Table of Contents

SQL BASE

OFFSET e Fetch

I comandi OFFSET e FETCH vengono utilizzati per restituire un sottoinsieme di righe da una tabella.

- **OFFSET** specifica il numero di righe da saltare prima di restituire i risultati.
- **FETCH** specifica il numero di righe da restituire.

Pseudocodice

```
SELECT
  *
FROM
  tabella
OFFSET
  offset_row_count
ROWS
FETCH
  fetch_row_count
ROWS
```

Esempio

Supponiamo di avere una tabella prodotti con i seguenti dati:

id	nome	prezzo
1	Cappello	10

id	nome	prezzo
2	Pantaloni	20
3	Camicia	30
4	Scarpe	40

La seguente query restituirà le prime 10 righe della tabella, partendo dalla quarta riga:

```
SELECT
  *
FROM
  prodotti
OFFSET
  3
ROWS
FETCH
  10
ROWS
```

TOP WITH TIES

Il comando TOP WITH TIES è simile al comando TOP, ma restituisce anche le righe con lo stesso valore del campo specificato nella clausola ORDER BY.

Pseudocodice

```
SELECT
  *
FROM
  tabella
ORDER BY
  campo
TOP
  n_righe
WITH TIES
```

Esempio

Supponiamo di avere una tabella studenti con i seguenti dati:

id	nome	cognome	voto
1	Mario	Rossi	20
2	Anna	Bianchi	20
3	Giovanni	Verdi	19

La seguente query restituirà i primi due studenti con il voto più alto:

```
SELECT
  *
FROM
  studenti
ORDER BY
  voto
TOP
  2
```

```
WITH TIES
```

GROUP BY

Il comando GROUP BY raggruppa le righe di una tabella in base a un valore comune.

Pseudocodice

```
SELECT
  *
FROM
  tabella
GROUP BY
  campo
```

Esempio

Supponiamo di avere una tabella ordini con i seguenti dati:

id	cliente	prodotto	quantità
1	Mario	Cappello	1
2	Anna	Pantaloni	2
3	Giovanni	Camicia	3

La seguente query restituirà il numero di ordini per ogni cliente:

```
SELECT
  cliente,
  COUNT(*) AS numero_ordini
FROM
  ordini
GROUP BY
  cliente
```

HAVING

La clausola HAVING viene utilizzata per filtrare i risultati di una query GROUP BY.

Pseudocodice

```
SELECT
  *
FROM
  tabella
GROUP BY
  campo
HAVING
  condizione
```

Esempio

La seguente query restituirà il numero di ordini per ogni cliente, ma solo se il numero di ordini è maggiore di 2:

```
SELECT
  cliente,
  COUNT(*) AS numero_ordini
FROM
  ordini
GROUP BY
  cliente
HAVING
```

```
  numero_ordini > 2
```

Spero che questa spiegazione sia stata utile.

GROUPING SET

Il comando GROUPING SET viene utilizzato per restituire più combinazioni di risultati da una query GROUP BY.

Pseudocodice

```
SELECT
  *
FROM
  tabella
GROUP BY
  campo1,
  campo2
GROUPING SET
  (campo1, campo2),
  (campo1),
  (campo2)
```

Esempio

Supponiamo di avere una tabella ordini con i seguenti dati:

id	cliente	prodotto	quantità
1	Mario	Cappello	1

id	cliente	prodotto	quantità
2	Anna	Pantaloni	2
3	Giovanni	Camicia	3

La seguente query restituirà il numero di ordini per ogni cliente e prodotto, nonché il numero totale di ordini:

```
SELECT
  cliente,
  prodotto,
  COUNT(*) AS numero_ordini
FROM
  ordini
GROUP BY
  cliente,
  prodotto
GROUPING SET
  (cliente, prodotto),
  (cliente),
  ()
```

CUBE

Il comando CUBE viene utilizzato per restituire tutte le combinazioni possibili di risultati da una query GROUP BY.

