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
```

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 TRAN
	BEGIN TRY
		... sql_statements ...
		IF @@ERROR ROLLBACK;
		COMMIT;
	END TRY
	BEGIN CATCH
		ROLLBACK;
	END CATCH
END
```

Demo

Transaction

icubed

ECOMMERCE

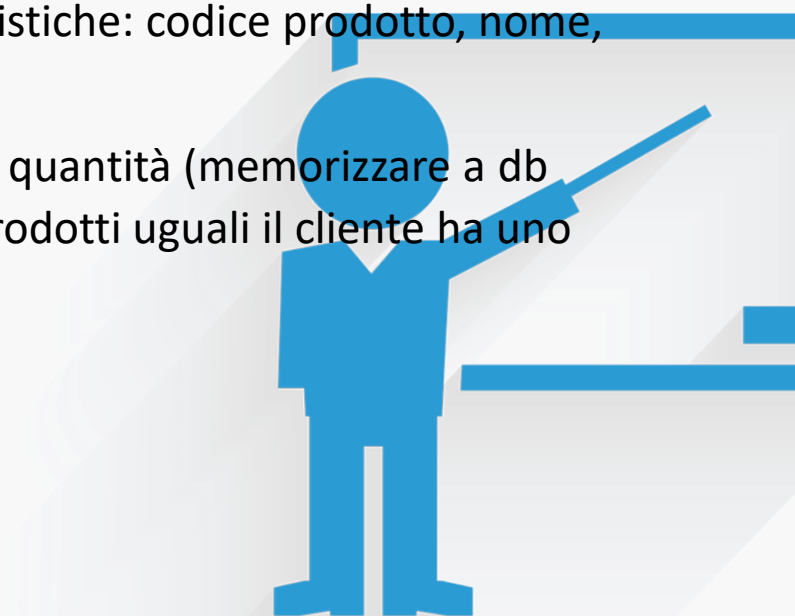
Progettare un database che consenta la gestione di un sito Ecommerce.

Di ogni cliente è necessario memorizzare: codice cliente, nome, cognome, data di nascita. Devono essere memorizzati a sistema anche:

- i suoi indirizzi. Ogni indirizzo è caratterizzato da Tipo (solo “Residenza”/”Domicilio”) Città, via, cap, numero civico, Provincia, nazione.
- Le sue carte (di credito/debito). Ogni carta è caratterizzata da Tipo (“Debito” “Credito”), scadenza, saldo.

Il sito Ecommerce mette a disposizione dei prodotti che hanno le seguenti caratteristiche: codice prodotto, nome, descrizione, quantità disponibile a magazzino, prezzo unitario.

Ogni cliente può aggiungere un prodotto alla volta al suo ordine, specificandone la quantità (memorizzare a db anche il subtotale rispetto a quantità e prodotto. In caso di acquisto di almeno 3 prodotti uguali il cliente ha uno sconto del 10% su quel prodotto).



icubed

Ogni cliente, a partire dal suo carrello/ordine può fare l'acquisto. Ogni ordine, oltre al riferimento ai prodotti che l'utente desidera acquistare e la relativa quantità, ha anche un totale complessivo dato dalla somma dei prezzi dei prodotti che deve acquistare.

L'ordine ha anche uno "stato" che, inizialmente impostato come "provvisorio", potrà passare a "confermato" se il cliente conclude l'acquisto relativo a quel codice ordine dopo aver specificato un indirizzo di spedizione e se, dopo aver selezionato con quale carta (non scaduta) vuole pagare, il saldo riesce a "coprire" la spesa totale.

Progettare e creare il DB partendo dal modello Concettuale/Logico ER.

Creare le funzioni e le stored procedure utili alla gestione del processo: iscrizione cliente con relativi indirizzi e carte, creazione ordine, modifica ordine "provvisorio", aggiunta prodotti all'ordine, Conferma acquisto.

TRIGGER

Il Trigger serve per definire un meccanismo automatico sui dati. Quando una determinata operazione viene effettuata sui dati il sistema verifica se esiste un trigger associato e lo esegue.

