T-SQL

Tipi dati

le versioni con var davanti indicano che l'occupazione in mem di quella variabile varia a seconda del contenuto fino alla dimensione specificata: quelli normali sono sempre della dimensione specificata

Dati binari:	**Dati ora e data:**	**Dati speciali:**
• binary[(n)] • varbinary[(n)]	• datetime • smalldatetime	• bit tipicamente è usato per rappresentare i flag, vero/false o true/false o si/no, perché può accettare solo due valori 0 o 1. Le colonne che hanno un tipo dati bit non possono avere valori nulli e non possono avere indici. • cursor sono usati come varibili in stored proc oppure come parametri di OUTPUT sempre in stored proc, fanno riferimento ai cursori. Possono essere nulli e non possono essere usati con le istruzioni CREATE TABLE. • sysname • timestamp occupa 8 bytes ed è un contatore incrementale per colonna assegnato automaticamente da SQL Server 7. • UNIQUEIDENTIFIER (GUID)
Dati carattere:	**Dati monetari:**	**Dati numerici esatti:**
• char[(n)] • varchar[(n)] • nchar[(n)] • nvarchar[(n)]	• money • smallmoney	• decimal[(p[, s])] • numeric[(p[, s])] decimal e numeric sono sinonimi per SQL Server 7, possono avere valori compresi tra 10^38 - 1 e -10^38 -1. La memoria che occupano per essere immagazzinati varia a seconda della precisione p - che rappresenta il numero massimo di cifre decimali che possono essere memorizzate (da entrambe le parti della virgola). Il massimo della precisione è 28 cifre. s - è la scala, che rappresenta il numero di massimo di cifre decimali dopo la virgola e deve essere minore od uguale alla precisione. • int • smallint • tinyint
Dati text ed image:	**Dati numerici approssimati:**	
• text • ntext • image	• float[(n)] • real	

Operatori

Matematici *decimal , numeric , int , smallint , tinyint*

+ (Aggiunge) , - (Sottrae) , * (Moltiplica) , / (Divide) , % (Modulo) resto divisione

Comparazione

Fanno comparazioni tra espressioni (numeriche e testuali). Non lavorano sui tipi di dati **text, ntext, image**.

= (Uguale) , > (Maggiore di) , < (Minore id) , >= (Maggiore uguale a) , <= (Minore uguale a) , <> (diverso da)

I risultati di una operazione di comparazione in T-SQL danni tre possibili risultati: **TRUE, FALSE, e UNKNOWN (NULL)**.

Logici

Saggiano la verità o falsità di alcune condizioni, ritornano un tipo dato booleano.

Operatore	Azione	Operatore	Azione
ALL	TRUE se tutte le condizioni sono TRUE.	IN	TRUE se l'operando è uguale ad uno della lista di espressioni.
AND	TRUE se tutte le espressioni booleane sono TRUE.	LIKE	TRUE se l'operando trova il valore cercato.
ANY	TRUE se almeno una delle condizioni è TRUE.	NOT	Inverte il valore di ogni operando booleano.
BETWEEN	TRUE se l'operando è all'interno del range.	OR	TRUE se ogni espressione booleana è TRUE.
EXISTS	TRUE se una subquery contiene una riga qualsiasi.	SOME	TRUE se alcuni set di comparazione sono TRUE.

IN

The IN operator may be used if you know the exact value you want to return for at least one of the columns.

```
SELECT column_name FROM table_name
WHERE column_name IN (value1,value2,..)
```

BETWEEN ... AND

The BETWEEN ... AND operator selects a range of data between two values. These values can be numbers, text, or dates.

```
SELECT column_name FROM table_name
WHERE column_name
BETWEEN value1 AND value2
```

LIKE
The following SQL statement will return persons with first names that start with an 'O':

```
SELECT * FROM Persons
WHERE FirstName LIKE 'O%'
```

The following SQL statement will return persons with first names that end with an 'a':

```
SELECT * FROM Persons
WHERE FirstName LIKE '%a'
```

The following SQL statement will return persons with first names that contain the pattern 'la':

```
SELECT * FROM Persons
WHERE FirstName LIKE '%la%'
```

Concatenazione

use pubs
SELECT au_lname + ' - ' , + au_fname from authors

Variabili

DECLARE dichiarare variabili
sintassi: *DECLARE @<nome_var> <tipo_var>, @<nome_var> <tipo_var>, ... ;*
visibilità in tutta la stored procedure, no costanti

