Viene definito anche inner join.

espressione

Qualsiasi combinazione di operatori, costanti, valori letterali, funzioni e nomi di campi, controlli e proprietà che restituisce un singolo valore. È possibile utilizzare le espressioni come impostazioni per numerosi argomenti di azione e proprietà, per impostare criteri o definire campi calcolati nelle query.

espressione numerica

Qualsiasi espressione che dia come risultato un numero. Può essere una qualsiasi combinazione di variabili, costanti, funzioni e operatori che restituisca un numero.

funzione di aggregazione

Funzione, ad esempio **Somma**, **Conteggio**, **Media** e **Max**, che può essere utilizzata per calcolare totali. Per la creazione di espressioni e durante la programmazione è possibile utilizzare funzioni di aggregazione SQL (incluse le quattro elencate) e funzioni di aggregazione sui domini per determinare diverse statistiche.

conflitto di sincronizzazione

Situazione in cui gli utenti di due repliche hanno modificato lo stesso record. Uno dei record aggiornati viene selezionato come il "vincitore" e il record della replica "sconfitta" viene aggiornato con i dati del vincitore. L'altro record aggiornato viene inserito in una tabella dei conflitti nella replica sconfitta.

formato di rappresentazione Unicode

Set di caratteri a 16 bit gestito da Unicode Consortium che include tutti i caratteri necessari per visualizzare ogni lingua esistente. Unicode viene utilizzato sempre più diffusamente per semplificare il funzionamento multilingue del software.

indice cluster

L'ordine fisico delle righe coincide con quello indicizzato.

incremento

Parametro facoltativo utilizzato con i tipi di dati **Counter** o **Indentity**. Disponibile nel linguaggio DDL SQL solo quando si utilizza il provider Microsoft OLE DB per Jet. Si tratta del valore utilizzato per assegnare il successivo valore di una nuova riga di una tabella che include un tipo di dati **Counter** o **Indentity**. L'incremento è il secondo dei due parametri facoltativi per i tipi di dati **Counter** o **Indentity**.

indice

Riferimento incrociato dinamico di uno o più campi di dati di tabella (colonne) che consente di recuperare più rapidamente record specifici da una tabella. Quando vengono aggiunti, modificati o eliminati record, il sistema di gestione di database aggiorna automaticamente l'indice per riflettere le modifiche. In una tabella possono esistere diversi indici definiti per i dati.

inner join

Join in cui i record di due tabelle vengono combinati e aggiunti a un **Recordset** solo se i valori dei campi collegati tramite join soddisfano una condizione specificata. Ad esempio, un equi-join è un inner join in cui i valori dei campi collegati tramite join devono essere uguali.

integrità referenziale

Regole impostate per stabilire e preservare relazioni tra tabelle quando si aggiungono, modificano o eliminano record. L'applicazione dell'integrità referenziale impedisce agli utenti di aggiungere record a una tabella collegata tramite join per cui non esiste alcuna chiave primaria, di modificare valori in una tabella primaria, generando, di conseguenza, record orfani in una tabella collegata tramite join, e di eliminare record da una tabella primaria, se esistono record correlati corrispondenti. Se si seleziona l'opzione **dbRelationDeleteCascade** o **dbRelationUpdateCascade** per una relazione, il modulo di gestione di database Microsoft Jet consente modifiche ed eliminazioni, ma modifica o elimina i record correlati per assicurare che le regole siano ancora applicate. Microsoft Jet versione 4.X consente di esporre, mediante il provider Microsoft OLE DB Provider per Jet, un set molto più ampio di SQL ANSI, oltre all'azione referenziale ANSI NULL a catena.

join sinistro

Un join esterno sinistro include tutti i record della prima di due tabelle (quella di sinistra), anche se non sono presenti valori corrispondenti per i record della seconda (quella di destra).

join esterno sinistro

Join esterno in cui tutti i record del lato sinistro dell'operazione LEFT JOIN nell'istruzione SQL della query vengono aggiunti all'oggetto **Recordset**, anche se non sono presenti valori corrispondenti nel campo collegato tramite join della tabella di destra. I record della tabella di destra vengono combinati con quelli della tabella di sinistra solo se sono presenti valori corrispondenti nei campi collegati tramite join. Se un record del lato sinistro non presenta alcuna corrispondenza, viene collegata tramite join una riga di valori **Null** sul lato destro.

