1 MySQL 简介

1.1 MySQL 是什么？

MySQL 是由 MySQL AB 公司（目前已经被 Oracle 公司收归麾下）自主研发的，目前 IT 行业最流行的开放源代码的数据库管理系统之一，它同时也是一个支持多线程高并发多用户的关系型数据库管理系统。

MySQL 所使用的 SQL 语言是用于访问数据库的最常用标准化语言。MySQL 软件采用了双授权政策，分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型网站的开发都选择 MySQL 作为网站数据库。

1.2 MySQL 通常应用在哪些场景？

MySQL 是目前最为流行的开源数据库管理系统软件之一。与其他的大型数据库例如 Oracle、DB2、SQL Server 等相比，MySQL 有它的不足之处，但是这丝毫也没有减少它受欢迎的程度。对于一般的个人使用者和中小型企业来说，MySQL 提供的功能已经绰绰有余，而且由于 MySQL 是开放源码软件，因此可以大大降低总体拥有成本。MySQL 主要应用于如下场景：

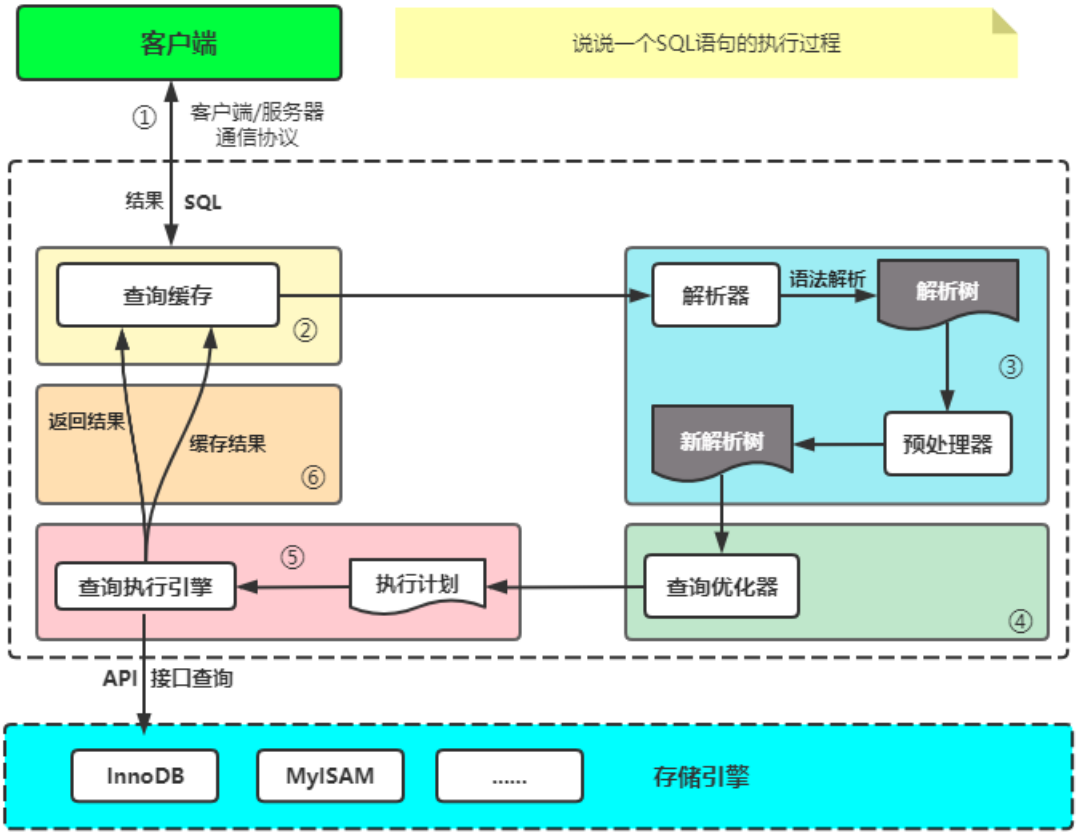
- Web 站点系统；
- 数据仓储系统；
- 嵌入式系统；

任何产品都不可能是万能的，也不可能适用于所有的应用场景。

2 MySQL 逻辑架构

2.1SQL 语句的执行逻辑是怎样的？

我们用一条 SQL SELECT 语句的执行轨迹是怎样的，如图所示：



其中：

①通过客户端/服务器通信协议与 MySQL 建立连接。

②查询缓存，这是 MySQL 的一个可优化查询的地方，如果开启了 Query Cache 且在查询缓存过程中查询到完全相同的 SQL 语句，则将查询结果直接返回给客户端；如果没有开启 Query Cache 或者没有查询到完全相同的 SQL 语句则会由解析器进行语法语义解析，并生成解析树。