SET assegnare valore a variabile
sintassi: *SET @<nome_var> = <valore>;*
valore può essere il risultato di una SELECT (se ritorna + righe assegna valore dell'ultima), la variabile può essere settata dentro la select_list di una SELECT

Controllo del flusso

BEGIN – END identificano un blocco_istruzioni (+ di una istruzione) se c'è solo una istruzione

IF (<condizione_booleana>) BEGIN <codice> END [ELSE BEGIN <codice> END]	**WHILE <espressione_booleana>** { istruzione_SQL | blocco_istruzioni } [BREAK] (esce dal ciclo) { istruzione_SQL | blocco_istruzioni } [CONTINUE] (torna all'inizio del ciclo)
CASE <*espressione_input*> WHEN *espressione_when* THEN *espressione_risultato* [*...n*] [ELSE *espressione_risultato*]	Esempio Case *USE pubs* *GO* *SELECT Categoria =* *CASE type* *WHEN 'popular_comp' THEN 'Computer'* *WHEN 'mod_cook' THEN 'Cucina moderna'*

END	*WHEN 'business' THEN 'Business'* *WHEN 'psychology' THEN 'Psicologia'* *WHEN 'trad_cook' THEN 'Cucina tradizionale'* *ELSE 'Nessuna categoria'* *END,* *title AS 'Titolo'* *FROM titles* *WHERE price IS NOT NULL* *ORDER BY type, price*

RETURN serve per terminare l'esecuzione del blocco di codice (query, batch procedure) e ritornare il controllo al blocco precedente. Può ritornare un valore intero per indicare lo stato di terminazione

WAITFOR DELAY <*tempoDelay*> ferma l'esecuzione per <*tempoDelay*> tempo (max 24 ore)
es. WAITFOR DELAY '00:00:02'; //*aspetta 2 sec*
WAITFOR TIME <*date*> ferma l'esecuzione finchè non è il momento <date>
es. WAITFOR TIME '22:00';

Creare VIEW

CREATE VIEW [< nome_*database* > .] [< *proprietario* > .] *nome_vista* [(*colonna* [,...*n*])]
[WITH < attributi_vista > [,...n]]
AS
istruzione_*SELECT*
[WITH CHECK OPTION]

esempio due modi per creare una VIEW: il primo esplicita ogni colonna, il secondo le dichiara all'inizio

CREATE VIEW V_ProdottiPerCategoria AS SELECT Categories.CategoryName *AS NomeCategoria*, Products.ProductName *AS NomeProdotto*, Products.QuantityPerUnit *AS Quantità* FROM Categories INNER JOIN Products ON Categories.CategoryID = Products.CategoryID	CREATE VIEW V_ProdottiPerCategoria *(NomeCategoria,NomeProdotto,Quantità)* AS SELECT Categories.CategoryName, Products.ProductName, Products.QuantityPerUnit FROM Categories INNER JOIN Products ON Categories.CategoryID = Products.CategoryID

Batch

serve per creare una sequenza di comandi che devono essere inviati per l'esecuzione sul server SQL
sintassi:
GO /* inizio batch*/
blocco_comandi
GO /* fine batch*/

STORED PROCEDURE

gruppi di istruzioni raggruppate in un modulo e mantenute nella cache del server per rapida esec. Ci sono anche procedure di sistema del server e procedure ad esec. automatica a determinate condizioni (avvio server, shutdown etc.)
devono avere input e stato di terminazione
sintassi:

```
CREATE PROCEDURE  nome_procedura
   [ { @parametro tipo_di_dato }
       [ = default ] [ OUTPUT ]
   ] [ ,...n ]
AS istruzione_sql [ ...n ]
```

esempi:

```
CREATE PROCEDURE HumanRes.uspGetEmployees
@LastName nvarchar(50)=0
AS
   SELECT FirstName,LastName,JobTitle,Department
   FROM HumanRes.vEmployedDepartment
   WHERE LastName = @LastName;

per eseguire
EXECUTE HumanRes.uspGetEmployees 'Ackerman';
```

```
CREATE PROCEDURE dbo.p_sel_autore2 ( @state
VARCHAR(2) , @contract BIT )
AS
 SELECT  au_lname + ' ' + au_fname AS Nome
 FROM authors
 WHERE state = @state AND contract = @contract
RETURN(0)

per eseguire
EXEC dbo.p_sel_autore2 'CA', '1'

oppure per catturare il valore di uscita e esplicitare i
parametri di input
DECLARE @ret INTEGER
EXEC @ret = dbo.p_sel_autore2 @contract = '1',
@state='CA'
PRINT @ret
```