join esterno

Join in cui tutti i record di una tabella vengono aggiunti all'oggetto **Recordset**, anche se non esistono valori corrispondenti nel campo collegato tramite join della seconda tabella. I record della seconda tabella vengono combinati con quelli della prima tabella solo quando esistono valori corrispondenti nei campi collegati tramite join. Se non esiste alcuna corrispondenza, vengono collegati tramite join i record Null.

join destro

Un join esterno destro include tutti i record della seconda di due tabelle (quella di destra), anche se non esistono valori corrispondenti per i record della prima tabella (quella di sinistra). Ad esempio, è possibile utilizzare l'operazione LEFT JOIN con le tabelle Dipartimenti (a sinistra) e Dipendenti (a destra) per selezionare tutti i dipartimenti, inclusi quelli a cui non sono stati assegnati dipendenti. Per selezionare tutti i dipendenti, inclusi quelli che non sono stati assegnati ad alcun dipartimento, si utilizzerà l'operazione RIGHT JOIN.

join esterno destro

Join esterno in cui tutti i record del lato destro dell'operazione RIGHT JOIN nell'istruzione SQL della query vengono selezionati, anche se non sono presenti valori corrispondenti nel campo collegato tramite join della tabella di sinistra. I record della tabella di sinistra vengono combinati con quelli della tabella di destra solo se esistono valori corrispondenti nei campi collegati tramite join.

modalità SQL ANSI-92

Questa modalità consente di utilizzare una sintassi SQL estesa che garantisce maggiore compatibilità con SQL ANSI-92 e Microsoft SQL Server. La modalità è disponibile solo se si utilizza il provider Microsoft OLE DB per Jet. Non può essere impostata se si utilizza DAO. Le nuove caratteristiche si trovano principalmente in DDL. Anche il supporto per i caratteri jolly ANSI, rappresentati dal segno di percentuale (%) e dal carattere di sottolineatura (\_), fa parte della modalità SQL ANSI-92.

database multiutente

Database che consente a più utenti di accedere e di modificare contemporaneamente lo stesso insieme di dati. In alcuni casi, l'utente aggiuntivo può essere rappresentato da un'altra istanza dell'applicazione o da un'altra applicazione, in esecuzione nel sistema, che accede agli stessi dati della prima applicazione.

NULL a catena

Nelle relazioni che applicano l'integrità referenziale tra tabelle, il NULL a catena è un'opzione mediante la quale se si elimina una riga nella tabella primaria, viene aggiornata automaticamente con un valore NULL la chiave esterna di tutti i record associati della tabella o delle tabelle esterne correlate. Tale processo è ora definibile anche mediante il linguaggio DDL SQL. Se si stabilisce, ad esempio, una relazione tra una tabella (principale) Clienti e una tabella (esterna) Ordini con l'opzione NULL a catena attivata, quando viene eliminata la chiave primaria in un record della tabella clienti, verrà incluso un valore NULL anche in tutti gli ordini della tabella esterna Ordini associati al cliente in questione.

nome di correlazione

Definizione ANSI di un nome che fa riferimento a una colonna di una tabella o vista (query archiviata). È noto anche come alias.

normalizzare

Ridurre al minimo la duplicazione di informazioni in un database relazionale tramite una strutturazione efficiente delle tabelle.

Null

Valore che indica dati mancanti o sconosciuti. I valori **Null** possono essere immessi in campi le cui informazioni sono sconosciute e in espressioni e query. In Visual Basic, la parola chiave **Null** indica un valore **Null**. Alcuni campi, quali quelli della chiave primaria, non possono includere valori **Null**.

join

Operazione del database che consente di combinare alcuni o tutti i record di due o più tabelle. Sono esempi di join gli equi-join, gli inner join, i join esterni o i self-join. In genere, un join fa riferimento a un'associazione tra un campo di una tabella e un campo con lo stesso tipo di dati di un'altra tabella. È possibile creare un join con un'istruzione SQL. Quando si definisce una relazione tra due tabelle, è possibile creare un join specificando i campi della tabella primaria e di quella esterna. Quando si aggiunge una tabella a una query, è necessario creare un join tra i campi appropriati nell'istruzione SQL che definisce la query.