Pseudocodice

```
SELECT
  *
FROM
  tabella
```

```
GROUP BY
  campo1,
  campo2
CUBE
```

Esempio

La seguente query restituirà il numero di ordini per ogni cliente e prodotto, nonché il numero totale di ordini:

```
SELECT
  cliente,
  prodotto,
  COUNT(*) AS numero_ordini
FROM
  ordini
GROUP BY
  cliente,
  prodotto
CUBE
```

ROLLUP

Il comando ROLLUP viene utilizzato per restituire tutte le combinazioni di risultati da una query GROUP BY, partendo dai gruppi più grandi fino ai gruppi più piccoli.

Pseudocodice

```
SELECT
  *
FROM
  tabella
GROUP BY
  campo1,
  campo2
ROLLUP
```

Esempio

La seguente query restituirà il numero di ordini per ogni cliente, il numero totale di ordini per ogni prodotto, e il numero totale di ordini:

```
SELECT
  cliente,
  prodotto,
  COUNT(*) AS numero_ordini
FROM
  ordini
GROUP BY
  cliente,
  prodotto
ROLLUP
```

CREATE/DROP DATABASE

I comandi CREATE DATABASE e DROP DATABASE vengono utilizzati per creare e rimuovere database.

Pseudocodice

```
CREATE DATABASE nome_database

DROP DATABASE nome_database
```

Esempio

La seguente query creerà un database chiamato mio_database:

```
CREATE DATABASE mio_database

DROP DATABASE mio_database
```

CREATE/ALTER/DROP SCHEMA

I comandi CREATE SCHEMA, ALTER SCHEMA e DROP SCHEMA vengono utilizzati per creare, modificare e rimuovere schemi.

Pseudocodice

```
CREATE SCHEMA nome_schema
```

```
ALTER SCHEMA nome_schema

DROP SCHEMA nome_schema
```

Esempio

La seguente query creerà uno schema chiamato mio_schema:

```
CREATE SCHEMA mio_schema
```

La seguente query aggiungerà una tabella chiamata ordini allo schema mio_schema:

```
ALTER SCHEMA mio_schema
ADD TABLE ordini (
  id INT,
  cliente VARCHAR(255),
  prodotto VARCHAR(255),
  quantità INT
);
```

La seguente query rimuoverà lo schema mio_schema:

```
DROP SCHEMA mio_schema
```

CREATE\DROP\ALTER\RENAME\TRUNCATE TABLE

I comandi CREATE TABLE, DROP TABLE, ALTER TABLE, RENAME TABLE e TRUNCATE TABLE vengono utilizzati per gestire le tabelle di un database.

CREATE TABLE

Il comando CREATE TABLE viene utilizzato per creare una nuova tabella.

Pseudocodice

```
CREATE TABLE nome_tabella (
  colonna1 tipo_dato,
  colonna2 tipo_dato,
  ...
);
```

Esempio

La seguente query creerà una tabella chiamata ordini con le seguenti colonne:

```
CREATE TABLE ordini (
  id INT,
  cliente VARCHAR(255),
  prodotto VARCHAR(255),
  quantità INT
);
```

DROP TABLE

Il comando DROP TABLE viene utilizzato per rimuovere una tabella.

Pseudocodice

```
DROP TABLE nome_tabella;
```

Esempio

La seguente query rimuoverà la tabella ordini:

```
DROP TABLE ordini;
```

ALTER TABLE

Il comando ALTER TABLE viene utilizzato per modificare una tabella.

Pseudocodice

```
ALTER TABLE nome_tabella
ADD colonna1 tipo_dato,
ADD colonna2 tipo_dato,
...;

ALTER TABLE nome_tabella
DROP colonna1,
DROP colonna2,
```

```
...;

ALTER TABLE nome_tabella
MODIFY colonna1 tipo_dato,
MODIFY colonna2 tipo_dato,
...;
```

Esempio

La seguente query aggiungerà una colonna chiamata data alla tabella ordini:

```
ALTER TABLE ordini
ADD data DATE;
```

RENAME TABLE

Il comando RENAME TABLE viene utilizzato per rinominare una tabella.

