关系模型和SQL

为了介绍关系模型，以MySQL数据库为例。

安装

MariaDB 安装

```
# yum list | grep mariadb
mariadb-libs.x86_64                     1:5.5.60-1.el7_5           @anaconda
mariadb.x86_64                          1:5.5.60-1.el7_5           base
mariadb-bench.x86_64                    1:5.5.60-1.el7_5           base
mariadb-devel.i686                      1:5.5.60-1.el7_5           base
mariadb-devel.x86_64                    1:5.5.60-1.el7_5           base
mariadb-embedded.i686                   1:5.5.60-1.el7_5           base
mariadb-embedded.x86_64                 1:5.5.60-1.el7_5           base
mariadb-embedded-devel.i686             1:5.5.60-1.el7_5           base
mariadb-embedded-devel.x86_64           1:5.5.60-1.el7_5           base
mariadb-libs.i686                       1:5.5.60-1.el7_5           base
mariadb-server.x86_64                   1:5.5.60-1.el7_5           base
mariadb-test.x86_64                     1:5.5.60-1.el7_5           base

安装mariadb 服务，会自动安装mairadb
# yum install mariadb-server

# systemctl start mariadb.service
# ss -tanl
State       Recv-Q Send-Q Local Address:Port               Peer Address:Port
LISTEN      0      50                 *:3306                        *:*

开机启动
# systemctl enable mariadb.service

为了安全设置Mysql服务
# mysql_secure_installation

数据库密码登录
# mysql -u root -p
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
3 rows in set (0.00 sec)

# 创建并授权用户
mysql> grant all on *.* to 'wayne'@'%' identified by 'wayne';
```

```
mysql> flush privileges;
```

导入测试脚本

```
# mysql -u root -p < test.sql
```

SQL语句

SQL是结构化查询语言Structured Query Language。1987年被ISO组织标准化。
所有主流的关系型数据库都支持SQL，NoSQL也有很大一部分支持SQL。

SQL语句分为

- DDL数据定义语言，负责数据库定义、数据库对象定义，由CREATE、ALTER与DROP三个语法所组成
- DML数据操作语言，负责对数据库对象的操作，CRUD增删改查
- DCL数据控制语言，负责数据库权限访问控制，由 GRANT 和 REVOKE 两个指令组成
- TCL事务控制语言，负责处理ACID事务，支持commit、rollback指令

语言规范

- SQL语句大小写不敏感
  - 一般建议，SQL的关键字、函数等大写
- SQL语句末尾应该使用分号结束
- 注释
  - 多行注释 `/*注释内容*/`
  - 单行注释 `--注释内容`
  - MySQL 注释可以使用#
- 使用空格或缩进来提高可读性
- 命名规范
  - 必须以字母开头
  - 可以使用数字、#、$和_
  - 不可使用关键字

DCL

GRANT授权、REVOKE撤销

```
GRANT ALL ON employees.* TO 'wayne'@'%' IDENTIFIED by 'wayne';
REVOKE ALL ON *.* FROM wayne;
```

* 为通配符，指代任意库或者任意表。`*.*`所有库的所有表；`employees.*`表示employees库下所有的表
% 为通配符，它是SQL语句的通配符，匹配任意长度字符串

DDL

删除用户（慎用）

```
DROP USER wayne;
```

创建数据库
库是数据的集合，所有数据按照数据模型组织在数据库中。

```
CREATE DATABASE IF NOT EXISTS test CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;
CREATE DATABASE IF NOT EXISTS test CHARACTER SET utf8;
```

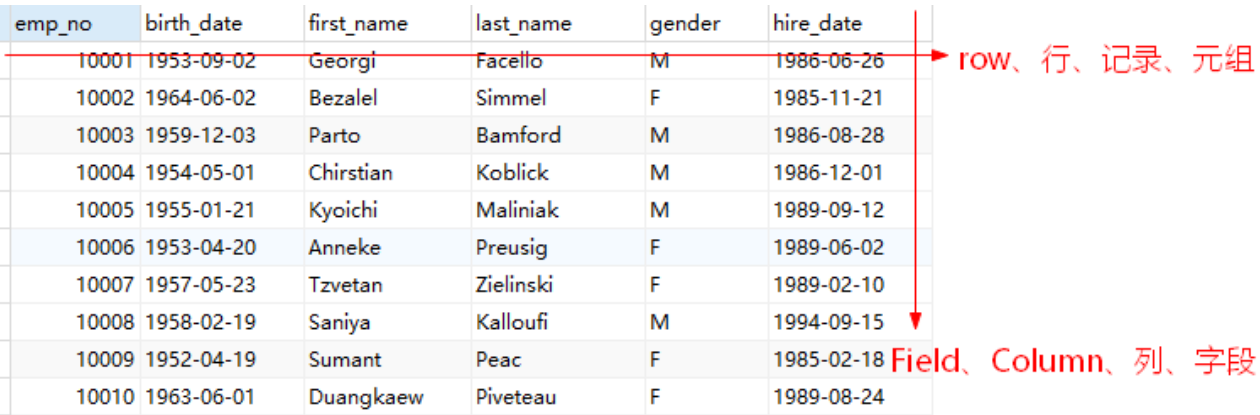
CHARACTER SET指定字符集。
utf8mb4是utf8的扩展，支持4字节utf8mb4，需要MySQL5.5.3+。
COLLATE指定字符集的校对规则，用来做字符串的比较的。例如a、A谁大?

删除数据库

```
DROP DATABASE IF EXISTS gogs;
```

创建表
表分为行和列，MySQL是行存数据库。数据是一行行存的，列必须固定多少列。
行Row，也称为记录Record，元组。
列Column，也称为字段Field、属性。
字段的取值范围叫做 域Domain。例如gender字段的取值就是M或者F两个值。

emp_no	birth_date	first_name	last_name	gender	hire_date
10001	1953-09-02	Georgi	Facello	M	1986-06-26
10002	1964-06-02	Bezalel	Simmel	F	1985-11-21
10003	1959-12-03	Parto	Bamford	M	1986-08-28
10004	1954-05-01	Chirstian	Koblick	M	1986-12-01
10005	1955-01-21	Kyoichi	Maliniak	M	1989-09-12
10006	1953-04-20	Anneke	Preusig	F	1989-06-02
10007	1957-05-23	Tzvetan	Zielinski	F	1989-02-10
10008	1958-02-19	Saniya	Kalloufi	M	1994-09-15
10009	1952-04-19	Sumant	Peac	F	1985-02-18
10010	1963-06-01	Duangkaew	Piveteau	F	1989-08-24



```
CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  PRIMARY KEY (`emp_no`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

反引号标注的名称，会被认为是非关键字，使用反引号避免冲突。

DESC
查看列信息
{DESCRIBE | DESC} tbl_name [col_name | wild]

```
DESC employees;
DESC employees '%name';
```

练习
设计一张表，记录登录账户的，应该存储用户的姓名、登录名、密码

```
DROP DATABASE IF EXISTS test;
CREATE DATABASE IF NOT EXISTS test CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;
```

```
CREATE TABLE `reg` (
  `id` int(11) NOT NULL,
  `loginname` varchar(50) NOT NULL,
  `name` varchar(64) DEFAULT NULL,
  `password` varchar(128) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;
```

关系

在关系数据库中，关系就是二维表，由行和列组成。

行Row，也称为记录Record，元组。
列Column，也称为字段Field、属性。
字段的取值范围叫做 域Domain。例如gender字段的取值就是M或者F两个值。

维数：关系的维数指关系中属性的个数
基数：元组的个数

注意在关系中，属性的顺序并不重要。理论上，元组顺序也不重要，但是由于元组顺序与存储相关，会影响查询效率。

候选键

关系中，能唯一标识一条元组的属性或属性集合，称为候选键。

PRIMARY KEY主键

表中一列或者多列组成唯一的key，也就是通过这一个或者多个列能唯一的标识一条记录。即被选择的候选键。
主键的列不能包含空值null。主键往往设置为整型、长整型，且自增AUTO_INCREMENT。
表中可以没有主键，但是，一般表设计中，往往都会有主键，以避免记录重复。

Foreign KEY外键

严格来说，当一个关系中的某个属性或属性集合与另一个关系（也可以是自身）的候选键匹配时，就称作这个属性或属性集合是外键。

索引Index

可以看做是一本字典的目录，为了快速检索用的。空间换时间，显著提高查询效率。
可以对一列或者多列字段设定索引。

主键索引，主键会自动建立主键索引，主键本身就是为了快速定位唯一记录的。
唯一索引，表中的索引列组成的索引必须唯一，但可以为空，非空值必须唯一
普通索引，没有唯一性的要求，就是建了一个字典的目录而已。

约束Constraint

为了保证数据的完整正确，数据模型还必须支持完整性约束。

必须有值约束
某些列的值必须有值，不许为空NULL。

域约束Domain Constraint

限定了表中字段的取值范围

实体完整性Entity Integrity

PRIMARY KEY约束定义了主键，就定义了**主键约束**。主键不重复且唯一，不能为空。

引用完整性Referential Integrity ***

外键定义中，可以不是引用另一张表的主键，但是，往往实际只会关注引用主键。

外键：在表B中的列，引用了表A中的主键，表B中的列就是外键。

A表称为主表，B表称为从表。

插入规则
不需要指定。
如果在表B插入一条记录，B的外键列插入了一个值，这个值必须是表A中存在的主键值。

更新规则
定义外键约束时指定该规则。

删除规则
定义外键约束时指定该规则。

外键约束的操作

设定值	说明
CASCADE	级联，从父表删除或更新会自动删除或更新子表中匹配的行
SET NULL	从父表删除或更新行，会设置子表中的外键列为NULL，但必须保证子表列没有指定NOT NULL，也就是说子表的字段可以为NULL才行
RESTRICT	如果从父表删除主键，如果子表引用了，则拒绝对父表的删除或更新操作
NO ACTION	标准SQL的关键字，在MySQL中与RESTRICT相同。拒绝对父表的删除或更新操作

外键约束，是为了保证数据完整性、一致性，杜绝数冗余、数据错误。

实体-联系E-R

数据库建立，需要收集用 户需求，设计符合企业要求的数据模型。而构建这种模型需要方法，这种方法需要成为E-R实体-联系建模。也出现了一种建模语言——UML（Unified Modeling Language）统一建模语言。

实体Entity：现实世界中具有相同属性的一组对象，可以是物理存在的事物或抽象的事物。

联系Relationship：实体之间的关联集合。

实体间联系类型

假设有实体部门，实体员工

类型	描述	解决方案
一对多联系 1:n	一个员工属于一个部门，一个部门有多个员工	员工外键；部门主键
多对多联系 m:n	一个员工属于多个部门，一个部门有多个员工	建立第三表
一对一联系 1:1	假设有实体管理者，一个管理者管理一个部门，一个部门只有一个管理者	字段建在哪张表都行

一对一关系用的较少，往往表示表A的一条记录唯一关联表B的一条记录，反之亦然。
它往往是为了将一张表多列分割并产生成了多张表，合起来是完整的信息，或为了方便查询，或为了数据安全隔离一部分字段的数据等等。

视图

视图，也称虚表，看起来像表。它是由查询语句生成的。可以通过视图进行CRUD操作。

视图的作用

- 简化操作，将复杂查询SQL语句定义为视图，可以简化查询。
- 数据安全，视图可以只显示真实表的部分列，或计算后的结果，从而隐藏真实表的数据

数据类型

MySQL中的数据类型

类型	含义
tinyint	1字节，带符号的范围是-128到127。无符号的范围是0到255。bool或boolean，就是tinyint，0表示假，非0表示真
smallint	2字节，带符号的范围是-32768到32767。无符号的范围是0到65535
int	整型，4字节，同Integer，带符号的范围是-2147483648到2147483647。无符号的范围是0到4294967295
bigint	长整型，8字节，带符号的范围是-92233720368547758O8到9223372036854775807。无符号的范围是0到18446744073709551615
float	单精度浮点数精确到大约7位小数位
double	双精度浮点数精确到大约15位小数位
DATE	日期。支持的范围为'1000-01-01'到'9999-12-31'
DATETIME	支持的范围是'1000-01-01 00:00:00'到'9999-12-31 23:59:59'
TIMESTAMP	时间戳。范围是'1970-01-01 00:00:00'到2037年
char(M)	固定长度，右边填充空格以达到长度要求。M为长度，范围为0~255。M指的是字符个数
varchar(M)	变长字符串。M 表示最大列长度。M的范围是0到65,535。但不能突破行最大字节数65535
text	大文本。最大长度为65535(2^16-1)个字符
BLOB	大字节。最大长度为65535(2^16–1)字节的BLOB列

LENGTH函数返回字节数。而char和varchar定义的M是字符数限制。
char可以将字符串变成等长的，空间换时间，效率略高；varchar变长，省了空间。

关系操作

关系：在关系数据库中，关系就是二维表。
关系操作就是对表的操作。

选择（selection）：又称为限制，是从关系中选择出满足给定条件的元组。
投影（projection）：在关系上投影就是从选择出若干属性列组成新的关系。
连接（join）：将不同的两个关系连接成一个关系。

DML —— CRUD 增删改查

Insert语句

```
INSERT INTO table_name (col_name,...) VALUES (value1,...);
-- 向表中插入一行数据，自增字段、缺省值字段、可为空字段可以不写

INSERT INTO table_name SELECT ... ;
-- 将select查询的结果插入到表中

INSERT INTO table_name (col_name1,...) VALUES (value1,...) ON DUPLICATE KEY UPDATE
col_name1=value1,...;
-- 如果主键冲突、唯一键冲突就执行update后的设置。这条语句的意思，就是主键不在新增记录，主键在就更新部分字段。

INSERT IGNORE INTO table_name (col_name,...) VALUES (value1,...);
-- 如果主键冲突、唯一键冲突就忽略错误，返回一个警告。
```

```
INSERT INTO reg (loginname, `name`, `password`) VALUES ('tom', 'tom', 'tom');
INSERT INTO reg (id, loginname, `name`, `password`) VALUES (5, 'tom', 'tom', 'tom');
INSERT INTO reg (id, loginname, `name`, `password`) VALUES (1, 'tom', 'tom', 'tom') ON DUPLICATE
KEY UPDATE name = 'jerry';
```

Update语句

```
UPDATE [IGNORE] tbl_name SET col_name1=expr1 [, col_name2=expr2 ...] [WHERE where_definition]
-- IGNORE 意义同Insert语句

UPDATE reg SET name='张三' WHERE id=5;
```

```
-- 注意这一句非常危险，会更新所有数据
UPDATE reg SET name = 'ben';

-- 更新一定要加条件
UPDATE reg SET name = 'ben', password = 'benpwd' WHERE id = 1;
```

Delete语句

```
DELETE FROM tbl_name [WHERE where_definition]
-- 删除符合条件的记录
```

```
-- 删除一定要有条件
DELETE FROM reg WHERE id = 1;
```

Select语句

```
SELECT
    [DISTINCT]
    select_expr, ...
    [FROM table_references
    [WHERE where_definition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_definition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC] , ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

FOR UPDATE会把行进行写锁定，这是排它锁。

查询
查询的结果成为结果集recordset。

```
SELECT 1;

-- 最简单的查询
SELECT * FROM employees;
-- 字符串合并
SELECT emp_no, first_name + last_name FROM employees;
SELECT emp_no, CONCAT(first_name,' ',last_name) FROM employees;

-- AS 定义别名，可选。写AS是一个好习惯
SELECT emp_no as `no`, CONCAT(first_name,' ',last_name) name FROM employees emp;
```

Limit子句

```
-- 返回5条记录
SELECT * FROM employees emp LIMIT 5;

-- 返回5条记录，偏移18条
SELECT * FROM employees emp LIMIT 5 OFFSET 18;
SELECT * FROM employees emp LIMIT 18, 5;
```

Where子句

运算符	描述
=	等于
<>	不等于
>、<、>=、<=	大于、小于、大于等于、小于等于
BETWEEN	在某个范围之内，between a and b等价于[a, b]
LIKE	字符串模式匹配，%表示任意多个字符，_表示一个字符
IN	指定针对某个列的多个可能值
AND	与
OR	或

注意：如果很多表达式需要使用AND、OR计算逻辑表达式的值的时候，由于有结合律的问题，建议使用小括号来避免产生错误

```
-- 条件查询
SELECT * FROM employees WHERE emp_no < 10015 and last_name LIKE 'P%';
SELECT * FROM employees WHERE emp_no BETWEEN 10010 AND 10015 AND last_name LIKE 'P%';
SELECT * FROM employees WHERE emp_no in (10001, 10002, 10010);
```

Order by子句
对查询结果进行排序，可以升序ASC、降序DESC。

```
-- 降序
SELECT * FROM employees WHERE emp_no in (10001, 10002, 10010) ORDER BY emp_no DESC;
```

DISTINCT
不返回重复记录

```
-- DISTINCT使用
SELECT DISTINCT dept_no from dept_emp;
SELECT DISTINCT emp_no from dept_emp;
SELECT DISTINCT dept_no, emp_no from dept_emp;
```

聚合函数

函数	描述
COUNT(expr)	返回记录中记录的数目，如果指定列，则返回非NULL值的行数
COUNT(DISTINCT expr,[expr...])	返回不重复的非NULL值的行数
AVG([DISTINCT] expr)	返回平均值，返回不同值的平均值
MIN(expr), MAX(expr)	最小值，最大值
SUM([DISTINCT] expr)	求和，Distinct返回不同值求和

```
-- 聚合函数
SELECT COUNT(*), AVG(emp_no), SUM(emp_no), MIN(emp_no), MAX(emp_no) FROM employees;
```

分组查询
使用Group by子句，如果有条件，使用Having子句过滤分组、聚合过的结果。

```
-- 聚合所有
SELECT emp_no, SUM(salary), AVG(salary), COUNT(emp_no) from salaries;
-- 聚合被选择的记录
SELECT emp_no, SUM(salary), AVG(salary), COUNT(emp_no) from salaries WHERE emp_no < 10003;

-- 分组
SELECT emp_no FROM salaries GROUP BY emp_no;
SELECT emp_no FROM salaries WHERE emp_no < 10003 GROUP BY emp_no;

-- 按照不同emp_no分组，每组分别聚合
SELECT emp_no, SUM(salary), AVG(salary), COUNT(emp_no) from salaries WHERE emp_no < 10003 GROUP
BY emp_no;

-- HAVING子句对分组结果过滤
SELECT emp_no, SUM(salary), AVG(salary), COUNT(emp_no) from salaries GROUP BY emp_no HAVING
AVG(salary) > 45000;
-- 使用别名
SELECT emp_no, SUM(salary), AVG(salary) AS sal_avg, COUNT(emp_no) from salaries GROUP BY emp_no
HAVING sal_avg > 60000;

-- 最后对分组过滤后的结果排序
SELECT emp_no, SUM(salary), AVG(salary) AS sal_avg, COUNT(emp_no) from salaries GROUP BY emp_no
HAVING sal_avg > 60000 ORDER BY sal_avg;
```

子查询
查询语句可以嵌套，内部查询就是子查询。
子查询必须在一组小括号中。
子查询中不能使用Order by。

```sql
-- 子查询
SELECT * FROM employees WHERE emp_no in (SELECT emp_no from employees WHERE emp_no > 10015)
ORDER BY emp_no DESC;

SELECT emp.emp_no, emp.first_name, gender FROM (SELECT * from employees WHERE emp_no > 10015) AS
emp WHERE emp.emp_no < 10019 ORDER BY emp_no DESC;
```

连接Join
交叉连接cross join
笛卡尔乘积，全部交叉
在MySQL中，CROSS JOIN从语法上说与INNER JOIN等同

```sql
-- 工资40行
SELECT * FROM salaries;
-- 20行
SELECT * FROM employees;
-- 800行
SELECT * from employees CROSS JOIN salaries;
-- 隐式连接，800行
SELECT * FROM employees, salaries;
```

注意：salaries和employees并没有直接的关系，做笛卡尔乘积只是为了看的清楚

内连接
inner join，省略为join。
等值连接，只选某些field相等的元组（行），使用On限定关联的结果
自然连接，特殊的等值连接，会去掉重复的列。用的少。

```sql
-- 内连接，笛卡尔乘积 800行
SELECT * from employees JOIN salaries;
SELECT * from employees INNER JOIN salaries;

-- ON等值连接 40行
SELECT * from employees JOIN salaries ON employees.emp_no = salaries.emp_no;

-- 自然连接，去掉了重复列，且自行使用employees.emp_no = salaries.emp_no的条件
SELECT * from employees NATURAL JOIN salaries;
```

外连接
outer join，可以省略为join
分为左外连接，即左连接；右外连接，即右连接；全外连接

```sql
-- 左连接
SELECT * from employees LEFT JOIN salaries ON employees.emp_no = salaries.emp_no;
-- 右连接
SELECT * from employees RIGHT JOIN salaries ON employees.emp_no = salaries.emp_no;
-- 这个右连接等价于上面的左连接
SELECT * from salaries RIGHT JOIN employees ON employees.emp_no = salaries.emp_no;
```

左外连接、右外连接
`SELECT * from employees RIGHT JOIN salaries ON employees.emp_no = salaries.emp_no;`
结果是先employees后salaries的字段显示，Right是看表的数据的方向，从salaries往employees看，以salaries为准，它的所有数据都显示

自连接
表，自己和自己连接

```
select manager.*  from emp manager,emp worker where manaer.empno=worker.mgr and worker.empno=1;
select manager.* from emp manager inner join emp worker on manaer.empno=worker.mgr where
worker.empno=1;
```

存储过程、触发器

存储过程（Stored Procedure），数据库系统中，一段完成特定功能的SQL语句。编写成类似函数的方式，可以传参并调用。支持流程控制语句。

触发器（Trigger），由事件触发的特殊的存储过程，例如insert数据时触发。

这两种技术，虽然是数据库高级内容，性能不错，但基本很少用了。
它们移植性差，使用时占用的服务器资源，排错、维护不方便。
最大的原因，不太建议把逻辑放在数据库中。

事务Transaction

InnoDB引擎，支持事务。
事务，由若干条语句组成的，指的是要做的一系列操作。

关系型数据库中支持事务，必须支持其四个属性（ACID）：

特性	描述
原子性（atomicity）	一个事务是一个不可分割的工作单位，事务中包括的所有操作要么全部做完，要么什么都不做
一致性（consistency）	事务必须是使数据库从一个一致性状态变到另一个一致性状态。一致性与原子性是密切相关的
隔离性（isolation）	一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰
持久性（durability）	持久性也称永久性（permanence），指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其有任何影响

> 原子性，要求事务中的所有操作，不可分割，不能做了一部分操作，还剩一部分操作；
> 一致性，多个事务并行执行的结果，应该和事务排队执行的结果一致。如果事务的并行执行和多线程读写共享资源一样不可预期，就不能保证一致性。
> 隔离性，就是指多个事务访问共同的数据了，应该互不干扰。隔离性，指的是究竟在一个事务处理期间，其

> 他事务能不能访问的问题
> 持久性，比较好理解，就是事务提交后，数据不能丢失。

MySQL隔离级别

隔离性不好，事务的操作就会互相影响，带来不同严重程度的后果。
首先看看隔离性不好，带来哪些问题：

1. 更新丢失Lost Update
   事务A和B，更新同一个数据，它们都读取了初始值100，A要减10，B要加100，A减去10后更新为90，B加100更新为200，A的更新丢失了，就像从来没有减过10一样。
2. **脏读**
   事务A和B，事务B读取到了事务A未提交的数据（这个数据可能是一个中间值，也可能事务A后来回滚事务）。事务A是否最后提交并不关心。只要读取到了这个被修改的数据就是脏读。
3. 不可重复读Unrepeatable read
   事务A在事务执行中相同查询语句，得到了不同的结果，**不能**保证同一条查询语句**重复读**相同的结果就是不可以重复读。
   例如，事务A查询了一次后，事务B**修改**了数据，事务A又查询了一次，发现数据不一致了。
   注意，脏读讲的是可以读到相同的数据的，但是读取的是一个未提交的数据，而不是提交的最终结果。
4. **幻读**Phantom read
   事务A中同一个查询要进行多次，事务B插入数据，导致A返回不同的结果集，如同幻觉，就是幻读。
   数据集有记录增加了，可以看做是**增加**了记录的不可重复读。

有了上述问题，数据库就必须要解决，提出了隔离级别。
隔离级别由低到高，如下表

隔离级别	描述
READ UNCOMMITTED	读取到未提交的数据
READ COMMITTED	读已经提交的数据，ORACLE默认隔离级别
REPEATABLE READ	可以重复读，**MySQL的 默认隔离级别。**
SERIALIZABLE	可串行化。事务间完全隔离，事务不能并发，只能串行执行

隔离级别越高，串行化越高，数据库执行效率低；隔离级别越低，并行度越高，性能越高。
隔离级别越高，当前事务处理的中间结果对其它事务不可见程度越高。

```
-- 设置会话级或者全局隔离级别
SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL
{READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE}
-- 查询隔离级别
SELECT @@global.tx_isolation;
SELECT @@tx_isolation;

SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-- 禁用自动提交
SET AUTOCOMMIT = 0
```

SERIALIZABLE，串行了，解决所有问题

REPEATABLE READ，事务A中同一条查询语句返回同样的结果，就是可以重复读数据了。例如语句为(select * from user)。解决的办法有：
1、对select的数据加锁，不允许其它事务删除、修改的操作
2、第一次select的时候，对最后一次确切提交的事务的结果做快照
解决了不可以重复读，但是有可能出现幻读。因为另一个事务可以增删数据。

READ COMMITTED，在事务中，每次select可以读取到别的事务刚提交成功的新的数据。因为读到的是提交后的数据，解决了脏读，但是不能解决 不可重复读 和 幻读 的问题。因为其他事务前后修改了数据或增删了数据。

READ UNCOMMITTED，能读取到别的事务还没有提交的数据，完全没有隔离性可言，出现了脏读，当前其他问题都可能出现。

事务语法
START TRANSACTION或BEGIN开始一个事务，START TRANSACTION是标准SQL的语法。
使用**COMMIT**提交事务后，变更成为永久变更。
ROLLBACK可以在提交事务之前，回滚变更，事务中的操作就如同没有发生过一样（原子性）。
SET AUTOCOMMIT语句可以禁用或启用默认的autocommit模式，用于当前连接。SET AUTOCOMMIT = 0禁用自动提交事务。如果开启自动提交，如果有一个修改表的语句执行后，会立即把更新存储到磁盘。

数据仓库和数据库的区别

本质上来说没有区别，都是存放数据的地方。
但是数据库关注数据的持久化、数据的关系，为业务系统提供支持，事务支持；
数据仓库存储数据的是为了分析或者发掘而设计的表结构，可以存储海量数据。

数据库存储在线交易数据OLTP（联机事务处理OLTP，On-line Transaction Processing）；数据仓库存储历史数据用于分析OLAP（联机分析处理OLAP，On-Line Analytical Processing）。

数据库支持在线业务，需要频繁增删改查；数据仓库一般囤积历史数据支持用于分析的SQL，一般不建议删改。

其它概念

游标Cursor

操作查询的结果集的一种方法。
可以将游标当做一个指针，指向结果集中的某一行。