SQL

Alice Colella

Junior Developer@icubedsrl

Alice.Colella@icubed.it

Roberto Ajolfi

Senior Developer@icubedsrl

Roberto.ajolfi@icubed.it

Database

Database: una struttura di dati organizzati seguendo un modello.

Per accedere a uno o più database di usa il DBMS:

Data**b**ase **M**anagement **S**ystem

Un set di software che permettono l'accesso, l'aggiornamento e eventuale recupero di dati.

icubed

Modelli di database

1. Relational

   Struttura tramite tabelle composte da campi e record.

   Relazioni: interne alla tabella e tra diverse tabelle

   ->Gestiti da RDBMS

2. Object Oriented

   Struttura tramite oggetti, usata soprattutto in ambito documentale (Json, XML..)

   ->Gestiti da ODBMS / OODBMS

3. Object-Relational

   Struttura mista

SQL – Cos’è

Structured Query Language

è un linguaggio standard per l'accesso e la manipolazione di database.

SQL – a cosa serve

- SQL può eseguire query su un database
- SQL può recuperare i dati da un database
- SQL può inserire record in un database
- SQL può aggiornare i record in un database
- SQL può eliminare i record da un database

SQL – a cosa serve

- SQL può creare nuovi database
- SQL può creare nuove tabelle in un database
- SQL può creare stored procedure in un database
- SQL può creare viste in un database
- SQL può impostare autorizzazioni per tabelle, procedure e viste

SQL è uno standard, ma …

Sebbene SQL sia uno standard ANSI / ISO, esistono diverse versioni del linguaggio SQL.

Tuttavia, per essere conformi allo standard ANSI, supportano tutti almeno i comandi principali in modo simile.

SQL - Data Definition Language

Data Definition Language (DDL) si occupa di schemi e descrizioni di database, di come i dati dovrebbero risiedere nel database.

- CREATE: creare il database e i suoi oggetti (tabelle, indici, viste, procedura di memorizzazione, funzioni e trigger)
- ALTER: modifica la struttura del database esistente
- DROP: elimina gli oggetti dal database
- TRUNCATE: rimuove tutti i record da una tabella, inclusi tutti gli spazi allocati per i record
- COMMENT: aggiungi commenti al dizionario dei dati
- RENAME: rinomina un oggetto

SQL - Data Manipulation Language

Data Manipulation Language (DML) si occupa della manipolazione dei dati e include le istruzioni SQL più comuni come SELECT, INSERT, UPDATE, DELETE ecc. E viene utilizzato per archiviare, modificare, recuperare, eliminare e aggiornare i dati nel database.

- SELECT: recupera i dati da un database
- INSERT: inserire i dati in una tabella
- UPDATE: aggiorna i dati esistenti all'interno di una tabella
- DELETE - Elimina tutti i record da una tabella del database
- MERGE - Funzionamento UPSERT (inserire o aggiornare)
- CALL: chiama un sottoprogramma PL / SQL o Java
- EXPLAIN PLAN - interpretazione del percorso di accesso ai dati
- LOCK TABLE - controllo della concorrenza

SQL - Data Control Language

Data Control Language (DCL) include comandi come GRANT e riguarda principalmente diritti, autorizzazioni e altri controlli del sistema di database.

- GRANT: consente agli utenti di accedere ai privilegi del database
- REVOKE: revoca agli utenti i privilegi di accesso forniti utilizzando il comando GRANT

SQL - Transaction Control Language

Transaction Control Language (TCL) si occupa delle transazioni all'interno di un database.

COMMIT: commette una transazione

ROLLBACK: rollback di una transazione in caso di errore

SAVEPOINT: per ripristinare i punti di transazione all'interno dei gruppi

SET TRANSACTION: specifica le caratteristiche per la transazione

SQL Data Types (alcuni)

Data Type	Definition
NCHAR(N)	Stringa con lunghezza fissa N
NVARCHAR(N)	Stringa di lunghezza variabile
BIT	0-1
INT	Numeri non decimali
DECIMAL(p,s)	Numeri decimali formati da p cifre di cui s decimali
DATE	Data
TIME	Orario
MONEY	Valuta

Demo

Accedere a SQL Server

Design of a Relational DB

icubed

SELECT

L'istruzione *SELECT* viene utilizzata per selezionare i dati da un database.

Gli operatori *UNION*, *EXCEPT* e *INTERSECT* possono essere utilizzati per combinare i risultati in un set di risultati.

```
SELECT *
FROM table_name;
```

```
SELECT column1, column2, ...
FROM table_name;
```

```
SELECT DISTINCT column1, ...
FROM table_name;
```

INSERT

```
INSERT INTO table_name (column1, column2, column3, ...)
        VALUES (value1, value2, value3, ...);

            INSERT INTO table_name
        VALUES (value1, value2, value3, ...);
```

L’istruzione INSERT aggiunge una o più righe a una tabella.

È anche possibile inserire dati solo in colonne specifiche.

UPDATE

```
UPDATE table_name SET column1 = value1, ...

UPDATE table_name SET column1 = value1, ...
WHERE condition;
```

L'istruzione UPDATE viene utilizzata per modificare i record esistenti in una tabella.