③预处理器生成新的解析树。

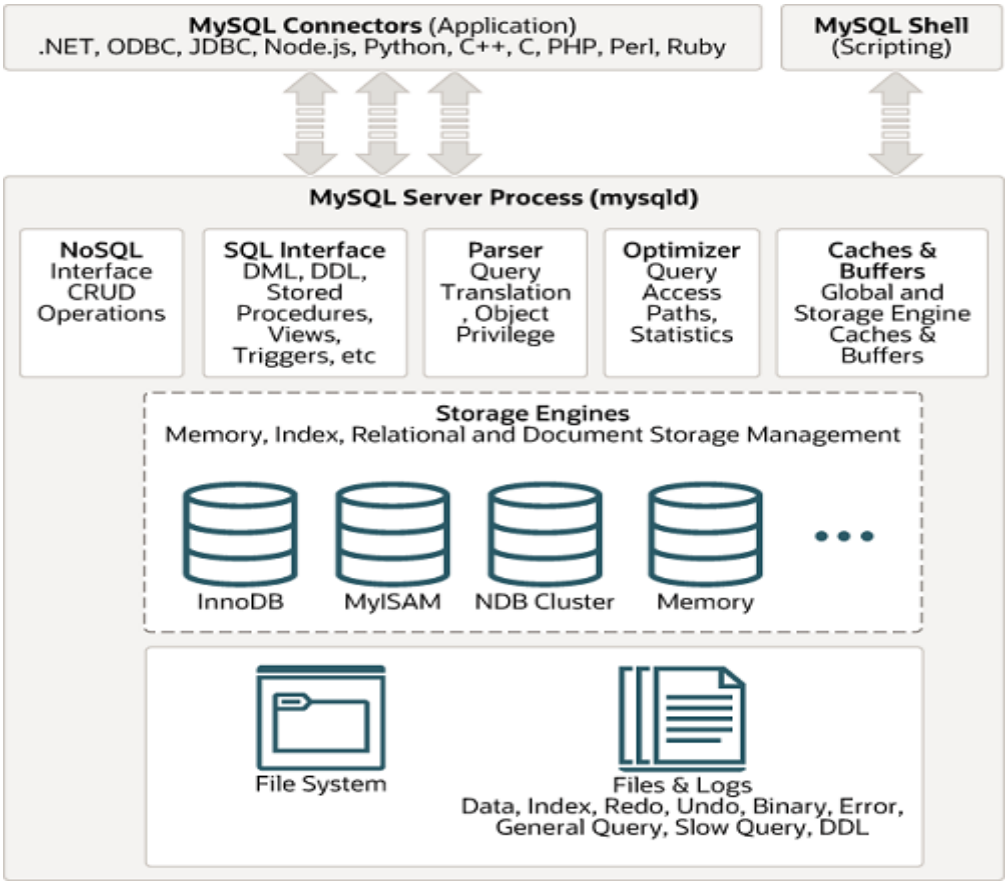
④查询优化器生成执行计划。

⑤执行引擎执行 SQL 语句，此时执行引擎会根据 SQL 语句中表的存储引擎类型，以及对应的 API 接口与底层存储引擎缓存或者物理文件的交互情况，得到查询结果，由 MySQL Server 过滤后将查询结果缓存并返回给客户端。若开启了 Query Cache，这时

也会将 SQL 语句和结果完整地保存到 Query Cache 中，以后若有相同的 SQL 语句执行则直接返回结果。

2.2 MySQL 逻辑架构是怎样的？

MySQL 是一种典型的 C/S 架构设计。接下来我们先先鸟瞰其全貌（注意，千万不要直接陷入细节里），从更高维度对 MySQL 有一个基本认知，如图所示：



图中，MySQL 体系结构由 Client Connectors 层、MySQL Server 层及存储引擎层组成。

- 连接层

负责接收客户端的连接请求，与服务端创建连接。目前 MySQL 几乎支持所有的连接类型，例如常见的 JDBC、Python、Go 等。

- 服务层

MySQL Server 层主要包括 Connection Pool、Service & utilities、SQL interface、Parser 解析器、Optimizer 查询优化器、Caches 缓存等模块。

- ✓ SQL interface，负责接收客户端发送的各种 SQL 语句，比如 DML、DDL 和存储过程等。
- ✓ Parser 解析器会对 SQL 语句进行语法解析、语义解析生成语法树。
- ✓ Optimizer 查询优化器会根据解析树生成执行计划，并选择合适的索引，生成执行计划,然后将执行计划交给存储引擎,最后通过存储引擎执行 SQL。
- ✓ Caches 缓存包括各个存储引擎的缓存部分，比如：InnoDB 存储的 Buffer Pool、MyISAM 存储引擎的 key buffer 等，Caches 中也会缓存一些权限，也包括一些 Session 级别的缓存。从 MySQL 5.7.20 开始，不推荐使用查询缓存，并在 MySQL 8.0 中删除 。

