INSERT [语句](http://doc.cockroachchina.baidu.com/#develop/sql-statements/overview/) 可往表中插入一行或者多行数据。遇到唯一性约束的值冲突时，加上 ON CONFLICT 分句会比单纯的INSERT更加好用。

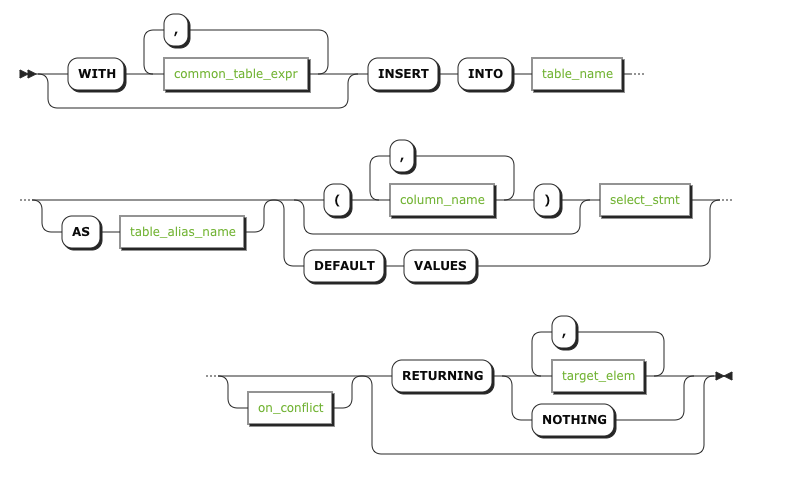
性能的最佳实践

* 执行一个单独的 [多行 INSERT](http://doc.cockroachchina.baidu.com/#develop/sql-statements/INSERT/#insert-multiple-rows-into-an-existing-table) 语句比多个单行的 INSERT 语句更快。往一张现有表中插入大量的数据时 ，建议使用多行 INSERT 语句代替多个单行INSERT 语句。
* 往一张新表写入数据时，[IMPORT](http://doc.cockroachchina.baidu.com/#develop/sql-statements/IMPORT/) 语句性能比 INSERT 语句更好。
* 在传统的SQL数据库中，生成和回收唯一性的IDs需要使用INSERT 和 SELECT。在CockroachDB中，用RETURNING 分句和 INSERT 代替。详情请参阅 [Insert and Return Values](http://doc.cockroachchina.baidu.com/#develop/sql-statements/INSERT/#insert-and-return-values)

所需权限

用户必须拥有表的 INSERT [权限](http://doc.cockroachchina.baidu.com/#deploy/access-management/privileges/) 。 如需使用 ON CONFLICT DO UPDATE，用户还要拥有表的 UPDATE 权限 。

概要



参数

| **参数** | **描述** |
| --- | --- |
| common\_table\_expr | 查看[Common Table Expressions](http://doc.cockroachchina.baidu.com/#develop/sql-syntax/common-table-expressions/). |
| table\_name | 要写入数据的表。 |
| AS table\_alias\_name | 表名的别名。当提供别名时，实际的表名会隐藏。 |
| column\_name | 写入数据时要填充的列名。 |
| select\_stmt | 选择查询。 每个值必须与对应列的数据类型匹配。 此外，如果列名称在INTO之后列出,值必须按相应的顺序排列; 否则，值的顺序必须遵循表中列的声明顺序。 |
| DEFAULT VALUES | 使用默认值填充所有列时，用DEFAULT VALUES代替select\_stmt。 使用默认值来填充特定列时，要将该值保留在select\_stmt之外（leave the value out of the select\_stmt），或在适当的位置使用DEFAULT。 请参阅下面的插入默认值示例。 |
| RETURNING target\_list | 基于插入的行返回值，其中target\_list可以是表中的特定列名，\*表示所有列，或使用scalar表达式计算。 请参阅下面的插入和返回值示例。  在事务中，使用RETURNING NOTHING代表响应中不返回任何内容，包括不返回受影响的行数。 |

ON CONFLICT 从句

正常情况下, 当插入数据的一个或多个列与唯一键约束冲突时，CockroachDB会返回错误。要更新受影响的行，请使用ON CONFLICT子句指定具有唯一键约束的列，DO UPDATE SET表达式指定要更新的列（支持所有[UPDATE](http://doc.cockroachchina.baidu.com/#develop/sql-statements/UPDATE/)语句支持的SET表达式，包括带有WHERE从句的）。为避免更新时影响要插入的新行，将ON CONFLICT设置为DO NOTHING。 请参阅下面的[ON CONFLICT时更新值] 和 [当ON CONFLICT时不要更新值] 示例。