Nota: fare attenzione quando si aggiornano i record in una tabella! La clausola WHERE specifica quali record devono essere aggiornati. Se si omette la clausola WHERE, tutti i record nella tabella verranno aggiornati!

DELETE

```
DELETE FROM table_name;

DELETE FROM table_name WHERE condition;
```

L'istruzione UPDATE viene utilizzata per eliminare record esistenti in una tabella.

Nota: fare attenzione quando si eliminano i record in una tabella! La clausola WHERE specifica quali record devono essere eliminati. Se si omette la clausola WHERE, tutti i record nella tabella verranno eliminati!

Demo

Inserire, aggiornare e eliminare dati

Select

icubed

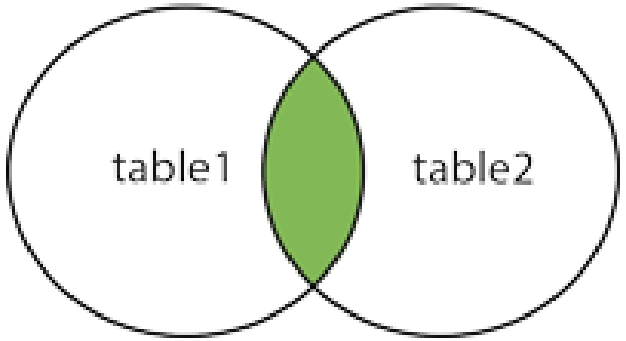
JOIN

Una clausola JOIN viene utilizzata per combinare righe da due o più tabelle, in base a una colonna correlata tra loro.

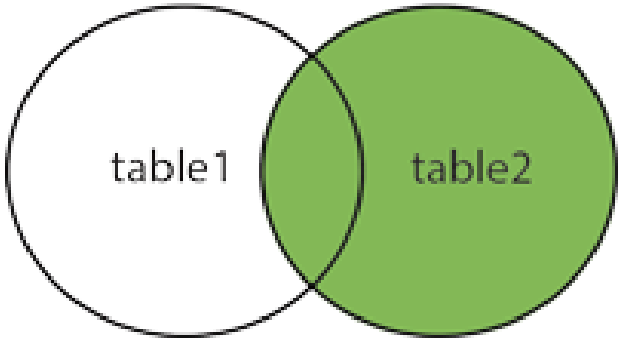
```
SELECT column1, column2, ...
FROM table1
INNER [ LEFT / RIGHT / FULL OUTER ] JOIN table2
  ON table1.column_name = table2.column_name;
```

JOIN

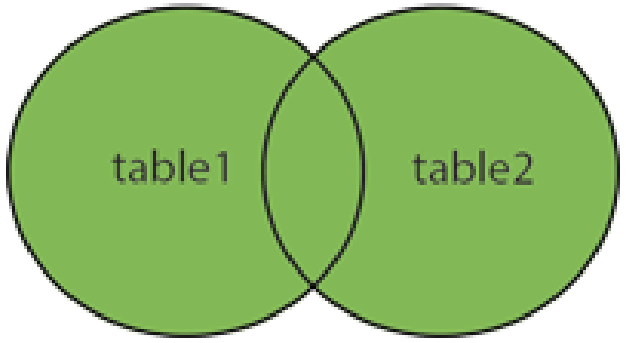
INNER JOIN



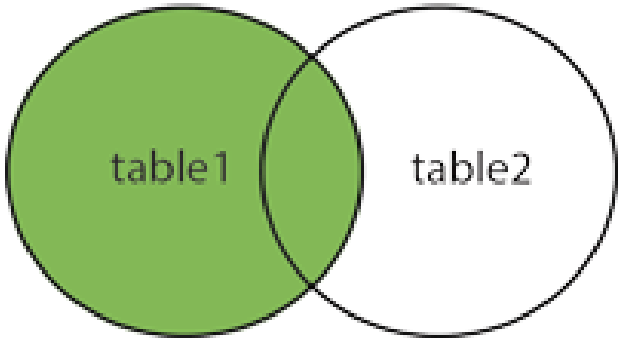
RIGHT JOIN



FULL OUTER JOIN



LEFT JOIN



GROUP BY

L'istruzione GROUP BY raggruppa righe con gli stessi valori in righe di riepilogo, ad esempio "trova il numero di clienti in ciascun paese".

```
SELECT column1, ...
  FROM table_name
GROUP BY column_name
```

L'istruzione GROUP BY viene spesso utilizzata con funzioni aggregate (COUNT, MAX, MIN, SUM, AVG) per raggruppare il set di risultati per una o più colonne.

GROUP BY

- **MIN()** restituisce il valore più piccolo della colonna selezionata

```
SELECT MIN(column1)
  FROM table_name
GROUP BY column_name
```

- **MAX()** restituisce il valore più grande della colonna selezionata

```
SELECT MAX(column1)
  FROM table_name
GROUP BY column_name
```

- **COUNT()** restituisce il numero di righe che corrisponde a un criterio specificato

```
SELECT COUNT(column1)
  FROM table_name
GROUP BY column_name
```

GROUP BY

- **AVG()** restituisce il valore medio di una colonna numerica

```
SELECT AVG(column1)
  FROM table_name
GROUP BY column_name
```

- **SUM()** restituisce la somma totale di una colonna numerica

```
SELECT SUM(column1)
  FROM table_name
GROUP BY column_name
```

HAVING

La clausola HAVING è stata aggiunta a SQL perché non è possibile utilizzare la parola chiave WHERE con le funzioni di aggregazione.

```
SELECT column1, MAX(column2), ...
    FROM table_name
    GROUP BY column1
    HAVING COUNT(column2) > 0
```

ORDER BY

La parola chiave ORDER BY viene utilizzata per ordinare il set di risultati in ordine crescente o decrescente.