- 存储引擎层。

存储引擎真正的负责了 MySQL 中数据的存储和提取，服务器通过 API 与存储引擎进行通信。不同的存储引擎具有的功能不同，常用的存储引擎包括 MyISAM、InnoDB，以及支持归档的 Archive 和内存的 Memory 等。

MySQL 提供了插件式存储引擎层（ Storage Engines），真正的负责了 MySQL 中数据的存储和提取。服务器通过 API 与存储引擎进行通信。不同的存储引擎具有的功能不同，常用的存储引擎包括 MyISAM、InnoDB，以及支持归档的 Archive 和内存的 Memory 等。我们可以根据自己的实际需要进行选取。

MySQL 8.0.25 默认支持的存储引擎如下：

```
mysql> show engines;
```

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

- 物理存储层

物理数据存储层，主要是将数据存储在运行于该设备的文件系统中，这些文件包括二进制日志、数据文件、错误日志、慢查询日志、全日志、redo/undo 日志等。

3 MySQL 数据类型分析

3.1MySQL 中有哪些数据类型?

数据类型定义了 MySQL 列中可以存储什么数据以及当前数据存储的基本规则。在使用 MySQL 的过程中，一定会根据数据类型的限制、特性有所取舍。MySQL 中可选的数据类型有很多，每一种数据类型都会有其使用限制与适合的使用场景

通常，我们会将 MySQL 的数据类型分为四类，*即字符串、日期 / 时间、数值以及二进制。*显然，根据这些分类的名称可以知道，分类是按照存储数据的类型来做的。那么，这些分类中又包含了哪些数据类型呢 ？

- 字符串类型：以 char、varchar、text 为代表，用于存储字符、字符串数据.
- 日期/时间类型：以 date、time、datetime、timestamp 为代表，用于存储日期或时间，这种数据类型也是比较难抉择的。
- 数值类型：以 tinyint、int、bigint、float、double、decimal 为代表，用于存储整数或小数。
- 二进制类型：以 tinyblob、blob、mediumblob、longblob 为代表，用于存

储二进制数据，适用场景最为受限。

说明，对数据类型的分类并不是绝对的，这取决于对存储数据的限制程度。例如对于数值类型又可以再去细分为整数型（int、bigint 等）、浮点型（float、double 等）、定点型（decimal 等）。所以，并不需要把过多的精力花在类型分类上，更多的是应该搞清楚这些类型怎么用，又为什么这样用。

3.2如何查看类型的具体信息?

对于平时写代码的你来说，Linux/Unix 环境一定不会陌生，当然，也就对 man 和 help 这样的命令不会陌生了。类似于这样的 “帮助命令” 在 MySQL 中也是有的。例如，你想知道 int 这种数据类型的使用范围，可以执行命令：

```
mysql> help int
Name: 'INT'
Description:
INT[(M)] [UNSIGNED] [ZEROFILL]
A normal-size integer. The signed range is -2147483648 to 2147483647.
The unsigned range is 0 to 4294967295.
URL: https://dev.mysql.com/doc/refman/5.7/en/numeric-type-overview.html
```

可以看到，help 打印了 int 数据类型的描述信息以及官方文档的链接地址。这对于学习使用数据类型来说，是非常方便的。当然，我们也可以在 help 后面加上 char、varchar 等等 MySQL 支持的数据类型。

此时，你可能会有疑问：这些打印的信息是从哪里来的 ？难道也是保存在 MySQL 表中的吗 ？确实，正如猜测的那样，MySQL 提供了 4 张表用于保存帮助信息（help 语法打印的即为帮助信息）。这些表位于 mysql 系统字典库

中，且表名都以 help_ 开头。如下所示：

```
mysql> show tables from mysql where Tables_in_mysql like 'help_%';
+-----------------+
| Tables_in_mysql |
+-----------------+
| help_category |
| help_keyword |
| help_relation |
| help_topic |
+-----------------+
```

这些表是在数据库初始化时通过内建脚本创建而成，其中：

- ✓ help_category：存储关于帮助主题类别的信息
- ✓ help_keyword：存储与帮助主题相关的关键字信息
- ✓ help_relation：存储帮助关键字信息和主题信息之间的映射
- ✓ help_topic：存储帮助主题的详细内容

