myqsl 12种锁，提供12个真实业务与流程图，轻松掌握运用场景与方式

MySQL的锁设计是为了在多用户环境中管理对数据库数据的并发访问，确保数据的完整性和一致性，提供了大概12种锁，每一种锁对应的业务场景不一样，肖哥带你掌握他的使用技巧。

1、读锁/共享锁（S锁）

流程图

* **设计目的**：允许多个事务并发读取同一资源，提高系统的并发性能。
* **解决问题**：防止脏读。
* **使用场景**：用户查询银行账户余额，同时不希望阻止其他用户查询。
* **业务案例**：在线教育平台，当教师查询学生的成绩记录时，多个教师可以同时访问成绩，但不允许修改。
* **MySQL操作案例**：

```
SELECT * FROM accounts WHERE user_id = 'user1' LOCK IN SHARE MODE;
```

2、写锁/排它锁/独占锁（X锁）

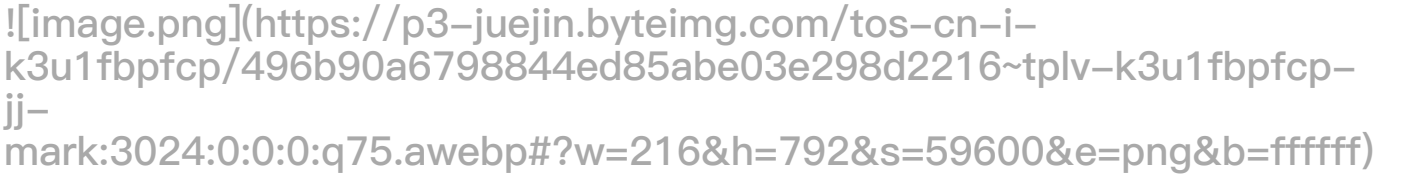
流程图

* **设计目的**：确保在数据被修改时，其他事务不能读取或修改该数据。
* **解决问题**：防止不可重复读和幻读。
* **使用场景**：用户更新银行账户余额，需要确保更新期间数据不被其他事务干扰。
* **业务案例**：电子商务平台，在处理用户订单时，当订单状态需要被更新时，需要确保在更新过程中数据的一致性。
* **MySQL操作案例**：

```
START TRANSACTION;
UPDATE accounts SET balance = balance - 100 WHERE user_id = 'user1';
COMMIT;
```

3、意向锁（Intention Locks）

流程图



* **设计目的**：在多粒度锁定系统中，用于表明事务在更高层次上的锁定意图。
* **解决问题**：协调行锁和表锁之间的关系。
* **使用场景**：事务需要在表的多行上设置锁，但表已被其他事务锁定。
* **业务案例**：金融交易系统，在进行大规模数据报表生成时，需要在表级别表明锁定意图，以协调行锁和表锁。
* **MySQL操作案例**： 意向锁通常由MySQL自动处理，不需要用户显式操作。

4、自增锁（Auto-increment Locks）

流程图

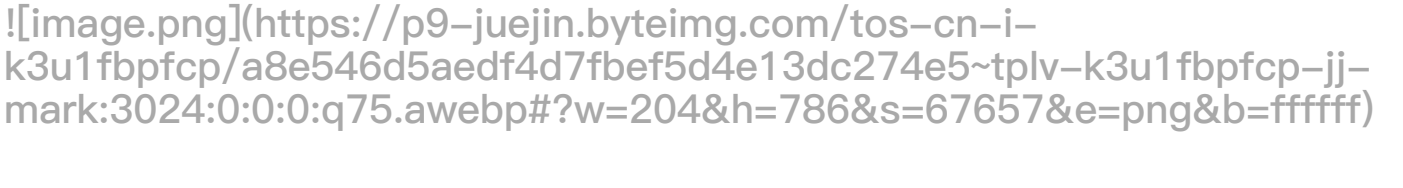
mark:3024:0:0:0:q75.awebp#?w=241&h=790&s=65189&e=png&b=ffffff)

* **设计目的**：确保自增字段在并发插入时能够生成唯一的序列号。
* **解决问题**：防止自增字段的冲突。
* **使用场景**：在插入新用户记录时自动分配唯一ID。
* **业务案例**：社交媒体平台，在创建新的帖子时，需要为每个帖子分配一个唯一的标识符。
* **MySQL操作案例**：

```
INSERT INTO users (username) VALUES ('new_user');
```

