


Prepared statement

```
1  PREPARE statement_name FROM  
2  "SELECT ..."  
3  FROM ...  
   ...";
```

A diagram illustrating a prepared statement. It features a dark rectangular box containing SQL code. Overlaid on the code are several blue circles. Two of these circles are larger and contain the number '3', positioned near the 'FROM' keyword on line 3. Other smaller blue circles are scattered around the code, some overlapping it.

```
1  PREPARE statement_name  
2  "SELECT ..."  
3  FROM ...  
4  ...";
```

A diagram illustrating a prepared statement, similar to the one on the left but with a different layout. It features a dark rectangular box containing SQL code. Overlaid on the code are several blue circles of varying sizes, some overlapping the text. The circles are distributed across the lines of code, with a notable cluster on line 2.

Soluzione

I prepared statement hanno la seguente sintassi dove la parola chiave FROM è fondamentale.

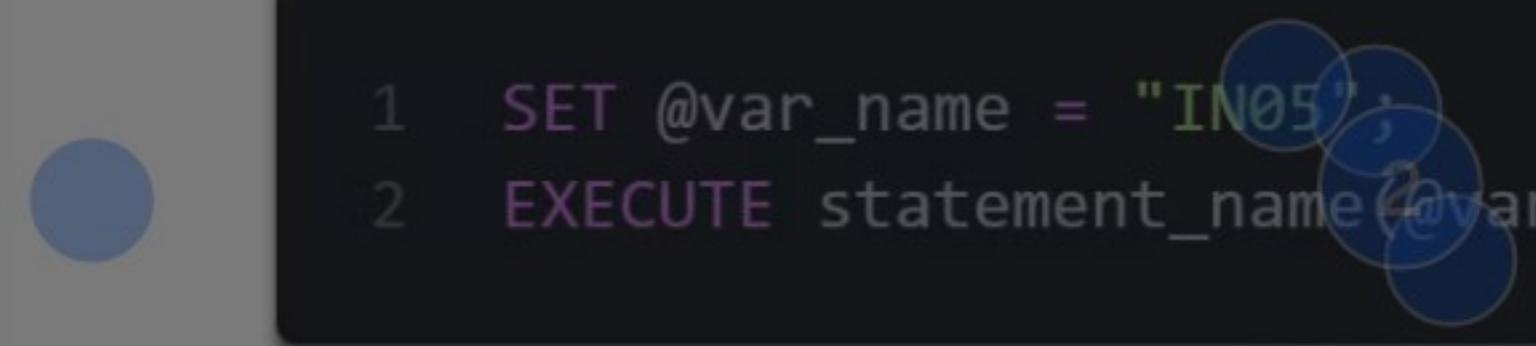
```
1  -- Come si crea un prepared statement
2  PREPARE statement_name FROM
3      "SELECT ...
4      FROM ...
5      ...";
```

Come usare lo statement

```
1 SET @var_name = 'IN05';  
2 EXECUTE statement_name USING @var_name;
```



```
1 SET @var_name = "IN05";  
2 EXECUTE statement_name(@var_name);
```