由此，可以知道，我们之前的 help int 信息来自于 mysql.help_topic 表中，也就可以通过查询表记录信息来获取帮

助信息了。如下所示：

```
--
\G 指示 MySQL 以列格式打印结果信息
mysql> SELECT * FROM mysql.help_topic WHERE name = 'int'\G
********1. row ************
help_topic_id: 13
name: INT
help_category_id: 2
description: INT[(M)] [UNSIGNED] [ZEROFILL]
A normal-size integer. The signed range is -2147483648 to 2147483647.
The unsigned range is 0 to 4294967295.
URL: https://dev.mysql.com/doc/refman/5.7/en/numeric-type-
overview.html
```

example:
url: https://dev.mysql.com/doc/refman/5.7/en/numeric-type-overview.html

关于其他的 “帮助表” 这里不再过多介绍，有兴趣的同学可以自行查询 MySQL 官网或其他渠道了解信息。

3.3 你是如何理解 MySQL 中一些常见类型的？

3.3.1 字符串

- **Char 类型**

char 数据类型用于定义一个固定长度的字符串，长度范围处于 1 ~ 255 之间，且必须是在创建表时指定。它有一个特殊的情况是，存储字符串时，如果未达到指定长度，则会使用空格填充到指定长度。所以，如果我们想要存储不同记录的字符串长度差别较大，会造成较大的空间浪费。根据对 char 类型的描述可以知道，当我们需要存储一些长度固定的数据列时，使用 char 是非常合适的。例如：手机号码、身份证号等等。

- **Varchar 类型**

相对于 char 来说，varchar 的 “出场率” 要稍微高一些。它定义了一个可变长度的字符串，创建时指定它所允许的最大长度。例如，如果创建时声明了 varchar (x)，则只能存储不超过 x 个字符的数据，且 x 的最大值是 65535。对于长度不固定的数据列，使用 varchar 就是最合适的。例如：姓名、邮箱地址等等。

char 和 varchar 是非常相似且常见的字符串类型，想要把它们用对、用好，不仅要能够理解它们各自的含义、特性，还要知道它们在使用上的区别：

- ✓ 定义了 char (x)，如果存入的字符个数小于 x，则以空格填充，查询时再将空格去掉（类似于 trim 操作）。所以，char 类型存储的字符串末尾不能有空格，而 varchar 则没有这一限制。
- ✓ char (x) 长度是固定的，不论存入什么，都会占用 x 个字节。但是 varchar

占用的字节数是存入的字符数 +1（x <= 255）或 + 2（x > 255）。

- ✓ char 由于长度固定，不需要考虑边界问题，检索速度要快于 varchar

- **文本类型**

text 是文本数据类型，它分为四类，都是变长字符串，最大的区别是存储空间的不同，其中：

- ✓ tinytext：最大长度是（2^8 - 1）个字符。
- ✓ text：最大长度是（2^16 - 1）个字符。
- ✓ mediumtext：最大长度是（2^24 - 1）个字符。
- ✓ longtext：最大长度是（2^32 - 1）个字符。

最简单的对文件数据类型的理解是：当我们要存储的数据量比较大，就应该考虑使用文本。这里，我建议当你的数据量超过 500 个字符时，就应该考虑使用文本。另外，文本类型不能有默认值，且在创建索引时需要指定前多少个字符。

3.3.2 日期

- Date 类型

正如这种数据类型的名称一样，它用于存储日期，存储范围是 '1000-01-01' 到 '9999-12-31'。这种数据类型比较简单，但同时适用场景也比较有限，因为它只能存储“年月日”。比较常见的用途是存储出生日期。

- Datetime 类型

它用于存储时间，不仅可以表示一天中的时间，也可以用于表示两个时间的时间间隔。它的取值范围是 '-838:59:59' to '838:59:59'。乍看起来，它的小时取值太特殊了，正常不应该是 [0, 23] 吗 ？这是因为 time 可以表示特殊的时间间隔，MySQL 将

time 的小时范围扩大了，而且支持负值。

除了基本的存储一天中的时间之外，time 允许以 “D HH:MM:SS” 的格式存储。其中，D 的取值是 0 ~ 34。如果要存储时间间隔，time 则会以（时间间隔 * 小时）作为小时进行存储。它的计算公式是：D * 24 + HH。例如，插入了 “2 19:20:00”，相当于插入 “67:20:00”。

- Timestamp 类型

同样用于存储日期时间数据，与 datetime 存储的数据格式是一样的，它的取值范围是：‘1970-01-0100:00:01.000000’ UTC 到 ‘2038-01-19 03:14:07.999999’ UTC。它与 datetime 的主要区别在于时间范围要小一些。

另外，timestamp 是与时区相关的，能够反映 “当前时间”。当插入时间时，会先转换为本地时区后再存储；查询时间时，会转换为本地时区后再显示。所以，不同时区的人看到的同一时间是不一样的。

