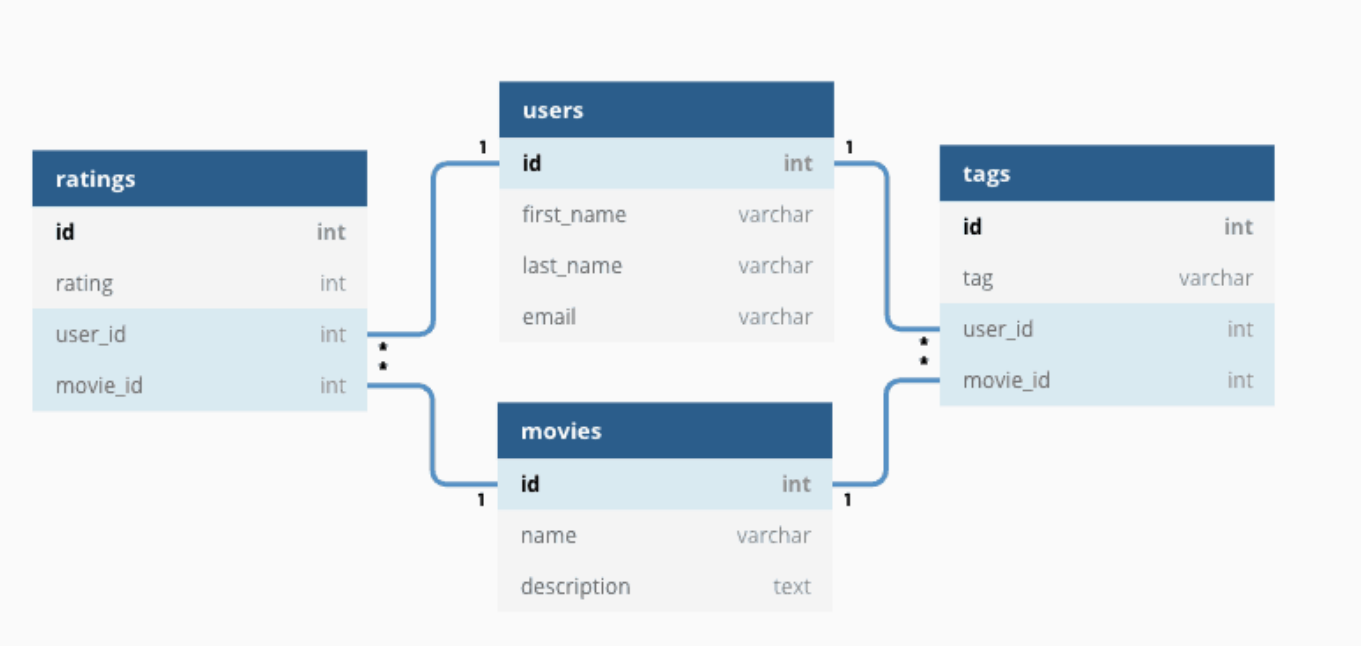


MySQL 基础

关系型数据库介绍

顾名思义，关系型数据库就是一种建立在关系模型的基础上的数据库。关系模型表明了数据库中所存储的数据之间的联系（一对一、一对多、多对多）。

关系型数据库中，我们的数据都被存放在了各种表中（比如用户表），表中的每一列就存放着一条数据（比如一个用户的信息）。



大部分关系型数据库都使用 SQL 来操作数据库中的数据。并且，大部分关系型数据库都支持事务的四大特性 (ACID)。

有哪些常见的关系型数据库呢？

MySQL、PostgreSQL、Oracle、SQL Server、SQLite（微信本地的聊天记录的存储就是用的 SQLite）

MySQL 介绍



MySQL 是一种关系型数据库，主要用于持久化存储我们的系统中的一些数据比如用户信息。

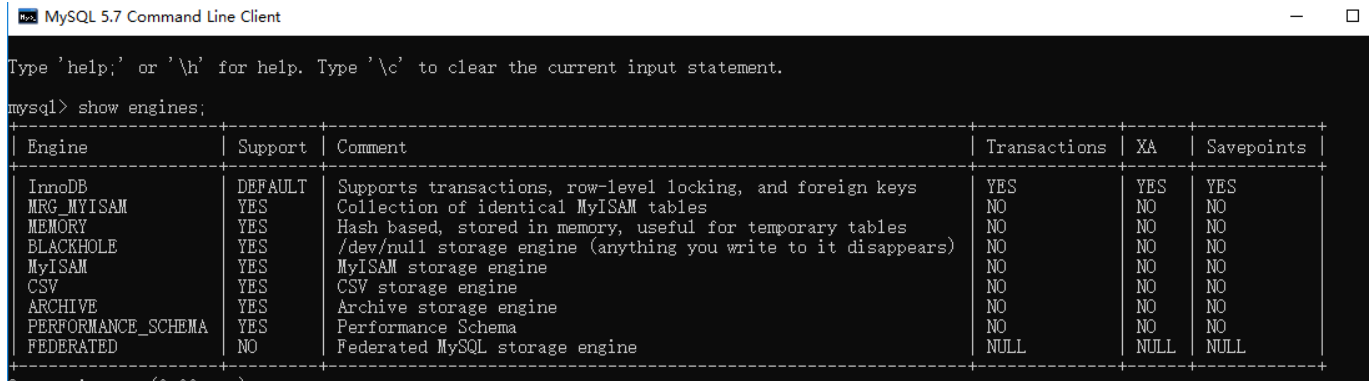
由于 MySQL 是开源免费并且比较成熟的数据库，因此，MySQL 被大量使用在各种系统中。任何人都可以在 GPL(General Public License) 的许可下下载并根据个性化的需要对其进行修改。MySQL 的默认端口号是**3306**。

存储引擎

存储引擎相关的命令

查看 MySQL 提供的所有存储引擎

```
mysql> show engines;
```



Engine	Support	Comment	Transactions	XA	Savepoints
InnoDB	DEFAULT	Supports transactions, row-level locking, and foreign keys	YES	YES	YES
MRG_MYISAM	YES	Collection of identical MyISAM tables	NO	NO	NO
