如果每个应用程序都各自写自己的读写数据的代码，一方面效率低，容易出错，另一方面，每个应用程序访问数据的接口都不相同，数据难以复用。

所以，数据库作为一种专门管理数据的软件就出现了。应用程序不需要自己管理数据，而是通过数据库软件提供的接口来读写数据。至于数据本身如何存储到文件，那是数据库软件的事情，应用程序自己并不关心。

数结构来组织、存储和管理数据，实际上，数据库一共有三种模型：

层次模型：层次模型就是以“上下级”的层次关系据模型。

数据库按照数据来组织数据的一种方式，层次模型的数据结构看起来就像一颗树。

网状模型：网状模型把每个数据节点和其他很多节点都连接起来，它的数据结构看起来就像很多城市之间的路网。

关系模型：关系模型把数据看作是一个二维表格，任何数据都可以通过行号+列号来唯一确定，它的数据模型看起来就是一个Excel表。

数据类型：对于一个关系表，除了定义每一列的名称外，还需要定义每一列的数据类型。

选择数据类型的时候，要根据业务规则选择合适的类型。

主流关系数据库

目前，主流的关系数据库主要分为以下几类：

商用数据库，例如：Oracle，SQL Server，DB2等；

开源数据库，例如：MySQL，PostgreSQL等；

桌面数据库，以微软Access为代表，适合桌面应用程序使用；

嵌入式数据库，以Sqlite为代表，适合手机应用和桌面程序。

SQL(Structured Query Language)是结构化查询语言的缩写，用来访问和操作数据库系统。

SQL语句既可以查询数据库中的数据，也可以添加、更新和删除数据库中的数据，还可以对数据库进行管理和维护操作。

不同的数据库，都支持SQL。

总的来说，SQL语言定义了这么几种操作数据库的能力：

DDL：Data Definition Language

DDL允许用户定义数据，也就是创建表、删除表、修改表结构这些操作。通常，DDL由数据库管理员执行。

DML：Data Manipulation Language

DML为用户提供添加、删除、更新数据的能力，这些是应用程序对数据库的日常操作。

DQL：Data Query Language

DQL允许用户查询数据，这也是通常最频繁的数据库日常操作。

SQL语言关键字不区分大小写！

使用MySQL时，不同的表还可以使用不同的数据库引擎。

如果你不知道应该采用哪种引擎，记住总是选择InnoDB就好了。

表的每一行称为记录（Record），记录是一个逻辑意义上的数据。

表的每一列称为字段（Column），同一个表的每一行记录都拥有相同的若干字段。

和Excel表有所不同的是，关系数据库的表和表之间需要建立“一对多”，“多对一”和“一对一”的关系，这样才能够按照应用程序的逻辑来组织和存储数据。

在关系数据库中，关系是通过主键和外键来维护的。

主键：

对于关系表，主键有个很重要的约束，就是任意两条记录不能重复。

不能重复不是指两条记录不完全相同，而是指能够通过某个字段唯一区分出不同的记录，这个字段被称为主键。

对主键的要求，最关键的一点是：记录一旦插入到表中，主键最好不要再修改，因为主键是用来唯一定位记录的，修改了主键，会造成一系列的影响。

所以，选取主键的一个基本原则是：不使用任何业务相关的字段作为主键。

作为主键最好是完全业务无关的字段，我们一般把这个字段命名为id。常见的可作为id字段的类型有：

自增整数类型：数据库会在插入数据时自动为每一条记录分配一个自增整数，这样我们就完全不用担心主键重复，也不用自己预先生成主键；

全局唯一GUID类型：使用一种全局唯一的字符串作为主键，类似8f55d96b-8acc-4636-8cb8-76bf8abc2f57。GUID算法通过网卡MAC地址、时间戳和随机数保证任意计算机在任意时间生成的字符串都是不同的，大部分编程语言都内置了GUID算法，可以自己预算出主键。