La parola chiave ORDER BY ordina i record in ordine crescente per impostazione predefinita.

Per ordinare i record in ordine decrescente, utilizzare la parola chiave DESC.

```
SELECT column1, column2, ...
FROM table1
ORDER BY column1, column2 DESC;
```

IN

L'operatore IN consente di specificare più valori in una clausola WHERE.

È una scorciatoia per sostituire più condizioni OR.

```
SELECT column1, column2, ...
FROM table1
WHERE column_name = value1 OR column_name = value2 OR ...;
```

```
SELECT column1, column2, ...
FROM table1
WHERE column_name IN (value1, value2, ...);
```

BETWEEN

L'operatore BETWEEN seleziona i valori all'interno di un determinato intervallo. I valori possono essere numeri, testo o date.

È un operatore inclusivo, ovvero i valori di inizio e fine sono inclusi nell'intervallo.

```
SELECT column1, column2, ...
FROM table1
WHERE column_name BETWEEN value1 AND value2;
```

EXISTS

L'operatore EXISTS viene utilizzato per verificare l'esistenza di qualsiasi record in una sottoquery.

Assume il valore `true` se la sottoquery restituisce uno o più record, altrimenti `false`.

```
SELECT column1, column2, ...
FROM table1
WHERE EXISTS
(SELECT column_name FROM table2 WHERE condition);
```

CASE

L'istruzione CASE restituisce il valore corrispondente alla prima delle condizioni specificate che risulta vera (come nel costrutto SWITCH() di C#).

```
SELECT CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END AS column_name, ...
FROM table1;
```

Quindi, una volta che una condizione è vera, interrompe la lettura e restituirà il risultato. Se nessuna condizione è vera, restituisce il valore nella clausola ELSE.

Demo

Join

Order By / Group by

Having

icubed

Funzioni

- SQL Server ha molte funzioni integrate.
- Vengono utilizzate per manipolare
  - Stringhe (CONCAT, FORMAT …)
  - Numeri (FLOOR, CEILING, ROUND …)
  - Date (DATEDIFF, DATEPART …)
  - Effettuare conversioni (CAST, CONVERT, …)
  - … and many more!

È possible anche definire delle proprie funzione (custom function).

Gestire i valori NULL

La funzione ISNULL consente di restituire un valore alternativo quando un'espressione è NULL.

Il valore alternativo deve essere di un tipo convertibile implicitamente nel tipo della colonna su cui va ad operare la funzione.

```
SELECT column1, ISNULL(column2, ''), ...
FROM table1;
```

Funzioni

Per creare una Funzione:

```
CREATE FUNCTION ufnprocedure_name(@param1 type, ...)
RETURNS [type | TABLE]
AS
BEGIN
      sql_statement ...
      RETURN value;
END;
```

Per invocare una Funzione:

```
    SELECT column1, dbo.ufnprocedure_name(param1) AS alias
                          FROM table1
```

Demo

Function

User-defined Function

icubed

Stored Procedures

Una Stored Procedure è un codice SQL che è possibile salvare e che quindi può essere riutilizzato più volte.

Se si dispone di una query SQL che si scrive più e più volte, conviente salvarla come Stored Procedure.

È inoltre possibile passare dei parametri a una Stored Proc e definire un valore ritornato.

Stored Procedures

Per creare una Stored Procedure:

```
CREATE PROCEDURE procedure_name @param1 type, @param2 type, ...
                                AS
                            sql_statement ...
                              RETURN value;
                                ...
                                GO;
```

Per invocare una Stored Procedure:

```
        @retval = EXEC procedure_name @param1=value, ...;
```

Gestione degli Errori

Il costrutto `TRY CATCH` consente di gestire le eccezioni in SQL Server.
Per utilizzare il costrutto `TRY CATCH`

- inserire un gruppo di istruzioni SQL, che potrebbero causare un'eccezione, in un blocco `BEGIN TRY ... END TRY`
- Quindi si utilizza un blocco `BEGIN CATCH ... END CATCH` immediatamente dopo il blocco `TRY`

Gestione degli Errori

All'interno del blocco CATCH è possibile utilizzare le seguenti funzioni per ottenere informazioni dettagliate sull'errore che si è verificato:

- `ERROR_LINE()`
- `ERROR_MESSAGE()`
- `ERROR_PROCEDURE()`
- `ERROR_NUMBER()`
- `ERROR_SEVERITY()`
- `ERROR_STATE()`

Gestione degli Errori

```
BEGIN TRY

... sql_statements ...

END TRY
BEGIN CATCH

SELECT ERROR_NUMBER(),ERROR_SEVERITY(),
ERROR_STATE(),ERROR_PROCEDURE(),ERROR_LINE(),ERROR_MESSAGE();

END CATCH
```

Demo

Stored Procedures

icubed

Transazioni

Una transazione è un'unità di lavoro che va trattata come "un tutto". Deve avvenire per intero o per niente.