如果SET表达式中的值导致唯一性冲突，CockroachDB将返回错误。

作为“ON CONFLICT”子句的简写替代，您可以使用[UPSERT](http://doc.cockroachchina.baidu.com/#develop/sql-statements/UPSERT/)语句。 但是，UPSERT不允许您指定具有唯一约束的列; 它假定这一列是主键。 因此，使用ON CONFLICT更灵活。

示例

下面的所有例子都假设你已经创建了一个表accounts：

> **CREATE** **TABLE** accounts(

id INT **DEFAULT** unique\_rowid(),

balance DECIMAL

);

插入一个单行

> **INSERT** **INTO** accounts (balance, id) **VALUES** (10000.50, 1);

> **SELECT** \* **FROM** accounts;

+----+---------+

| id | balance |

+----+---------+

| 1 | 10000.5 |

+----+---------+

如果未列出列名，则INSERT语句将按声明顺序指定表的列：

> **SHOW** **COLUMNS** **FROM** accounts;

+---------+---------+-------+----------------+

| Field | Type | Null | Default |

+---------+---------+-------+----------------+

| id | INT | false | unique\_rowid() |

| balance | DECIMAL | true | NULL |

+---------+---------+-------+----------------+

> **INSERT** **INTO** accounts **VALUES** (2, 20000.75);

> **SELECT** \* **FROM** accounts;

+----+----------+

| id | balance |

+----+----------+

| 1 | 10000.50 |

| 2 | 20000.75 |

+----+----------+

在现有表中插入多行

多行插入比多个单行INSERT语句更快。 为了达到最佳性能，我们建议在一个多行INSERT语句中对多行进行批处理，而不是使用多个单行INSERT语句。 监视不同批量大小（10行，100行，1000行）的性能，基于实验结果确定你的服务最佳批量大小。

> **INSERT** **INTO** accounts (id, balance) **VALUES** (3, 8100.73), (4, 9400.10);

> **SELECT** \* **FROM** accounts;

+----+----------+

| id | balance |

+----+----------+

| 1 | 10000.50 |

| 2 | 20000.75 |

| 3 | 8100.73 |

| 4 | 9400.10 |

+----+----------+

在新建表中插入多行数据

往一张新表写入数据时， [IMPORT](http://doc.cockroachchina.baidu.com/#develop/sql-statements/IMPORT/) 语句性能会比INSERT 性能更加好 。

通过SELECT 语句插入

> **SHOW** COLUMS **FROM** other\_accounts;

+*--------+---------+-------+---------+*

| Field | Type | Null | Default |

+*--------+---------+-------+---------+*

| number | INT | false | NULL |

| amount | DECIMAL | true | NULL |

+*--------+---------+-------+---------+*

> **INSERT** **INTO** accounts (id, balance) **SELECT** number, amount **FROM** other\_accounts **WHERE** id > 4;

> **SELECT** \* **FROM** accounts;

+*----+----------+*

| id | balance |

+*----+----------+*

| 1 | 10000.5 |

| 2 | 20000.75 |

| 3 | 8100.73 |

| 4 | 9400.1 |

| 5 | 350.1 |

| 6 | 150 |

| 7 | 200.1 |

+*----+----------+*

插入一个默认值

> **INSERT** **INTO** accounts (id) **VALUES** (8);

> **INSERT** **INTO** accounts (id, balance) **VALUES** (9, **DEFAULT**);

> **SELECT** \* **FROM** accounts **WHERE** id **in** (8, 9);

+----+---------+

| id | balance |

+----+---------+

| 8 | NULL |

| 9 | NULL |

+----+---------+

> **INSERT** **INTO** accounts **DEFAULT** **VALUES**;

> **SELECT** \* **FROM** accounts;