MEMORY	YES	Hash based, stored in memory, useful for temporary tables	NO	NO	NO
BLACKHOLE	YES	/dev/null storage engine (anything you write to it disappears)	NO	NO	NO
MyISAM	YES	MyISAM storage engine	NO	NO	NO
CSV	YES	CSV storage engine	NO	NO	NO
ARCHIVE	YES	Archive storage engine	NO	NO	NO
PERFORMANCE_SCHEMA	YES	Performance Schema	NO	NO	NO
FEDERATED	NO	Federated MySQL storage engine	NULL	NULL	NULL

从上图我们可以查看出 MySQL 当前默认的存储引擎是 InnoDB,并且在 5.7 版本所有的存储引擎中只有 InnoDB 是事务性存储引擎,也就是说只有 InnoDB 支持事务。

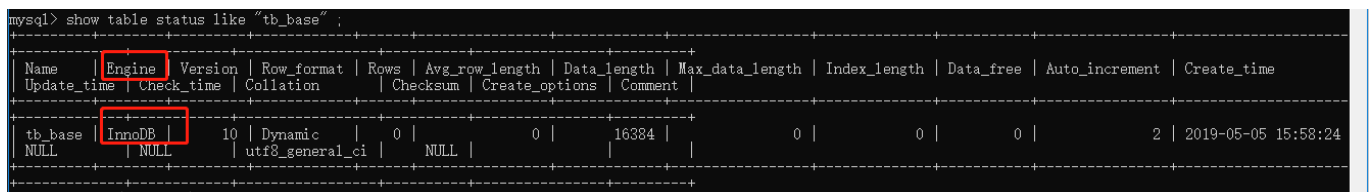
查看 MySQL 当前默认的存储引擎

我们也可以通过下面的命令查看默认的存储引擎。

```
mysql> show variables like '%storage_engine%';
```

查看表的存储引擎

```
show table status like "table_name" ;
```



Name	Engine	Version	Row_format	Rows	Avg_row_length	Data_length	Max_data_length	Index_length	Data_free	Auto_increment	Create_time
tb_base	InnoDB	10	Dynamic	0	0	16384	0	0	0	2	2019-05-05 15:58:24