Un esempio classico è il trasferimento di denaro da un conto bancario a un altro:

- prelevare l'importo dall'account di origine
- quindi depositarlo sull'account di destinazione

L'operazione deve riuscire a pieno. Se ci si ferma a metà strada, i soldi andranno persi ...

```
beginTransaction;

accountB += 100;
accountA -= 100;

endTransaction;
```

Transazioni

Le transazioni sono caratterizzate da quattro proprietà chiamate proprietà **ACID**. Per superare questo test ACID, una transazione deve essere Atomica, Coerente, Isolata e Durevole.

- *Atomico*: Tutti i passaggi della transazione dovrebbero avere esito positivo o negativo insieme
- *Consistenza*: La transazione porta il database da uno stato stabile a un nuovo stato stabile
- *Isolamento*: Ogni transazione è un'entità indipendente
- *Durevolezza*: i risultati delle transazioni impegnate sono permanenti

Transazioni

Sono possibili solo due esiti di una transizione:

- **COMMIT**: l'intera unità di lavoro è stata completata con successo. Tutte le modifiche applicate ai dati vengono confermate e il database passa con successo ad un nuovo stato 'stabile'
- **ROLLBACK**: uno o più operazioni dell'unità di lavoro sono fallite. Tutte le operazione completate con successo vengono annullate e il database ritorna allo stato 'stabile' iniziale

Transazioni

Le transazioni esplicite iniziano con l'istruzione `BEGIN TRANSACTION` e terminano con l'istruzione `COMMIT` o `ROLLBACK`

```
BEGIN TRANSACTION
[ transaction_name | @tran_name_variable
    [ WITH MARK [ 'description' ] ] ]

… sql statements …

COMMIT;

ROLLBACK;
```

Transazioni

Per utilizzare una Transazione in una Stored Procedure:

```
CREATE PROCEDURE procedure_name @param1 type, @param2 type, ...
AS
BEGIN
	BEGIN TRANS
	BEGIN TRY
		... sql_statements ...
		IF @@ERROR ROLLBACK;
		COMMIT;
	END TRY
	BEGIN CATCH
		ROLLBACK;
	END CATCH
END
```

