

Academy Agritech Developer

Database

Davide Maggiulli

Agenda

- Database e DBMR
- SQL
- MySql
- ADO.NET
- Entity Framework

Database

Database, DBMR, SQL, MySql

Database

- Database: una struttura di dati organizzati seguendo un modello.
- Per accedere a uno o più database si usa il DBMS:
 - **Database Management System**
- Un set di software che permettono l'accesso, l'aggiornamento e eventuale recupero di dati.



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

Installare MySql

Accedere a MySql

Client Linea di comando

Design of a Relational DB

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

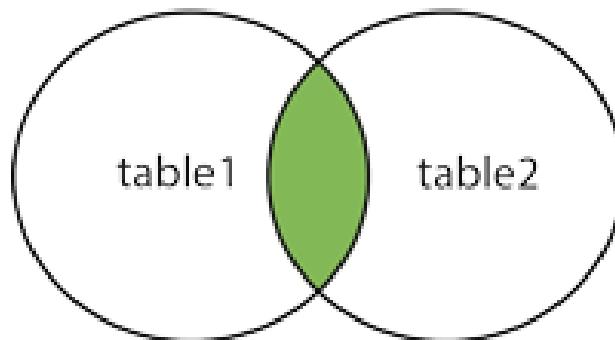
JOIN

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

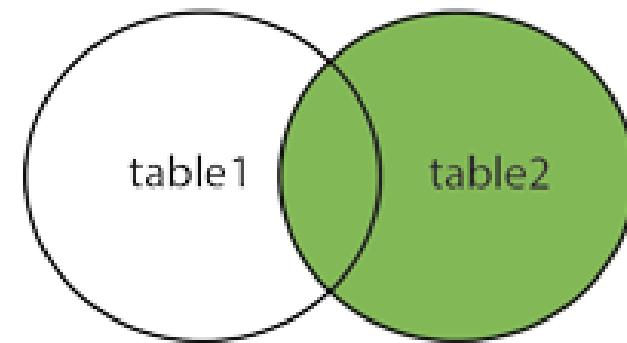
```
SELECT column1, column2, ...
      FROM table1
INNER [ LEFT / RIGHT / FULL OUTER ] JOIN table2
      ON table1.column_name = table2.column_name;
```

JOIN

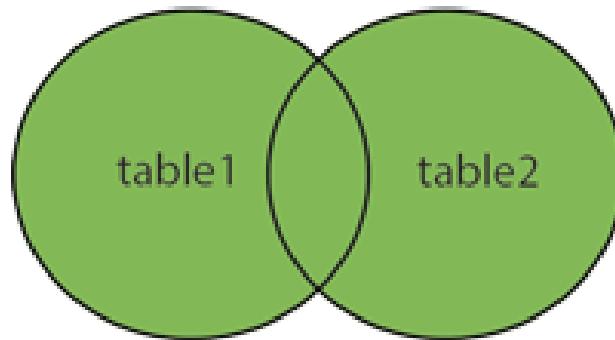
INNER JOIN



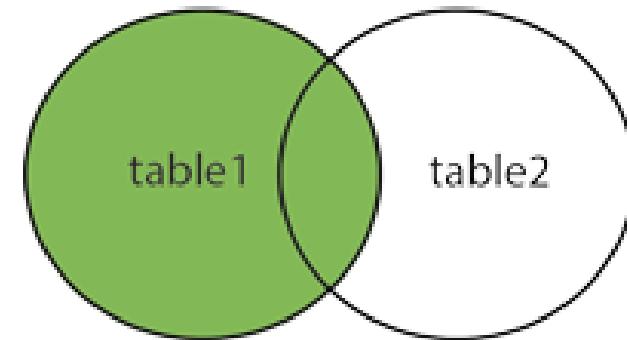
RIGHT JOIN



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".

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.