MyISAM 和 InnoDB 的区别



MySQL 5.5 之前，MyISAM 引擎是 MySQL 的默认存储引擎，可谓是风光一时。

虽然，MyISAM 的性能还行，各种特性也还不错（比如全文索引、压缩、空间函数等）。但是，MyISAM 不支持事务和行级锁，而且最大的缺陷就是崩溃后无法安全恢复。

5.5 版本之后，MySQL 引入了 InnoDB（事务性数据库引擎），MySQL 5.5 版本后默认的存储引擎为 InnoDB。小伙子，一定要记好这个 InnoDB，你每次使用 MySQL 数据库都是用的这个存储引擎吧？

言归正传！咱们下面还是来简单对比一下两者：

1.是否支持行级锁

MyISAM 只有表级锁(table-level locking)，而 InnoDB 支持行级锁(row-level locking)和表级锁,默认为行级锁。

也就说，MyISAM 一锁就是锁住了整张表，这在并发写的情况下是多么滴憋憋啊！这也是为什么 InnoDB 在并发写的时候，性能更牛皮了！

2.是否支持事务

MyISAM 不提供事务支持。

InnoDB 提供事务支持，具有提交(commit)和回滚(rollback)事务的能力。

3.是否支持外键

MyISAM 不支持，而 InnoDB 支持。

📖 拓展一下：

一般我们也是不建议在数据库层面使用外键的，应用层面可以解决。不过，这样会对数据的一致性造成威胁。具体要不要使用外键还是要根据你的项目来决定。

4.是否支持数据库异常崩溃后的安全恢复

MyISAM 不支持，而 InnoDB 支持。

使用 InnoDB 的数据库在异常崩溃后，数据库重新启动的时候会保证数据库恢复到崩溃前的状态。这个恢复的过程依赖于 redo log。

📖 拓展一下：

- MySQL InnoDB 引擎使用 **redo log(重做日志)** 保证事务的**持久性**，使用 **undo log(回滚日志)** 来保证事务的**原子性**。
- MySQL InnoDB 引擎通过 **锁机制**、**MVCC** 等手段来保证事务的隔离性（默认支持的隔离级别是 **REPEATABLE-READ**）。
- 保证了事务的持久性、原子性、隔离性之后，一致性才能得到保障。

5.是否支持 MVCC

MyISAM 不支持，而 InnoDB 支持。

讲真，这个对比有点废话，毕竟 MyISAM 连行级锁都不支持。

MVCC 可以看作是行级锁的一个升级，可以有效减少加锁操作，提供性能。

关于 MyISAM 和 InnoDB 的选择问题

大多数时候我们使用的都是 InnoDB 存储引擎，在某些读密集的情况下，使用 MyISAM 也是合适的。不过，前提是你的项目不介意 MyISAM 不支持事务、崩溃恢复等缺点（可是~我们一般都会介意啊！）。

《MySQL 高性能》上面有一句话这样写到：

不要轻易相信“MyISAM 比 InnoDB 快”之类的经验之谈，这个结论往往不是绝对的。在很多我们已知场景中，InnoDB 的速度都可以让 MyISAM 望尘莫及，尤其是用到了聚簇索引，或者需要访问的数据都可以放入内存的应用。

一般情况下我们选择 InnoDB 都是没有问题的，但是某些情况下你并不在乎可扩展能力和并发能力，也不需要事务支持，也不在乎崩溃后的安全恢复问题的话，选择 MyISAM 也是一个不错的选择。但是一般情况下，我们都是需要考虑到这些问题的。

因此，对于咱们日常开发的业务系统来说，你几乎找不到什么理由再使用 MyISAM 作为自己的 MySQL 数据库的存储引擎。

锁机制与 InnoDB 锁算法

MyISAM 和 InnoDB 存储引擎使用的锁：

- MyISAM 采用表级锁(table-level locking)。
- InnoDB 支持行级锁(row-level locking)和表级锁,默认为行级锁