Per modificare una stored procedure già creata si usa l'istruzione ALTER
esempio

```
ALTER PROCEDURE dbo.p_sel_autore ( @au_id VARCHAR(11) = '' )
AS
  SELECT TOP 10
  au_lname + ' ' + au_fname AS Nome
  FROM  authors
  WHERE au_id = @au_id
  ORDER BY au_fname DESC
```

Per eliminare una stored procedure già memorizzata si usa l'istruzione DROP
esempio:

```
DROP PROCEDURE dbo.p_sel_autore
```

I TRIGGER

Un trigger è una stored procedure che si attiva automaticamente quando viene eseguita una particolare operazione (INSERT / UPDATE / DELETE) su una determinata Tabella. Una tabella può avere + trigger associati ma non si può stabilire a priori l'ordine di esecuzione.
Ci sono due tabelle speciali che è utile usare nei trigger: **deleted** e **inserted.** La tabella **deleted** conterrà le righe che sono state appena eliminate al contrario con una INSERT la tabella **inserted** conterrà le righe appena inserite. In caso di UPDATE, entrambe le tabelle contengono valori, perchè la **deleted** conterrà i dati prima della modifica (le vecchie righe) mentre la **inserted** conterrà i dati dopo la modifica (le nuove righe)

Un trigger viene eseguito una sola volta per tabella indipendentemente da quante righe sono interessate: si può controllarne il numero con la variabile di sistema @@ROWCOUNT
sintassi:

```
CREATE TRIGGER [schema_name.]trigger_name
ON {table|view}
 {FOR|AFTER| INSTEAD OF}
{[INSERT] [,] [UPDATE] [,] [DELETE]}
AS {sql_statement [;] [,...n]}
```

esempio:

```
CREATE TRIGGER TR_DEL_Employees
ON Employees
FOR DELETE
AS
INSERT CronologiaImpiegati
SELECT EmployeeID,FirstName,LastName,'Eliminato' AS Azione
FROM deleted
```

nei trigger è comodo l'utilizzo dell' istruzione ROLLBACK TRAN per annullare le modifiche eseguite dalla procedure che ha scatenato il trigger
esempio:

- ho due tabelle T1 e T2, su T2 ho un trigger che impedisce l'inserimento dei caratteri '--' nella colonna valore.
-
```
CREATE TRIGGER TR_UPD_test
ON t2
FOR INSERT,UPDATE
AS
  IF EXIST(SELECT 1 FROM inserted WHERE valore='--')
    BEGIN
    ROLLBACK TRAN /*la transazione implicita del trigger*/
    PRINT 'Errore valore: -- non è permesso!'
  END
```

I Cursori

un cursore è un tipo di variabile che si può utilizzare per scorrere un determinato set di righe di una tabella.Come costruirli:

- dichiarare una variabile per contenere i dati da estrarre col cursore: il cursore contiene una riga quindi ci devono essere delle variabili per ogni colonna che interessa della riga (con il tipo di dati giusto)
- associare il cursore ad una SELECT con il comando DECLARE CURSOR
- utilizzare l'istruzione OPEN per eseguire la SELECT e popolare il cursore
- usare la FETCH INTO per recuperare i dati dal cursore e mettere il contenuto delle colonne nelle corrispettive variabili
- quando finito, usare CLOSE per liberare le risorse del cursore (tranne il nome), per riusarlo bisognerà rifare la OPEN. Se si vuole cancellare totalmente il cursore usare DEALLOCATE

esempi:

```
DECLARE @MyVariabile CURSOR
DECLARE MyCursor CURSOR FOR
   SELECT LastName FROM Person.Contact
SET @MyVariabile =MyCursor
GO
```