processo a catena

Processo di un'azione che ne avvia un'altra. Se viene definita, ad esempio, una relazione di aggiornamento a catena per due o più tabelle, l'aggiornamento della chiave primaria della tabella primaria avvierà automaticamente (a catena) modifiche nella tabella esterna. Tale processo è ora definibile anche mediante il linguaggio DDL SQL.

proprietà dei campi

Attributi di un campo che determinano il tipo di dati in esso incluso. Sono esempi di attributi **Dimensione** e **Tipo**.

parametro

Elemento che include un valore che è possibile modificare per influenzare i risultati della query. Ad esempio, una query che restituisce dati relativi a un dipendente può avere un parametro per il nome del dipendente

parola riservata

Parola che fa parte di un linguaggio, quale il linguaggio SQL. Le parole riservate includono i nomi di istruzioni, funzioni predefinite, tipi di dati, metodi, operatori e oggetti. Ne sono esempi SELECT, UPDATE, BETWEEN, SET e INSERT.

query di comando

Query che copia o modifica i dati. Le query di comando includono query di accodamento, di eliminazione, di creazione tabella e di aggiornamento. Le query di eliminazione e di aggiornamento modificano dati esistenti, quelle di accodamento e creazione tabella, invece, copiano dati esistenti. In SQL, tali query sono note come query DML e sono rappresentate dalle parole chiave DELETE, INSERT e UPDATE. Microsoft Jet supporta la parola chiave SELECT INTO, che viene anche considerata una query DDL. Le query di selezione, invece, restituiscono record di dati. Anche una query SQL pass-through può rappresentare una query di comando.

query di accodamento

Query di comando che aggiunge nuovi record alla fine di una tabella o una query esistente. Le query di accodamento non restituiscono record (righe). La sintassi SQL includerà la parola chiave INSERT.

query asincrona

Tipo di query in cui le query SQL vengono restituite immediatamente, anche se i risultati sono ancora in sospeso. In questo modo in un'applicazione possono continuare le altre attività di elaborazione mentre si attende il completamento della query.

query composta

Query composta da almeno una query di comando (query che copia o modifica dati) e una query di selezione (query che opera senza modificare dati).

query a campi incrociati

Query che calcola una somma, una media, un conteggio o un altro tipo di totale dei record e quindi raggruppa i risultati in base a due tipi di informazioni, uno lungo il lato sinistro di una griglia (intestazioni di riga) e l'altro nella parte superiore della griglia (intestazioni di colonna). La query Ordini trimestrali per prodotto del database di esempio Northwind è un esempio di query a campi incrociati.

query di definizione dati

Query specifica del linguaggio SQL che consente di creare, modificare o eliminare una tabella oppure di creare o di eliminare un indice di un database. Lo standard ANSI definisce tali query "query DDL" e stabilisce che i token utilizzati per il linguaggio sono CREATE, DROP e ALTER.

query di eliminazione

Query di comando che elimina un set di righe che soddisfano i criteri specificati. Una query di eliminazione non restituisce righe.

query di creazione tabella

Query di comando che consente di creare una nuova tabella.

query con parametri

Query che per essere eseguita richiede che siano specificati uno o più valori che servono da criteri, ad esempio Redmond per Città. Una query con parametri non è un tipo particolare di query, ma piuttosto aumenta la flessibilità di altre query.

query procedurale

Istruzione SQL che esegue una stored procedure.

query

Istruzione formalizzata per un database che consente di restituire un set di record o di eseguire una particolare azione su un set di record, secondo quanto specificato nella query. Ad esempio, l'istruzione query SQL seguente restituisce record: SELECT NomeAzienda FROM Editori WHERE Zona = 'CA'. È possibile creare ed eseguire query di selezione, di comando, a campi incrociati, con parametri e specifiche di SQL.

protezione

Serie di caratteristiche utilizzate per specificare o limitare l'accesso di utenti o gruppi di utenti specifici ai dati e agli oggetti di un database.

tipo di ordinamento

Principio per la creazione di una sequenza utilizzato per ordinare dati secondo un criterio alfabetico o numerico. Il tipo di ordinamento può essere crescente e decrescente.

query di unione

Query di selezione specifica di SQL che crea un oggetto contenente dati di tutti i record specificati di due o più tabelle, rimuovendo eventuali record duplicati. Per includere i duplicati, aggiungere la parola chiave ALL. Ad esempio, una query di unione delle tabelle Clienti e Fornitori restituisce una selezione che include tutti i fornitori che sono anche clienti.