Demo

Transaction

icubed

ADO.NET

Alice Colella

Junior Developer@icubedsrl

Alice.Colella@icubed.it

Roberto Ajolfi

Senior Developer@icubedsrl

Roberto.ajolfi@icubed.it

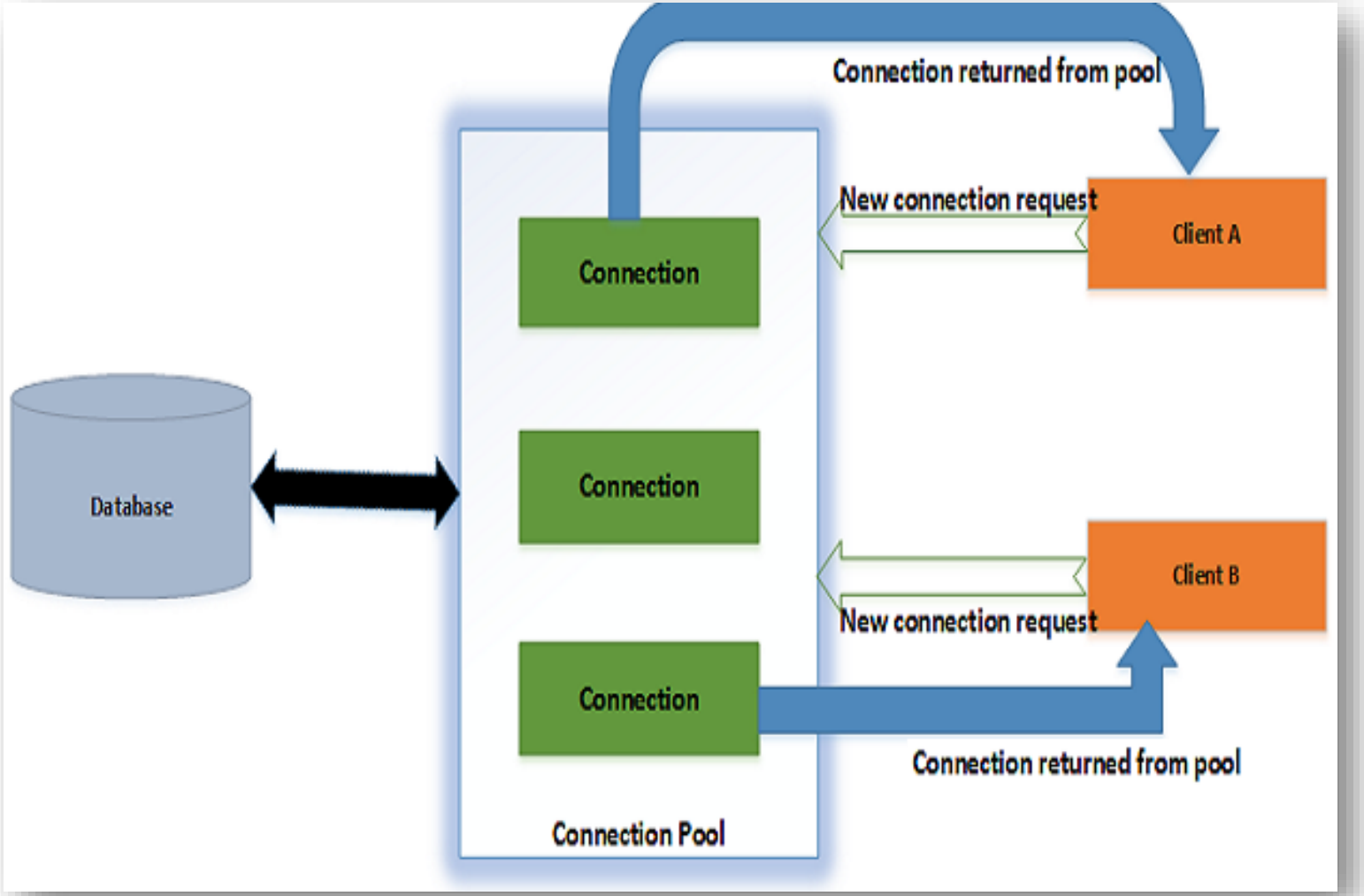
ADO.NET

- **ADO.NET** è una tecnologia di accesso ai dati di Microsoft
- È costituito da un insieme di componenti software che i programmatori possono utilizzare per accedere ai dati e ai servizi dati da un database
- Fornisce un accesso coerente
  - alle origini dati come SQL Server
  - alle origini dati esposte tramite OLE DB e ODBC

ADO.NET - Provider

- ADO.NET include i **Provider** di dati per
  - la connessione a un database
  - l'esecuzione di comandi
  - il recupero dei risultati
- I **provider** di dati .NET Framework forniscono una serie di oggetti per gestire l'accesso ai dati
  - L'oggetto **Connection** fornisce connettività a un'origine dati tramite una *Connection String*
  - L'oggetto **Command** consente l'accesso ai comandi del database per restituire dati, modificare dati, eseguire procedure memorizzate e inviare o recuperare informazioni sui parametri

ADO.NET - Connection Pool



ADO.NET - Connection Pool

- Ogni volta che un utente chiama OPEN() su una connessione, il pooler cerca una connessione disponibile nel pool
  - Se è disponibile una connessione in pool, la restituisce al chiamante invece di aprire una nuova connessione.
- Quando l'applicazione chiama CLOSE() sulla connessione, il pooler la restituisce al set di connessioni attive in pool anziché chiuderlo
- Una volta che la connessione viene restituita al pool, è pronta per essere riutilizzata alla successiva chiamata Open

ADO.NET – Connection String

Una stringa di connessione contiene le informazioni di inizializzazione fondamentali per creare una connessione con un database.

```
Server=tcp:platone.database.windows.net,1433; Initial Catalog=University; Persist
Security Info=False; User ID={your user}; Password={your_password};
MultipleActiveResultSets=False; Encrypt=True; TrustServerCertificate=False;
Connection Timeout=30;
```

ADO.NET - Modalità

ADO.NET consente l'accesso ai dati in due modalità distinte:

- **Connected Mode**
  (Connection, Commands, DataReader …)

