索引

- 用来快速地寻找那些具有特定值的记录。如果没有索引，执行查询时MySQL就会从第一个记录开始扫描整个表的所有记录，直至找到符合要求的记录。表里面的记录数量越多，查询成本越大。如果作为搜索条件的列上创建了索引，MySQL无需扫描任何记录即可迅速得到目标记录所在的位置。

- 创建索引

```
CREATE index sc_c_id_index on SC(c_id);
CREATE index sc_score_index on SC(score);

//CREATE INDEX index_name ON table_name (column_list)
```

- 查看索引

```
//show index from tblname;

show index from sc;
```

- 删除索引

```
//drop index index_name on table_name;
```

- 查询计划（SQL语句前加上 **EXPLAIN**）

```
EXPLAIN SELECT name,college FROM student  WHERE student_id
in (select s_id FROM sc WHERE c_id = 24 AND score = 2)
```

- Type：表使用的访问方式：（**性能由差到好**）
  - ALL: 全表扫描
  - index: 索引全扫描
  - range:索引范围扫描
  - ref:使用非唯一索引扫描
  - eq_ref:使用唯一索引扫描
  - const，system:单表中最多只有一个匹配行
- **索引的应用**
  - **匹配全值: 索引中所有列都指定具体值，即是对索引中的所有列都有等值匹配的条件。**

```
EXPLAIN SELECT * FROM student WHERE name = '匡俊霖' AND student_id =
'2017211903' LIMIT 1
```

- **匹配值的范围查询**
- **匹配列前缀**

```
create index index_student_name on student(`name` (10))
```

- **索引匹配精确而其他列范围匹配**

```
EXPLAIN SELECT name,student_id,college from student where name like
'匡%' AND college = '外国语学院'
```

- **Tips**
  - 只需要一条记录的时候，使用LIMIT
    - 这样能使的EXPLAIN中type达到const
  - 查看索引使用情况

```
SHOW STATUS LIKE 'Handler_read%';
```

    - **Handler_read_key**：如果索引正在工作，Handler_read_key的值将很高。

      Handler_read_rnd_next：数据文件中读取下一行的请求数，如果正在进行大量的表扫描，值将较高，则说明索引利用不理想。
  - 被查询的列是索引的话，数据能直接从索引中取得，而如果被查询的列不是索引，就会回表查询，所以要避免 **SELECT *** 。
  - **更新十分频繁的字段上不宜建立索引**

    更新会变更B+树，维护索引的消耗太大。

事务

- 事务就是一段sql 语句的批处理，但是这个批处理是一个atom（原子），不可分割，要么都执行，要么回滚（rollback）都不执行。
- 应用场景例子：人员管理系统中删除员工信息的同时，与之相关的信息也要删除。转账时，一个用户的账户减去多少钱，就要保证被转入的账户增加相应数额的钱。这些SQL 语句必须作为一个整体来执行
- 启用事务

```
//手动开启事务
start transaction;

//手动关闭事务
commit ;

//回滚到事务执行前
rollback;
```

- 事务的**ACID**（Atomic、Consistent、Isolated、Durable ）
  - A: **原子性**（不可分割），一个事务必须被视为一个不可分割的**最小工作单元**，要么执行成功进入下一个状态，要么失败rollback 到执行前的状态。
  - C：**一致性**，数据库总是从一个一致性的状态转换到另一个一致性的状态。
  - I：**隔离性**，一个事务无法知道另外一个事务的执行情况
  - D：**持久性**，在事务一旦提交，该事务对数据库所作的修改便会永久保存到数据库，无法被回滚。

隔离性

- 读未提交
  - **最低的隔离性级别**：**事务中的修改，即使没有提交，对其他事务也都是可见的。**

    读未提交面临**脏读**的问题：*事务可以读取未提交的数据，而该数据可能在未来因回滚而消失*。
- 读已提交(不可重复读)
  - **一个事务所做的修改在最终提交以前，对其他事务是不可见的。**

    两次执行同样的查询，如果第二次读到了其他事务提交的结果，则会得到不一样的结果。
- 可重复读
  - 在读已提交的基础上，**可重复读**解决了部分不可重复的问题：**同一个事务中多次读取同样记录结果是一致的。记录**指具体的数据行。
  - 但是存在**幻读**：*当某个事务在读取目标**范围**内的记录时，另一个事务又在该范围内**插入**了新的记录，当之前的事务再次读取该范围的记录时，会产生第一次读取范围时不存在的**幻行**（Phantom Row）。*

> MySQL的默认隔离级别是可重复读，有幻读问题。

- 可序列化
  - **可序列化**是最高的隔离级别：**强制事务序列化执行。**

    *可序列化会在目标**范围**加独占锁，将并发读写相同范围数据的请求序列化。*可序列化会导致大量的超时和锁争用问题，实际应用中很少用到这个隔离级别，只有**在非常需要确保数据的一致性**而且可以接受没有并发的情况下，才考虑使用该级别。