È un tipo speciale di stored procedure che viene eseguita automaticamente quando si verifica un evento.

TRIGGER

SQL SERVER prevede tre tipi di TRIGGER: DDL, LOGON e DML.

- I trigger DDL vengono eseguiti in risposta a vari eventi DDL (Data Definition Language). Corrispondono principalmente alle istruzioni CREATE, ALTER e DROP su oggetti del database;
- I trigger DML vengono eseguiti quando un utente tenta di agire sui dati di una o più tabelle, mediante un evento DML (Data Manipulation Language). Gli eventi che innescano questi trigger sono le istruzioni INSERT, UPDATE o DELETE eseguite su una tabella o una vista.
- I trigger LOGON vengono attivati in risposta all'evento LOGON generato quando viene stabilita una sessione utente.

TRIGGER

```
CREATE TRIGGER [ SCHEMA_NAME . ]TRIGGER_NAME
ON { NOME-TABELLA | NOME-VISTA }
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS
BEGIN
{ SQL_STATEMENT  [ ; ] [ ,...N ]
END
```

Le opzioni FOR, AFTER e INSTEAD OF sono alternative esclusive :
FOR serve per indicare che deve essere eseguito prima del comando a cui è legato (per esempio FOR INSERT significa prima di un inserimento);
AFTER, si usa nel caso di voler effettuare delle azioni prima che i controlli sui vincoli della tabella siano stati controllati. È utile per impostare valori di chiave primaria calcolati o per riempire dei campi lasciati vuoti prima che siano effettuati i normali controlli;
INSTEAD OF, serve per indicare che deve essere eseguito al posto del comando a cui è legato (per esempio INSTEAD OF INSERT significa in sostituzione di un inserimento);

Autorità e privilegi

Nei DBMS SQL ogni operazione deve essere autorizzata, ovvero l'utente che esegue l'operazione deve avere i privilegi necessari.

I privilegi vengono concessi e revocati per mezzo delle istruzioni **GRANT e REVOKE**

Un principio fondamentale è che un utente che ha ricevuto un certo privilegio può a sua volta accordarlo ad altri utenti solo se è stato esplicitamente autorizzato a farlo.

Mediante GRANT e REVOKE si controllano anche le autorità, ovvero il diritto ad eseguire azioni amministrative di un certo tipo

Ad esempio, se si ha l'autorità SYSADM (che include anche quella di DBADM) è possibile passare ad altri utenti l'autorità DBADM (Database Administrator Authority):

GRANT DBADM ON DATABASE TO Pippo WITH GRANT OPTION;

la clausola WITH GRANT OPTION autorizza l'utente Pippo a passare l'autorità ad altri utenti

GRANT: dettagli sui privilegi

CONTROL: comprende tutti i privilegi (su una view sono solo SELECT, INSERT, DELETE e UPDATE). Inoltre permette di conferire tali privilegi ad altri utenti; può essere conferito solo da qualcuno che ha autorità SYSADM o DBADM

ALTER: attribuisce il diritto di modificare la definizione di una tabella

DELETE: attribuisce il diritto di cancellare righe di una tabella

INDEX: attribuisce il diritto di creare un indice sulla tabella

INSERT: attribuisce il diritto di inserire righe nella tabella

REFERENCES: attribuisce il diritto di definire foreign keys in altre tabelle che referenziano la tabella

SELECT: attribuisce il diritto di eseguire query sulla tabella/vista e di definire VIEW

UPDATE: attribuisce il diritto di modificare righe della tabella/vista

Nota: Per eseguire una query, è necessario avere il privilegio di SELECT o di CONTROL su tutte le table e le view referenziate dalla query

Grant. Esempi

- Paperino autorizza Pippo e Topolino a leggere la relazione Employee e a modificare i valori di Salary; inoltre concede loro di passare questo privilegio ad altri utenti:

```
Paperino>  GRANT SELECT, UPDATE(Salary)
           ON TABLE Employee TO USER Pippo, USER Topolino
           WITH GRANT OPTION
```