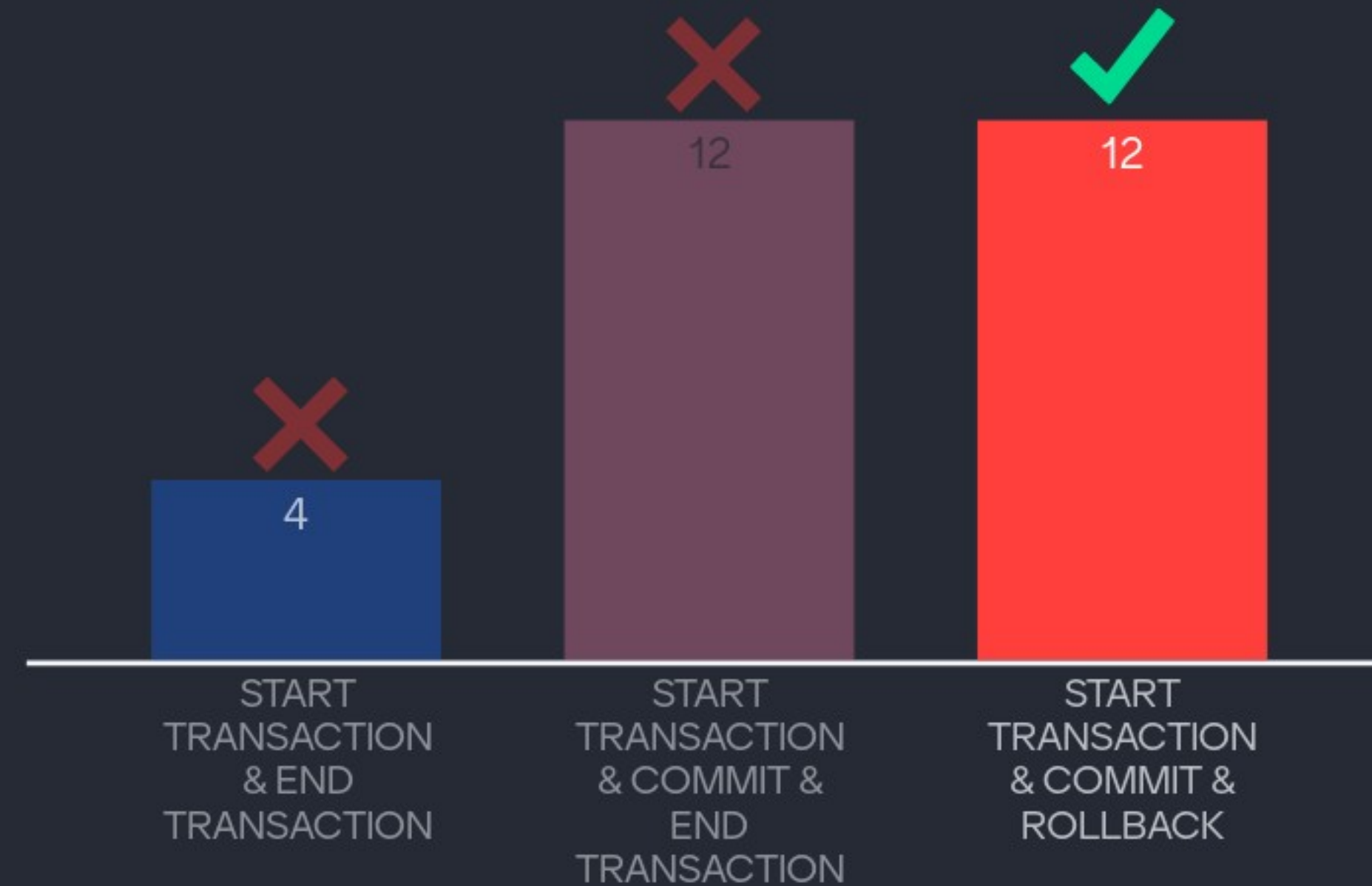


Soluzione

I prepared statement si possono chiamare utilizzando la parola chiave EXECUTE e passando la variabile da utilizzare tramite la parola chiave USING.

```
1  -- Come si usa uno statement
2  SET @var_name = "IN05";
3  EXECUTE statement_name USING @var_name;
```

Elementi della TRANSAZIONE? (in MySQL)



Soluzione

Le Transazioni ci garantiscono queste proprietà:

- Atomicità
- Consistenza
- Isolamento
- Durabilità / Persistenza

La sintassi delle transazioni è fortemente dipendente dal motore, ma in MySQL si utilizzano le seguenti parole chiave:

- `START TRANSACTION`: specifica l'inizio della transazione (le operazioni non vengono ancora eseguite sul DB)
- `COMMIT`: le operazioni specificate a partire dal `begin transaction` vengono eseguite.
- `ROLLBACK`: si rinuncia all'esecuzione delle operazioni specificate dopo l'ultimo `begin transaction`.