在 MySQL 表中存储时间（可以是日期、时间或日期时间）是非常常见的需求，但是如何合理的选择数据类型却也是个难题。这里我给出一个建议：通常 datetime 是最佳选择。理由如下：

- ✓ 时间范围跨度足够大，能够满足所有的时间需求。
- ✓ 即使是只用于存储日期或时间，也可以存储日期时间，只需要在代码中处理即可。避免将来需求变更时对数据表的 Schema 有所变动。

3.3.3 数值

- 整数类型

MySQL 主要支持 5 个整数类型：tinyint、smallint、mediumint、int、bigint。这些数据类型我们基本上认为它们有共同的特性，不同之处只在于存储空间，即存储数值的取值范围。同时，在定义时可以使用 UNSIGNED 关键字规定字段只保存正值。下面，我

将这几种整数类型的特性用表格展示出来。

数据类型	占据空间	范围（有符号）	范围（无符号）	描述
tinyint	1 个字节	-2^7 - 2^7 - 1	0 - 255	小整数值
smallint	2 个字节	-2^15 - 2^15 - 1	0 - 65535	大整数值
mediumint	3 个字节	-2^23 - 2^23 - 1	0 - 16777215	大整数值
int	4 个字节	-2^31 - 2^31 - 1	0 - 4294967295	大整数值
bigint	8 个字节	-2^63 - 2^63-1	0 - 18446744073709551615	极大整数值

由于这几种数据类型除了取值范围不同之外，并没有其他的不同，所以，在使用上，根据需要选择 “足够大” 的空间就可以了。另外，关于整数类型还有一个特性：显示宽度。例如，我们在定义 Schema 时，常常会看到类似这样的写法：

```
`a` bigint(20) NOT NULL COMMENT 'a',
`b`int(11) NOT NULL COMMENT 'b'
```

其中，20 和 11 就是可选的显示宽度，这会让 MySQL 对 SQL 标准进行扩展，当从数据库检索一个值时，可以把这个值延长到指定的宽度。例如，这里的 b 定义的类型为 int (11)，就可以保证 b 这一列少于 11 个字符宽度时自动使用空格填充。但同时，需要注意，定义宽度并不会影响字段的大小和存储值的取值范围。

- 浮点类型

MySQL 支持两个浮点类型：float、double。其中，float 用于表示单精度浮点数值，占用 4 个字节；double 用于表示双精度浮点数值，占用 8 个字节。因为它们只能保存近似值（不精确的值），所以，通常也叫做非标准类型。

float 相较于 double 类型来说，由于占据的空间小，精度较低，取值范围也相对较小。它们的定义格式及说明如下：

✓ float (M, D)：其中 M 定义显示长度，D 定义小数位数。但是它们是可选的，

且默认值是 float (10, 2)，2 是小数的位数，10 是数字的总长（包括小数）。它的小数精度可以到 24 个浮点。

- ✓ double (M, D)：M 和 D 的含义与 float 是相同的，默认值是 double (16, 4)。它的小数精度可以达到 53 位。

- 定点类型

MySQL 中的 decimal 被称为定点数据类型，由于它保存的是精确值，所以它通常用于精度要求非常高的计算中。另外，也可以利用 decimal 去保存比 bigint 还要大的整数值。

CPU 并不支持对 decimal 的直接计算，而是 MySQL 自身实现了对 decimal 的高精度计算。底层存储方面，MySQL 将 decimal 类型的数字使用二进制字符串存储，每 4 个字节可以存储 9 个数字。假如我们定义了 decimal (18, 9)。

- ✓ 则代表不包含小数点的数字总数（整数位数 + 小数位数）位数是 18，不指定的情况下默认是 10
- ✓ 9 则代表小数的位数，如果不指定，默认是 0。

由于小数点两边各有 9 个数字，所以占据 2 * 4 = 8 个字节，小数点自身占用一个字节，最终，decimal (18, 9) 一共占用 9 个字节。需要注意，如果存储的位数不够，则小数末尾会补零。但是，如果超出了声明的位数，则会报错。

由于 decimal 需要比较大的空间和计算开销，它的计算效率也就没有 float 和 double 那么高，所以应该只有要求精确计算的场景下才考虑去使用 decimal。

3.3.4 二进制

二进制数据类型理论上可以存储任何数据，可以是文本数据，也可以存储图像或者其他多媒体数据。二进制数据类型相对于其他的数据类型来说，使用频率是比较低的。

MySQL 一共提供了四种二进制类型：tityblob、blob、mediumblob、longblob，它们的区别只在于存储范围的不同。

- ✓ tinyblob：最大支持 255 字节
- ✓ blob：最大支持 64KB
- ✓ mediumblob：最大支持 16MB
- ✓ longblob：最大支持 4GB