Pseudocodice

```
RENAME TABLE nome_tabella TO nuovo_nome_tabella;
```

Esempio

La seguente query rinominerà la tabella ordini in ordini_aggiornati:

```
RENAME TABLE ordini TO ordini_aggiornati;
```

TRUNCATE TABLE

Il comando TRUNCATE TABLE viene utilizzato per eliminare tutti i dati da una tabella.

Pseudocodice

```
TRUNCATE TABLE nome_tabella;
```

Esempio

La seguente query rimuoverà tutti i dati dalla tabella ordini:

```
TRUNCATE TABLE ordini;
```

SEQUENCE

Il comando SEQUENCE viene utilizzato per creare una sequenza.

Pseudocodice

```
CREATE SEQUENCE nome_sequenza
INCREMENT BY 1
START WITH 1
MINVALUE 1
MAXVALUE 9999999999
CYCLE
NO CACHE;
```

Esempio

La seguente query creerà una sequenza chiamata id_ordini che inizia a 1 e incrementa di 1 a ogni valore successivo:

```
CREATE SEQUENCE id_ordini
INCREMENT BY 1
START WITH 1
MINVALUE 1
MAXVALUE 9999999999
CYCLE
NO CACHE;
```

TEMPORARY TABLE

Il comando TEMPORARY TABLE viene utilizzato per creare una tabella temporanea.

Pseudocodice

```
CREATE TEMPORARY TABLE nome_tabella (
```

```
  colonna1 tipo_dato,
  colonna2 tipo_dato,
  ...
);
```

Esempio

La seguente query creerà una tabella temporanea chiamata ordini_temporanei:

```
CREATE TEMPORARY TABLE ordini_temporanei (
  id INT,
  cliente VARCHAR(255),
  prodotto VARCHAR(255),
  quantità INT
);
```

SYNONYMS

Il comando SYNONYMS viene utilizzato per creare un sinonimo per una tabella, una vista o un'altra risorsa di database.

Pseudocodice

```
CREATE SYNONYM nome_sinonimo FOR nome_risorsa;
```

Esempio

La seguente query creerà un sinonimo chiamato ordini_sinonimo per la tabella ordini:

```
CREATE SYNONYM ordini_sinonimo FOR ordini;
```

COALESCE

La funzione COALESCE viene utilizzata per restituire il primo valore non NULL di una lista di valori.

Pseudocodice

```
COALESCE(valore1, valore2, ..., valore_n)
```

Esempio

Supponiamo di avere una tabella ordini con le seguenti colonne:

id	cliente	prodotto	quantità

La seguente query restituirà il nome del cliente per l'ordine con l'id 1, se il valore della colonna cliente è non NULL, altrimenti restituirà il valore "Anonimo":

```
SELECT COALESCE(cliente, 'Anonimo')
FROM ordini
WHERE id = 1;
```

CASE

La clausola CASE viene utilizzata per eseguire un'azione in base al valore di un'espressione.

Pseudocodice

```
CASE
WHEN condizione THEN azione1
WHEN condizione THEN azione2
...
ELSE azione_default
END
```

Esempio

Supponiamo di avere una tabella ordini con le seguenti colonne:

id	cliente	prodotto	quantità

La seguente query restituirà la categoria del prodotto per l'ordine con l'id 1, in base alla quantità ordinata:

```SQL
SELECT
  CASE
    WHEN quantità > 10 THEN 'Prodotto grande'
    WHEN quantità > 5 THEN 'Prodotto medio'
    ELSE 'Prodotto piccolo'
  END AS categoria
FROM ordini
WHERE id = 1;
```

NULLIF

La funzione NULLIF viene utilizzata per restituire NULL se due espressioni hanno lo stesso valore, altrimenti restituisce la prima espressione.