query di aggiornamento

Query di comando che modifica un set di record in base a criteri specificati. Una query di aggiornamento non restituisce alcun record.

relazione uno-a-molti

Associazione fra due tabelle in cui il valore della chiave primaria di ogni record della tabella primaria corrisponde al valore del campo o dei campi corrispondenti di molti record nella tabella collegata oppure in cui il valore della chiave primaria di ogni record della tabella collegata corrisponde al valore del campo o dei campi corrispondenti di uno e solo un record nella tabella primaria.

relazione uno-a-uno

Associazione fra due tabelle in cui il valore della chiave primaria di ogni record della tabella primaria corrisponde al valore del campo o dei campi corrispondenti di uno e solo un record nella tabella collegata oppure in cui il valore della chiave primaria di ogni record della tabella collegata corrisponde al valore del campo o dei campi corrispondenti di uno e solo un record nella tabella primaria.

procedura

Simile a una vista tranne per il fatto che la sintassi SQL può consentire alle istruzioni SELECT (query che restituiscono righe) di passare parametri e può consentire l'utilizzo di istruzioni DML UPDATE, INSERT, DELETE e SELECT INTO (query che non restituiscono righe).

record

Set di dati correlati relativi a una persona, un luogo, un evento o un altro elemento. I dati delle tabelle sono memorizzati in record (righe) nel database. Ogni record è composto da una serie di campi correlati (colonne). Ogni campo definisce un attributo di informazioni per il record. Nel suo complesso, un record definisce una specifica unità di informazioni recuperabili in un database. I record vengono inoltre definiti righe.

relazione

Associazione stabilita tra campi comuni (colonne) di due tabelle. Una relazione può essere di tipo uno-a-uno, molti-a-molti o uno-a-molti.

replica

Copia di un database che include tabelle, query, maschere, report, macro e moduli. Una replica fa parte di un set di repliche e può essere sincronizzata con le altre repliche del set. Le modifiche apportate ai dati di una tabella replicata di una replica vengono inviate e applicate agli altri membri del set di repliche.

riga

Gruppo di colonne o campi correlati utilizzato per memorizzare dati. Riga è sinonimo di record. Una tabella è composta da zero o più righe di dati.

set di caratteri ANSI

Set di caratteri a 8 bit utilizzato da Microsoft? Windows? per rappresentare un massimo di 256 caratteri, da zero a 255, utilizzando la tastiera. Il set di caratteri ASCII è un sottoinsieme del set di caratteri ANSI.

set di caratteri ASCII

Set di caratteri a 7 bit ASCII (American Standard Code for Information Interchange) ampiamente utilizzato per rappresentare lettere e simboli di una tastiera US standard. Il set di caratteri ASCII corrisponde ai primi 128 caratteri (da 0 a 127) del set di caratteri ANSI.

base

Parametro facoltativo utilizzato con i tipi di dati **Counter** o **Indentity**. Disponibile nel linguaggio DDL SQL solo quando si utilizza il provider Microsoft OLE DB per Jet. Viene denominato anche valore iniziale. La base è il primo dei due parametri facoltativi dei tipi di dati **Counter** o **Identity**. L'istruzione SQL seguente è un esempio in cui vengono utilizzati i parametri: CREATE TABLE Clienti (IDCliente IDENTITY (100, 10) CONSTRAINT cpClienti PRIMARY KEY, NomeCl VARCHAR(10), CognomeCl VARCHAR(15)); La colonna IDCliente della prima riga avrà un valore di 100. La seconda riga avrà un valore di 110. Se vengono utilizzati i parametri facoltativi e questi non includono valori di (1,1), la base non verrà reimpostata sul valore massimo della tabella. Questo comportamento è diverso da quello delle precedenti versioni di Jet. Se i parametri facoltativi non vengono passati o non viene utilizzato un valore di (1,1), quando il database è compattato, la base verrà reimpostata sul valore massimo della colonna **Counter** o **Identity** della tabella.

query di selezione

Query che esegue un'interrogazione sui dati memorizzati nelle tabelle e restituisce un oggetto **Recordset** senza modificare i dati. Dopo aver recuperato i dati dell'oggetto **Recordset**, è possibile esaminare e apportare modifiche ai dati delle tabelle sottostanti. Le query di comando, invece, possono apportare modifiche ai dati, ma non restituiscono record di dati.