5、 悲观锁（Pessimistic Locks）

流程图

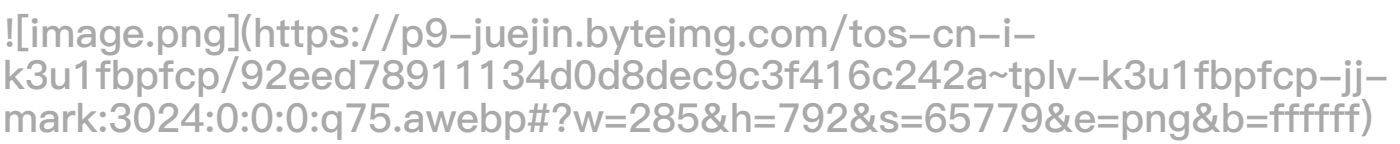


* **设计目的**：在事务可能发生冲突的环境中，通过锁定资源来保证数据一致性。
* **解决问题**：避免数据竞争。
* **使用场景**：银行转账，需要确保资金的准确性和一致性。
* **业务案例**：银行系统，在处理资金转账时，为了避免并发操作导致的数据不一致，采取悲观锁策略。
* **MySQL操作案例**：

```
SELECT * FROM accounts WHERE id = 1 FOR UPDATE;
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
COMMIT;
```

6、乐观锁（Optimistic Locks）

流程图



* **设计目的**：在冲突较少的环境中，通过检测数据在读取后是否被修改来避免锁的开销。
* **解决问题**：减少锁的使用，提高系统性能。
* **使用场景**：在线票务系统，用户查看票务信息后决定是否购买。
* **业务案例**：在线协作文档编辑，当多个用户可以同时编辑同一文档时，使用版本号来确保操作的一致性。
* **MySQL操作案例**： 乐观锁通常通过应用程序逻辑实现，例如使用数据表中的版本号或时间戳字段。

1. **初始读取记录** - 应用程序读取记录以及版本号。

```
SELECT id, data_column, version_column FROM table_name WHERE id = record_id;
```

这里，`data_column` 是你想要读取的数据列，`version_column` 是用于乐观锁的版本号列。

2. **更新记录** - 应用程序在本地进行更改，并尝试更新数据库中的记录，同时检查版本号是否一致。

```
UPDATE table_name
SET data_column = new_data_value,
    version_column = version_column + 1
WHERE id = record_id
  AND version_column = original_version_value;
```

在这个例子中，`new_data_value` 是应用程序更新后的值，`original_version_value` 是最初读取的版本号。如果`version_column` 在尝

试更新时没有改变，这条语句会成功更新记录并增加版本号。如果版本号已经改变（意味着另一个事务已经更新了记录），这个更新就不会执行。

3. **检查更新是否成功** - 应用程序检查更新操作是否影响了预期数量的行。

```
-- 这通常在应用程序逻辑中处理，例如：
IF (update_affected_rows > 0) THEN
    -- 更新成功，版本号增加
ELSE
    -- 更新失败，版本号未改变，可能需要重新读取记录或抛出异常
END IF;
```

7、间隙锁（Gap Lock）

流程图

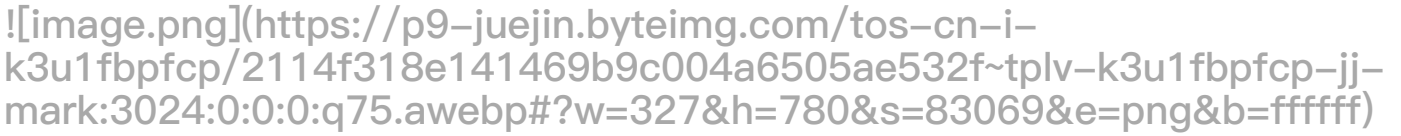
* **设计目的**：防止幻读，确保范围查询的一致性。
* **解决问题**：避免在锁定范围内插入新记录。
* **使用场景**：适用于需要在某个范围内进行范围查询并保证结果一致性的情况。
* **业务案例**: 银行账户交易记录查询，需要确保在查询的日期范围内，不会有新的交易记录被插入，从而保证查询结果的一致性。
* **MySQL操作案例**： 间隙锁通常在执行范围查询并加锁时隐式使用，例如：

```
SELECT * FROM table_name WHERE column_name BETWEEN val1 AND val2 LOCK IN SHARE MODE;
```

这个查询会锁定`val1`和`val2`之间的所有记录，并且防止其他事务在这个范围内插入新记录。

8、页锁（Page Lock）

流程图



* **设计目的**：锁定数据库页，以减少锁的粒度。
* **解决问题**：在行锁和表锁之间提供一种折中的锁定策略。
* **使用场景**：适用于对表的某个部分进行操作，而不需要锁定整个表。
* **业务案例**:数据库维护任务，如索引重建，需要对表的特定部分进行锁定，以减少对整个表的影响。
* **MySQL操作案例**： 页锁通常由InnoDB自动使用，不需要显式的SQL操作。但是，可以通过指定行锁来间接使用页锁。