对于大部分应用来说，通常自增类型的主键就能满足需求。我们在students表中定义的主键也是BIGINT NOT NULL AUTO\_INCREMENT类型。

如果使用INT自增类型，那么当一张表的记录数超过2147483647（约21亿）时，会达到上限而出错。使用BIGINT自增类型则可以最多约922亿亿条记录。

总结：

主键是关系表中记录的唯一标识。

主键的选取非常重要：主键不要带有业务含义，而应该使用BIGINT自增或者GUID类型。

主键也不应该允许NULL。

可以使用多个列作为联合主键，但联合主键并不常用。

外键：

把数据与另一张表关联起来，这种列称为外键。

外键并不是通过列名实现的，而是通过定义外键约束实现的：

ALTER TABLE students

ADD CONSTRAINT fk\_class\_id

FOREIGN KEY (class\_id)

REFERENCES classes (id);

其中，外键约束的名称fk\_class\_id可以任意，FOREIGN KEY (class\_id)指定了class\_id作为外键，REFERENCES classes (id)指定了这个外键将关联到classes表的id列（即classes表的主键）。

通过定义外键约束，关系数据库可以保证无法插入无效的数据。

由于外键约束会降低数据库的性能，大部分互联网应用程序为了追求速度，并不设置外键约束，而是仅靠应用程序自身来保证逻辑的正确性。

这种情况下，class\_id仅仅是一个普通的列，只是它起到了外键的作用而已。

要删除一个外键约束，也是通过ALTER TABLE实现的：

ALTER TABLE students

DROP FOREIGN KEY fk\_class\_id;

通过一个表的外键关联到另一个表，我们可以定义出一对多关系。

多对多关系实际上是通过两个一对多关系实现的，即通过一个中间表，关联两个一对多关系，就形成了多对多关系。

一对一关系是指，一个表的记录对应到另一个表的唯一一个记录。

还有一些应用会把一个大表拆成两个一对一的表，目的是把经常读取和不经常读取的字段分开，以获得更高的性能。

总结：

关系数据库通过外键可以实现一对多、多对多和一对一的关系。

外键既可以通过数据库来约束，也可以不设置约束，仅依靠应用程序的逻辑来保证。

索引：

通过对数据库表创建索引，可以提高查询速度。

通过创建唯一索引，可以保证某一列的值具有唯一性。

数据库索引对于用户和应用程序来说都是透明的。

在关系数据库中，如果有上万甚至上亿条记录，在查找记录的时候，想要获得非常快的速度，就需要使用索引。

索引是关系数据库中对某一列或多个列的值进行预排序的数据结构。通过使用索引，可以让数据库系统不必扫描整个表，而是直接定位到符合条件的记录，这样就大大加快了查询速度。

ALTER TABLE students

ADD INDEX idx\_score (score);

使用ADD INDEX idx\_score (score)就创建了一个名称为idx\_score，使用列score的索引。索引名称是任意的，索引如果有多列，可以在括号里依次写上。

索引的效率取决于索引列的值是否散列，即该列的值如果越互不相同，那么索引效率越高。

可以对一张表创建多个索引。索引的优点是提高了查询效率，缺点是在插入、更新和删除记录时，需要同时修改索引，因此，索引越多，插入、更新和删除记录的速度就越慢。

对于主键，关系数据库会自动对其创建主键索引。使用主键索引的效率是最高的，因为主键会保证绝对唯一。

在设计关系数据表的时候，看上去唯一的列，例如身份证号、邮箱地址等，因为他们具有业务含义，因此不宜作为主键。但是，这些列根据业务要求，又具有唯一性约束：即不能出现两条记录存储了同一个身份证号。这个时候，就可以给该列添加一个唯一索引。

ALTER TABLE students

ADD UNIQUE INDEX uni\_name (name);