SQL (Structured Query Language)

Linguaggio utilizzato per l'esecuzione di query, l'aggiornamento e la gestione di database relazionali. È possibile utilizzare SQL per recuperare, ordinare e filtrare dati specifici da estrarre dal database. È possibile utilizzare istruzioni SQL SELECT ovunque sia accettato un nome di tabella, di query o di campo.

tabella esterna

Tabella che include una chiave esterna per un'altra tabella del database. Una tabella esterna viene utilizzata, in genere, per stabilire o applicare l'integrità referenziale e si trova di norma sul lato "molti" di una relazione uno-a-molti. Una tabella di ordini dei clienti è un esempio di tabella esterna.

tabella

Unità base di memorizzazione dati di un database relazionale. In una tabella i dati sono memorizzati in record (righe) e campi (colonne) e sono in genere relativi a una particolare categoria di elementi, ad esempio dipendenti o parti. Una tabella viene denominata anche tabella di base. Un oggetto **Recordset** di tipo tabella è una rappresentazione logica di una tabella di base in un database.

tipo di dati Contatore

Attributo di un campo che consente di memorizzare automaticamente un numero univoco per ogni record aggiunto a una tabella. In un campo Contatore viene utilizzato il tipo di dati **Long** e la proprietà **Attributes** è impostata su **dbAutoIncrement**. I numeri generati da un campo Contatore non possono essere modificati, a meno che non sia stata concessa una speciale autorizzazione per la tabella mediante l'istruzione Grant o Revoke della sintassi DDL SQL. Il campo Contatore è noto anche come tipo di dati Contatore o Identità.

tipo di dati Big Integer

Tipo di dati che consente di memorizzare un valore numerico esatto con segno, con precisione 19 (con segno) o 20 (senza segno) e scala 0 (con segno: -263 = n = 263-1; senza segno: 0 = n = 264-1).

tipo di dati Binary

Tipo di dati che consente di memorizzare dati binari a lunghezza fissa. La lunghezza massima è di 255 byte.

tipo di dati Boolean

Valore **True**/**False** o yes/no. I valori **booleani** vengono memorizzati, in genere, in campi di bit di un database di Microsoft Jet; tuttavia, in alcuni database questo tipo di dati non è supportato in modo diretto.

tipo di dati Byte

Tipo di dati fondamentale che consente di memorizzare numeri interi positivi di piccole dimensioni compresi tra 0 e 255.

tipo di dati Char

Tipo di dati che consente di memorizzare una stringa di caratteri a lunghezza fissa. La lunghezza viene impostata mediante la proprietà **Size**.

tipo di dati Valuta

Tipo di dati utile per calcoli monetari o per calcoli a virgola fissa in cui la precisione è estremamente importante. Questo tipo di dati viene utilizzato per memorizzare numeri con un massimo di 15 cifre a sinistra della virgola decimale e 4 a destra. Poiché nel tipo di dati **Valuta** vengono utilizzati valori discreti per tutti gli importi, non è possibile utilizzare l'arrotondamento binario come fattore per il calcolo dei totali.

tipo di dati

Caratteristiche di una variabile che determinano quale tipo di dati può essere contenuto nella variabile. I tipi di dati includono: **Big Integer**, **Binary**, **Byte**, **Boolean**, **Char**, **Currency**, **Date**, **Decimal**, **Double**, **Float**, **Guid**, **Integer**, **Long**, **Long Binary** (**Oggetto OLE**), **Memo**, **Numeric**, **Single**, **string**, **Text**, **Time**, **TimeStamp**, **VarBinary**, **Variant** (impostazione predefinita), tipi definiti dall'utente (creati con l'istruzione **Type**) e tipi di dati oggetto, che includono tipi di dati oggetto definiti dall'host e tipi di oggetti DAO.

tipo di dati Decimal

Tipo di dati che consente di memorizzare un valore numerico esatto con segno, con precisione p e scala s (1 = p =15; 0 = s = p).

tipo di dati Double

Tipo di dati fondamentale per la memorizzazione di numeri a virgola mobile e precisione doppia in formato IEEE. Una variabile **Double** viene memorizzata come numero a 64 bit (8 byte) nell'intervallo compreso tra -1,79769313486231E308 e -4,94065645841247E-324 per i valori negativi, 4,94065645841247E-324 e 1,79769313486231E308 per i valori positivi e 0.