Pseudocodice

```
NULLIF(espressione1, espressione2)
```

Esempio

Supponiamo di avere una tabella ordini con le seguenti colonne:

id	cliente	prodotto	quantità

La seguente query restituirà NULL se il valore della colonna prodotto è uguale a "Cappello",

altrimenti restituirà il valore della colonna prodotto:

```
SELECT NULLIF(prodotto, 'Cappello')
FROM ordini;
```

IIF

La funzione IIF è simile alla clausola CASE, ma è più concisa.

Pseudocodice

```
IIF(condizione, azione1, azione2)
```

Esempio

L'esempio della clausola CASE può essere riscritto utilizzando la funzione IIF come segue:

```
SELECT
  IIF(quantità > 10, 'Prodotto grande',
    IIF(quantità > 5, 'Prodotto medio', 'Prodotto piccolo')) AS
categoria
FROM ordini
WHERE id = 1;
```

INSERT INTO

Il comando INSERT INTO viene utilizzato per inserire nuove righe in una tabella.

Pseudocodice

```
INSERT INTO tabella (colonna1, colonna2, ...)
VALUES (valore1, valore2, ...);
```

Esempio

Supponiamo di avere una tabella ordini con le seguenti colonne:

id	cliente	prodotto	quantità

```
INSERT INTO ordini (id, cliente, prodotto, quantità)
VALUES (1, 'Mario', 'Cappello', 1);
```

UPDATE

Il comando UPDATE viene utilizzato per aggiornare le righe di una tabella.

Pseudocodice

```
UPDATE tabella
SET colonna1 = valore1,
  colonna2 = valore2,
  ...
WHERE condizione;
```

Esempio

Supponiamo di avere una tabella ordini con le seguenti colonne:

id	cliente	prodotto	quantità

```
UPDATE ordini
SET quantità = 2
WHERE cliente = 'Mario'
AND prodotto = 'Cappello';
```

DELETE

Il comando DELETE viene utilizzato per eliminare le righe da una tabella.

Pseudocodice

```
DELETE FROM tabella
WHERE condizione;
```

Esempio

Supponiamo di avere una tabella ordini con le seguenti colonne:

id	cliente	prodotto	quantità

La seguente query eliminerà tutte le righe dalla tabella ordini:

```
DELETE FROM ordini;
```

MERGE

Il comando MERGE viene utilizzato per combinare i dati da due tabelle.

Pseudocodice

```
MERGE INTO tabella1
USING tabella2
ON condizione
WHEN MATCHED THEN UPDATE SET colonna1 = valore1,
  colonna2 = valore2,
  ...
WHEN NOT MATCHED THEN INSERT (colonna1, colonna2, ...)
VALUES (valore1, valore2, ...);
```

Esempio

Supponiamo di avere due tabelle ordini_vecchi e ordini_nuovi con le seguenti colonne:

id	cliente	prodotto	quantità

La seguente query aggiornerà le righe della tabella ordini_vecchi con i dati della tabella ordini_nuovi, se le due righe hanno lo stesso valore per la colonna id:

```
MERGE INTO ordini_vecchi
USING ordini_nuovi
ON ordini_vecchi.id = ordini_nuovi.id
WHEN MATCHED THEN UPDATE SET
  ordini_vecchi.cliente = ordini_nuovi.cliente,
  ordini_vecchi.prodotto = ordini_nuovi.prodotto,
  ordini_vecchi.quantità = ordini_nuovi.quantità;
```

SUBQUERIES

Le sottoquery sono query contenute all'interno di altre query.

Pseudocodice

```
SELECT
  colonna1,
  colonna2,
  ...
FROM
  tabella
WHERE condizione = (
    SELECT
      colonna1,
      colonna2,
      ...
    FROM
      tabella
    WHERE condizione
  );
```

Esempio

Supponiamo di avere una tabella ordini con le seguenti colonne:

id	cliente	prodotto	quantità

La seguente query restituirà i nomi dei clienti che hanno effettuato più di un ordine:

```
SELECT
  cliente
FROM
  (
    SELECT
      cliente,
      COUNT(*) AS numero_ordini
    FROM
      ordini
    GROUP BY
      cliente
    HAVING
      numero_ordini > 1
  ) AS t;
```