- **Disconnected Mode**
  - *(Connection, Adapter, DataSet, DataTable ...)*

ADO.NET – Connected Mode

Il **Connected Mode** fornisce

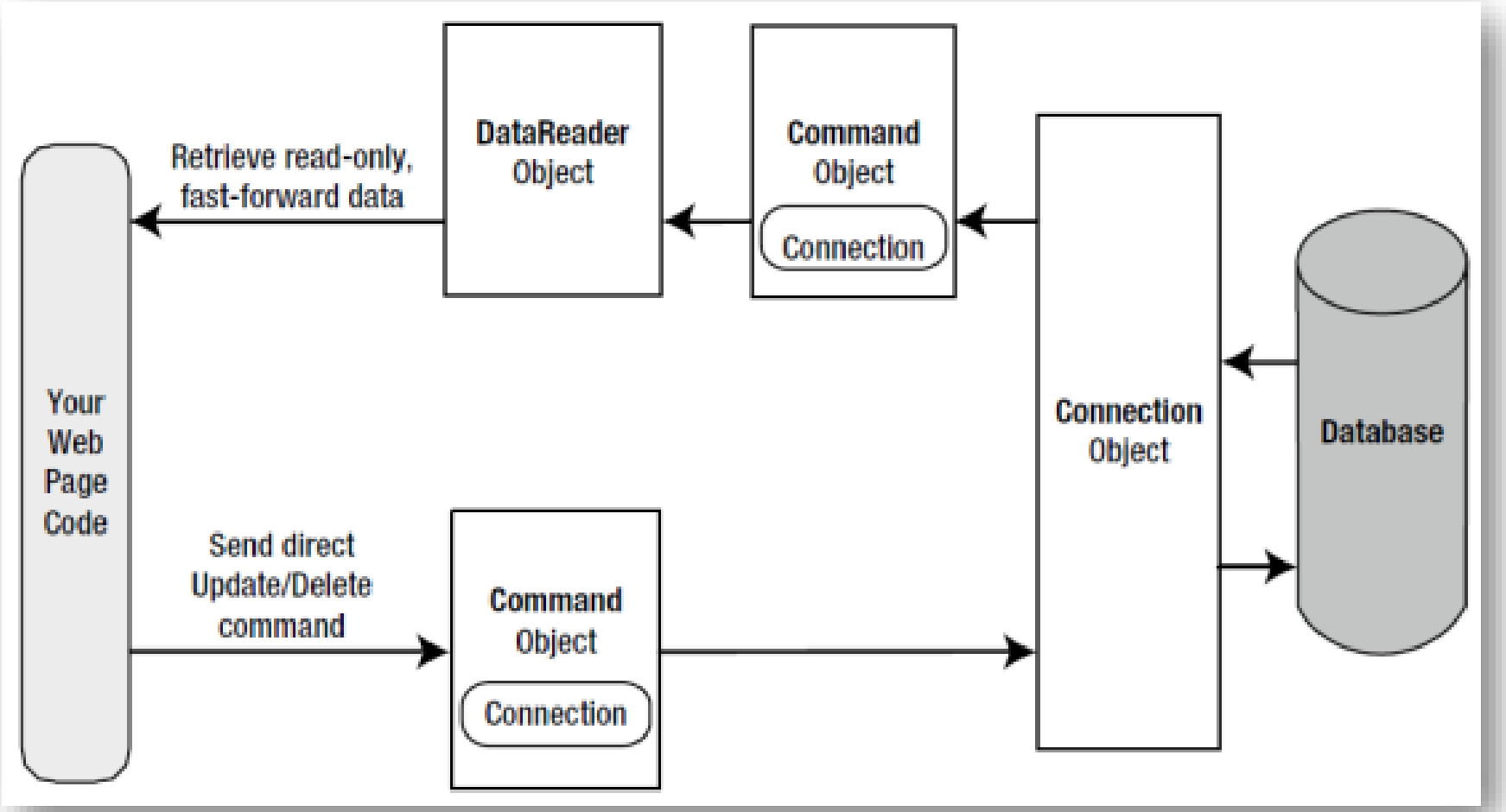
- accesso di sola lettura (e forward-only) ai dati nell'origine dati
- la possibilità di eseguire comandi sull'origine dati

ADO.NET – Connected Mode

Principali classi utilizzate in Connected Mode:

- Connection (*SqlConnection*)
- Command (*SqlCommand*)
- DataReader (*SqlDataReader*)
- Parameter (*SqlParameter*)

ADO.NET – Connected Mode



ADO.NET – Connected Mode

1. Creare Connessione
2. Aprire Connessione
3. Creare Command
    1. Creare Paramentri se necessario
4. Eseguire Command (DataReader/NonQuery/Scalar)
5. Lettura Dati a schermo
6. Chiudere Connessione

Demo

Connected Mode

icubed

ADO.NET – Disconnected Mode

Il **Disconnected Mode** consente di

- manipolare i dati recuperati dall'origine dati
- successivamente riconciliarli con l'origine dati

Le classi disconnesse forniscono un modo comune di lavorare con i dati disconnessi, indipendentemente dall'ambiente di origine data.

ADO.NET – Disconnected Mode

Principali classi utilizzate in Disconnected Mode:

- DataSet
- DataTable
- DataColumn
- DataRow
- Constraint
- DataRelation

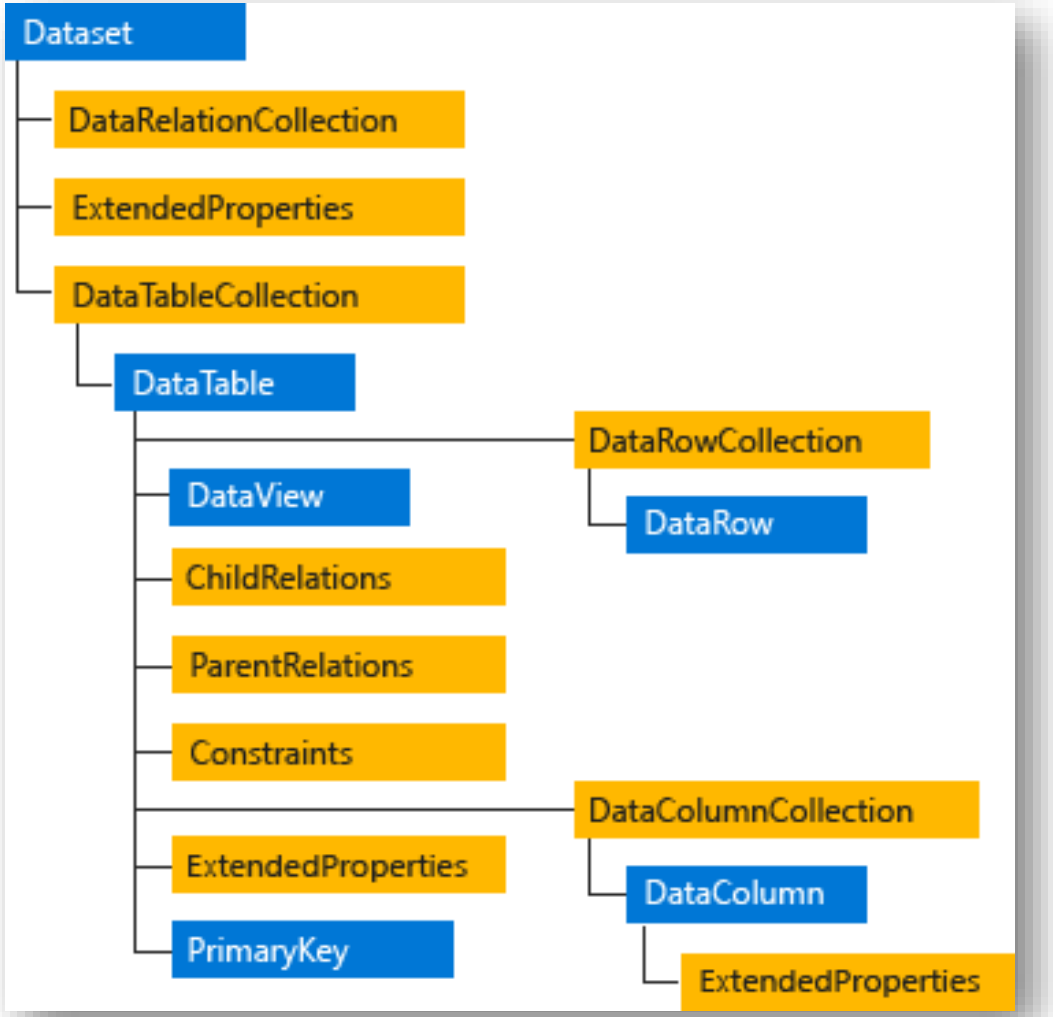
ADO.NET - DataSet

Il **DataSet** è progettato esplicitamente per l'accesso ai dati indipendentemente da qualsiasi origine dati: può essere utilizzato con origini dati multiple e diverse o utilizzato per gestire i dati locali per l'applicazione.

Il **DataSet** contiene una raccolta di uno o più oggetti **DataTable** costituiti da

- righe e colonne di dati
- informazioni relative a chiave primaria, chiavi esterne, vincoli e relazione sui dati