tipi di dati dei campi

Di seguito sono riportati i tipi di dati **Field**: Big Integer, Binary,Boolean, Byte, Char, Currency, Date/Time, Decimal, Double, Float, GUID, Integer, Long, Long Binary (OLE Object), Memo, Numeric, Single, Text, Time, TimeStamp e VarBinary.

tipo di dati Float

Tipo di dati che consente di memorizzare un valore numerico approssimato con segno, con precisione mantissa di 15 (zero o valore assoluto da 10-308 a 10308).

tipo di dati Long Binary

Tipo di campo che consente di memorizzare 2,14 GB di dati. I campi Long Binary possono contenere qualsiasi tipo di dati binari. In Microsoft Access questo tipo di dati viene denominato anche Oggetto OLE.

tipo di dati Long

Tipo di dati fondamentale per la memorizzazione di numeri interi lunghi. Una variabile **Long** viene memorizzata come numero a 32 bit (4 byte) compreso tra -2.147.483.648 e 2.147.483.647.

tabella lato "molti" di relazioni uno-a-molti

Tabella che non viene visualizzata come lato "uno" di un join in una query.

tipo di dati Memo

Tipo di dati di un campo. I campi di questo tipo possono includere fino a 2,14 GB di dati di testo.

tipo di dati Numeric

Tipo di dati che consente di memorizzare un valore numerico esatto con segno, con precisione p e scala s (1 = p =15; 0 = s = p).

tipo di dati Single

Tipo di dati che consente la memorizzazione di numeri a virgola mobile e precisione singola in formato IEEE. Una variabile **Single** viene memorizzata come numero a 32 bit (4 byte) compreso tra -3,402823E38 e -1,401298E-45 per i valori negativi, tra 1,401298E-45 e 3,402823E38 per i valori positivi e 0.

tipo di dati Testo(n) (con specifica della lunghezza)

Tipo di dati di campo. I campi di Testo possono includere fino a 255 caratteri o il numero di caratteri specificato nella proprietà**Size** dell'oggetto **Field** se questo è inferiore a 255. Se la proprietà **Size** del campo di tipo Testo è impostata su 0, potranno essere memorizzati fino a 255 caratteri di dati.

tipo di dati Testo (senza specifica della lunghezza)

Tipo di dati di campo. I campi di tipo Testo definiti senza una lunghezza esplicita sono identici ai campi Memo e possono includere fino a 2,14 GB di caratteri.

tipo di dati relativo all'ora

Tipo di dati che consente di memorizzare un valore orario. Il valore dipende dall'impostazione dell'orologio dell'origine dati.

vincolo CHECK

Concetto SQL ANSI-92 che consente la creazione di regole business valide per un'intera tabella o più tabelle. È possibile, ad esempio, creare una tabella Cliente e una tabella Ordine e impostare in quest'ultima un vincolo CHECK per impedire che gli ordini relativi a un cliente superino un limite di credito definito nella tabella Cliente.

vincolo

Limitazione dei valori che possono essere immessi in un campo.

valore letterale di data

Sequenza di caratteri con un formato valido racchiusa tra simboli di cancelletto (#). I formati validi sono quelli specificati dalle impostazioni internazionali o il formato di data universale. Ad esempio, #12/31/92# è il valore letterale di data che rappresenta il 31 dicembre 1992, quando le impostazioni internazionali dell'applicazione corrispondono a Inglese (Stati Uniti). Utilizzare i valori di data letterali per ottimizzare la portabilità per le lingue nazionali.

vista

Definizione ANSI di tabella virtuale. Vista è sinonimo di query archiviata in Access e una query archiviata è rappresentata da un'istruzione SELECT senza parametri. Una vista è una definizione memorizzata che viene materializzata durante l'esecuzione. È diversa da una tabella in quanto in essa non vengono fisicamente memorizzati dati. Restituisce solo dati. Una vista può essere creata con la nuova sintassi SQL ANSI CREATE VIEW, esposta mediante il provider Microsoft OLE DB per Jet. In alternativa, è possibile creare una routine che consenta l'utilizzo di istruzioni SQL DML e istruzioni SELECT con parametri. A questo scopo, è possibile utilizzare la nuova sintassi CREATE PROCEDURE, esposta mediante il provider Microsoft OLE DB per Jet.