通过UNIQUE关键字我们就添加了一个唯一索引。

也可以只对某一列添加一个唯一约束而不创建唯一索引：

ALTER TABLE students

ADD CONSTRAINT uni\_name UNIQUE (name);

这种情况下，name列没有索引，但仍然具有唯一性保证。

无论是否创建索引，对于用户和应用程序来说，使用关系数据库不会有任何区别。这里的意思是说，当我们在数据库中查询时，如果有相应的索引可用，数据库系统就会自动使用索引来提高查询效率，如果没有索引，查询也能正常执行，只是速度会变慢。因此，索引可以在使用数据库的过程中逐步优化。

《查询数据》

基本查询：

SELECT \* FROM <表名>可以查询到一张表的所有记录。

SELECT \* FROM students;

students 是表名。

条件查询：

SELECT语句可以通过WHERE条件来设定查询条件，查询结果是满足查询条件的记录。

SELECT \* FROM <表名> WHERE <条件表达式>

SELECT \* FROM students WHERE score >= 80

条件表达式可以用 <条件1> AND <条件2> 表达满足条件1并且满足条件2。

SELECT \* FROM students WHERE score >= 80 AND gender = 'M';

条件表达式可以用 <条件1> OR <条件2> 表达满足条件1或者满足条件2。

SELECT \* FROM students WHERE score >= 80 OR gender = 'M';

条件表达式可以用 NOT <条件> 表示“不符合该条件”的记录。

SELECT \* FROM students WHERE NOT class\_id = 2;

NOT条件NOT class\_id = 2其实等价于class\_id <> 2，因此，NOT查询不是很常用。

要组合三个或者更多的条件，就需要用小括号()表示如何进行条件运算。

例如，编写一个复杂的条件：分数在80以下或者90以上，并且是男生：

SELECT \* FROM students WHERE (score < 80 OR score > 90) AND gender = 'M';

常用的条件表达式：

条件 表达式举例1 表达式举例2

使用=判断不相等 score = 80 name = 'abc'

使用>判断大于或相等 score > 80 name > 'abc'

使用>=判断大于或相等 score >= 80 name >= 'abc'

使用<判断小于或相等 score < 80 name <= 'abc'

使用<=判断小于或相等 score <= 80 name <= 'abc'

使用<>判断不相等 score <> 80 name <> 'abc'

使用LIKE判断相似 name LIKE 'ab%' name LIKE '%bc%' %表示任意字符，例如'ab%'将匹配'ab'，'abc'，'abcd'

说明：

字符串需要用单引号括起来

字符串比较根据ASCII码，中文字符比较根据数据库设置

投影查询：

如果我们只希望返回某些列的数据，而不是所有列的数据，我们可以用SELECT 列1, 列2, 列3 FROM ...，让结果集仅包含指定列。这种操作称为投影查询。

SELECT 列1, 列2, 列3 FROM ...

例如，从students表中返回id、score和name这三列：

SELECT id, score, name FROM students;

这样返回的结果集就只包含了我们指定的列，并且，结果集的列的顺序和原表可以不一样。

SELECT 列1 别名1, 列2 别名2, 列3 别名3 FROM ...。

例如，以下SELECT语句将列名score重命名为points，而id和name列名保持不变：

SELECT id, score points, name FROM students;

排序查询：

我们使用SELECT查询时，查询结果集通常是按照id排序的，也就是根据主键排序。

这也是大部分数据库的做法。

如果我们要根据其他条件排序怎么办？

可以加上 ORDER BY 子句。

例如按照成绩从低到高进行排序：

SELECT id, name, gender, score FROM students ORDER BY score;

如果要反过来，按照成绩从高到底排序，我们可以加上DESC表示“倒序”：

SELECT id, name, gender, score FROM students ORDER BY score DESC;

如果score列有相同的数据，要进一步排序，可以继续添加列名。

例如，使用ORDER BY score DESC, gender 表示先按score列倒序，如果有相同分数的，再按gender列排序：

SELECT id, name, gender, score FROM students ORDER BY score DESC, gender;

默认的排序规则是ASC：“升序”，即从小到大。ASC可以省略，即ORDER BY score ASC和ORDER BY score效果一样。

如果有WHERE子句，那么 ORDER BY 子句要放到WHERE子句后面。

例如，查询一班的学生成绩，并按照倒序排序：

SELECT id, name, gender, score

FROM students

WHERE class\_id = 1

ORDER BY score DESC;

分页查询：

使用LIMIT <M> OFFSET <N>可以对结果集进行分页，每次查询返回结果集的一部分；

分页查询需要先确定每页的数量和当前页数，然后确定LIMIT和OFFSET的值。

OFFSET是可选的，如果只写LIMIT 15，那么相当于LIMIT 15 OFFSET 0。

在MySQL中，LIMIT 15 OFFSET 30还可以简写成LIMIT 30, 15。

使用LIMIT <M> OFFSET <N>分页时，随着N越来越大，查询效率也会越来越低。

聚合查询：

对于统计总数、平均数这类计算，SQL提供了专门的聚合函数，使用聚合函数进行查询，就是聚合查询，它可以快速获得结果。

以查询students表一共有多少条记录为例，

我们可以使用SQL内置的COUNT()函数查询：

SELECT COUNT(\*) FROM students;

COUNT(\*)表示查询所有列的行数，要注意聚合的计算结果虽然是一个数字，但查询的结果仍然是一个二维表，只是这个二维表只有一行一列，并且列名是COUNT(\*)。

通常，使用聚合查询时，我们应该给列名设置一个别名，便于处理结果：

SELECT COUNT(\*) num FROM students;

COUNT(\*)和COUNT(id)实际上是一样的效果。

另外注意，聚合查询同样可以使用WHERE条件，

因此我们可以方便地统计出有多少男生、多少女生、多少80分以上的学生等：

SELECT COUNT(\*) boys FROM students WHERE gender = 'M';

函数 说明

SUM 计算某一列的合计值，该列必须为数值类型

AVG 计算某一列的平均值，该列必须为数值类型

MAX 计算某一列的最大值

MIN 计算某一列的最小值

注意，MAX()和MIN()函数并不限于数值类型。如果是字符类型，MAX()和MIN()会返回排序最后和排序最前的字符。

使用聚合查询计算男生平均成绩:

SELECT AVG(score) average FROM students WHERE gender = 'M';

如果聚合查询的WHERE条件没有匹配到任何行，

COUNT()会返回0，而SUM()、AVG()、MAX()和MIN()会返回NULL：

SELECT AVG(score) average FROM students WHERE gender = 'X';

分组：

对于聚合查询，SQL还提供了“分组聚合”的功能。

按class\_id分组:

SELECT COUNT(\*) num FROM students GROUP BY class\_id;

GROUP BY子句指定了按class\_id分组，因此，执行该SELECT语句时，会把class\_id相同的列先分组，再分别计算。

聚合查询的列中，只能放入分组的列。

用多个列进行分组。

SELECT class\_id, gender, COUNT(\*) num

FROM students GROUP BY class\_id, gender;

多表查询：

SELECT查询不但可以从一张表查询数据，还可以从多张表同时查询数据。

查询多张表的语法是：SELECT \* FROM <表1> <表2>。

SELECT \* FROM students, classes;

这种一次查询两个表的数据，查询的结果也是一个二维表，它是students表和classes表的“乘积”，即students表的每一行与classes表的每一行都两两拼在一起返回。

结果集的列数是students表和classes表的列数之和，行数是students表和classes表的行数之积。

这种多表查询又称笛卡尔查询，使用笛卡尔查询时要非常小心，由于结果集是目标表的行数乘积，对两个各自有100行记录的表进行笛卡尔查询将返回1万条记录，对两个各自有1万行记录的表进行笛卡尔查询将返回1亿条记录。

利用投影查询的“设置列的别名”来给两个表各自的id和name列起别名：

SELECT

students.id sid,

students.name,

students.gender,

students.score,

classes.id cid,

classes.name cname

FROM students, classes;

FROM子句给表设置别名的语法是FROM <表名1> <别名1>, <表名2> <别名2>。

多表查询也是可以添加WHERE条件的：

SELECT

s.id sid,

s.name,

s.gender,

s.score,

c.id cid,

c.name cname

FROM students s, classes c

WHERE s.gender = 'M' AND c.id = 1；

连接查询：

连接查询是另一种类型的多表查询。

连接查询对多个表进行JOIN运算，简单地说，就是先确定一个主表作为结果集，然后，把其他表的行有选择性地“连接”在主表结果集上。

使用最常用的一种内连接——INNER JOIN来实现：

SELECT s.id, s.name, s.class\_id, c.name class\_name, s.gender, s.score

FROM students s

INNER JOIN classes c

ON s.class\_id = c.id;

INNER JOIN查询的写法是：

1. 先确定主表，仍然使用FROM <表1>的语法；

2. 再确定需要连接的表，使用INNER JOIN <表2>的语法；

3. 然后确定连接条件，使用ON <条件...>，这里的条件是s.class\_id = c.id，表示students表的class\_id列与classes表的id列相同的行需要连接；

4. 可选：加上WHERE子句、ORDER BY等子句。

外连接（OUTER JOIN）

RIGHT OUTER JOIN返回右表都存在的行。

LEFT OUTER JOIN则返回左表都存在的行。

FULL OUTER JOIN，它会把两张表的所有记录全部选择出来，并且，自动把对方不存在的列填充为NULL。

《修改数据》

关系数据库的基本操作就是增删改查，即CRUD：Create、Retrieve、Update、Delete。其中，对于查询，我们已经详细讲述了SELECT语句的详细用法。

而对于增、删、改，对应的SQL语句分别是：

INSERT：插入新记录： INSERT INTO <表名> (字段1, 字段2, ...) VALUES (值1, 值2, ...);

注意到我们并没有列出id字段，也没有列出id字段对应的值，这是因为id字段是一个自增主键，它的值可以由数据库自己推算出来。

此外，如果一个字段有默认值，那么在INSERT语句中也可以不出现。

字段顺序不必和数据库表的字段顺序一致，但值的顺序必须和字段顺序一致。

还可以一次性添加多条记录，只需要在VALUES子句中指定多个记录值，每个记录是由(...)包含的一组值。

UPDATE：更新已有记录： UPDATE <表名> SET 字段1=值1, 字段2=值2, ... WHERE ...;

在UPDATE语句中，更新字段时可以使用表达式。

如果WHERE条件没有匹配到任何记录，UPDATE语句不会报错，也不会有任何记录被更新。

要特别小心的是，UPDATE语句可以没有WHERE条件，

在执行UPDATE语句时要非常小心，最好先用SELECT语句来测试WHERE条件是否筛选出了期望的记录集，然后再用UPDATE更新。

DELETE：删除已有记录： DELETE FROM <表名> WHERE ...;

注意到DELETE语句的WHERE条件也是用来筛选需要删除的行，因此和UPDATE类似，DELETE语句也可以一次删除多条记录。

如果WHERE条件没有匹配到任何记录，DELETE语句不会报错，也不会有任何记录被删除。

要特别小心的是，和UPDATE类似，不带WHERE条件的DELETE语句会删除整个表的数据。

在执行DELETE语句时也要非常小心，最好先用SELECT语句来测试WHERE条件是否筛选出了期望的记录集，然后再用DELETE删除。

MySQL:

安装完MySQL后，除了MySQL Server，即真正的MySQL服务器外，还附赠一个MySQL Client程序。

MySQL Client是一个命令行客户端，可以通过MySQL Client登录MySQL，然后，输入SQL语句并执行。

打开命令提示符，输入命令mysql -u root -p，提示输入口令。

填入MySQL的root口令，如果正确，就连上了MySQL Server，同时提示符变为mysql>

输入exit断开与MySQL Server的连接并返回到命令提示符。

MySQL Client的可执行程序是mysql，MySQL Server的可执行程序是mysqld。

在MySQL Client中输入的SQL语句通过TCP连接发送到MySQL Server。

默认端口号是3306，即如果发送到本机MySQL Server，地址就是127.0.0.1:3306。

也可以只安装MySQL Client，然后连接到远程MySQL Server。

假设远程MySQL Server的IP地址是10.0.1.99，那么就使用-h指定IP或域名：

mysql -h 10.0.1.99 -u root -p

管理MySQL

要管理MySQL，可以使用可视化图形界面MySQL Workbench。

MySQL Workbench可以用可视化的方式查询、创建和修改数据库表，

但是，归根到底，MySQL Workbench是一个图形客户端，它对MySQL的操作仍然是发送SQL语句并执行。

因此，本质上，MySQL Workbench和MySQL Client命令行都是客户端，和MySQL交互，唯一的接口就是SQL。

数据库：

在一个运行MySQL的服务器上，实际上可以创建多个数据库（Database）。

要列出所有数据库，使用命令：

SHOW DATABASES;

e.g.:

mysql> SHOW DATABASES;

+--------------------+

| Database |

+--------------------+

| information\_schema |

| mysql |

| performance\_schema |

| shici |

| sys |

| test |

| school |

+--------------------+

其中，information\_schema、mysql、performance\_schema和sys是系统库，不要去改动它们。其他的是用户创建的数据库。

要创建一个新数据库，使用命令：

CREATE DATABASE test;

要删除一个数据库，使用命令：

DROP DATABASE test;

注意：删除一个数据库将导致该数据库的所有表全部被删除。

对一个数据库进行操作时，要首先将其切换为当前数据库：

USE test;

表

列出当前数据库的所有表，使用命令：

SHOW TABLES;

e.g.:

mysql> SHOW TABLES;

+---------------------+

| Tables\_in\_test |

+---------------------+

| classes |

| statistics |

| students |

| students\_of\_class1 |

+---------------------+

要查看一个表的结构，使用命令：

DESC <表名>

e.g.:

mysql> DESC students;

+----------+--------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+----------+--------------+------+-----+---------+----------------+

| id | bigint(20) | NO | PRI | NULL | auto\_increment |

| class\_id | bigint(20) | NO | | NULL | |

| name | varchar(100) | NO | | NULL | |

| gender | varchar(1) | NO | | NULL | |

| score | int(11) | NO | | NULL | |

+----------+--------------+------+-----+---------+----------------+

还可以使用以下命令查看创建表的SQL语句：

SHOW CREATE TABLE <表名>；

e.g.:

mysql> SHOW CREATE TABLE students;

+----------+-------------------------------------------------------+

| students | CREATE TABLE `students` ( |

| | `id` bigint(20) NOT NULL AUTO\_INCREMENT, |

| | `class\_id` bigint(20) NOT NULL, |

| | `name` varchar(100) NOT NULL, |

| | `gender` varchar(1) NOT NULL, |

| | `score` int(11) NOT NULL, |

| | PRIMARY KEY (`id`) |

| | ) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8 |

+----------+-------------------------------------------------------+

创建表使用CREATE TABLE语句，而删除表使用DROP TABLE语句。

修改表：

ALTER TABLE students ADD COLUMN birth VARCHAR(10) NOT NULL； 要给students表新增一列birth。

ALTER TABLE students CHANGE COLUMN birth birthday VARCHAR(20) NOT NULL； 要修改birth列，例如把列名改为birthday，类型改为VARCHAR(20)。

ALTER TABLE students DROP COLUMN birthday； 删除列。

退出MySQL： EXIT。

实用语句：

插入或替换

如果我们希望插入一条新记录（INSERT），但如果记录已经存在，就先删除原记录，再插入新记录。

此时，可以使用REPLACE语句，这样就不必先查询，再决定是否先删除再插入：

REPLACE INTO students (id, class\_id, name, gender, score) VALUES (1, 1, '小明', 'F', 99);

插入或更新

如果我们希望插入一条新记录（INSERT），但如果记录已经存在，就更新该记录。

此时，可以使用INSERT INTO ... ON DUPLICATE KEY UPDATE ...语句：

INSERT INTO students (id, class\_id, name, gender, score) VALUES (1, 1, '小明', 'F', 99) ON DUPLICATE KEY UPDATE name='小明', gender='F', score=99;

插入或忽略

如果我们希望插入一条新记录（INSERT），但如果记录已经存在，就啥事也不干直接忽略，此时，可以使用INSERT IGNORE INTO ...语句：

INSERT IGNORE INTO students (id, class\_id, name, gender, score) VALUES (1, 1, '小明', 'F', 99);

快照

如果想要对一个表进行快照，即复制一份当前表的数据到一个新表，可以结合CREATE TABLE和SELECT：

CREATE TABLE students\_of\_class1 SELECT \* FROM students WHERE class\_id=1;

写入查询结果集

如果查询结果集需要写入到表中，可以结合INSERT和SELECT，将SELECT语句的结果集直接插入到指定表中。

例如，创建一个统计成绩的表statistics，记录各班的平均成绩：

CREATE TABLE statistics (

id BIGINT NOT NULL AUTO\_INCREMENT,

class\_id BIGINT NOT NULL,

average DOUBLE NOT NULL,

PRIMARY KEY (id)

);

然后，我们就可以用一条语句写入各班的平均成绩：

INSERT INTO statistics (class\_id, average) SELECT class\_id, AVG(score) FROM students GROUP BY class\_id;

> SELECT \* FROM statistics;

+----+----------+--------------+

| id | class\_id | average |

+----+----------+--------------+

| 1 | 1 | 86.5 |

| 2 | 2 | 73.666666666 |

| 3 | 3 | 88.333333333 |

+----+----------+--------------+

强制使用指定索引

在查询的时候，数据库系统会自动分析查询语句，并选择一个最合适的索引。

但是很多时候，数据库系统的查询优化器并不一定总是能使用最优索引。

如果我们知道如何选择索引，可以使用FORCE INDEX强制查询使用指定的索引。

SELECT \* FROM student FOCRE INDEX (idx\_class\_id) WHERE class\_id = 1 ORDER BY id DESC;

事务

在执行SQL语句的时候，某些业务要求，一系列操作必须全部执行，而不能仅执行一部分。

例如，一个转账操作：

-- 从id=1的账户给id=2的账户转账100元

-- 第一步：将id=1的A账户余额减去100

UPDATE accounts SET balance = balance - 100 WHERE id = 1;

-- 第二步：将id=2的B账户余额加上100

UPDATE accounts SET balance = balance + 100 WHERE id = 2;

这两条SQL语句必须全部执行，或者，由于某些原因，如果第一条语句成功，第二条语句失败，就必须全部撤销。

把多条语句作为一个整体进行操作的功能，被称为数据库事务。

数据库事务可以确保该事务范围内的所有操作都可以全部成功或者全部失败。

如果事务失败，那么效果就和没有执行这些SQL一样，不会对数据库数据有任何改动。

可见，数据库事务具有ACID这4个特性：

A：Atomic，原子性，将所有SQL作为原子工作单元执行，要么全部执行，要么全部不执行；

C：Consistent，一致性，事务完成后，所有数据的状态都是一致的，即A账户只要减去了100，B账户则必定加上了100；

I：Isolation，隔离性，如果有多个事务并发执行，每个事务作出的修改必须与其他事务隔离；

D：Duration，持久性，即事务完成后，对数据库数据的修改被持久化存储。

数据库事务具有ACID特性，用来保证多条SQL的全部执行。

对于单条SQL语句，数据库系统自动将其作为一个事务执行，这种事务被称为隐式事务。

要手动把多条SQL语句作为一个事务执行，使用BEGIN开启一个事务，使用COMMIT提交一个事务，这种事务被称为显式事务。

例如，把上述的转账操作作为一个显式事务：

BEGIN;

UPDATE accounts SET balance = balance - 100 WHERE id = 1;

UPDATE accounts SET balance = balance + 100 WHERE id = 2;

COMMIT;

很显然多条SQL语句要想作为一个事务执行，就必须使用显式事务。

COMMIT是指提交事务，即试图把事务内的所有SQL所做的修改永久保存。

如果COMMIT语句执行失败了，整个事务也会失败。

有些时候，我们希望主动让事务失败，这时，可以用ROLLBACK回滚事务，整个事务会失败：

BEGIN;

UPDATE accounts SET balance = balance - 100 WHERE id = 1;

UPDATE accounts SET balance = balance + 100 WHERE id = 2;

ROLLBACK;

隔离级别

对于两个并发执行的事务，如果涉及到操作同一条记录的时候，可能会发生问题。

因为并发操作会带来数据的不一致性，包括脏读、不可重复读、幻读等。

数据库系统提供了隔离级别来让我们有针对性地选择事务的隔离级别，避免数据不一致的问题。

SQL标准定义了4种隔离级别，分别对应可能出现的数据不一致的情况：

Isolation Level 脏读（Dirty Read） 不可重复读（Non Repeatable Read） 幻读（Phantom Read）

Read Uncommitted Yes Yes Yes

Read Committed - Yes Yes

Repeatable Read - - Yes

Serializable - - -

Read Uncommitted是隔离级别最低的一种事务级别。

在这种隔离级别下，一个事务会读到另一个事务更新后但未提交的数据，如果另一个事务回滚，那么当前事务读到的数据就是脏数据，这就是脏读（Dirty Read）。

在Read Committed隔离级别下，一个事务可能会遇到不可重复读（Non Repeatable Read）的问题。

不可重复读是指，在一个事务内，多次读同一数据，在这个事务还没有结束时，如果另一个事务恰好修改了这个数据，那么，在第一个事务中，两次读取的数据就可能不一致。

在Read Committed隔离级别下，事务不可重复读同一条记录，因为很可能读到的结果不一致。

在Repeatable Read隔离级别下，一个事务可能会遇到幻读（Phantom Read）的问题。

幻读是指，在一个事务中，第一次查询某条记录，发现没有，但是，当试图更新这条不存在的记录时，竟然能成功，并且，再次读取同一条记录，它就神奇地出现了。

幻读就是没有读到的记录，以为不存在，但其实是可以更新成功的，并且，更新成功后，再次读取，就出现了。

Serializable是最严格的隔离级别。在Serializable隔离级别下，所有事务按照次序依次执行，因此，脏读、不可重复读、幻读都不会出现。

虽然Serializable隔离级别下的事务具有最高的安全性，但是，由于事务是串行执行，所以效率会大大下降，应用程序的性能会急剧降低。如果没有特别重要的情景，一般都不会使用Serializable隔离级别。

Serializable是最严格的隔离级别。

在Serializable隔离级别下，所有事务按照次序依次执行，因此，脏读、不可重复读、幻读都不会出现。

虽然Serializable隔离级别下的事务具有最高的安全性，但是，由于事务是串行执行，所以效率会大大下降，应用程序的性能会急剧降低。

如果没有特别重要的情景，一般都不会使用Serializable隔离级别。

默认隔离级别

如果没有指定隔离级别，数据库就会使用默认的隔离级别。在MySQL中，如果使用InnoDB，默认的隔离级别是Repeatable Read。