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

GROUP BY

- **MIN()** restituisce il valore più piccolo della colonna selezionata
- **MAX()** restituisce il valore più grande della colonna selezionata
- **COUNT()** restituisce il numero di righe che corrisponde a un criterio specificato

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

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

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

GROUP BY

- **AVG()** restituisce il valore medio di una colonna numerica
- **SUM()** restituisce la somma totale di una colonna numerica

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

```
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#).

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.

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

Demo

Join
Order By / Group by
Having

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 possibile 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;
```

```
SELECT * FROM dbo.ufnprocedure_name(param1)
```

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(column2) AS alias  
      FROM table1
```

Demo

Function

User-defined Function

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, conviene 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 storedProcedure_name @par1 type, @par2 type  
    OUTPUT, ...  
        AS BEGIN  
            sql_statement ...  
            RETURN value;  
            ...  
        END;
```

Per invocare una Stored Procedure:

```
EXEC @retval = 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

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 operazioni 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

Triggers

- Un DBMS si dice **attivo** quando dispone di un sottosistema integrato per definire e gestire **regole**.
- Un trigger si attiva a fronte di un dato evento. Quando tale evento viene emesso allora vien eseguita una data **azione**.
- Gli eventi possono essere generati da istruzioni DDL (trigger DDL) o da istruzioni DML (trigger DML).
- Nel caso di istruzioni DML, gli eventi per i quali si attiva un trigger sono l'esecuzione di istruzioni INSERT / UPDATE / DELETE su una tabella.
- Il trigger viene ancorato ad una tabella e qualora si verifichi uno di questi eventi si attiva eseguendo il codice T-SQL contenuto al suo interno (similmente a una SP).
- Il corpo di un trigger viene eseguito transazionalmente e quindi atomicamente.

Trigger: Motivazioni

- I trigger sono utilizzati per diversi scopi nella progettazione di un database, e principalmente:
 - per mantenere l'integrità referenziale tra le varie tabelle
 - per mantenere l'integrità dei dati della singola tabella
 - per monitorare i campi di una tabella ed eventualmente generare eventi ad hoc
 - per creare tabelle di auditing per i record che vengono modificati o eliminati

CREATE TRIGGER

- Consente di creare un nuovo trigger che viene attivato automaticamente ogni volta che si verifica un evento come INSERT, DELETEo UPDATEin una tabella

```
CREATE TRIGGER [schema_name.]trigger_name  
          ON table_name  
      AFTER { [INSERT], [UPDATE], [DELETE] }  
      [NOT FOR REPLICATION]  
          AS  
      {sql_statements}
```

Tabelle Virtuali per Triggers

- SQL Server fornisce due tabelle virtuali disponibili specificamente per i trigger chiamate INSERTED e DELETED. SQL Server utilizza queste tabelle per acquisire i dati della riga modificata prima e dopo il verificarsi dell'evento.

Evento DML	INSERTED	DELETED
INSERT	Righe da inserire	-
UPDATE	Nuove righe modificate	Righe originali prima dell'aggiornamento
DELETE	-	righe da eliminare

TRIGGER: Esempio

- ALTER TRIGGER CostoMedicinaMonitor
- ON Medicina
- FOR UPDATE, INSERT
- AS
 - INSERT INTO Medicina_History(codice, data, costoPrec, costoAtt, deltaPrezzo)
 - SELECT I.codice, GETDATE(), ISNULL(D.costo,0), I.costo, I.costo - ISNULL(D.costo,0)
 - FROM inserted I
 - LEFT JOIN deleted D ON I.codice = D.codice

TRIGGER FOR vs TRIGGER INSTEAD OF

- I Trigger FOR (o AFTER) vengono eseguiti dopo che l'evento DML si verifica e quindi a valle della specifica istruzione CREATE/INSERT/DELETE
- I Trigger INSTEAD OF vengono attivati prima che l'istruzione DML, sostituendosi di fatto all'istruzione stessa che viene saltata per eseguire il corpo del trigger: l'effettiva operazione di inserimento, eliminazione o aggiornamento non si verifica affatto.

Trigger INSTEAD OF

- CREATE TRIGGER [schema_name.] trigger_name
- ON {table_name | view_name }
- INSTEAD OF {[INSERT] [,] [UPDATE] [,] [DELETE] }
- AS
- {sql_statements}

Demo

Triggers

Disabilitazione dei Trigger

• Un Trigger viene subito attivato sulla tabella una volta creato.

- Può essere necessario a volte disabilitare uno specifico trigger con il comando DISABLE (e riabilitarlo con ENABLE)

```
DISABLE TRIGGER [schema_name.][trigger_name]
ON [object_name | DATABASE | ALL SERVER]
```

- Per disabilitare tutti i trigger di una tabella:

```
DISABLE TRIGGER ALL ON table_name
```

- Per disabilitare tutti i trigger di un database:

