CockroachDB支持将多个SQL语句放到一个all-or-nothing事务中。 每个事务都保证跨表和行的ACID语义，即使在分布式数据库中也可以。 如果事务成功，则所有变动同时生效，如果事务的任何部分失败，则中止整个事务，并保持数据库不变。 CockroachDB保证在事务处于挂起状态时，它与具有serializable隔离的其他并发事务隔离。

有关CockroachDB事务语义的详细讨论，请参阅[How CockroachDB Does Distributed Atomic Transactions](https://www.cockroachlabs.com/blog/how-cockroachdb-distributes-atomic-transactions/) 和 [Serializable, Lockless, Distributed: Isolation in CockroachDB](https://www.cockroachlabs.com/blog/serializable-lockless-distributed-isolation-cockroachdb/)。 请注意，此博客文章中描述的事务模型的说明略有过时。 有关详细信息，请参阅“事务重试”部分。

SQL语句

以下每个SQL语句都以某种方式控制事务。

| **Statement** | **Function** |
| --- | --- |
| [BEGIN](http://doc.cockroachchina.baidu.com/#develop/sql-statements/BEGIN/) | 启动事务，并控制其优先级和隔离级别。 |
| [SET TRANSACTION](http://doc.cockroachchina.baidu.com/#develop/sql-statements/SET-TRANSACTION/) | 控制事务的优先级和隔离级别。 |
| [SAVEPOINT cockroach\_restart](http://doc.cockroachchina.baidu.com/#develop/sql-statements/SAVEPOINT/) | 将事务声明为可重试。 这使得你可以在由于更高优先级的事务同时或最近访问了相同数据而导致事务失败时重试该事务 |
| [RELEASE SAVEPOINT cockroach\_restart](http://doc.cockroachchina.baidu.com/#develop/sql-statements/RELEASE-SAVEPOINT/) | 提交可重试事务 |
| [COMMIT](http://doc.cockroachchina.baidu.com/#develop/sql-statements/COMMIT/) | 提交不可重试的事务或在提交可重试的事务后清除连接。 |
| [ROLLBACK TO SAVEPOINT cockroach\_restart](http://doc.cockroachchina.baidu.com/#develop/sql-statements/ROLLBACK/) | 通过回滚事务的变更并提高其优先级来处理可重试的错误。 |
| [ROLLBACK](http://doc.cockroachchina.baidu.com/#develop/sql-statements/ROLLBACK/) | 中止事务并将数据库回滚到事务开始之前的状态。 |
| [SHOW](http://doc.cockroachchina.baidu.com/#develop/sql-statements/SHOW(session-variables)/) | 显示当前的事务设置。 |

语法

在CockroachDB中，事务是通过使用BEGIN和COMMIT语句包围SQL语句来组织的。

要使用客户端事务重试，还应包括SAVEPOINT cockroach\_restart，ROLLBACK TO SAVEPOINT cockroach\_restart和RELEASE SAVEPOINT语句。

> **BEGIN**;

> **SAVEPOINT** cockroach\_restart;

<transaction statements>

> **RELEASE** **SAVEPOINT** cockroach\_restart;

> **COMMIT**;

在提交之前的任何时候，你都可以通过执行ROLLBACK语句来中止事务。

使用事务的客户端还必须包含处理重试的逻辑。

错误处理

要处理事务中的错误，应检查以下类型的服务器端错误：

| **Type** | **Description** |
| --- | --- |
| **Retryable Errors** | 代码为40001或字符串“retry transaction”的错误，表明事务失败，因为它与访问相同数据的另一个并发或最近的事务冲突，事务需要由客户重试。更多信息请查看 [client-side transaction retries](http://doc.cockroachchina.baidu.com/#develop/transactions/#client-side-transaction-retries) |
| **Ambiguous Errors** | 代码40003的错误是为了响应RELEASE SAVEPOINT（或当不使用SAVEPOINT的COMMIT）而返回的，它表明事务的状态是不明确的，即你不能认为它已经提交或者失败。 如何处理这些错误取决于你希望如何解决歧义。 更多这类错误请查看 [here](https://www.cockroachlabs.com/docs/stable/common-errors.html#result-is-ambiguous) |
| **SQL Errors** | 所有其他错误，表明事务中的语句失败了。 例如，违反Unique约束会产生23505错误。 遇到这些错误后，你可以发出COMMIT或ROLLBACK来中止事务并将数据库恢复到事务开始之前的状态。  如果你想再次尝试同一组语句 ，你必须开始一个全新的事务。 |