- … e Pippo ne approfitta subito:

```
Pippo>     GRANT SELECT
           ON TABLE Employee TO USER Pluto
```

- Pluto può eseguire query su Employee, ma non aggiornamenti; inoltre non può passare lo stesso privilegio ad altri

Grant. Esempi

- Se ora Topolino esegue:

```
Topolino>  GRANT UPDATE(Salary)
           ON TABLE Employee TO USER Pluto
           WITH GRANT OPTION
```

allora Pluto può anche modificare i valori di Salary e passare lo stesso privilegio ad altri

- Quindi:

```
Pluto>     GRANT ALL
           ON TABLE Employee TO USER Minnie
```

trasferisce a Minnie (ma senza GRANT OPTION) il solo privilegio sulla modifica dei valori di Salary

Grant. Esempi

- Pippo crea una vista su Employee:

```
Pippo>      CREATE VIEW NomiEmp(NOME,COGNOME)
            AS SELECT LASTNAME,FIRSTNME
               FROM EMPLOYEE
```

e permette a Orazio di fare query su tale vista:

```
Pippo>      GRANT SELECT
            ON TABLE NomiEmp TO USER Orazio
```

- Quindi ora Orazio può interrogare `NomiEmp`, ma non `Employee`!

Revoke

- Il formato dell'istruzione REVOKE per revocare privilegi su tables e views è:

```
REVOKE { ALL | < lista di privilegi > }
ON [ TABLE ] <table name>
FROM { <lista di utenti e gruppi> | PUBLIC }
```

- A differenza del GRANT, per eseguire REVOKE bisogna avere l'autorità **SYSADM** o **DBADM**, oppure il privilegio di **CONTROL** sulla relazione
- Il REVOKE non agisce a livello di singoli attributi; pertanto non si possono revocare privilegi di **UPDATE** solo su un attributo e non su altri (per far ciò è quindi necessario revocarli tutti e poi riassegnare solo quelli che si vogliono mantenere)

Revoke. Esempi

- Se Pippo, che non ha autorità DBADM o SYSADM, né CONTROL su Employee, prova ad eseguire:

```
Pippo>      REVOKE SELECT
            ON TABLE Employee FROM Pluto
```

  si verifica un errore

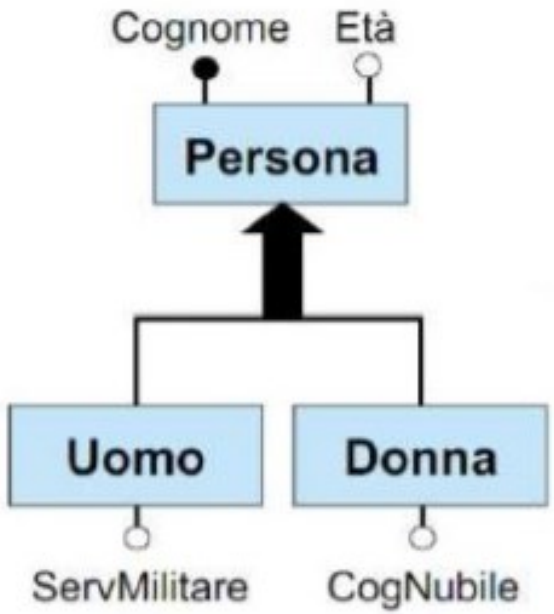
- Viceversa, se Paperino ha autorità `DBADM` ed esegue

```
Paperino>   REVOKE SELECT
            ON TABLE Employee FROM Pippo, Topolino
```