```
DISABLE TRIGGER ALL ON DATABASE;
```

Indici

- Gli indici sono una struttura dati che contiene puntatori ai contenuti di una tabella disposti in un ordine specifico, per aiutare il database a ottimizzare le query. Sono simili all'indice del libro, dove le pagine (righe della tabella) sono indicizzate dal loro numero di pagina.
- Esistono diversi tipi di indici e possono essere creati su una tabella. Quando un indice esiste sulle colonne utilizzate nella clausola WHERE di una query, clausola JOIN o clausola ORDER BY, può migliorare sostanzialmente le prestazioni della query.

Indici

- Gli indici sono un modo per accelerare le query di lettura ordinando le righe di una tabella in base a una colonna.
- L'effetto di un indice non è evidente per i piccoli database come nell'esempio, ma se c'è un numero elevato di righe, può migliorare notevolmente le prestazioni. Invece di controllare ogni riga della tabella, il server può eseguire una ricerca binaria sull'indice.
- Il compromesso per la creazione di un indice è la velocità di scrittura e la dimensione del database. La memorizzazione dell'indice richiede spazio. Inoltre, ogni volta che viene eseguito un INSERT o la colonna viene aggiornata, l'indice deve essere aggiornato. Questa operazione non è costosa quanto la scansione dell'intera tabella su una query SELECT, ma è ancora qualcosa da tenere a mente.

Creare un indice

- Indice per la colonna *EmployeeId* nella tabella Cars .

```
CREATE INDEX ix_cars_employee_id ON Cars (EmployeeId);
```

- Questo indice migliorerà la velocità delle query che chiedono al server di ordinare o selezionare in base ai valori in EmployeeId , ad esempio

```
SELECT * FROM Cars WHERE EmployeeId = 1
```

- Indice multi colonna

```
CREATE INDEX ix_cars_e_c_o_ids ON Cars (EmployeeId, CarId, OwnerId);
```

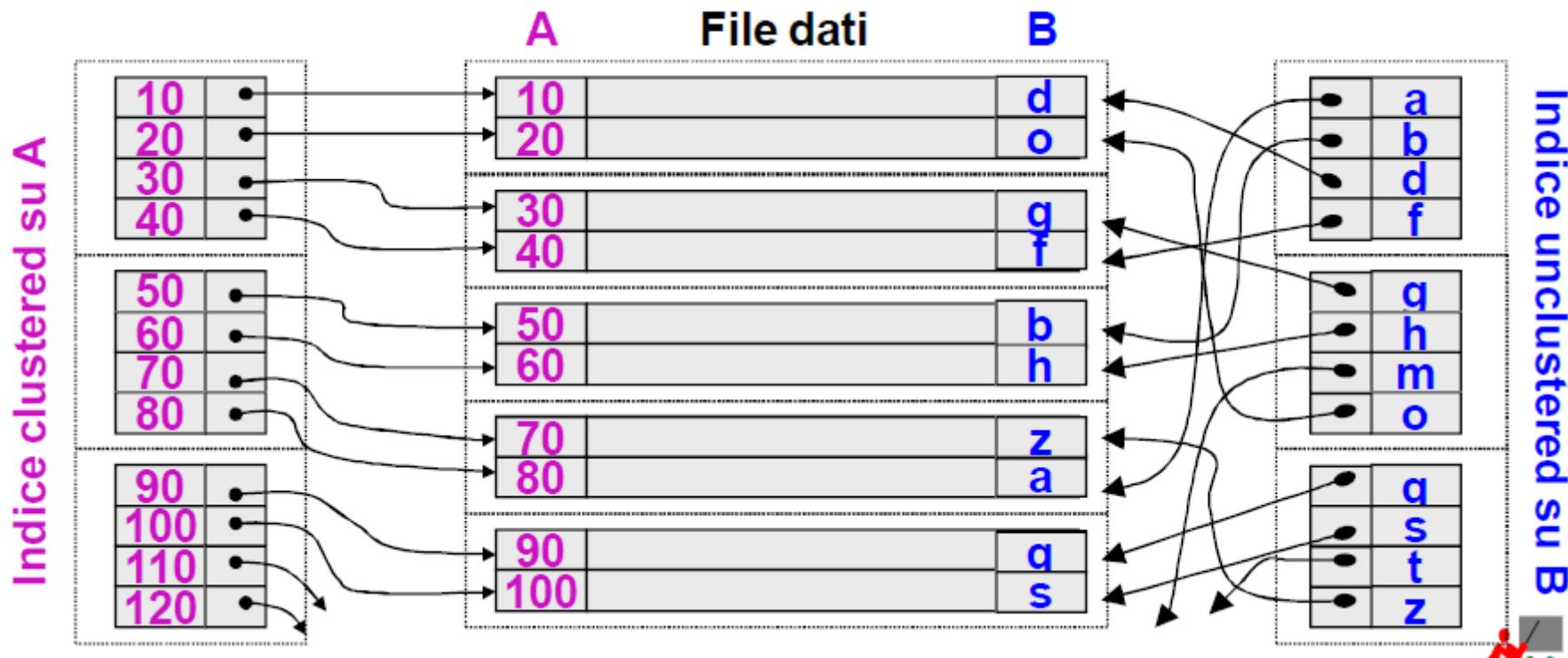
Indici Clustered e non clustered

- Gli indici possono avere diverse caratteristiche che possono essere impostate alla creazione o alterando gli indici esistenti.