9、MDL锁（Metadata Lock）

流程图

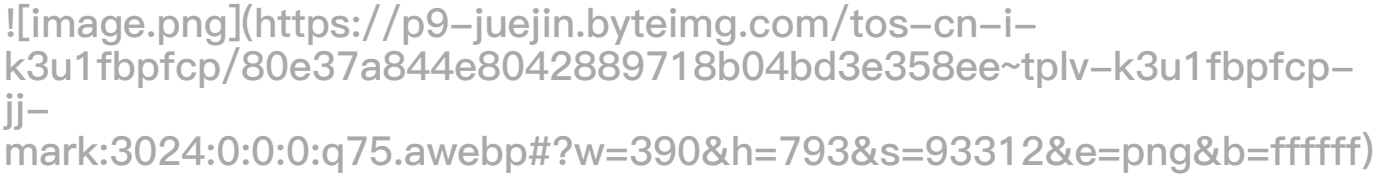
* **设计目的**：锁定数据库对象的元数据。
* **解决问题**：防止在执行某些操作时元数据被更改。
* **使用场景**：适用于修改表结构或统计信息收集等操作。
* **业务案例**：数据库管理员在调整数据库结构，如添加新列或索引，以确保结构变更期间数据定义的一致性。
* **MySQL操作案例**： MDL锁由MySQL自动管理，例如：

```
ALTER TABLE table_name ADD COLUMN new_column datatype;
```

这个操作会隐式地使用MDL锁。

10、RL锁（Record Lock）

流程图



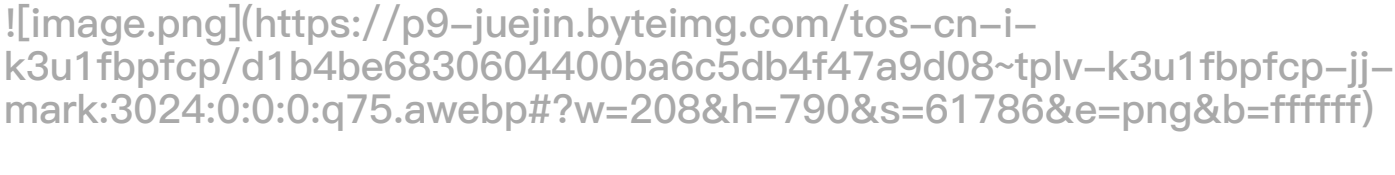
* **设计目的**：锁定单个记录。
* **解决问题**：确保单个记录的一致性和完整性。
* **使用场景**：适用于需要更新单个记录并保证更新期间记录不被其他事务更改的情况
* **业务案例**：电子商务网站在处理订单，需要锁定特定订单记录，以确保订单信息在处理过程中不被其他操作更改。
* **MySQL操作案例**：

```
SELECT * FROM table_name WHERE id = record_id FOR UPDATE;
```

这个查询会锁定指定ID的记录，防止其他事务修改它。

11、GL锁（Gap Lock）

流程图



* **设计目的**：锁定一个范围内的记录，但不包括记录本身。
* **解决问题**：防止在范围内插入新记录，解决幻读问题。
* **使用场景**：适用于需要保证某个范围内数据一致性的情况。
* **业务案例**：股票交易系统，在查询某个价格区间内的股票记录时，需要

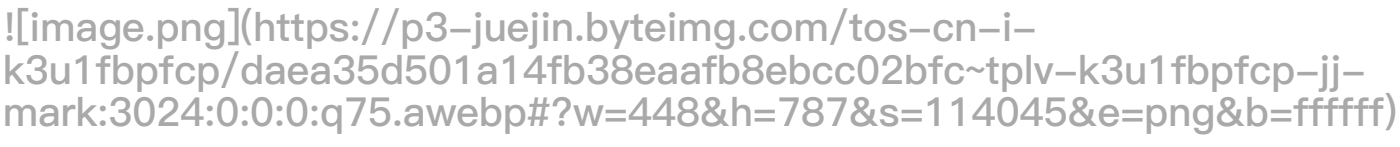
防止新的交易记录在该价格区间内被插入。
* **MySQL操作案例**： GL锁通常在执行范围查询并加共享锁时隐式使用，例如：

```
SELECT * FROM table_name WHERE column_name > val LOCK IN SHARE MODE;
```

这个查询会锁定大于`val`的所有记录，但不包括`val`本身的记录。

12、NKL锁（Next-Key Lock）

流程图



* **设计目的**：结合了记录锁和间隙锁的特性。
* **解决问题**：同时防止在同一记录上发生冲突和在范围内插入新记录。
* **使用场景**：适用于需要同时保证记录一致性和范围一致性的情况。
* **业务案例**：在线预订系统，在查询并锁定特定日期的房间时，需要同时防止该日期的房间被其他用户预订，以及防止在相邻日期插入新的预订记录。
* **MySQL操作案例**： NKL锁通常由InnoDB自动使用，不需要显式的SQL操作。例如，在执行以下查询时：

```
SELECT * FROM table_name WHERE column_name = val FOR UPDATE;
```

原文链接: https://juejin.cn/post/7385491648081707048