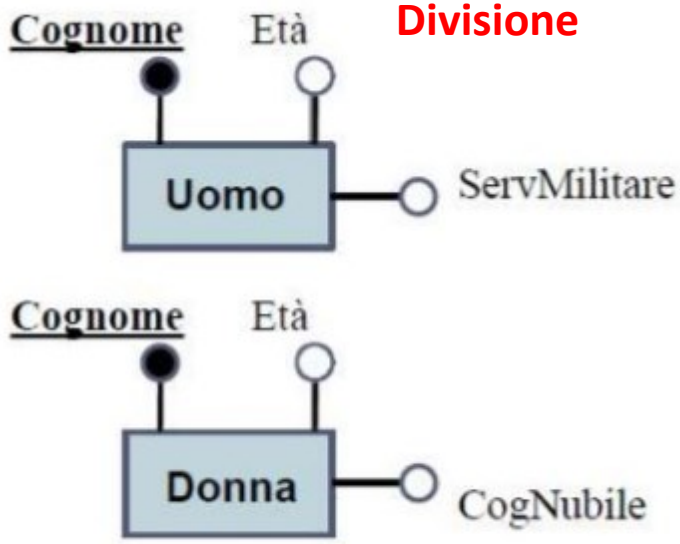
  né Pippo né Topolino possono più eseguire query su Employee, ma continuano a poter aggiornare Salary

- La vista NomiEmp definita da Pippo su Employee diventa "inoperativa", ovvero non più utilizzabile (e quindi Orazio non può più interrogarla)
- Si noti che Pluto mantiene il privilegio SELECT su Employee
- Lo standard SQL prevede una gestione del REVOKE più complessa, che include anche effetti di revoca dei privilegi "in cascata" (in cui quindi Pluto perderebbe il privilegio di SELECT)