+--------------------+----------+

| id | balance |

+--------------------+----------+

| 1 | 10000.5 |

| 2 | 20000.75 |

| 3 | 8100.73 |

| 4 | 9400.1 |

| 5 | 350.1 |

| 6 | 150 |

| 7 | 200.1 |

| 8 | NULL |

| 9 | NULL |

| 142933248649822209 | NULL |

+--------------------+----------+

插入并返回值

在这个例子中，RETURNING子句返回插入行的id值，id值由unique\_rowid（）函数在服务器端生成。 不同语言版本需要你安装相应的[客户端驱动程序](http://doc.cockroachchina.baidu.com/#develop/install-client-drivers/)。

这种 RETURNING 的用法对应MySQL的 last\_insert\_id（）函数。  
某些语言驱动（例如Go）执行 query()方法用于会返回语句执行结果 ，执行exec()方法不会返回语句执行结果。在执行带RETURNING子句的 INSERT 语句时，需要使用query()方法。

> **INSERT** **INTO** accounts (id, balance)

**VALUES** (**DEFAULT**, 1000), (**DEFAULT**, 250)

RETURNING id;

+--------------------+

| id |

+--------------------+

| 190018410823680001 |

| 190018410823712769 |

+--------------------+

(2 rows)

*# Import the driver.*

**import** psycopg2

*# Connect to the "bank" database.*

conn = psycopg2.connect(

database='bank',

user='root',

host='localhost',

port=26257

)

*# Make each statement commit immediately.*

conn.set\_session(autocommit=**True**)

*# Open a cursor to perform database operations.*

cur = conn.cursor()

*# Insert two rows into the "accounts" table*

*# and return the "id" values generated server-side.*

cur.execute(

'INSERT INTO accounts (id, balance) '

'VALUES (DEFAULT, 1000), (DEFAULT, 250) '

'RETURNING id'

)

*# Print out the returned values.*

rows = cur.fetchall()

print('IDs:')

**for** row **in** rows:

print([str(cell) **for** cell **in** row])

*# Close the database connection.*

cur.close()

conn.close()

打印的值如下所示：

IDs:

['190019066706952193']

['190019066706984961']

*# Import the driver.*

**require** 'pg'

*# Connect to the "bank" database.*

conn = PG.connect(

user: 'root',

dbname: 'bank',

host: 'localhost',

port: 26257

)

*# Insert two rows into the "accounts" table*

*# and return the "id" values generated server-side.*

conn.exec(

'INSERT INTO accounts (id, balance) '\

'VALUES (DEFAULT, 1000), (DEFAULT, 250) '\

'RETURNING id'

) **do** |res|

*# Print out the returned values.*

puts "IDs:"

res.each **do** |row|

puts row

**end**

**end**

*# Close communication with the database.*

conn.close()

打印的值如下所示：

**IDs**:

{"id"=>"190019066706952193"}

{"id"=>"190019066706984961"}

**package** main

**import** (

"database/sql"

"fmt"

"log"

\_ "github.com/lib/pq"

)

**func** main() {

*//Connect to the "bank" database.*

db, err := sql.Open(

"postgres",

"postgresql://root@localhost:26257/bank?sslmode=disable"

)

**if** err != nil {

log.Fatal("error connecting to the database: ", err)

}

*// Insert two rows into the "accounts" table*

*// and return the "id" values generated server-side.*

rows, err := db.Query(

"INSERT INTO accounts (id, balance) " +

"VALUES (DEFAULT, 1000), (DEFAULT, 250) " +

"RETURNING id",

)

**if** err != nil {

log.Fatal(err)

}

*// Print out the returned values.*

**defer** rows.Close()

fmt.Println("IDs:")

**for** rows.Next() {

**var** id int

**if** err := rows.Scan(&id); err != nil {

log.Fatal(err)

}

fmt.Printf("%d\n", id)

}

}

打印的值如下所示：

IDs:

190019066706952193

190019066706984961

**var** async = require('async');

*// Require the driver.*

**var** pg = require('pg');

*// Connect to the "bank" database.*

**var** config = {

user: 'root',

host: 'localhost',

database: 'bank',

port: 26257

};

pg.connect(config, **function** (err, client, done) {

*// Closes communication with the database and exits.*

**var** finish = **function** () {

done();

process.exit();

};

**if** (err) {

console.error('could not connect to cockroachdb', err);

finish();

}

async.waterfall([

**function** (next) {

*// Insert two rows into the "accounts" table*

*// and return the "id" values generated server-side.*

client.query(

`INSERT INTO accounts (id, balance)

VALUES (DEFAULT, 1000), (DEFAULT, 250)