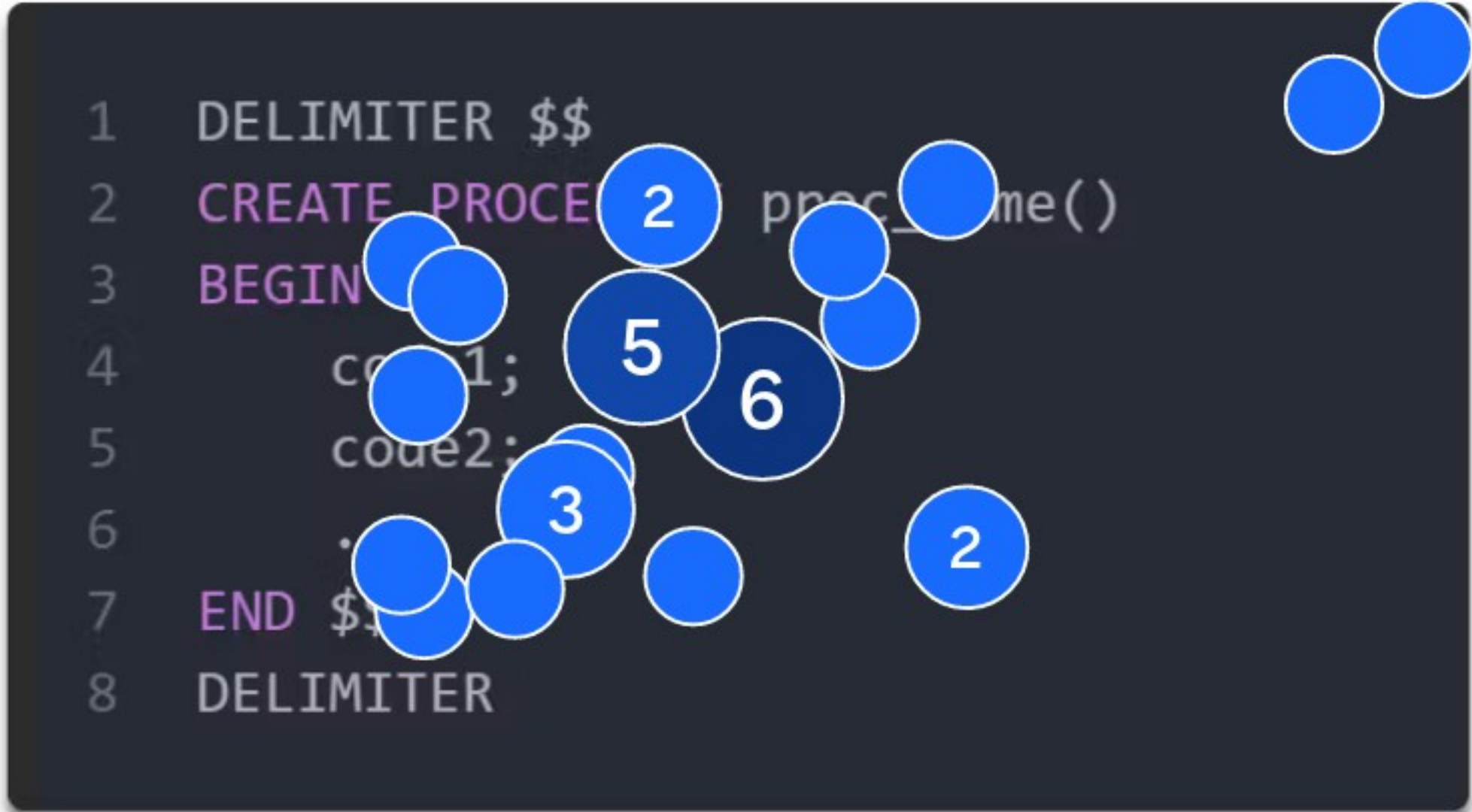
Soluzione

Quindi la corretta sintassi per una transazione può essere:

```
1  -- Transizione
2  START TRANSACTION;
3
4  operazioni
5
6  COMMIT;
```