需要注意，虽然 MySQL 提供并支持大文件存储，但是这样会急剧降低数据库的性能。所以，应该谨慎使用这些数据类型，能不用的情况下尽量不用。

3.4 你是如何使用 MySQL 数据类型的？

mysql 数据类型应用时有一些技巧性经验，但是这些经验并不一定适用于所有的情况。在做实际的选择时，我们不仅要考虑这些技巧，也要对应到具体的需求。

3.4.1 使用 Not Null 和 Comment

建议所有字段类型的定义，都要指定 Not null 和 Comment，为什么呢？

MySQL 在定义索引值为 NULL 的列时，需要额外的存储空间。另外，在进行比较和计算时，MySQL 要对 NULL 值做特别的处理，使用效率较低。

COMMENT 用于定义列的注释信息，就好像我们在写代码一样，把重要的或者不易理解的地方，加上一些注释，方便以后查阅。

3.4.2 选择简单数据类型

这里的 “简单” 二字听上去会比较奇怪，我以一个例子去说明。假如说我想在一列中存储 10、100、201 这样的数据，我们可以选择使用 int 或 varchar 来存储。但是整型要比字符型的操作复杂度小太多，那么，选择整型（例如 int）就是最简单的数据类型。

3.4.3 应用最小数据类型

这里所说的最小数据类型并不是直接选择最小的，而是在满足需求的同时选择最小的。例如，要存储事件状态，可以选择 tinyint；要存储班级人数，可以选择 smallint 等等。关于最小数据类型，它有两大优势：

- ✓ 越小的数据类型占用的磁盘、内存、CPU 缓存都会更小，存取速度也会更快。
- ✓ 小的数据类型建立索引时所需要的空间也相对较小，这样一页中所能存储的索引节点数量也就越多，遍历时 IO 次数就会越少，索引的性能也就越好。

3.4.4 基于 Decimal 类型存储小数

虽然我并不建议在数据库中存储小数，但是，在一些场景中小数不可避免，最常见的例子就是订单的金额。由于小数本身在计算时就很复杂，而且很多时候你需要去考虑精度问题。所以，最直接的方式就是把这种管理交给数据库。

这里我提出一个扩展建议，也就是不要在数据库中存储小数。那么，假如订单的精度到分（元、角、分）级别，我们可以考虑在存储时，把数据值 * 100 再去存储。之后，在代码中处理分的逻辑，也就是自己去控制处理小数的精度问题。

3.4.5 尽量避免使用 text 和 blob 类型

MySQL 内存临时表并不支持 text、blob 这样的大数据类型，如果查询时包含有这样的数据，则排序操作必须使用磁盘临时表，性能会下降很多。而且对于这种数据，MySQL 还要做二次查询（因为 MySQL 实际保存的是指针，而不是真实数据），会使 SQL 性能变得很差。

但是，也并不是说我们一定就不能用 text 和 blob。如果确实有需求需要使用这样的数据类型，那么在查询时一定不要直接 SELECT *，而是取出需要的列。这样 MySQL 就不会去主动查询这些数据列，也是提高性能的一种惯用手段。

最后，还需要注意，因为 MySQL 对索引长度的限制，text 类型只能用到前缀索引，

并且由于存储的是指针，text 列上不能有默认值。

3.5小节总结说明

数据类型是 MySQL 的基础，看起来也比较简单，但常常也就是觉得简单才会忽略它们的特性与限制。可以肯定的说，想要选择正确的、合理的数据类型并不是一件简单的事。不过，也并不需要追求完美的选型。能够解决实际的问题，或多或少存在一些瑕疵，当然也是可以接受的。先去学习并理解，再去大胆的使用，遇到瓶颈了再回过头仔细分析问题，并解决掉，这就是很好的学习方法。

4 MySQL 数据库及表设计

4.1什么是数据库设计？

我们所说的数据库设计一般指的是对库和表的设计。也就是，在对 MySQL 基本的使用中，如何根据业务需求去创建数据库、创建数据表。

4.2数据库设计的目标是怎样的？

通俗的说，不论是 MySQL 还是其他工具也好，最基本的设计目标肯定是可用。如果可以，就在可用的基础之上，再去追求好用。下面，我来详细的对可用和好用的设计目标进行解读。

如果你设计的数据库和数据表能够支撑当前的业务需求，且在技术实现上没有太大的弊端，那么，我们就可以说它是可用的。更深层的看，这个设计目标的核心其实是对需求的理解。确实，理清了需求，你会得出结论：应该存储哪些数据、这些数据是什么类型、在代码中怎样使用这些数据等等。余下的建库建表也自然就是水到渠成了。