uso:

```
OPEN MyCursor
FETCH NEXT FROM MyCursor INTO @contact_name
```

Funzioni già incluse

Funzioni per il parsing ed il calcolo della lunghezza di una stringa	
Datalength (*char_expr*)	Ritorna un intero che indica la lunghezza di una stringa.
substring (*expression,start, length*)	Ritorna una parte di una stringa.
right (*char_exp, int_expr*)	Recupera n caratteri (indicati da un numero intero) dalla parte destra di una stringa
left (*char_exp, int_expr*)	Recupera n caratteri (indicati da un numero intero) dalla parte sinistra di una stringa
Funzioni per la manipolazione di una stringa	
upper (*char_expr*)	Converte tutti i caratteri della stringa in maiscolo
lower (*char_expr*)	Converte tutti i caratteri della stringa in minuscolo.
space (*int_expr*)	Crea una stringa di spazi, la sua lunghezza è indicata dal numero intero nell'argomento .
replicate (*char_expr, int_expr*)	Ripete una stringa un certo numero di volte, indicato dal numero intero nell'argomento.
stuff (*char_exp1, start, length, char_expr2*)	Elimina una determinata porzione di stringa sostituendolo con un'altra, partendo da uno specifico punto di partenza.
reverse (*char_expr*)	Inverte una stringa.
ltrim (*char_expr*) e rtrim (*char_expr*)	Ritorna una stringa dopo averla ripulita degli spazi più a sinistra.
Funzioni per la conversione di una stringa	
ascii (*char_expr*)	Ritorna il codice ASCII per il carattere più a sinistra della stringa.
char (*int_expr*)	Converte un intero rappresentante il codice ASCII in una stringa.
Funzioni per la ricerca all'interno di una stringa	

charindex **(** *expression1* **,** *expression2* [**,** *start_location*] **)**	Ritorna la posizione di partenza di una stringa all'interno di un'altra stringa, come numero intero.
Patindex **(** '*%pattern%*' **,** *expression* **)**	Ritorna la posizione di partenza di una pattern all'interno di un'altra stringa, come numero intero.

Funzioni per le date
Servono a lavorare e soprattutto a manipolare le date:

getdate ()	Data corrente.
datename(*datepart, date_expr*)	Ritorna una specifica parte di una data come nome, il valore è una stringa.
datepart (*datepart, date_expr*)	Ritorna una specifica parte di una data come numero, il valore è un intero.
datediff (*datepart, date_expr1, date_expr2*)	Ritorna l'intervallo tra due date, la misurazione fa riferimento alla parte di data specificata.
dateadd (*datepart, number, date_expr*)	Ritorna una nuova data, creata aggiungendo un valore intero ad un data di input, l'aggiunta fa riferimento alla parte di data specificata

Per le parti di data presenti come argomento delle funzioni facciamo riferimento alla tabella sottostante:

Parte di data	**Abbrevizione**	**Parte di data**	**Abbrevizione**
Year	yy, yyyy	Week	wk, ww
quarter	qq, q	Hour	hh
Month	Mm, m	minute	Mi,n
dayofyear	dy, y	second	ss,s
Day	dd, d	millisecond	Ms

Ecco alcuni esempi:

```
use pubs
go
/* Esempio: getdate() */
select getdate()
/* Esempio: datediff() */
select title,
datediff(yy, pubdate, getdate()) as [Anni dalla data di pubblicazione]
from titles
/* Esempio: datetime() */
select title, datename (mm, pubdate) from titles
select title, datename (dd, pubdate) from titles
/* Esempio: dateadd() */
select dateadd (dd, 2, getdate()) as [Data tra due giorni]
```

Funzioni matematiche

Sono funzioni scalari utili per fare trasformazioni matematiche su input vari (di tipo numerico).

abs (*numeric_expr*)	Data corrente.
ceiling (*numeric_expr*)	Ritorna l'intero più grande od uguale all'espressione numerica di input
rand , (*int_expr*)	Ritorna un numero float casuale tra 0 e 1

Esempi di utilizzo:

```
use pubs
go
/* Esempio: abs() */
select abs(-123) as [Valore assoluto]
/* Esempio: ceiling() */
select ceiling(-123.23) as [Valore assoluto]
/* Esempio: rand() */
Select round(rand() * 10,0) as [Numero casuale]
```

Ci sono anche altre funzioni matematiche ricordiamo tra le altre: exp (*float_expr*), pi (), , round (*numeric_expr, int_expr*), sprt (*float_expr*) e le funzioni trigonometriche.