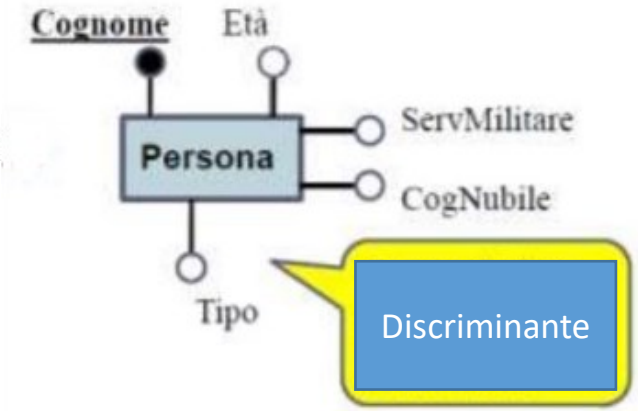
Generalizzazione e tabelle DB



Divisione



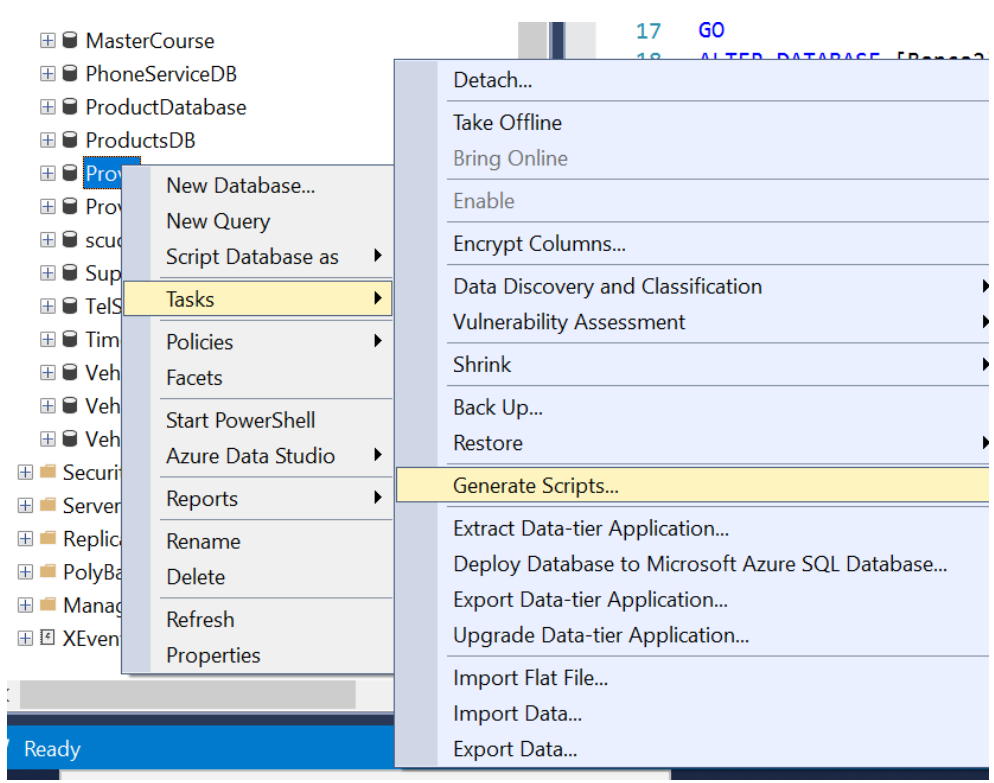
Fusione

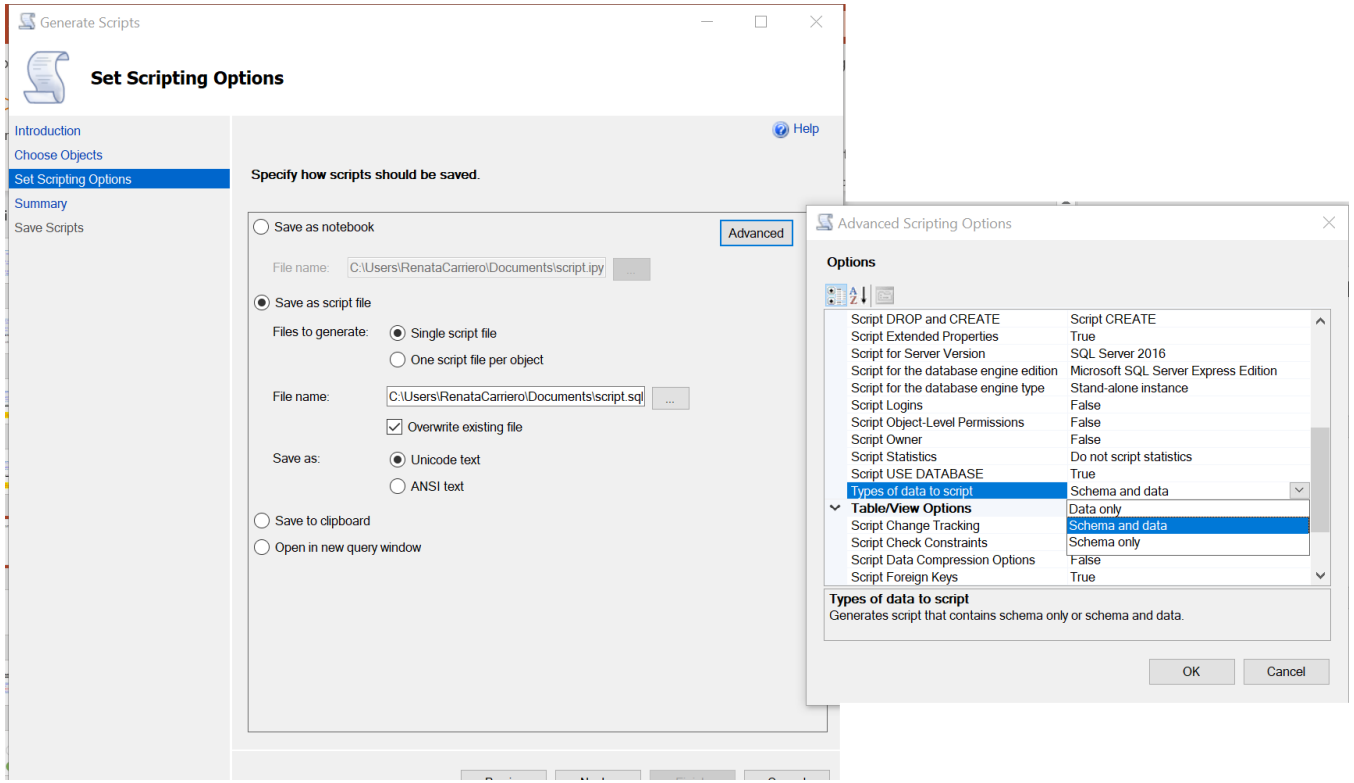


icubed

Esportare Database (struttura e/o dati)

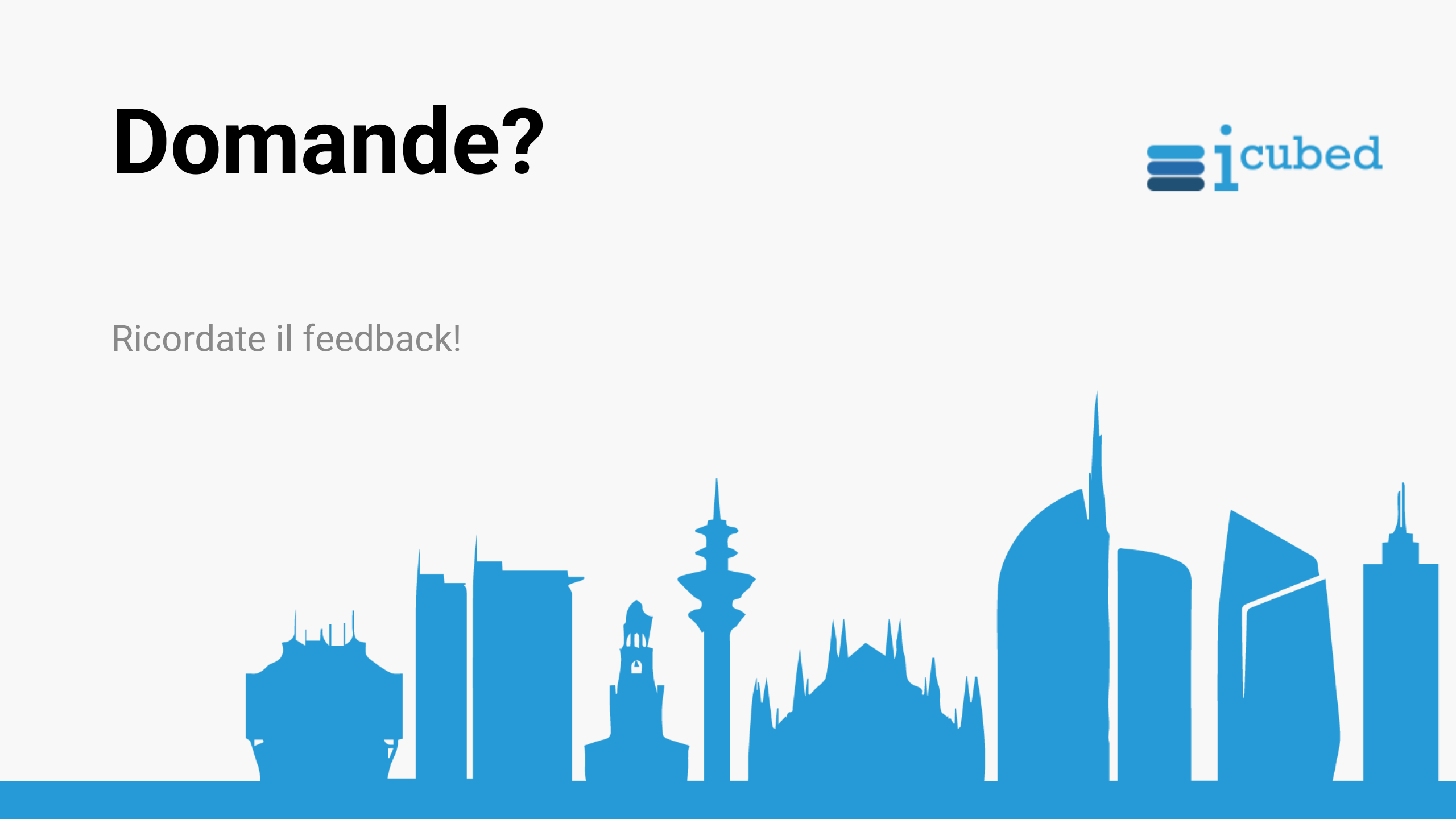
Viene generato un file .sql con gli scripts che SQL Server Management Studio crea automaticamente.





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