SP corretta? (in MySQL)

```
1 DELIMITER $$
2 CREATE PROCEDURE proc_name()
3     code1;
4     code2;
5     ...
6 $$
7 DELIMITER
8
```



The diagram shows the same MySQL stored procedure definition as the left panel, but with several blue circles and numbers indicating syntax errors:

- Line 2: "CREATE PROCEDURE" is circled with a "2".
- Line 3: "BEGIN" is circled with a "3".
- Line 4: "code1;" is circled with a "5".
- Line 5: "code2;" is circled with a "6".
- Line 6: "..." is circled with a "2".
- Line 7: "END \$\$" is circled with a "3".
- Line 8: "DELIMITER" is circled with a "2".

Soluzione

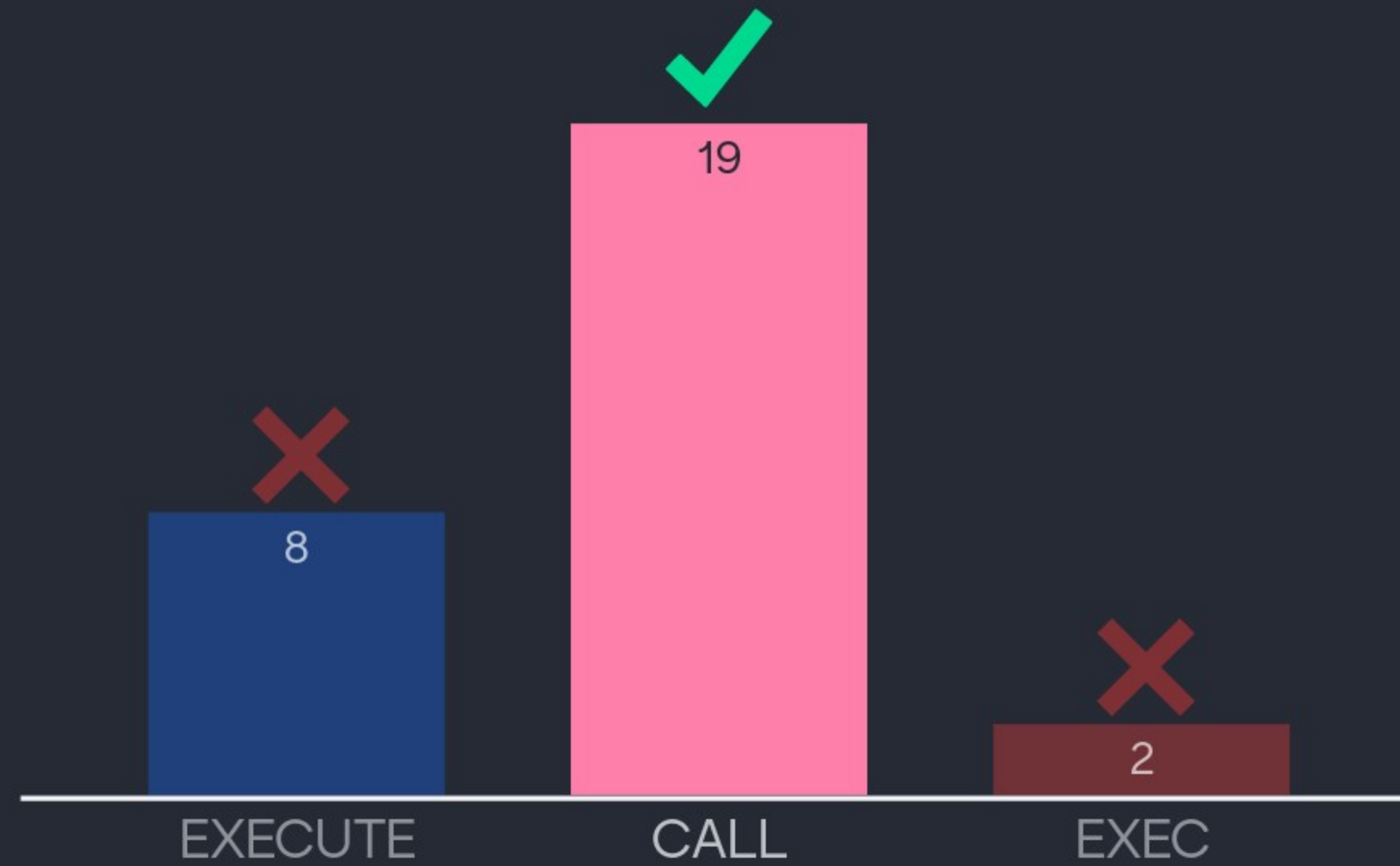
Una Stored Procedure può anche essere vista come un'insieme di comandi SQL con parametri di input e output che possono restituire dei recordset. Le SP risultano essere fondamentali per incapsulare la logica di accesso alle tabelle e per la manipolazione dei dati, aiutando così la creazione di un livello di astrazione del modello fisico del database.

```
1  -- Create stored procedure
2  DELIMITER $$
3  CREATE PROCEDURE proc_name(DIREZIONE NOME_var TIPO, IN var1 INT(4))
4  BEGIN
5      code1;
6      code2;
7      ...
8  END $$
9  DELIMITER
```