SQL

Create table

sintassi:

```
CREATE table nome_tabella (
nome_colonna tipo_colonna [ clausola_default ] [ vincoli_di_colonna ]
[ , nome_colonna tipo_colonna [ clausola_default ] [ vincoli_di_colonna ] ... ]
[ , [ vincolo_di tabella] ... ] )
```

nome_colonna: e' il nome della colonna che compone la tabella.

tipo_colonna: e' l'indicazione del tipo di dato che la colonna potra' contenere. I principali tipi previsti dallo standard SQL sono:

- CHARACTER(n) / CHAR(n)

Una stringa a lunghezza fissa di esattamente n caratteri.
- CHARACTER VARYING(n) / VARCHAR(n)
  Una stringa a lunghezza variabile di al massimo n caratteri.
- INTEGER / INT
- SMALLINT
- FLOAT(p)
  Un numero a virgola mobile, con precisione p.
- DECIMAL(p,q)
  Un numero a virgola fissa di almeno p cifre e segno, con q cifre dopo la virgola.
- INTERVAL
  Un periodo di tempo (anni, mesi, giorni, ore, minuti, secondi e frazioni di secondo).
- DATE, TIME e TIMESTAMP
  Un preciso istante temporale. DATE permette di indicare l'anno, il mese e il giorno. Con TIME si possono specificare l'ora, i minuti e i secondi. TIMESTAMP e' la combinazione dei due precedenti.

clausola_default: indica il valore di default che assumera' la colonna se non gliene viene assegnato uno esplicitamente nel momento della creazione della riga. La sintassi da utilizzare e' la seguente:

DEFAULT { <valore> | NULL }

vincoli_di_colonna: sono vincoli di integrita' che vengono applicati al singolo attributo. Sono:

- NOT NULL, che indica che la colonna non puo' assumere il valore NULL.
- PRIMARY KEY, che indica che la colonna e' la chiave primaria della tabella.
- una definizione di riferimento, con cui si indica che la colonna e' una chiave esterna verso la tabella e i campi indicati nella definizione. La sintasi e' la seguente:

  REFERENCES nome_tabella [(colonna1 [, colonna2 ...])]
  [ON DELETE { CASCADE | SET DEFAULT | SET NULL }]
  [ON UPDATE { CASCADE | SET DEFAULT | SET NULL }]

  Le clausole ON DELETE e ON UPDATE indicano quale azione deve essere compiuta nel caso in cui una tupla nella tabella referenziata venga eliminata o aggiornata. Infatti in tali casi nella colonna referenziante (che e' quella che si sta definendo) potrebbero esserci dei valori inconsistenti. Le azioni possono essere:

  - CASCADE: eliminare la tupla contenente la colonna referenziante (nel caso di ON DELETE) o aggiornare anche la colonna referenziante (nel caso di ON UPDATE).
  - SET DEFAULT: assegnare alla colonna referenziante il suo valore di default.
  - SET NULL: assegnare alla colonna referenziante il valore NULL.
- un controllo di valore, con il quale si permette o meno l'assegnazione di un valore alla colonna, in base al risultato di un espressione. La sintassi da usare e':

  CHECK (espressione_condizionale)

  dove espressione_condizionale e' un'espressione che restituisce vero o falso.
  Ad esempio, se stiamo definendo la colonna COLONNA1, definendo il seguente controllo:

  CHECK (COLONNA1 < 1000)

  in tale colonna potranno essere inseriti solo valori inferiori a 1000.

vincolo_di_tabella: sono vincoli di integrita' che possono riferirsi a piu' colonne della tabella. Sono:

- la definizione della chiave primaria:

  PRIMARY KEY (colonna1 [, colonna2 ...])

  Si noti che in questo caso, a differenza della definizione della chiave primaria come vincolo

di colonna, essa puo' essere formata da piu' di un attributo.

- le definizioni delle chiavi esterne:

  FOREIGN KEY (colonna1 [, colonna2 ...]) definizione_di_riferimento

  La definizione_di_riferimento ha la stessa sintassi e significato di quella che puo' comparire come vincolo di colonna.

- un controllo di valore, con la stessa sintassi e significato di quello che puo' essere usato come vincolo di colonna.