```
CREATE CLUSTERED INDEX ix_clust_employee_id ON Employees(EmployeeId, Email);
```

- Un indice cluster è un tipo di indice in cui i record della tabella vengono fisicamente riordinati per corrispondere all'indice (può esserci al massimo un solo indice cluster su una tabella). Un indice non cluster, d'altra parte, è un tipo speciale di indice in cui l'ordine logico dell'indice non corrisponde all'ordine fisico memorizzato delle righe sul disco

Indici Clustered vs unclustered



Indici univoci

```
CREATE UNIQUE INDEX uq_customers_email ON Customers(Email);
```

- Crea un indice univoco per la colonna Email nella tabella Customers. Questo indice, oltre a velocizzare le query come un normale indice, imporrà anche l'unicità di ogni indirizzo email in quella colonna. Se una riga viene inserita o aggiornata con un valore di Email non univoco, l'inserimento o l'aggiornamento, per impostazione predefinita, falliranno.
- Indice univoco che consente NULL:

```
CREATE UNIQUE INDEX idx_license_id  
ON Person(DrivingLicenseID) WHERE DrivingLicenseID IS NOT NULL
```

Indice Ordinato

- Se si utilizza un indice ordinato, l'istruzione SELECT non eseguirà ulteriori ordinamenti durante il recupero.