需求也许不会变化，但是随着业务量的增长触发数据和并发的增长，数据库是否还能保持相对较高的性能是个值得思考的问题，同时也是衡量设计目标是否好用的重要指标。无论什么时候，我们对 MySQL（数据库）的使用都肯定是围绕数据的增删改查。而这些基本的操作，当数据量加速膨胀的过程中，也会引起性能瓶颈。所以，好用的设计目标讲究能够“预见未来”，能够对未来做出预判。例如：将通用信息单独使用一张表存储、建立适当的索引等等。

说明，规范是一种通用的建议，并不一定适用于所有的场景，一定要仔细分析需求再做出合适的取舍。

4.3 MySQL 应用时有哪些设计原则？

鉴于 MySQL 的一些固有属性（特性），在使用上我们通常都会遵守一些 “共识”，而这些是与具体的业务没有相关性的。下面，我将会介绍一些通用的设计原则，它们有些是关于库的，有些是关于表的。

- 使用小写的名称，且只有英文字母：不论是库、表还是数据列，应该是只包含英文字母的名称，不要出现特殊字符或者是数字。这也比较好理解，英文字母不论是阅读还是编码都非常的便捷。另外，由于 MySQL 是大小写不敏感的，选择一律小写的名称能够统一书写规则，避免不必要的书写错误。

- 取一个有意义的名称，单词之间使用下划线连接：除了基本的名称书写规范之外，取一个有意义的名称是非常有必要的。例如：我们需要创建学生表，表的名称叫做 student 就会比 other 更易理解。当然，可能有些时候我们无法用一个单词表达清楚想要的含义，此时，可以使用多个单词，且单词之间使用下划线连接，例如：insert_time。最后，名称不要过长，最长不要超过 32 个字符。

- 记住 “够用且尽量小” 的原则：很明显，这条原则对应的是数据表列的数据类型选择问题。占用空间少的数据类型最直接的优势就是减少了用户数据存储空间和索引存储空间，这对于数据传输与检索的性能提高有着巨大意义。

- 不要使用物理外键：物理外键是说让数据库去管理表与表之间的关联关系，而它相对

的逻辑外键，则是我们自己用代码去管理这种关系。这是因为物理外键存在两个重大缺陷：消耗数据库资源，降低数据库实例可扩展性；母表一旦受损，子表很难恢复，造成数据丢失。

- 表一定要有主键：MySQL 并不要求表一定要有主键，但是主键的作用是能够唯一区分表中的每一行。没有主键，更新或删除表中的特定行将会很困难，因为没有安全的方法保证只涉及相关的行。并且，主键能够为方便扩展、高可用的数据库系统做铺垫。

- 保持一致的字符集：库、表、数据列的字符集都应该是一致的，统一为 utf8 或 utf8mb4。字符集编码不仅影响数据存储，还会影响客户端与数据库之间的交互，最常见的问题就是字符集导致的乱码。所以，相同的字符集更利于管理，也更方便去排查问题。

4.4数据库设计规范是怎样的？

4.4.1 字符集

创建数据库时，建议指定字符集，例如

```
CREATE DATABASE [IF NOT EXISTS] database_name CHARACTER SET utf8;
```

4.4.2 表的个数

首先，我们可以使用如下的 SQL 语句查看系统库 mysql 中定义了多少张表：

```
mysql> SELECT COUNT(*) TABLES, table_schema
FROM information_schema.TABLES
WHERE table_schema = 'mysql'
GROUP BY table_schema;

+--------+--------------+
| TABLES | table_schema |
```

```
+--------+--------------+
| 31 | mysql |
+--------+--------------+
```

可以看到，mysql 库（注意区分是系统库）中定义了 31 张表。由于它是 MySQL 的系统库，所以我们可以理所应当的认为它是合理的。换句话说，一个库中至少是可以存储 31 张表的。但是，得出这样的结论仅仅是靠猜测，没有任何站得住脚的依据。

MySQL 自身并没有对库的容量做出限制，也就是说，你几乎不用考虑表的数量上限问题。但是，当表的数量越多，越容易产生以下问题（以下所讨论的都是单个库）：

- ✓ 表越多，需要维护的元数据（表结构、统计信息等）就会越多。即使是这些元数据只占据很少的空间，但是也会让管理这些元数据变得很复杂，且通常也是不合理的需求分析造成的；
- ✓ 表越多，可能存储的数据量也会越大，这无疑会给数据库造成压力。且大量的数据聚集在同一个库中也是非常危险的，一旦出现库损坏，丢失的数据量也会更多。

综上所述，我们讨论了单库中表太多的缺陷，再去结合日常的工作实践来说，建议大家在一个库中创建的表数量不要超过 200。更常见的情况是，一个库中只维护几十到 100 张表。