Come riceve i parametri un SP?



Eseguire SP in MySQL



Soluzione

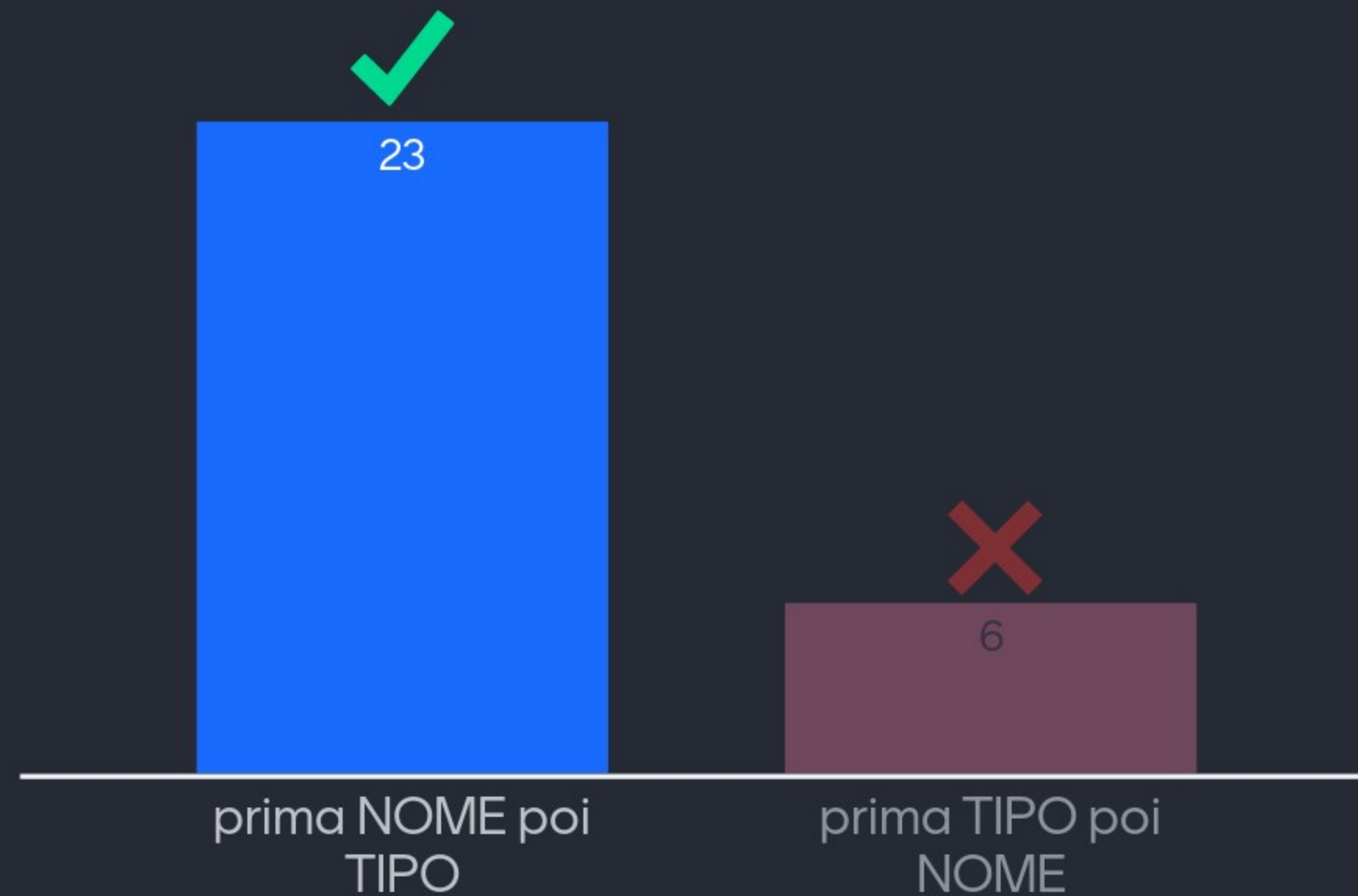
Una SP riceve i parametri come DIREZIONE NOME TIPO.