RETURNING id;`,

next

);

}

],

**function** (err, results) {

**if** (err) {

console.error('error inserting into and selecting from accounts', err);

finish();

}

*// Print out the returned values.*

console.log('IDs:');

results.rows.forEach(**function** (row) {

console.log(row);

});

finish();

});

});

打印的值如下所示：

IDs:

{ id: '190019066706952193' }

{ id: '190019066706984961' }

ON CONFLICT时更新值

检测到唯一性冲突时，CockroachDB将该行存储在名为excluded的临时表中。以下示例演示在发生冲突时，如何使用临时表excluded中的列来应用更新：

> **INSERT** **INTO** accounts (id, balance)

**VALUES** (8, 500.50)

**ON** CONFLICT (id)

**DO** **UPDATE** **SET** balance = excluded.balance;

> **SELECT** \* **FROM** accounts **WHERE** id = 8;

+----+---------+

| id | balance |

+----+---------+

| 8 | 500.50 |

+----+---------+

你还可以使用现有值更新行：

> **INSERT** **INTO** accounts (id, balance)

**VALUES** (8, 500.50)

**ON** CONFLICT (id)

**DO** **UPDATE** **SET** balance = accounts.balance + excluded.balance;

> **SELECT** \* **FROM** accounts **WHERE** id = 8;

+----+---------+

| id | balance |

+----+---------+

| 8 | 1001.00 |

+----+---------+

你还可以使用WHERE子句来有条件地应用DO UPDATE SET表达式：

> **INSERT** **INTO** accounts (id, balance)

**VALUES** (8, 700)

**ON** CONFLICT (id)

**DO** **UPDATE** **SET** balance = excluded.balance

**WHERE** excluded.balance > accounts.balance;

> **SELECT** \* **FROM** accounts **WHERE** id = 8;

+----+---------+

| id | balance |

+----+---------+

| 8 | 800 |

+----+---------+

(1 row)

当ON CONFLICT时不要更新值

在这个示例中，我们遇到了一个唯一性冲突的错误：

> **SELECT** \* **FROM** accounts **WHERE** id = 8;

+----+---------+

| id | balance |

+----+---------+

| 8 | 500.5 |

+----+---------+

> **INSERT** **INTO** accounts (id, balance) **VALUES** (8, 125.50);

pq: duplicate key value (id)=(8) violates unique constraint "primary"

在这个例子中，我们使用ON CONFLICT DO NOTHING来忽略唯一性错误，避免冲突行被更新：

> **INSERT** **INTO** accounts (id, balance)

**VALUES** (8, 125.50)

**ON** CONFLICT (id)

**DO** NOTHING;

> **SELECT** \* **FROM** accounts **WHERE** id = 8;

+----+---------+

| id | balance |

+----+---------+

| 8 | 500.5 |

+----+---------+

在这个例子中，ON CONFLICT DO NOTHING阻止第一行更新，同时允许插入第二行：

> **INSERT** **INTO** accounts (id, balance)

**VALUES** (8, 125.50), (10, 450)

**ON** CONFLICT (id)

**DO** NOTHING;

> **SELECT** \* **FROM** accounts **WHERE** id **in** (8, 10);

+----+---------+

| id | balance |

+----+---------+

| 8 | 500.5 |

| 10 | 450 |

+----+---------+

其他

* [Selection Queries](http://doc.cockroachchina.baidu.com/#develop/sql-syntax/selection-queries/)
* [DELETE](http://doc.cockroachchina.baidu.com/#develop/sql-statements/DELETE/)
* [UPDATE](http://doc.cockroachchina.baidu.com/#develop/sql-statements/UPDATE/)
* [UPSERT](http://doc.cockroachchina.baidu.com/#develop/sql-statements/UPSERT/)
* [TRUNCATE](http://doc.cockroachchina.baidu.com/#develop/sql-statements/TRUNCATE/)
* [ALTER TABLE](http://doc.cockroachchina.baidu.com/#develop/sql-statements/ALTER-TABLE/)
* [DROP TABLE](http://doc.cockroachchina.baidu.com/#develop/sql-statements/DROP-TABLE/)
* [DROP DATABASE](http://doc.cockroachchina.baidu.com/#develop/sql-statements/DROP-DATABASE/)
* [Other SQL Statements](http://doc.cockroachchina.baidu.com/#develop/sql-statements/overview/)