4.5数据库表的设计规范是怎样的？

4.5.1 范式与反范式

相信我们在刚开始接触数据库的时候就听过范式的概念，它的核心思想是数据只出现一次，不存在信息冗余。而反范式的概念也就是破坏了范式的规范，它允许出现冗余的数据。所以，对于这个问题来说，它聚焦的点在于：冗余字段是否是可取的。

对于任何给定的数据来说，我们可以设计各种各样的存储方案，从完全范式化到完全反范式化，或者兼顾两者。对于范式化的设计，有着这样的优点：

- ✓ 使用更少的存储空间（现在很少有人特别在乎这个点存储空间了）
- ✓ 由于没有冗余存储，增删改查的速度相对较快（有冗余时为了保证数据的一致性还有更新冗余）

但是，如果我们想要的数据出现在两张或者多张表中，对于范式不存在冗余的设计，就不得不采用关联查询。而这恰恰是反范式设计最大的优势，适当的冗余设计，可以减少或避免表关联，提高查询效率。

所以，没有冗余就未必是好的，有时为了提高工作效率（对于查询大于更新的业务），就必须采用反范式的设计，适当的让数据存在冗余。

对于设计范式的理解也可以参考，老齐课堂：

https://mp.weixin.qq.com/s?__biz=Mzg4MzIxMDE2Mg==&mid=2247484210&idx=1&sn=22cb02bdd4847cf5b1ff239d91bfe807&chksm=cf4ba1eff83c28f96656396cb4cc0c179be7e7ce5e2cbd0db062913c8fc1f4a71a6d6f52ed13&token=635438270&lang=zh_CN#rd

4.5.2 宽表与窄表

字面意思理解，宽表就是数据列比较多的表，而窄表刚好相反，数据列比较少。MySQL 对于每张表有 4096 个列的硬限制，而真正在使用上的限制又会取决于你所使用的存储引擎。例如：对于 InnoDB 来说，一张表最多可以有 1017 列。在不考虑 “宽和窄” 的问题上来说，MySQL 和存储引擎支持的列数目肯定是足够的了。

在讲解宽表和窄表的优缺点之前，我这里给出一个定义：以 40 列为界，超过 40 列的表，我们可以称之为宽表，相对的，少于 40 列的表，我们称之为窄表。但同时，需要知道，这里的数字是人为定义的，MySQL 规范中并没有这种定义。我这里的划分是基于工作经验和总结，当然，你也可以有自己的数字界限。那么，不论是宽表还是窄表，它们一定各自都会有相应的优缺点：

- 窄表较多，数据列会更加分散，编写关联查询的难度就会很大。
- 数据项会有不同的安全级别，宽表中涉及的列过多，数据权限的管理会带来很大

的挑战。

- 窄表数据量通常较少，但是等量的数据项会创建更多的表，管理难度大。
- 宽表数据量通常较大，单表占据的存储空间过大，会降低排序、分组等查询的性能。

综上所述，我们应该从多个角度去分析问题，完美的选择几乎是不存在的，我们在拥抱有利之处的时候，也不可避免的会摄入弊端，也正所谓鱼和熊掌不可兼得。

4.5.3 合理应用索引

我们对索引的概念一定不会陌生，它能够加速表数据的查询，但是相应的，它也会占据一定的存储空间，也就是典型的以空间换时间的优化策略。另外，索引的存在，也会使插入、删除、更新的性能降低，因为这些操作都会伴随着索引的修改。所以，这一条设计规范所要追求的是空间与时间的平衡，达到既不占用过多的存储空间，也有较高的查询性能。

索引要建的合理，就必须要知晓并理解 MySQL 中索引创建和使用的特性：

- 一定要为作为搜索条件的字段创建索引，不是搜索条件的字段建索引反而会降低使用性能
- 选择区分度高的字段作为索引字段，重复性高的字段不要加索引
- 联合索引存在 “最左前缀” 的特性，不要建多余的索引。

最后，如果一张表中已经存在了大量的数据，再去创建索引的过程会相当漫长，且可能会影响线上服务。此时，应该评估是否是在原表上增加索引还是创建新表并迁移数据。

5 总结(Summary)

5.1重难点分析

- xxxxxxxxx

5.2 FAQ 分析

- xxxxxxxxx

5.3 参考

- 《高性能 MySQL（第三版）》
- 《MySQL 技术内幕：InnoDB 存储引擎（第 2 版）》
- https://dev.mysql.com/doc/refman/5.7/en/innodb-architecture.html
- https://dev.mysql.com/doc/refman/5.7/en/innodb-storage-engine.html
- https://dev.mysql.com/doc/refman/5.7/en/innodb-in-memory-structures.html
- https://dev.mysql.com/doc/refman/5.7/en/innodb-on-disk-structures.html