- Dove la DIREZIONE può essere di sola lettura (IN) o di in sola scrittura (OUT) o sia leggibili che scrivibili (INOUT).
- Il NOME che identificherà quella variabile e ci sarà utile per utilizzarla all'interno della SP.
- Il TIPO invece è il tipo della variabile, ad esempio un INT(4) o un CHAR(9) e così avanti.

Invece per chiamare una procedura si utilizza la parola chiave CALL passando tra parentesi i parametri.

```
1  -- Chiamare una SP
2  CALL nome_sp(parametro1, parametro2)
3  CALL MonteOre(41, @ore);
```

Come riceve i parametri un UDF?



Soluzione

```
1  -- UDF
2  CREATE FUNCTION func_name
3  (param1 tipo1, param2 tipo2, ...)
4  RETURNS tipo
5  [NOT] DETERMINISTIC
6  BEGIN
7      statements
8  END
9
```


Soluzione

Una User Defined Function (UDF) viene definita come nell'immagine sotto. Con i parametri sempre specificati con prima il NOME e poi il TIPO. In più per il valore che vogliamo ritornare è bene specificare DETERMINISTIC o NOT DETERMINISTIC i quali aiutano il motore a capire come ottimizzare il codice.

DETERMINISTIC: se l'input e lo stato del DB non variano, l'output non varia.

NOT DETERMINISTIC: l'output può variare anche se l'input non varia. Un esempio è quando si utilizzano funzioni come `now()` o `rand()`.

Creazione di un Trigger

```
1 CREATE TRIGGER nome quando
2 ON nome_tabella
3 FOR EACH ROW
4 BEGIN
5     codice
6 END
```



```
1 CREATE TRIGGER nome quando
2 ON nome_tabella
3 BEGIN
4     codice
5 END
6
```

Soluzione

Il Trigger è un'operazione da eseguire quando si verifica un certo evento. È utile perché non è necessario chiamarlo, ma parte automaticamente quando si effettua una operazione su una certa tabella. Se si utilizzano i Trigger in MySQL bisogna ricordarsi non possono usare SP, UDF e prepared statement. Inoltre la parola chiave FOR EACH ROW è obbligatoria.

```
1  -- Trigger
2  CREATE TRIGGER nome quando
3  ON nome_tabella
4  FOR EACH ROW
5  BEGIN
6      codice
7  END
```