表级锁和行级锁对比：

- **表级锁**：MySQL 中锁定 **粒度最大** 的一种锁，对当前操作的整张表加锁，实现简单，资源消耗也比较少，加锁快，不会出现死锁。其锁定粒度最大，触发锁冲突的概率最高，并发度最低，MyISAM 和 InnoDB 引擎都支持表级锁。
- **行级锁**：MySQL 中锁定 **粒度最小** 的一种锁，只针对当前操作的行进行加锁。行级锁能大大减少数据库操作的冲突。其加锁粒度最小，并发度高，但加锁的开销也最大，加锁慢，会出现死锁。

InnoDB 存储引擎的锁的算法有三种：

- Record lock：记录锁，单个行记录上的锁
- Gap lock：间隙锁，锁定一个范围，不包括记录本身
- Next-key lock：record+gap+临键锁，锁定一个范围，包含记录本身

查询缓存

执行查询语句的时候，会先查询缓存。不过，MySQL 8.0 版本后移除，因为这个功能不太实用

`my.cnf` 加入以下配置，重启 MySQL 开启查询缓存

```
query_cache_type=1
query_cache_size=600000
```

MySQL 执行以下命令也可以开启查询缓存

```
set global query_cache_type=1;
set global query_cache_size=600000;
```

如上，**开启查询缓存后在同样的查询条件以及数据情况下，会直接在缓存中返回结果**。这里的查询条件包括查询本身、当前要查询的数据库、客户端协议版本号等一些可能影响结果的信息。因此任何两个查询在任何字符上的不同都会导致缓存不命中。此外，如果查询中包含任何用户自定义函数、存储函数、用户变量、临时表、MySQL 库中的系统表，其查询结果也不会被缓存。

缓存建立之后，MySQL 的查询缓存系统会跟踪查询中涉及的每张表，如果这些表（数据或结构）发生变化，那么和这张表相关的所有缓存数据都将失效。

缓存虽然能够提升数据库的查询性能，但是缓存同时也带来了额外的开销，每次查询后都要做一次缓存操作，失效后还要销毁。因此，开启查询缓存要谨慎，尤其对于写密集的应用来说更是如此。如果开启，要注意合理控制缓存空间大小，一般来说其大小设置为几十 MB 比较合适。此外，**还可以通过 `sql_cache` 和 `sql_no_cache` 来控制某个查询语句是否需要缓存**：

```
select sql_no_cache count(*) from usr;
```

事务

何为事务？

一言蔽之，**事务是逻辑上的一组操作，要么都执行，要么都不执行**。

可以简单举一个例子不？

事务最经典也经常被拿出来说例子就是转账了。假如小明要给小红转账 1000 元，这个转账会涉及到两个关键操作就是：

1. 将小明的余额减少 1000 元
2. 将小红的余额增加 1000 元。

事务会把这两个操作就可以看成逻辑上的一个整体，这个整体包含的操作要么都成功，要么都要失败。

这样就不会出现小明余额减少而小红的余额却并没有增加的情况。

何为数据库事务？

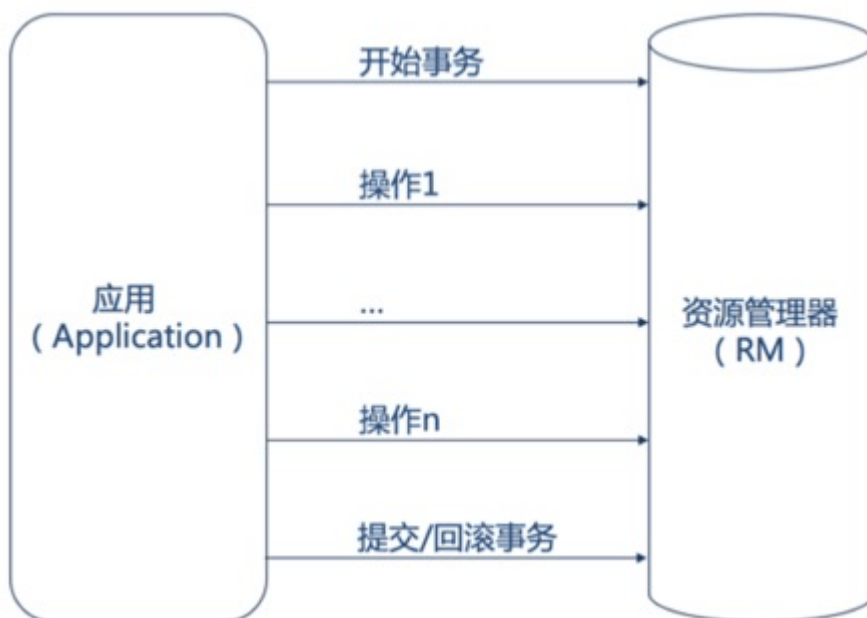
数据库事务在我们日常开发中接触的最多了。如果你的项目属于单体架构的话，你接触到的往往就是数据库事务了。