事务竞争

CockroachDB中的事务在执行期间锁定写入的数据资源。 当一个事务的挂起写入与并发事务的写入冲突时，并发事务必须等待先前的事务完成才能继续。 当在事务之间检测到依赖性循环时，具有较高优先级的事务中止依赖事务以避免死锁，被中止事务将被重试。

有关事务竞争和避免竞争的最佳实践的更多详细信息，请参阅 [Understanding and Avoiding Transaction Contention](http://doc.cockroachchina.baidu.com/#develop/performance-optimization/sql-best-practices/#understanding-and-avoiding-transaction-contention).

事务重试

如果事务遇到死锁或与其他并发事务的读/写竞争时可能需要重试，如果不允许potential serializable anomalies（潜在serializable异常?），则无法解析这些事务。 （但是可以通过使用AS OF SYSTEM TIME执行读取来缓解读写冲突。）

处理事务重试有两种情况：

* CockroachDB为你处理的自动重试。
* 客户端介入，你的应用程序必须能处理。

自动重试

CockroachDB会自动重试从客户端发送的单个语句和事务作为单个批处理。

单个语句

单个语句被视为隐式事务，例如：

> **DELETE** **FROM** customers **WHERE** id = 1;

批量语句

事务可以作为单个批处理（batch）从客户端发送。 批处理意味着CockroachDB接收多个语句而不被要求返回它们中间的结果; 而是在执行所有语句后返回结果（除非累积的结果溢出内部缓冲区，在这种情况下，它们会更快返回并且无法再执行自动重试）。

批处理通常由你的驱动程序或客户端的行为控制。 从技术上讲，它可以通过两种方式实现，都支持自动重试：

1. 当客户端/驱动程序使用PostgreSQL扩展查询协议时，批处理由两个Sync消息之间发送的所有查询组成。 许多驱动程序通过显式批处理构造支持此类批处理。 自CockroachDB v2.0起支持自动重试此类批处理。
2. 客户端/驱动程序使用PostgreSQL简单查询协议，批处理由分号分隔的字符串组成，作为一个单元发送到CockroachDB。 例如，在Go中，此代码将发送一个批处理（将自动重试）：

``` go db.Exec( "BEGIN;

**DELETE** **FROM** customers **WHERE** id = 1;

**DELETE** orders **WHERE** customer = 1;

**COMMIT**;"

) ```

在一批语句中，CockroachDB假设语句不以先前语句的结果为条件，因此它可以重试所有语句。 但是，如果事务依赖于条件逻辑（例如，仅对语句1的某些结果执行语句2），事务中的某些语句的结果已经传递给客户端（例如，语句1的结果已经传递 ），CockroachDB不能单独自动重试语句2。 相反，你应该编写事务以使用客户端干预，以便客户端重试语句1。

客户端干预

在单独发送语句时，你的应用程序应包括客户端重试处理，例如：

> **BEGIN**;

> **UPDATE** products **SET** inventory = 0 **WHERE** sku = '8675309';

> **INSERT** **INTO** orders (customer, **status**) **VALUES** (1, 'new');

> **COMMIT**;

为了表明必须重试事务，CockroachDB使用代码40001和以字符串retry transaction开头的错误消息。

要处理这些类型的错误，你有两种选择：

* 建议：使用SAVEPOINT cockroach\_restart函数创建可重试的事务。 可重试事务可以提高性能，因为每次重试时它们的优先级都会提高，使得它们在系统中的时间越长就成功概率越高。

有关更多信息，请参阅[客户端事务重试](http://doc.cockroachchina.baidu.com/#develop/transactions/#client-side-transaction-retries)。

* 使用ROLLBACK中止事务，然后重新发出事务中的所有语句。 这不会自动增加事务的优先级，因此在高竞争工作负载中，事务可能需要非常长的时间才能成功。

客户端事务重试

作为提高竞争事务的性能的一种方法，CockroachDB包含一组允许你重试这些事务的语句。 重试事务有利于每次重试时提高其优先级，从而增加其成功的可能性。

重新编写的事务也在稍晚一点的时间戳发出，因此事务现在在数据库的稍晚一点的快照上运行，因此读取可能返回更新的数据。

实现客户端重试需要三个语句：

* 如果存在竞争错误，SAVEPOINT cockroach\_restart声明客户端将要重试事务。 它必须在BEGIN之后且在第一个操作数据库的语句之前执行。
* 当你的应用程序检测到40001/retry transaction错误时，使用ROLLBACK TO SAVEPOINT cockroach\_restart。 它使你有机会通过将数据库的状态回滚到事务的开头并增加事务的优先级来“重试”事务。

在发出ROLLBACK TO SAVEPOINT cockroach\_restart后，你必须发出你希望事务包含的所有语句。 通常，这意味着重新计算值并将一组类似的语句重新发送。

* RELEASE SAVEPOINT cockroach\_restart提交交易。 此时，CockroachDB会检查事务是否与其他事务为了访问相同的值发生竞争; 优先级最高的事务成功，其他事务返回40001 / retry transaction 错误。

你还必须在之后执行COMMIT以清除下一个事务的连接。

你可以在本页的“语法”部分或“使用CockroachDB构建应用程序”教程中找到此示例。

如果你使用以下语言构建应用程序，我们有一些软件包可以简化客户端重试：

* Go开发人员可以使用CockroachDB Go客户端的crdb包。详见[Build a Go App with CockroachDB](https://www.cockroachlabs.com/docs/stable/build-a-go-app-with-cockroachdb.html#transaction-with-retry-logic)
* Python开发人员可以使用sqlalchemy包。详见[Building an Application With CockroachDB and SQLAlchemy](https://www.cockroachlabs.com/blog/building-application-cockroachdb-sqlalchemy-2/)

同样重要的是，要注意重试的事务在稍晚一点的时间戳重新启动。 这意味着事务在稍晚一点的数据库快照上运行，相关的读取可能会检索更新的数据。

有关更多详细信息，请参阅可重试事务的过程。

1.事务以BEGIN语句开头。

1. SAVEPOINT cockroach\_restart语句声明将在竞争错误的情况下重试该事务。 请注意，CockroachDB的savepoint实现不支持所有savepoint功能，例如嵌套事务。
2. 执行事务中的语句。
3. 如果一个语句返回一个可重试的错误（通过错误消息开头的40001错误代码或retry transaction），你可以发出ROLLBACK TO SAVEPOINT cockroach\_restart 语句来重启事务。 或者，可以重新发出的SAVEPOINT cockroach\_restart语句以重新启动事务。

你现在必须重新发出事务中的语句。

如果你不希望应用程序重试事务，你可以在此时发出ROLLBACK。 任何其他语句都将被服务器拒绝，通常是在遇到错误并且事务尚未关闭之后的情况。

1. 一旦事务执行所有语句而没有遇到竞争错误，执行RELEASE SAVEPOINT cockroach\_restart来提交更改。 如果成功，则事务所做的所有变更对后续事务都是可见的，并且在crash时保证变更是持久的。

在某些情况下，RELEASE SAVEPOINT语句本身可能会因可重试错误而失败，主要是因为CockroachDB中的事务只意识到在它们尝试提交时需要重新启动它们。 如果发生这种情况，将按步骤4中所述处理可重试错误。

事务参数

每个事务由两个参数控制：其优先级和隔离级别。 以下两节将进一步详述。

事务优先级

CockroachDB中的每个事务都被分配了初始优先级。 默认情况下，该优先级为NORMAL，但对于在高竞争场景中应优先考虑的事务，客户端可以在BEGIN语句中设置优先级：

> **BEGIN** PRIORITY <LOW | NORMAL | HIGH>;

或者，客户端可以在事务启动后立即设置优先级，如下所示：

> **SET** **TRANSACTION** PRIORITY <LOW | NORMAL | HIGH>;

客户端还可以使用SHOW TRANSACTION PRIORITY显示事务的当前优先级。

当两个事务间接竞争相同的资源时，它们可能会创建一个导致死锁情况的依赖循环，两个事务都在等待另一个事务完成。 在这些情况下，CockroachDB允许具有更高优先级的事务中止另一个事务，被中止的事务需要重试。 在重试时，事务继承更高的优先级。 这意味着每次重试都会使事务在再次遇到死锁的情况下成功概率更高。

隔离级别

CockroachDB高效支持最严格的ANSI事务隔离级别：SERIALIZABLE。 所有其他ANSI事务隔离级别（例如，READ UNCOMMITTED，READ COMMITTED和REPEATABLE READ）将自动升级为SERIALIZABLE。 历史使用中较弱的隔离级别常用于最大化事务吞吐量。 但是，最近的研究表明，使用弱隔离级别会导致基于并发攻击的严重漏洞。 CockroachDB继续支持额外的非ANSI隔离级别SNAPSHOT，尽管它已被弃用。 客户端可以在启动事务时显式设置事务的隔离：

> **BEGIN** **ISOLATION** **LEVEL** <**SERIALIZABLE** | **SNAPSHOT**>;

或者，客户端可以在事务启动后立即设置隔离级别：

> **SET** **TRANSACTION** **ISOLATION** **LEVEL** <**SERIALIZABLE** | **SNAPSHOT**>;

客户端还可以使用SHOW TRANSACTION ISOLATION LEVEL显示事务的当前隔离级别

有关CockroachDB事务中隔离的详细讨论，请参阅[Serializable, Lockless, Distributed: Isolation in CockroachDB.](https://www.cockroachlabs.com/blog/serializable-lockless-distributed-isolation-cockroachdb/)

Serializable隔离

使用SERIALIZABLE隔离，事务的行为就好像它在事务执行期间占有整个数据库。 这意味着没有并发writers可以影响事务，除非它们在启动之前提交，并且在成功提交之前，并不会有任何并发读取器受事务影响。 这是CockroachDB提供的最强隔离级别，它是默认的隔离级别。

与SNAPSHOT不同，SERIALIZABLE隔离不允许任何异常。 为了防止写入偏斜异常，SERIALIZABLE隔离可能需要重新启动事务。

Snapshot隔离

使用SNAPSHOT隔离（**弃用**），事务就像在固定的时间点一致地读取数据库的状态一样。 与SERIALIZABLE级别不同，SNAPSHOT隔离允许写入偏斜异常。 仍然支持此隔离级别以实现向后兼容性，但你应该避免使用它。 它在性能方面并没有优势，并且可能导致某些复杂工作负载下的状态不一致。 基于并发的攻击可以将不一致性强制转化为对对系统状态造成负面影响。 出于同样的原因，CockroachDB将对较弱的ANSI隔离级别READ UNCOMMITTED，READ COMMITTED和REPEATABLE READ的所有请求升级为SERIALIZABLE。

与ANSI SQL隔离级别的比较

CockroachDB使用的隔离级别与ANSI SQL隔离级别略有不同。

别名

* READ UNCOMMITTED, READ COMMITTED, 和 REPEATABLE READ 是 SERIALIZABLE的别名。

对比

* CockroachDB的SERIALIZABLE比 ANSI SQL READ UNCOMMITTED, READ COMMITTED, 和 REPEATABLE READ 更严格，与ANSI SQL SERIALIZABLE 相同。
* CockroachDBSNAPSHOT级别（**已弃用**）强于ANSI SQLREAD UNCOMMITTED和READ COMMITTED级别。

有关这些级别之间关系的更多信息，see [this paper](http://arxiv.org/ftp/cs/papers/0701/0701157.pdf).

See Also

* [BEGIN](http://doc.cockroachchina.baidu.com/#develop/sql-statements/BEGIN/)
* [COMMIT](http://doc.cockroachchina.baidu.com/#develop/sql-statements/COMMIT/)
* [ROLLBACK](http://doc.cockroachchina.baidu.com/#develop/sql-statements/ROLLBACK/)
* [SAVEPOINT](http://doc.cockroachchina.baidu.com/#develop/sql-statements/SAVEPOINT/)
* [RELEASE SAVEPOINT](http://doc.cockroachchina.baidu.com/#develop/sql-statements/RELEASE-SAVEPOINT/)
* [SHOW](http://doc.cockroachchina.baidu.com/#develop/sql-statements/SHOW(session-variables)/)
* [Retryable function code samples](http://doc.cockroachchina.baidu.com/#quick-start/build-an-app/overview/)