```
CREATE INDEX ix_scoreboard_score ON scoreboard (score DESC);
```

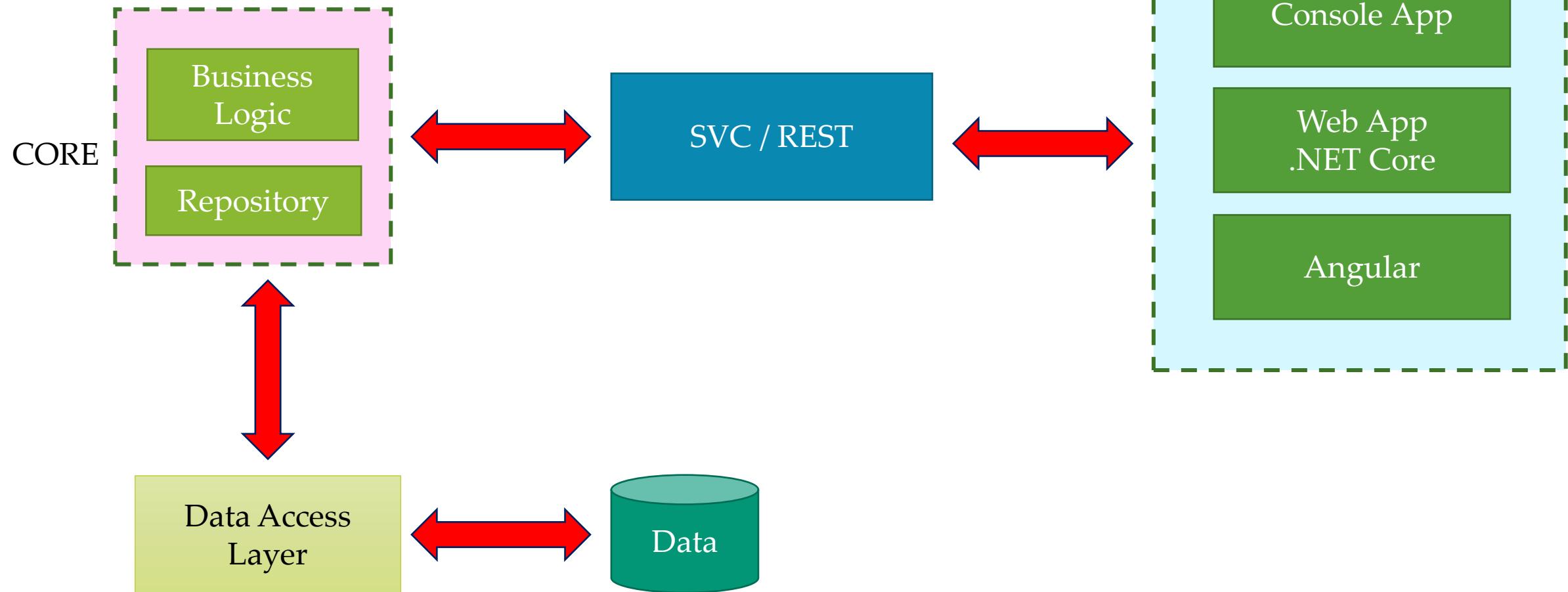
- Quando si esegue la query
- SELECT * FROM scoreboard ORDER BY score DESC;
- Il sistema di database non eseguirà ulteriori ordinamenti, poiché può eseguire una ricerca dell'indice in tale ordine.

Demo

Indici

ADO.NET e EntityFramework

Repository Pattern



ADO.NET

- **ADO.NET** è una tecnologia di accesso ai dati di Microsoft
- È costituito da un insieme di componenti software che i programmatorei 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 - Provider

- Insieme di oggetti che ereditano da queste classi astratte di base:
 - *DbConnection*
 - *DbCommand*
 - *DbDataReader*
 - *DbDataRecord*
 - *DbDataAdapter*
- Creati come layer leggero tra la fonte dati e l'applicazione

ADO.NET - Provider

- Realizzati come codice managed all'interno del .NET Framework (namespace *System.Data.Common*)
- Costituiscono la parte “connessa” alle varie fonti dati
- Permettono l'utilizzo di comandi diretti verso le fonti dati attraverso gli oggetti *Command*
 - Chiamata a Stored Procedure
 - Codice SQL dinamico

ADO.NET - Provider

Nel namespace *System.Data.Common*

DbCommand	DbCommandBuilder	DbConnection
DataAdapter	DbDataAdapter	DbDataReader
DbParameter	DbParameterCollection	DbTransaction