Esempio:

CREATE table Publication (ID INTEGER PRIMARY KEY, type CHAR(18) NOT NULL);	La precedente istruzione crea la tabella Publication, formata dalle due colonne ID di tipo INTEGER, e type di tipo CHAR(18). ID e' la chiave primaria della relazione. Sull'attributo type e' posto un vincolo di non nullita'.
CREATE table Book (ID INTEGER PRIMARY KEY REFERENCES Publication(ID), title VARCHAR(160) NOT NULL, publisher INTEGER NOT NULL REFERENCES Publisher(ID), volume VARCHAR(16), series VARCHAR(160), edition VARCHAR(16), pub_month CHAR(3), pub_year INTEGER NOT NULL, note VARCHAR(255));	Crea la relazione Book, formata da nove attributi. La chiave primaria e' l'attributo ID, che e' anche una chiave esterna verso la relazione Publication. Sugli attributi title, publisher e pub_year sono posti dei vincoli di non nullita'. Inoltre l'attributo publisher e' una chiave esterna verso la tabella Publisher.
CREATE table Author (publicationID INTEGER REFERENCES Publication(ID), personID INTEGER REFERENCES Person(ID), PRIMARY KEY (publicationID, personID));	Crea la relazione Author, composta da due attributi: publicationID e personID. La chiave primaria in questo caso e' formata dalla combinazione dei due attributi, come indicato dal vincolo di tabella PRIMARY KEY. PublicationID e' una chiave esterna verso la relazione Publication, mentre personID lo e' verso la relazione Person.

INSERT

sintassi:

```
INSERT INTO nome_tabella [ ( elenco_campi ) ]
VALUES ( elenco_valori )
```

elenco_campi e' l'elenco dei nomi dei campi a cui deve essere assegnato un valore, separati fra loro da una virgola. I campi non compresi nell'elenco assumeranno il loro valore di default o NULL se non hanno un valore di default. *elenco_valori* e' l'elenco dei valori che verranno assegnati ai campi della tabella, nell'ordine e numero specificati dall'elenco_campi o in quello della definizione della tabella (se elenco_campi non viene specificato). I valori possono essere un'espressione scalare del tipo appropriato per il campo o le keyword DEFAULT o NULL, se il campo prevede un valore di default o ammette il valore NULL.

Esempio:

```
INSERT INTO Person VALUES ( 4, 'Spaccapietra', 'S.' );
INSERT INTO Person VALUES ( 5, 'Maryansky', 'F.' );

INSERT INTO Institution ( ID, name, city, country )
VALUES ( 1, '7th IFIP 2.6 Working Conference on Database Semantics (DS-7)',
'Leysin', 'Switzerland' );
```

SELECT

sintassi:

```
SELECT [ ALL | DISTINCT ] lista_elementi_selezione
FROM lista_riferimenti_tabella
[ WHERE espressione_condizionale ]
[ GROUP BY lista_colonne ]
[ HAVING espressione_condizionale ]
[ ORDER BY lista_colonne ]
```

- produce una tabella ottenuta come prodotto cartesiano delle tabelle specificate nella clausola FROM. Ogni elemento della lista_riferimenti_tabella segue la seguente sintassi:

  riferimento_tabella [[AS] alias_tabella]

  Il riferimento puo' essere il nome di una tabella o un' espressione (posta fra parentesi tonde) il cui risultato e' una tabella, quindi, ad esempio, anche un'altra SELECT. L'alias e' un nome che serve per indicare in breve un riferimento di tabella. Nel caso in cui il riferimento di tabella sia un espressione, e' obbligatorio specificare un alias.
- dalla tabella precedente elimina tutte le righe che non soddisfano l'espressione condizionale (cioè le righe per cui l'espressione condizionale restituisce falso come risultato) della clausola WHERE.
- (se e' presente la clausola GROUP BY) le righe della tabella risultante dal passo 2 vengono raggruppate secondo i valori presenti nelle colonne specificate nella clausola GROUP BY. Righe con valori uguali vengono accorpate in un'unica riga. Le colonne non comprese nella clausola devono contenere delle espressioni con funzioni di aggregazione (come ad esempio AVG, che calcola la media) che quindi vengono calcolate producendo un singolo valore per ogni gruppo.
- (se e' presente la clausola HAVING) dal risultato del punto 3 vengono eliminate le righe che non soddisfano l'espressione condizionale della clausola HAVING.
- Vengono calcolate le colonne presenti nella clausola SELECT (quelle nella lista_elementi_selezione). In particolare vengono calcolate le colonne con le funzioni di aggregazione derivanti dal raggruppamento avvenuto al punto 3. Ogni elemento della lista_elementi_selezione segue la seguente sintassi:

  espressione_scalare [[AS] alias_colonna]