ADO.NET - DataSet



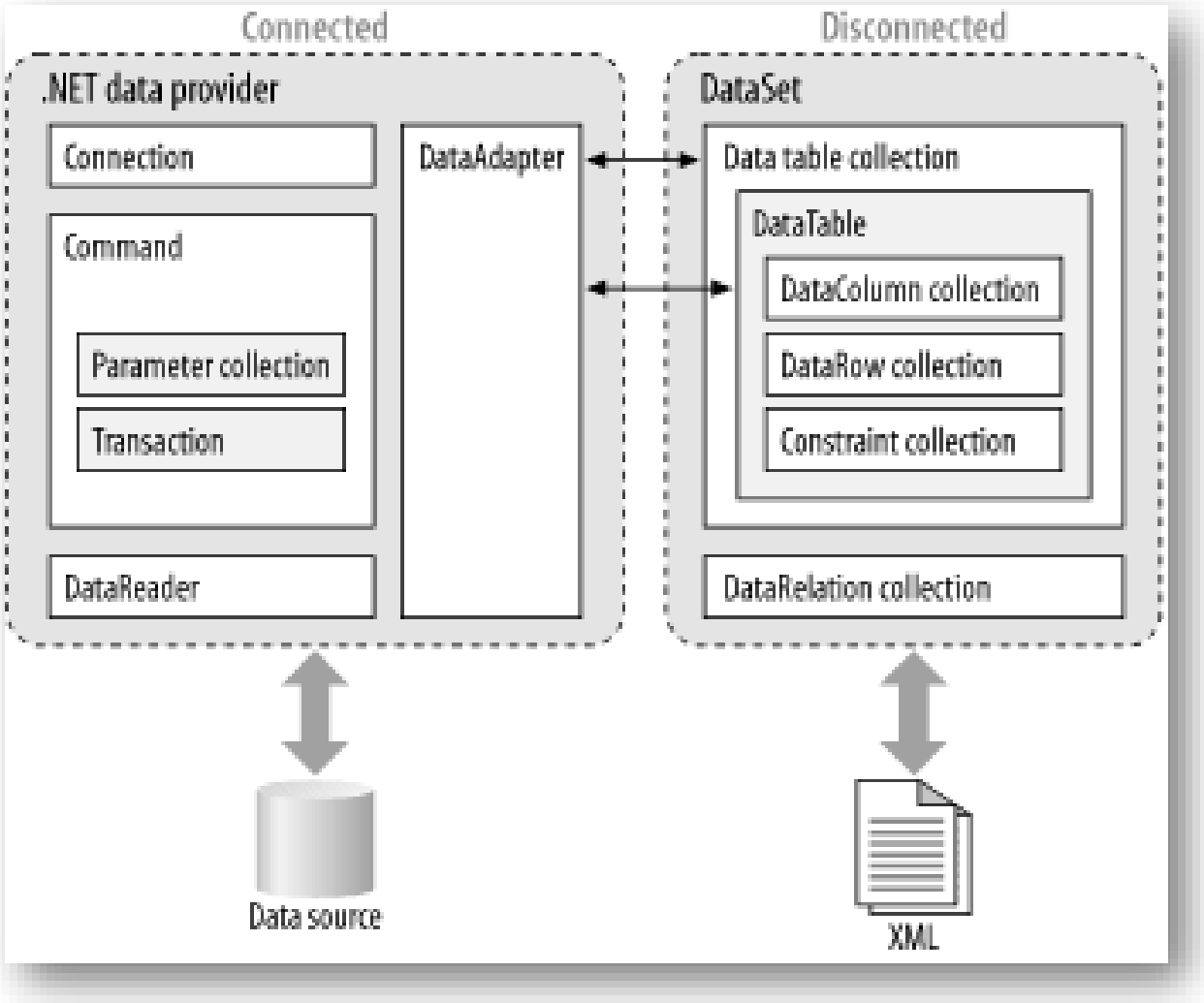
ADO.NET - Adapter

Il **DataAdapter** fornisce il ponte tra l'oggetto **DataSet** e l'origine dati.

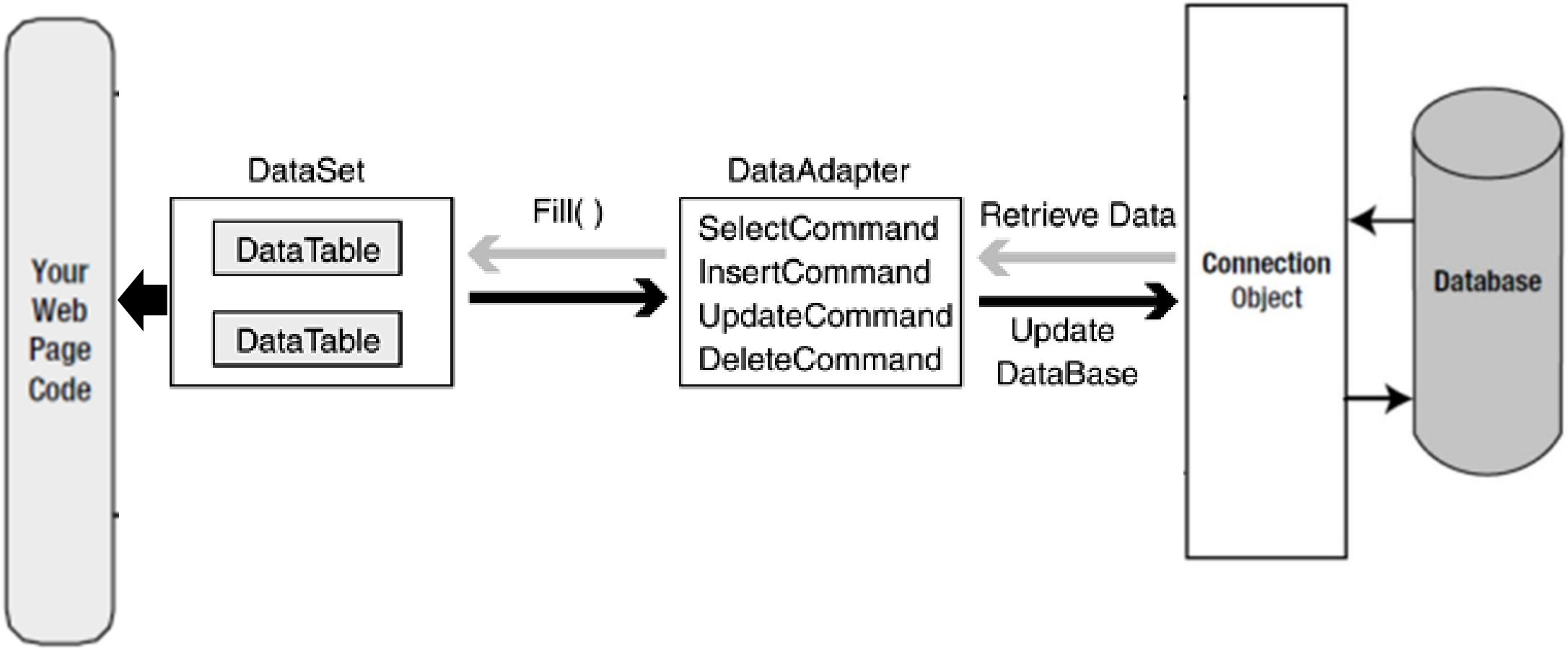
DataAdapter utilizza oggetti *Command* per eseguire comandi SQL sull'origine dati per caricare sia *DataSet* con dati sia per riconciliare le modifiche apportate ai dati nel *DataSet* con l'origine dati

La classe **DataAdapter** permette di collegare l'origine dati e le classi disconnesse tramite le classi connesse.

ADO.NET - Adapter



ADO.NET – Disconnected Mode



icubed

Demo

Disconnected Mode

icubed

Domande?

Ricordate il feedback!

© 2020 iCubed Srl

iCubed s.r.l.

Piazza Duca D’Aosta, 12 20124 MILANO

Phone: +39 02 57501057

P.IVA 07284390965