平时，我们在谈论事务的时候，如果没有特指**分布式事务**，往往指的就是**数据库事务**。

那数据库事务有什么作用呢？

简单来说：数据库事务可以保证多个对数据库的操作（也就是 SQL 语句）构成一个逻辑上的整体。构成这个逻辑上的整体的这些数据库操作遵循：**要么全部执行成功,要么全部不执行**。

```
# 开启一个事务
START TRANSACTION;
# 多条 SQL 语句
SQL1, SQL2...
## 提交事务
COMMIT;
```



另外，关系型数据库（例如：[MySQL](#)、[SQL Server](#)、[Oracle](#) 等）事务都有 **ACID** 特性：



何为 ACID 特性呢？

1. **原子性 (Atomicity)**：事务是最小的执行单位，不允许分割。事务的原子性确保动作要么全部完成，要么完全不起作用；
2. **一致性 (Consistency)**：执行事务前后，数据保持一致，例如转账业务中，无论事务是否成功，转账者和收款人的总额应该是不变的；
3. **隔离性 (Isolation)**：并发访问数据库时，一个用户的事务不被其他事务所干扰，各并发事务之间数据库是独立的；
4. **持久性 (Durability)**：一个事务被提交之后。它对数据库中数据的改变是持久的，即使数据库发生故障也不应该对其有任何影响。

数据事务的实现原理呢？

我们这里以 MySQL 的 InnoDB 引擎为例来简单说一下。

MySQL InnoDB 引擎使用 **redo log(重做日志)** 保证事务的**持久性**，使用 **undo log(回滚日志)** 来保证事务的**原子性**。

MySQL InnoDB 引擎通过 **锁机制**、**MVCC** 等手段来保证事务的隔离性（默认支持的隔离级别是 **REPEATABLE-READ**）。

保证了事务的持久性、原子性、隔离性之后，一致性才能得到保障。

并发事务带来哪些问题？

在典型的应用程序中，多个事务并发运行，经常会操作相同的数据来完成各自的任务（多个用户对同一数据进行操作）。并发虽然是必须的，但可能会导致以下的问题。

- **脏读 (Dirty read)**：当一个事务正在访问数据并且对数据进行了修改，而这种修改还没有提交到数据库中，这时另外一个事务也访问了这个数据，然后使用了这个数据。因为这个数据是还没有提交的数据，那么另外一个事务读到的这个数据是“脏数据”，依据“脏数据”所做的操作可能是不正确的。
- **丢失修改 (Lost to modify)**：指在一个事务读取一个数据时，另外一个事务也访问了该数据，那么在第一个事务中修改了这个数据后，第二个事务也修改了这个数据。这样第一个事务内的修改结果就被丢失，因此称为丢失修改。例如：事务 1 读取某表中的数据 $A=20$ ，事务 2 也读取 $A=20$ ，事务 1 修改 $A=A-1$ ，事务 2 也修改 $A=A-1$ ，最终结果 $A=19$ ，事务 1 的修改被丢失。
- **不可重复读 (Unrepeatable read)**：指在一个事务内多次读同一数据。在这个事务还没有结束时，另一个事务也访问该数据。那么，在第一个事务中的两次读数据之间，由于第二个事务的修改导致第一个事务两次读取的数据可能不太一样。这就发生了在一个事务内两次读到的数据是不一样的情况，因此称为不可重复读。

- **幻读 (Phantom read)** : 幻读与不可重复读类似。它发生在一个事务 (T1) 读取了几行数据, 接着另一个并发事务 (T2) 插入了一些数据时。在随后的查询中, 第一个事务 (T1) 就会发现多了一些原本不存在的记录, 就好像发生了幻觉一样, 所以称为幻读。

不可重复读和幻读区别:

不可重复读的重点是修改比如多次读取一条记录发现其中某些列的值被修改, 幻读的重点在于新增或者删除比如多次读取一条记录发现记录增多或减少了。

事务隔离级别有哪些?