  Le espressioni scalari degli elementi della SELECT normalmente coinvolgono le colonne della tabella risultante dal punto 4. Nel caso in cui siano presenti delle ambiguita' a causa di colonne con nomi uguali in due o piu' tabelle comprese nella clausola FOR, esse possono essere risolte prefissando il nome o l'alias della colonna con il nome o l'alias della tabella, separati da un punto. Ad esempio, T.C indica la colonna C della tabella T. L'alias di colonna e' il nome che viene assegnato alla colonna.
  L'intero elenco delle colonne di una tabella puo' essere specificato utilizzando il carattere '*'.
- (se e' presente l'opzione DISTINCT) vengono eliminate le righe che risultano duplicate. Nel caso non sia presente ne' ALL ne' DISTINCT, viene assunto ALL.
- (se e' presente la clausola ORDER BY) le righe della tabella vengono ordinate secondo i valori presenti nelle colonne specificate nella clausola. La sintassi da utilizzare e' la

seguente:

ORDER BY nome_colonna [ASC | DESC] [, nome_colonna [ASC | DESC] ...]

L'ordinamento di default e' quello ascendente. Nel caso si desideri effettuare quello decrescente bisogna specificare l'opzione DESC.

Esempi:

SELECT PUB.*, PER.surname AS S, PER.given_names FROM Publication PUB, Author AUT, Person PER WHERE PUB.ID = AUT.publicationID AND AUT.personID = PER.ID AND PUB.type = 'Book' ORDER BY S	In questo caso la tabella risultante contiene tutte le colonne della tabella Publication (indicata con l'alias PUB definito nella clausola FROM) e le colonne surname e given_names della tabella Person. La clausola FROM genera il prodotto cartesiano delle tabelle Publication, Author e Person, di cui vengono selezionate solo le righe in cui l'identificativo della pubblicazione e quello dell'autore corrispondono. Inoltre ci si limita a considerare solo le pubblicazioni di tipo 'Book'. Per finire la tabella viene ordinata secondo il cognome dell'autore, indicato mediante l'alias S, definito nella clausola SELECT.
SELECT title, volume, pub_year FROM Book WHERE ID IN (SELECT PUB.ID FROM Publication PUB, Author AUT, Person PER WHERE PUB.ID = AUT.publicationID AND AUT.personID = PER.ID AND PUB.type = 'Book' AND PER.surname = 'Knuth')	In questo esempio si vede l'utilizzo di un espressione condizionale che contiene l'operatore IN, il quale restitisce il valore vero se il valore dell'operando alla sua sinistra e' contenuto nella tabella risultato dell'espressione alla sua destra. La query fra parentesi produce una tabella di un'unica colonna contentente gli identificativi delle pubblicazioni di tipo 'Book' di cui Knuth e' authore. La query piu' esterna quindi estrae dalla tabella Book le informazioni dei libri con tali identificativi.

Le funzioni COUNT, SUM, AVG, MAX e MIN, calcolano rispettivamente il numero, la somma, la media aritmetica, il massimo e il minimo dei valori scalari presenti nella colonna a cui vengono applicate

UPDATE

sintassi:

```
UPDATE nome_tabella
SET elenco_assegnamenti
[ WHERE espressione_condizionale ]
```

Gli assegnamenti vengono specificati nella forma:

nome_colonna = espressione_scalare

Esempio:

UPDATE Person SET given_names = 'Stefano' WHERE surname = 'Spaccapietra'	La precedente istruzione cambia il valore della colonna given_name della tabella Person nelle righe (nel nostro caso e' una sola) in cui la

	colonna surname ha valore 'Spaccapietra'.

DELETE

sintassi:

```
DELETE FROM nome_tabella
[ WHERE espressione_condizionale ]
```

L'istruzione delete elimina da una tabella tutte le righe che soddisfano l'espressione condizionale della clausola WHERE. Se WHERE non viene specificata, vengono cancellate tutte le righe della tabella.