DbProviderFactory	DbProviderFactories	DbException

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

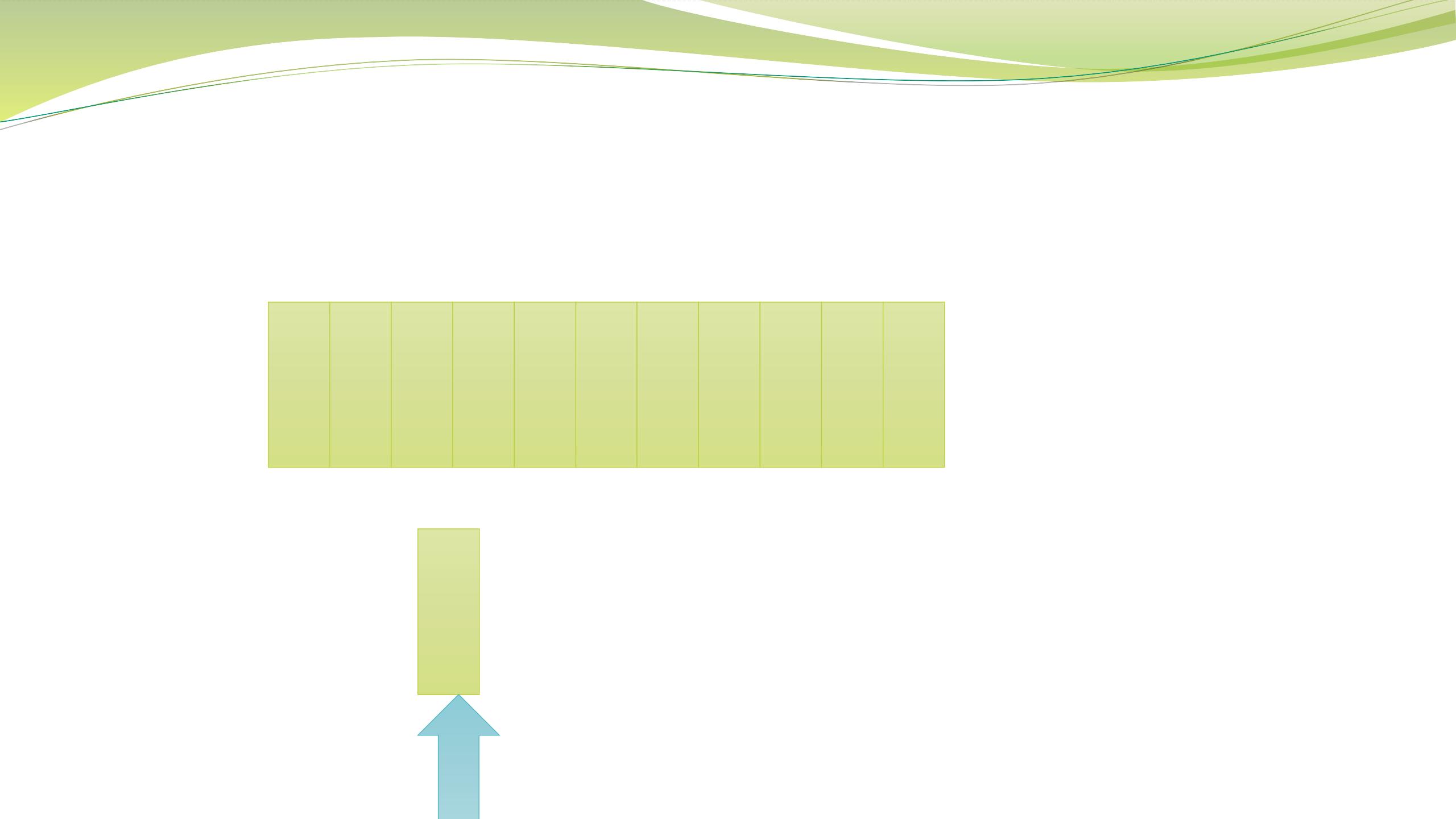
- Il **Connected Mode** fornisce
 - accesso di sola lettura (e forward-only) ai dati nell'origine dati
 - la possibilità di eseguire comandi sull'origine dati

ADO.NET – Connected Mode

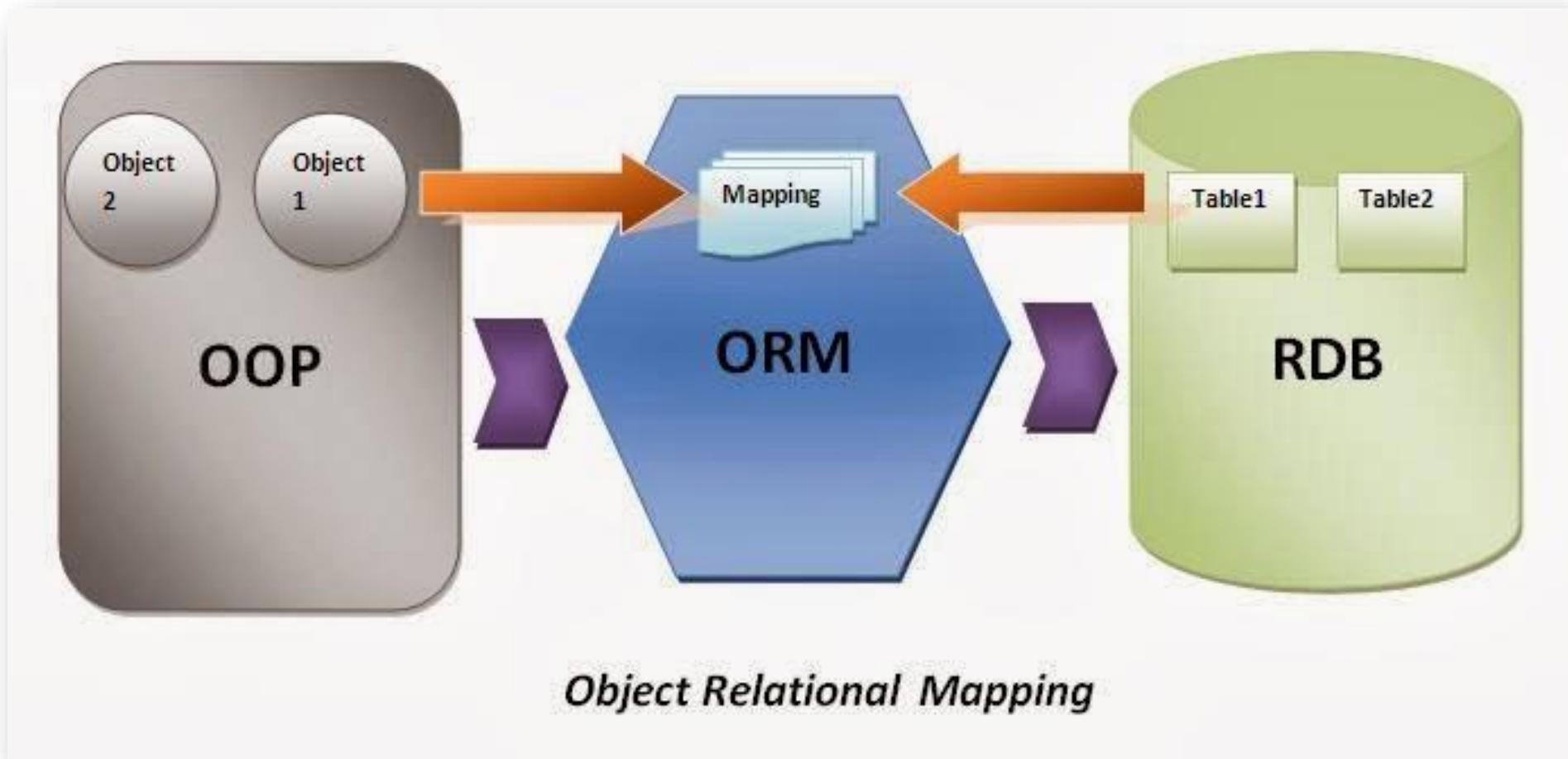
- Principali classi utilizzate in Connected Mode:
 - Connection (*SqlConnection*)
 - Command (*SqlCommand*)
 - DataReader (*SqlDataReader*)
 - Parameter (*SqlParameter*)

ADO.NET – Connected Mode

- Step
 - 1. Creare e Aprire una Connessione
 - 2. Creare un Command
 - Creare Parametri se necessario
 - 3. Eseguire Command (DataReader/NonQuery/Scalar)
 - 4. Lettura Dati a schermo
 - 5. Chiudere Connessione



Cos'è un ORM?



Esercitazione

Creare un nuovo database *Ticketing* con una sola tabella *Tickets*. Le colonne sono:

- *ID* (int, PK, auto-incrementale)
- *Descrizione* (varchar(500))
- *Data* (datetime)
- *Utente* (varchar(100))
- *Stato* (varchar(10)) – New, OnGoing, Resolved

Realizzare una Console app che acceda al database *Ticketing* utilizzando il Connected Mode di ADO.NET o EntityFramework e che:

- Stampi la lista dei Ticket in ordine cronologico (dal più recente al più vecchio)
- Permetta l'inserimento di nuovi Ticket (i dati devono essere inseriti dall'utente)
- Permetta la cancellazione di un Ticket (utilizzare l'*ID* univoco per identificarlo)

Esercitazione – Agenzia Immobiliare

Creare un'applicazione console per la gestione semplificata di un'agenzia immobiliare.

L'applicazione gestisce degli immobili in vendita. Ogni immobile è caratterizzato da:

- Codice
- Tipo (appartamento, villa, garage)
- Superficie in mq
- Numero vani (1, 2, 3, etc.)
- Anno di fabbricazione
- Prezzo richiesto
- Proprietario
- Stato: in vendita, venduto, non disponibile
- Data di inserimento

Esercitazione – Agenzia Immobiliare

E' necessario catalogare i proprietari. Un proprietario è caratterizzato da

- Codice
- Nome
- Cognome
- Gli immobili di cui è proprietario (gli immobili per semplicità hanno un unico proprietario).

L'applicazione deve offrire le seguenti funzionalità:

1. Elenco di tutti gli immobili in vendita, ordinati per data di inserimento
2. Elenco di tutti gli immobili per tipologia
3. Elenco di tutti gli immobili dato il codice del proprietario
4. Elenco di tutti i proprietari
5. Inserimento proprietario

Esercitazione – Agenzia Immobiliare

6. Inserimento immobile
7. Aggiorna anagrafica proprietario
8. Aggiornamento dati immobile
9. Cambia stato immobile (da in vendita a venduto ad esempio)

L'applicazione dovrà sfruttare una SOA