SQL 标准定义了四个隔离级别:

- **READ-UNCOMMITTED(读取未提交)**: 最低的隔离级别, 允许读取尚未提交的数据变更, **可能会导致脏读、幻读或不可重复读**。
- **READ-COMMITTED(读取已提交)**: 允许读取并发事务已经提交的数据, **可以阻止脏读, 但是幻读或不可重复读仍有可能发生**。
- **REPEATABLE-READ(可重复读)**: 对同一字段的多次读取结果都是一致的, 除非数据是被本身事务自己所修改, **可以阻止脏读和不可重复读, 但幻读仍有可能发生**。
- **SERIALIZABLE(可串行化)**: 最高的隔离级别, 完全服从 ACID 的隔离级别。所有的事务依次逐个执行, 这样事务之间就完全不可能产生干扰, 也就是说, **该级别可以防止脏读、不可重复读以及幻读**。

隔离级别	脏读	不可重复读	幻读
READ-UNCOMMITTED	√	√	√
READ-COMMITTED	×	√	√
REPEATABLE-READ	×	×	√
SERIALIZABLE	×	×	×

MySQL 的默认隔离级别是什么?

MySQL InnoDB 存储引擎的默认支持的隔离级别是 **REPEATABLE-READ (可重读)**。我们可以通过 `SELECT @@tx_isolation;` 命令来查看, MySQL 8.0 该命令改为 `SELECT @@transaction_isolation;`

```
mysql> SELECT @@tx_isolation;
+-----+
| @@tx_isolation |
+-----+
| REPEATABLE-READ |
+-----+
```

这里需要注意的是: 与 SQL 标准不同的地方在于 InnoDB 存储引擎在 **REPEATABLE-READ (可重读)** 事务隔离级别下使用的是 Next-Key Lock 锁算法, 因此可以避免幻读的产生, 这与其他数据库系统(如 SQL Server)是不同的。所以说 InnoDB 存储引擎的默认支持的隔离级别是 **REPEATABLE-READ (可重读)** 已经可以完全保证事务的隔离性要求, 即达到了 SQL 标准的 **SERIALIZABLE(可串行化)** 隔离级别。

🐞 问题更正: **MySQL InnoDB 的 REPEATABLE-READ (可重读) 并不保证避免幻读, 需要应用使用加锁读来保证。而这个加锁度使用到的机制就是 Next-Key Locks。**

因为隔离级别越低, 事务请求的锁越少, 所以大部分数据库系统的隔离级别都是 **READ-COMMITTED(读取提交内容)**, 但是你要知道的是 InnoDB 存储引擎默认使用 **REPEATABLE-READ (可重读)** 并不会有任何性能损失。

InnoDB 存储引擎在 **分布式事务** 的情况下一般会用到 **SERIALIZABLE(可串行化)** 隔离级别。

📖 拓展一下(以下内容摘自《MySQL 技术内幕: InnoDB 存储引擎(第 2 版)》7.7 章):

InnoDB 存储引擎提供了对 XA 事务的支持, 并通过 XA 事务来支持分布式事务的实现。分布式事务指的是允许多个独立的事务资源 (transactional resources) 参与到一个全局的事务中。事务资源通常是关系型数据库系统, 但也可以是其他类型的资源。全局事务要求在其中的所有参与的事务要么都提交, 要么都回滚, 这对于事务原有的 ACID 要求又有了提高。另外, 在使用分布式事务时, InnoDB 存储引擎的事务隔离级别必须设置为 SERIALIZABLE。

参考

- 《高性能 MySQL》
- <https://www.omnisci.com/technical-glossary/relational-database>