Se nella definizione della tabella si sono specificate le clausole ON UPDATE o ON DELETE, nel momento in cui vengono eseguite queste operazioni viene eseguita anche l' azioni che era stata prevista sulle colonne referenziate (CASCADE, SET DEFAULT o SET NULL).

DROP TABLE

sintassi:

```
DROP table nome_tabella { REStrICT | CASCADE }
```

nome_tabella e' il nome della tabella che deve essere eliminata.
Se si specifica la clausola CASCADE vengono automaticamente eliminati i vincoli di integrita' e le viste (view) in cui la tabella e' coinvolta. Viceversa, se si specifica la clausola REStrICT ed esistono dei vincoli di integrita' o delle viste che si riferiscono alla tabella, l'operazione fallisce.

ALTER

Aggiunta di una nuova colonna nella tabella

```
ALTER table nome_tabella ADD [ COLUMN ] definizione_colonna
```

nome_tabella e' il nome della tabella che si vuole modificare.
La definizione della colonna segue la stessa sintassi vista nella lezione "Creare il database" nella spiegazione dell'istruzione CREATE table.

Eliminazione di una colonna dalla tabella

```
ALTER table nome_tabella
DROP [ COLUMN ] nome_colonna { REStrICT | CASCADE }
```

nome_colonna e' il nome della colonna che si vuole eliminare. Le clausole RESStrIC e CASCADE si comportano esattamente come nell'istruzione DROP table vista precedentemente.

Modifica del valore di default di una colonna

```
ALTER table nome_tabella
ALTER [ COLUMN ] nome_colonna { SET clausola_default | DROP DEFAULT }
```

Eliminazione di un vincolo della tabella

```
ALTER table nome_tabella
DROP CONStrAINT nome_vincolo { REStrICT | CASCADE }
```

Elimina il vincolo identificato dal nome specificato. L'operazione fallisce se e' stata specificata la clausola REStrICT ed esistono altri vincoli che dipendono da quello che si intende eliminare. Specificando la clausola CASCADE l'operazione verra' sempre completata con successo, cancellando inoltre i vincoli dipendenti da quello eliminato.

Aggiunta di un vincolo alla tabella

ALTER table nome_colonna
ADD vincolo_di_tabella

JOIN

Employees:

Employee_ID	Name
01	Hansen, Ola
02	Svendson, Tove
03	Svendson, Stephen
04	Pettersen, Kari

Orders:

Prod_ID	Product	Employee_ID
234	Printer	01
657	Table	03
865	Chair	03

select data from two tables with the JOIN keyword, like this:

INNER JOIN
Sintassi:

```
SELECT field1, field2, field3
FROM first_table
INNER JOIN second_table
ON first_table.keyfield = second_table.foreign_keyfield
```

Esempio:

```
SELECT Employees.Name, Orders.Product
FROM Employees
INNER JOIN Orders
ON Employees.Employee_ID=Orders.Employee_ID
```

The INNER JOIN returns all rows from both tables where there is a match. If there are rows in Employees that do not have matches in Orders, those rows will **not** be listed.

Result

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair

LEFT JOIN

Sintassi:

```
SELECT field1, field2, field3
FROM first_table
LEFT JOIN second_table
ON first_table.keyfield = second_table.foreign_keyfield
```

Esempio:

```
SELECT Employees.Name, Orders.Product
FROM Employees
LEFT JOIN Orders
ON Employees.Employee_ID=Orders.Employee_ID
```

The LEFT JOIN returns all the rows from the first table (Employees), even if there are no matches in the second table (Orders). If there are rows in Employees that do not have matches in Orders, those rows **also** will be listed.

Result

Name	Product
Hansen, Ola	Printer
Svendson, Tove	
Svendson, Stephen	Table
Svendson, Stephen	Chair
Pettersen, Kari	

RIGHT JOIN
Sintassi:

```
SELECT field1, field2, field3
FROM first_table
RIGHT JOIN second_table
ON first_table.keyfield = second_table.foreign_keyfield
```

Esempio:

```
SELECT Employees.Name, Orders.Product
FROM Employees
RIGHT JOIN Orders
ON Employees.Employee_ID=Orders.Employee_ID
```

The RIGHT JOIN returns all the rows from the second table (Orders), even if there are no matches in the first table (Employees). If there had been any rows in Orders that did not have matches in Employees, those rows **also** would have been listed.

Result

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair