- 了解SQL
  - 1.1数据库基础
    - 1.1.1什么是数据库
    - 1.1.2表
    - 1.1.3列和数据类型
    - 1.1.4行
    - 1.1.5主键
  - 1.2什么是SQL
  - 1.3手动实践
  - 1.4小结
- MySQL简介
  - 2.1什么是MySQL
    - 2.1.1客户机-服务器软件
    - 2.1.2MySQL版本
  - 2.2MySQL工具
    - 2.2.1mysql命令行实用程序
      - help命令　help select
      - quit命令　quit
    - 2.1.2MySQL Administrator
    - 2.2.3MySQL Query Browser
  - 2.3小结
- 使用MySQL
  - 3.1连接
  - 3.2选择数据库　use crashcourse;
  - 3.3了解数据库和表
    - show databases;　我有哪些数据库
    - show tables;　我有哪些数据表
    - show colums from customers;　这张数据表里都有哪些字段（列）
    - describe customers;　描述一下这张数据表
    - show status;
    - show create database crashcourse;　数据库的创建信息
    - show create table customers;　数据表的创建信息
    - show grants;　安全权限；
    - show errors;
    - show warnings;
  - 3.4小结　对当前数据库内容的基础了解：内容、状态、权限
- 检索数据
  - 4.1SELECT语句　作用是检索信息，必须给出两条信息：想选择什么，从哪里选择
  - 4.2检索单个列　select prod_name from products;
  - 4.3检索多个列　select prod_id,prod_name,prod_price from products;
  - 4.4检索所有列　select * from products;
  - 4.5检索不同的行　select DISTINCT vend_id from products;　distinct需要放在列名前面，它告诉MYSQL只选择不同（唯一）值
  - 4.6限制结果
    - select prod_name from products limit 5;
    - select prod_name from products limit 5,5;　从行5开始，选五行（初始索引为0，实际选取第6-10行）
  - 4.7使用完全限定的表名
    - select products.prod_name from products;
    - select products.prod_name from crashcourse.products;
    - select crashcourse.products.prod_name from crashcourse.products;　数据库-表-列这种层级关系是完整存在的，为了方便使用,习惯上省略层级关系
  - 4.8小结

- 排序检索数据
  - 5.1排序数据
    - select prod_name from products ORDER BY prod_name; 使用检索的列排序；
    - select prod_name from products ORDER BY prod_price ; 使用未检索的列排序；
  - 5.2按多个列排序 select prod_name,vend_id,prod_price from products ORDER BY prod_price, prod_name; 使用多列排序；(by a列,b列)
  - 5.3指定排序方向
    - 关键字DESC=Descending
    - select prod_id,prod_price,prod_name from products order by prod_price DESC;
    - select prod_id,prod_price,prod_name from products order by prod_price DESC,prod_name; DESC只作用于prod_price,而prod_price仍是升序（默认）
    - 关键字ASC=Ascending 默认的
  - 5.4小结
    - Order By +Limit 找出一列中最高/ 最低值
    - Select prod_price from products ORDER BY prod_price DESC LIMIT 1;
    - # select max(prod_price) from products;
- 过滤数据
  - 6.1使用WHERE语句
    - select prod_name,prod_price from products WHERE prod_price =2.5;
    - 条件操作符
      - = 逻辑是
      - <> / != — *逻辑非*
      - < / <= / > / >= / BETWEEN AND — *逻辑包含*
      - IS NULL / IS NOT NULL — *逻辑空*
  - 6.2WHERE子句操作符
    - 6.2.1检查单个值
      - select prod_name,prod_price from products WHERE prod_name="fuses"; 字符串要用引号
      - select prod_name,prod_price from products where prod_price<=10;
    - 6.2.2不匹配检查
      - select vend_id,prod_name from products where vend_id <>1003;
      - select vend_id,prod_name from products where vend_id !=1003;
    - 6.2.3范围值检查 select prod_name,prod_price from products where prod_price BETWEEN 5 AND 10;
    - 6.2.4空值检查
      - select prod_name from products where prod_price IS NULL;
      - select cust_id,cust_email from customers where cust_email IS NULL; IS NULL
  - 6.3小结 用where语句加单条件判断，过滤出需要的数据，包括：逻辑包含、逻辑是、逻辑非、逻辑空
- 数据过滤
  - 7.1组合WHERE语句
    - 7.1.1 AND操作符 select prod_id,prod_price,prod_name from products where vend_id=1003 AND prod_price<=10;
    - 7.1.2 OR操作符 select prod_name,prod_price,vend_id from products where vend_id=1002 OR vend_id=1003;
    - *多个条件来筛选*
    - 7.1.3 计算优先次序 select vend_id,prod_price from products where (vend_id=1002 or vend_id=1003) and prod_price>=10;
      - 默认and优先级高于or,括号的优先级高于 and
      - 多用括号，好习惯，不过分依赖默认次序
  - 7.2IN操作符
    - select prod_name,prod_price from products where vend_id IN (1001,1003) order by prod_name;
    - 结合OR操作符理解 等价于：select vend_id prod_name,prod_price from products where vend_id=1001 or vend_id=1003 order by prod_name;
    - IN 的优点：1、直观，字面理解，只要在里面就满足 / 2、计算次序清晰（没有or） /3、一般比or更快
  - 7.3NOT操作符 select prod_name,prod_price from products where vend_id Not in (1002,1003) order by prod_price; NOT IN 是 IN的否定形式
  - 7.4小结 AND OR 优先顺序 IN

- 用通配符进行过滤
  - 8.1LIKE操作符
    - 8.1.1百分号通配符
      - %表示任何字符出现任何次数
      - select prod_id,prod_name from products where prod_name LIKE 'jet%'; 开头是jet
      - select prod_id,prod_name from products where prod_name LIKE '%anvil%'; 包含anvil
      - select prod_name from products where prod_name LIKE 's%e'; 开头和结尾
      - 空格有时候会干扰匹配
    - 8.1.2下划线通配符
      - _表示单个字符
      - select prod_id,prod_name from products where prod_name LIKE '_ ton anvil'; 返回两行1,2
      - 对照： select prod_id,prod_name from products where prod_name LIKE '% ton anvil'; 返回三行1,2,.5
  - 8.2使用通配符的技巧 它会降低查询速度，慎用
  - 8.3小结 只知道一部分关键字的情况下，用字符匹配位置来查询相关的内容 %是任意多次，_是一次
- 用正则表达式进行搜索 ⑫
- 创建计算字段
  - 10.1计算字段 计算字段并不存在于表中，它是运行时创建的
  - 10.2拼接字段
    - concat()函数
    - 多数数据库管理系统用+或者||，mysql用concat()函数
    - select concat(vend_name,'(',vend_country,')') from vendors order by vend_name; 拼接了四个元素
    - 使用别名AS select concat(Rtrim(vend_name),'(',Rtrim(vend_country),')') AS vend_title from vendors order by vend_name;
  - 10.3执行算数计算
    - select prod_id,quantity,item_price from orderitems where order_num=20005; select prod_id,quantity,item_price,quantity*item_price AS expanded_price from orderitems where order_num=20005;
    - 上方代码两句是做对比的，关键在第二句的计算字段的创建——通过两个字段的乘法创造了一个新字段
  - 10.4小结
- 使用数据处理函数
  - 11.1函数
    - 11.2.1文本处理函数
      - Upper select vend_name,Upper(vend_name) AS vend_name_upcase from vendors order by vend_name;
      - Left ()
      - Right()
      - Lengh()
      - Locate() select locate("ball","football"); #返回值5
      - Lower
      - Upper()
      - Ltrim()
      - Rtrim()
      - soundex()
        - 根据近似的读音去匹配
        - select cust_name,cust_contact from customers where cust_contact ='Y.Lie'; 返回 empty
        - select cust_name,cust_contact from customers where soundex(cust_contact) =soundex('Y.Lie'); 返回Y Lee
      - substring()
        - 提取子字符串
        - select SUBSTRING('MySQL SUBSTRING' ，10);
        - select SUBSTRING('MySQL SUBSTRING' FROM -10);
    - 应该总是使用四位数字的年份

- 11.2使用函数
  - 11.2.2日期和时间处理函数
    - Date()函数
      - select cust_id from orders where DATE(order_date)="2005-09-01";
      - 限定只取日期（不取时间）
    - Time()函数
      - 限定只取时间，不取日期
    - 取2005年9月的订单
      - 方法1
        - select cust_id from orders where DATE(order_date) BETWEEN "2005-09-01" AND "2005-09-30";
      - 方法2
        - select cust_id from orders where YEAR(order_date)=2005 and MONTH(order_date)=9;
    - 常用函数 16
  - 11.2.3数值处理函数
    - Abs()
    - Cos()
    - Mod()
    - Exp()
    - Pi()
    - Rand()
    - Sin()
    - Squrt()
    - Tan()
- 11.3小结

- 汇总数据
  - 12.1聚集函数
    - 12.1.1 Avg()函数
      - select AVG(prod_price) as avg_price from products;
      - select AVG(prod_price) as avg_price from products where vend_id = 1003;
    - 12.1.2 Count()函数
      - count(*)统计行数，无论是否空都计数。
        - select count(*) as row_count from products;
      - count（列名）统计某列非空值数量
        - select count(cust_email) as num_cust from customers;
    - 12.1.3 Max()函数
      - select MAX(prod_price) as max_price from products;
    - 12.1.4 Min()函数
      - select MIN(prod_price) as min_price from products;
    - 12.1.5 Sum()函数
      - select SUM(quantity) as total_amout from orderitems where order_num = 20005;
      - select SUM(quantity*item_price) as total_price from orderitems where order_num=20005;
  - 12.2聚集不同值
    - Distinct()函数
      - 例子1
        - select avg(prod_price) from products where vend_id=1003;
        - select avg(DISTINCT prod_price) from products where vend_id=1003;
        - select avg(ALL prod_price) from products where vend_id=1003;
        - 默认为ALL，可省略
      - 例子2
        - select count(distinct(prod_price)) from products;
        - select count(*) from products;
      - 说明
        - distinct 可以有两种形式，一种是定语，一种是函数，都可以得到想要的结果
  - 12.3组合聚集函数
    - 同时创建多个聚合变量
    - select count(*) as num_items,min(prod_price) as min_price,max(prod_price) as max_price,avg(prod_price) as avg_price from products;
  - 12.4小结
  - 13.1数据分组
    - 略
  - 13.2创建分组
    - group by 关键字
      - select vend_id,count(*) as num_items from products GROUP BY vend_id;
    - with rollup 关键字
      - select vend_id,count(*) as num_items from products GROUP BY vend_id WITH ROLLUP;
      - 在最后一行增加汇总（求和）
  - 13.3过滤分组
    - having 关键字
      - having在某种程度上能替代where 关键字
      - select cust_id,count(*) as orders from orders group by cust_id HAVING count(*)>=2;
        - 通过having关键字对分组结果进行了筛选
      - select vend_id,count(*) as num_prod from products where prod_price > 10 group by vend_id having count(*)>=2;
        - where在分组前过滤，过滤对象是原始数据；
        - having在分组后过滤，过滤对象是分组结果；
  - 13.4分组和排序
    - select order_num,sum(quantity*item_price) as price_total from orderitems group by order_num having price_total>=50 order by price_total;
    - 对分组的结果进行排序

- 分组数据
  - 13.5SELECT子句顺序

	子句	说明	是否必须使用
	select	要返回的列或者表达式	是
	from	从中检索数据的表	仅在从表检索数据时使用
	where	行级别过滤	否
	group by	分组说明	仅在按组聚集时使用
	having	输出排序顺序	否
	order by	要检索的行数	否

  - 13.6小结
    - 从需求出发
      - 1、分组的目的是实现分类。
      - 2、在分组功能的实现过程中隐含了一个“双重过滤”的概念，即对原始数据过滤（where）+对分组结果过滤（having）。
      - 其实都是为了更好地完成分组任务服务的。
      - 3、分组得到的结果是统计结果。
    - ANKI总结到这里了（这里还没总结）
- 使用子查询
  - 14.1子查询
    - select cust_id
      from orders
      where order_num in (select order_num
      from orderitems
      where prod_id = "TNT2");
  - 14.2使用子查询过滤
    - select cust_name,cust_contact
      from customers
      where cust_id in (select cust_id
      from orders
      where order_num in (select order_num
      from order_items
      where prod_id="TNT2"));
    - from：两层子查询；内层先执行；
    - where：一般用IN，也可以用=、<>
  - 14.3作为计算字段使用子查询
    - select cust_name,
      cust_state,
      (select count(*)
      from orders
      where orders.cust_id=customers.cust_id) as orders
      from customers
      order by cust_name;
    - 等：select ABCfromX;
      C跨表匹配（相关子查询）
      orders.cust_id=customers.cust_id
      orders是一个计算字段。
    - 有一个顾客名单，还有一个历史购物单想要统计顾客购买的次数。
  - 14.4小结
    - 1、子查询的本质
      可以理解为一种嵌套查询。
      即A嵌套着B，被嵌套的语句B先执行，然后输出一个结果供A使用。
      2、如何使用这个输出的结果
      可以作为一个数据集和，参与A的包含逻辑
      也可以作为一个集合，参与A的字段构建。
- 联结表
  - 15.1联结
    - 15.1.1关系表
      - 分解数据为多个表，能更有效的储存，更方便的处理，并且具有强大的可伸缩性（但是都是有代价的）
    - 15.1.2为什么要使用联结
      - 等：
        - 不设定where的联结
          - select vend_name,prod_name,prod_price
            from vendors,products
            order by vend_name,prod_name;
          - 14行*6行=84行，笛卡尔积
        - 设定where的联结
          - select vend_name,prod_name,prod_price
            from vendors,products
            where vendors.vend_id=products.vend_id
            order by vend_name,prod_name;
        - WHERE子句的重要性
          - 联结两个表，过程是第一个表的每一行和第二个表的每一行逐条匹配，where作为过滤条件，只保留匹配给定条件的行。如果不设定where过滤，就会得到两表的笛卡尔积。
  - 15.2创建联结
    - 15.2.1WHERE子句：where
    - 15.2.2内部联结：inner join
      - select vend_name,prod_name,prod_price
        from vendors inner join products
        on vendors.vend_id=products.vend_id;
      - inner join实际上形成了一个新表
    - 15.2.3联结多个表
      - select prod_name,vend_name,prod_price,quantity
        from orderitems,products,vendors
        where products.vend_id = vendors.vend_id
        and orderitems.prod_id =products.prod_id
        and order_num = 20005;
  - 15.3小结
    - 联结是SQL中最重要也是最强大的特性
    - 联结的对象是两个表，或者两个表里的若干字段。

- 创建高级联结
  - 16.1使用表别名
    - SELECT Concat(Rtrim(vend_name),"(",Rtrim(vend_country),")") AS vend_title FROM vendors ORDER BY vend_name;
    - select cust_name,cust_contact from customers as c,orders as o,orderitems as oi where c.cust_id =o.cust_id and oi.order_num = o.order_num and prod_id = "TNT2";
      - 表别名，对应的是列别名，表别名不返回到客户机，只是中间变量列别名会返回到客户机（呈现在屏幕上）
  - 16.2使用不同类型的联结
    - 16.2.1自联结
      - 自己联结自己，表A联结表A
      - 案例 ≡
        - 子查询实现
          - select prod_id,prod_name from products where vend_id = (select vend_id from products where prod_id = "DTNTR")；
        - 自联结实现
          - select p1.prod_id,p1.prod_name from products AS p1,products AS p2 where p1.vend_id = p2.vend_id and p2.prod_id = "DTNTR";
            - 再复习，略难理解
        - 重命名的形式
          - SELECT continent, name, population FROM world x WHERE population >= ALL (SELECT population FROM world y WHERE y.continent=x.continent AND population>0)
    - 16.2.2自然联结
      - 只能选择那些唯一的列，没有重复的列被检索出来
      - select c.*,o.order_num,o.order_date,oi.prod_id,oi.quantity,oi.item_price from customers AS c,orders as o,orderitems as oi where c.cust_id = o.cust_id and oi.order_num = o.order_num and prod_id = "FB";
    - 16.2.3外部联结
      - 左外部联结 LEFT OUTER JOIN
        - select customers.cust_id,orders.order_num from customers LEFT OUTER JOIN orders ON customers.cust_id = orders.cust_id;
      - 右外部联结 RIGHR OUTER JOIN
        - select customers.cust_id,orders.order_num from customers RIGHT OUTER JOIN orders ON customers.cust_id = orders.cust_id;
  - 16.3使用带聚集函数的联结
    - select customers.cust_name,customers.cust_id, count(orders.order_num) as num_ord from customers inner join orders on customers.cust_id=orders.cust_id group by customers.cust_id;
      - 再复习，略难理解
    - select customers.cust_name,customers.cust_id, count(orders.order_num) as num_ord from customers left outer join orders on customers.cust_id=orders.cust_id group by customers.cust_id;
      - 再复习，略难理解
  - 16.4使用联结和联结条件
  - 16.5小结
- 组合查询
  - 17.1组合查询
  - 17.2创建组合查询
    - 关键语句 Union、Union All、Where +or
    - 17.2.1使用UNION ≡
      - select vend_id,prod_id,prod_price from products where prod_price<=5 UNION select vend_id,prod_id,prod_price from products where vend_id in(1001,1002);
      - 对比where/OR语句
        - select vend_id,prod_id,prod_price from products where prod_price <=5 OR vend_id in(1001,1002);
    - 17.2.2UNION规则
      - 1、两条以上的select语句组成
      - 2、每个查询必须包含相同的列、表达式或者聚集函数
    - 17.2.3包含或取消重复的行 ≡
      - 若几条语句中包含互相重复的行，则返回值会默认只显示其中一行
      - 使用 UNION ALL以显示所有行（无论是否重复）
      - Union All能做到的，where做不到
    - 17.2.4对组合查询结果排序
      - 不允许使用多条ORDER BY语句，只能用一条即只能对最终的结果集统一排序
      - select vend_id,prod_id,prod_price from products where prod_price<=5 UNION select vend_id,prod_id,prod_price from products where vend_id in(1001,1002) ORDER BY vend_name,vend_price;
  - 17.3小结

- 全文本搜索
  - 18.1理解全文本搜索
    - 并非所有的引擎都支持全文搜索
      - myISAM支持，InnoDB不支持
  - 18.2使用全文本搜索
    - 18.2.1启用全文本搜索支持
      - 等
        - create table productnotes
(
note_id int NOT NULL AUTO_INCREMENT,
prod_id chat910) NOT NULL,
note_data datetime NOT NULL,
note_text text NULL,
PRIMARY KEY(note_id),
FULLTEXT(note_text)
) ENGINE=MyISAM;
    - 18.2.2进行全文本搜索
      - 关键词
        - Match(note_text) Against('rabbit')
      - select note_text
from productnotes
where Match(note_text) Against('rabbit');
        - 等
          - 1、不区分大小写；
          - 2、传递给Match的值必须与FULLTEXT定义中相同，如果指定多个列则必须列出它们且次序正确。
      - select note_text,
Match(note_text) Against('rabbit') As rank
from productnotes;
        - 排序的确认，结果中目标词靠前的，等级高；不含目标词的等级统统为0
    - 18.2.3使用查询拓展
      - 关键词
        - Against('rabbit' WITH QUERY EXPANSION)
      - select note_text
from productnotes
where Match(note_text) Against('rabbit' WITH QUERY EXPANSION);
    - 18.2.4布尔文本搜索
      - select note_text
from productnotes
where match(note_text) against('heavy' IN BOOLEAN MODE);
      - select note_text
from productnotes
where match(note_text) against('heavy -rope*' IN BOOLEAN MODE);
        - 排除以rope开始的行
      - +,包含，词必须存在
-,排除，词必须不出现
>,包含，且中增加等级值
<,包含，且减少等级值
(),把词组成子表达式（允许这些子表达式作为一个组被包含、排除、排列等）
_,取消一个词的排序值
*,词尾的通配符
"",定义一个短语（与单个词的列表不一样，它匹配整个短语以便包含或排除这个短语）
      - select note_text
from productnotes
where Match(note_text) Against('+rabbit +bait' IN BOOLEAN MODE);
        - 两个必须都包含
      - select note_text
from productnotes
where Match(note_text) Against('rabbit bait' IN BOOLEAN MODE);
        - 至少包含一个即可
      - select note_text
from productnotes
where Match(note_text) Against('"rabbit bait"' IN BOOLEAN MODE);
        - 必须匹配短语（而不是词）
      - select note_text
from productnotes
where Match(note_text) Against('>rabbit <carrot' IN BOOLEAN MODE);
        - 提高前者等级，降低后者等级
      - 理解
        - 布尔文本搜索，首先是针对文本进行搜索，之后通过特定的表达式定义索索范围，可以实现多种搜索条件的组合。
    - 18.2.5全文本搜索的使用说明
  - 18.3小结

- 19.1数据插入
  - INSERT INTO customers
values(NULL,'Pep E.LaPew','100 Main Street','Los Angeles','CA','90046','USA',NULL,NULL);
    - 一般不这么用，潜在风险忒多
- 19.2插入完整的行
  - INSERT INTO
customers(cust_name,cust_address,cust_city,cust_state,cust_zip,cust_country,cust_contact,cust_email)
values('Pep E.LaPew','100 Main Street','Los Angeles','CA','90046','USA',NULL,NULL);
    - 指定列插入，不再受到原始字段顺序的限制
    - 每个列都要给对应的值，必须一一对应
  - INSERT INTO
customers(cust_contact,cust_email,cust_name,cust_address,cust_city,cust_state,cust_zip,cust_country)
values(NULL,NULL,'Pep E.LaPew','100 Main Street','Los Angeles','CA','90046','USA');
    - 再次证明，乱序插入，只要指定好列名即可
    - 即使表的结构变了，只要字段还在，就没问题
  - insert Low_priority into
    - 当查询和插入同时存在时，可以降低插入的优先级，让查询任务优先

- 插入数据
  - 19.3插入多个行
    - 推荐方式
      - INSERT INTO customers(cust_name,cust_address,cust_city,cust_state,cust_zip,cust_country) values('Pep E.LaPew','100 Main Street','Los Angeles','CA','90046','USA'), ('M.Martian','42 Galaxy way','New York','NY','11213','USA');
      - 一个括号对应一行内容。
    - 反对方式
      - INSERT INTO customers(cust_name,cust_address,cust_city,cust_state,cust_zip,cust_country) values('Pep E.LaPew','100 Main Street','Los Angeles','CA','90046','USA');
      - INSERT INTO customers(cust_name,cust_address,cust_city,cust_state,cust_zip,cust_country) values('M.Martian','42 Galaxy way','New York','NY','11213','USA');
      - 每一条一个完整的插入语句，看上去严谨，感觉有点机械。缺陷是相对慢的多
  - 19.4插入检索出的数据
    - INSERT SELTCT语句
      - INSERT into customers(cust_name,cust_address,cust_city,cust_state,cust_zip,cust_country) SELECT cust_name,cust_address,cust_city,cust_state,cust_zip,cust_country from custnew;
      - custnew是新创建的表，将其中的对应列插入customers
      - 并不要求列名一一对应，插入是根据列顺序来的，和列名没有关系！！！
      - SELECT的目的是选出列，也可以用where语句进行约束
  - 19.5小结
    - 理解插入语句
      - 本质上，这也是一种命令。
      - 形式上：INSERT INTO 表名（字段1，字段2、字段3...）值（值1，值2，值3）
      - 涉及内容 - 告诉程序要插入值，在哪个表，哪个字段，插入什么值
      - 插入一条数据还是多条数据，顺序会不会乱，插入的是筛选来的行不行
- 更新和删除数据
  - 20.1更新数据
    - update
      - 万万不能省略where,否则更新所有行
    - UPDATE SET
      - UPDATE customers set cust_email ='elekk@fund.com' where cust_id =10005;
      - 先列后行交叉定位
    - 更新多个数据
      - UPDATE customers SET cust_name='The Fudds', cust_email = "goodday@gmail.com" WHERE cust_id = 10005;
      - 更新的时候，这里的语言习惯问题，英语习惯把重要的定语后置。理解上就先列后行吧
    - 设为NULL
      - UPDATE customers SET cust_email=NULL where cust_id =10005;
      - 将某字段能够值设置为NULL，相当于删除
  - 20.2删除数据
    - DELETE
      - DELETE from customers WHERE cust_id=10006;
      - 删除指定的行，如果不指定，就全删；DELETE只实现删行，不删表本身
  - 20.3更新和删除的指导原则
    - MySQL没有撤销按钮
    - 在删东西前，要先测试对应的SELECT语句，保证目标准确
    - 要删东西必须带where语句
  - 20.4小结
    - 关键字：修改元素、修改数据、update set 删除行、delete from where
  - 21.1.1创建基础表
    - CREATE TABLE
      - 若创建的表与已经存在的表重名，则会报错（mysql避免覆盖的机制）
    - CREATE TABLE IF NOT EXISTS students
      - 仅在表不存在时创建
    - CREATE TABLE customer_test(cust_id int not null auto_increment, cust_name char(50) not null, primary key(cust_id))ENGINE = InnoDB;
      - 表名（字段及属性，主键）引擎；——表名和引擎是并列关系
    - insert into orders_test(order_num,order_date,cust_id) values('1','2019-01-01','1209');
  - 21.1.2使用NULL值
    - create table vendors_test(vend_id int NOT NULL AUTO_INCREMENT, vend_name char(50) NOT NULL, vend_address char(50) NULL, vend_city char(50) NULL, vend_state char(5) NULL, vend_zip char(10) NULL, vend_country char(50) NULL, PRIMARY KEY(vend_id))ENGINE = InnoDB;
      - NOT NULL 将阻止插入没有值的列 若不指定，则默认为NOT NULL
  - 21.1.3主键再介绍
    - 主键必须是唯一值：单列情况下值唯一；多列情况下组合值唯一。
    - 多列组合情况指定主键
      - PRIMARY KEY(order_num,order_item)

- 创建和操纵表
  - 21.1创建表
    - 21.1.4使用AUTO__INCREMNT
      - 每个表只允许一个列使用AUTO_INCREMENT
      - 插入新行的时候，如果指定一个值代替自动生成的值，则后续的增量将开始于手工插入的值；
      - 确定AUTO_INCREMENT值　SELECT last_insert_id();
    - 21.1.5指定默认值
      - 在创建表的时候指定某一列的默认值是1，插入数据时，如果不没有给该列值，则会自动填充1
        - create table orderitems(
          order_num int NOT NULL,
          order_item int NOT NULL,
          prod_id char(10) NOT NULL,
          quantity int NOT NULL DEFAULT 1,
          item_price decimal(8,2) NOT NULL,
          PRIMARY KEY(order_num,order_item)
)ENGINE = InnoDB;
      - 默认值不允许指定函数，只允许指定常量
      - 许多数据库开发人员使用默认值而不是NULL列，对于那些用于计算或者数据分组的列更是如此
    - 21.1.6引擎类型
      - 1、innoDB　是一个可靠的事务处理引擎，它不支持全文本搜索
      - 2、MEMORRY　功能等同于MyISAM，但是由于数据存储在内存（而不是磁盘）中，速度很快，特别适合于临时表
      - 3、MyISAM　是一个性能极高的引擎，它支持全文本搜索，但不支持事务处理
      - 注意　引擎类型可以混合使用，譬如有三张表，其中两张用引擎A，一张使用引擎B；
        混合使用引擎的缺陷是：外键不能跨引擎，即使用一个引擎的表不能引用具有使用不同引擎表的外键
  - 21.2更新表　ALTER TABLE
    - ALTER TABLE vendors
      ADD column vend_phone char(20);　增加列，必须明确数据类型
    - ALTER TABLE vendors
      DROP COLUMN vend_phone;　删除列
    - ALTER TABLE vendors
      CHANGE column vend_phone phone varchar(30);　重命名列/修改列名
    - ALTER TABLE orderitems
      ADD CONSTRAINT fk_orderitems_orders
      FOREIGN KEY (order_num) REFERENCES orders (order_num);　P166本书所用的外键的定义
  - 21.3删除表　DROP TABLE　DROP TABLE customers2;
  - 21.4重命名表　RENAME TO
    - RENAME TABLE customers2 TO customers3;　重命名一张表
    - RENAME TABLE customers2 TO customers3，
      A to B，
      C to D;
- 使用视图
  - 22.1视图
    - 22.1.1为什么使用视图
      - 重用SQL语句
      - 简化复杂的SQL操作
      - 使用表的组成部分而不是整个表
      - 保护数据（权限）、
      - 更改数据格式和表示
    - 22.1.2视图的规则和限制
  - 22.2使用视图
    - 创建　CREATE VIEW viewname AS
    - 查看　SHOW CREATE VIEW viewname
    - 删除　DROP VIEW viewname
    - 更新
      - 先DROP再CREATE
      - 或者
      - CREATE OR REPLCAE VIEW
    - 22.2.1利用视图简化复杂的联结
      - create view productcustomers as
        select cust_name,cust_contact,prod_id
        from customers,orders,orderitems
        where customers.cust_id = orders.cust_id
        and orderitems.order_num = orders.order_num;　返回已经订购了任意产品的所有的客户的列表
      - select * from productcustomers;
      - select cust_name,cust_name,cust_contact
        from productcustomers
        where prod_id = 'TNT2';　使用视图大大简化查询语句
    - 22.2.2用视图重新格式化检索出的数据
      - select Concat(Rtrim(vend_name),'(',Rtrim(vend_country),')')
        as vend_title
        from vendors
        order by vend_name;
      - 转换为视图
        - Create VIEW vendorlocations as
          select Concat(Rtrim(vend_name),'(',Rtrim(vend_country),')')
          as vend_title
          from vendors
          order by vend_name;
      - 使用视图　select * from vendorlocations;
      - 总结　将格式转换的语句封装为视图，之后使用的时候可以简单调用视图即可

- 22.2.3用视图过滤不想要的数据
  - 定义customeremaillist视图，过滤没有电子邮件地址的客户
  - create view customeremaillist as
    select cust_id,cust_name,cust_email
    from customerswhere cust_email IS NOT NULL;
  - select * from customeremaillist;
- 22.2.4使用视图与计算字段
  - select prod_id,quantity,item_price,quantity*item_price AS expanded_price
    from orderitems
    where order_num =20005;
  - 转化为视图
    - CREATE VIEW orderitemsexpanded AS
      select order_num,order_id,quantity,itemprice,quantity*item_price AS expanded_price
      from orderitems;
  - 使用视图
    - select * from orderitemsexpanded
      where order_num = 20005;
- 22.2.5更新视图
  - 通常可更新，但是由于视图主要是用来做查询，所并不建议更新
  - 若视图定义中有以下操作，则不能更新（分组、联结、子查询、并、聚集函数min()count()sum()等）、DISTINCT、导出（计算）列

使用存储过程

- 本质
  - 存储过程本质是函数；
- 调用CALL
  - 存储过程的执行称为调用
  - CALL productpricing(@pricelow,@pricehigh,@priceaverage) #计算并返回产品最低、最高和平均价格
- 不带参数的存储过程
  - 创建
    - CREATE PROCEDURE productpricing()
      BEGIN
      select avg(prod_price) as priceaverage
      from products;
      END;
      =
      - 优化
        - 解决分隔符的执行
          - DELIMITER // #重新定义分隔符
            CREATE PROCEDURE productpricing()
            BEGIN
            select avg(prod_price) as priceaverage
            from products;
            END // #创建过程（保证遇到;不执行）
            DELIMITER ；#重新定义分隔符
  - 调用
    - CALL productpricing();
      - 就和调用函数一样
  - 删除
    - drop procedure productpricing;
- 带参数的存储过程
  - delimiter //

    CREATE PROCEDURE productpricing(
    OUT pl Decimal(8,2),
    OUT ph Decimal(8,2),
    OUT pa Decimal(8,2)
)
    BEGIN
    SELECT Min(prod_price)
    INTO pl
    From products;
    SELECT Max(prod_price)
    INTO ph
    From products;
    SELECT Avg(prod_price)
    Into pa
    From products;
    END;//
    delimiter ;
  - 带变量的过程
    - CALL productpricing(@pricelow,@pricehigh,@priceaverage);
      - 没有任何返回值
    - select @pricaverage;
    - select @pricelow;
    - select @pricehigh;
  - CREATE PROCEDURE ordertotal(
    IN onumber INT,
    OUT ototal Decimal(8,2))
    BEGIN
    SELECT Sum(item_price*quantity)
    FROM orderitems
    where order_num = onumber
    INTO ototal;
    END;
  - 调用带变量的过程2
    - CALL ordertotal(20005,@total);
      SELECT @total;
    - CALL ordertotal(20009,@total)
      SELECT @total;
  - -- name:prdertotal
    -- parameters: onumber = order number
    -- taxable = 0 if not taxable,1 if taxable
    -- ototal = order total variable
    CREATE PROCEDURE ordertotal(
    IN onumber INT,
    IN taxable BOOLEAN,
    OUT ototal DECIMAL(8,2)) COMMENT 'Obtain order total,optionally adding tax'

    BEGIN
    -- Declare variable for total
    DECLARE total Decimal(8,2);
    -- Declare tax percentage
    DECLARE taxrate INT DEFAULT 6;

- 建立智能存储过程
  - 代码
```
-- Get the order total
SELECT Sum(item_price*quantity)
FROM orderitems
WHERE order_num = onumber
INTO total;

-- Is this taxable?
IF taxable THEN
-- Yes,so add taxrate to the total
SELECT total+(total/100*taxrate) INTO total;
END IF;
-- And finally,save to out variavle
SELECT total INTO ototal;
END;
```
  - 验证
```
call ordertotal(20005,0,@total);
select @total;
call ordertotal(20005,1,@total);
select @total;
```
- 检查存储过程
  - SHOW CREATE PROCEDURE ordertotal;
  - SHOW PROCEDURE STATUS;
  - SHOW PROCEDURE STATUS LIKE ‘ordertotal;

使用游标　暂时卡在open ordernumbers;P188

使用触发器
- 你想要某条语句在事件发生时自动自行 ≡
- MySQL响应特定语句而自动执行的一条MySQL语句（或位于BEGIN和END之间的一组语句，包括DELETE，INSERT，UPDATE，其他语句不支持触发器
- 只有表支持触发器（视图、临时表不支持），每个表最多支持6个触发器（每条INSERT、UPDATE、DELETE的之前和之后），单一触发器不能与多个事件、或多个表关联
- 创建
  - CREATE TRIGGER
  - CREATE TRIGGER newproduct
AFTER INSERT ON products
FOR EACH ROW
SELECT 'product added';　≡　每插入一行都显示 'product added'
- 删除　DROP TRIGGER newproduct;
- 最新到P195

管理事务处理
- 案例　给系统添加订单的过程
- 本质是一种机制（管理方案），保证一组操作不糊中途听通知，它们或者作为整体执行，或者不执行。如果没有错误发生，整组语句提交给数据库表；如果发生错误，则进行回退以恢复数据库到某个已知且安全的状态。
- 事务　transaction,指的是一组SQL语句
- 回退　rollback,指的是撤销指定SQL语句的过程
- 提交　commit,指的是将未存储的SQL语句结果写入数据库表
- 保留点　savepoint,指的是事务处理中设置的临时占位符(placeholder)你可以对它发布回退
- 回退案例
```
select * from text; --查看表内容
start transaction;  --开启事务
delete from text;   --删除有一个表中所有数据
select * from text; --确认已经执行
rollback;           --回退
select * from text; --确认回退成功
```
- 回退的范围
  - 包括：insert、update、delete
  - 不包括：select、create、drop
- 隐含提交　implicit commit，即提交动作是自动进行的（在事务处理中则需要明确提交）
- 提交
```
start transaction;
delete from text where vend_id = 1001;
delete from text where prod_price=50;
commit;
```
如果第一条delete起作用而第二条delete失败，则两条delete都不会提交；
- 隐含事务关闭　当commit或者rollback语句执行后，事务会自动关闭（将来的更改会隐含提交）
- 保留点
```
select * from text;
start transaction;
delete from text where vend_id = 1002;
savepoint delete1;  --设置保留点
delete from text where vend_id = 1003;
rollback to delete1;  --回撤到保留点
select * from text;
```
保留点越多越好
- 释放保留点　保留点在事务处理完成后自定释放（也可以用release savepoint）明确释放
- 更改默认的提交行为　默认状态下，mysql行为是自动提交所有更改；可指示mysql不自动提交：set autocommit=0;

- 全球化和本地化
  - 字符集和校对顺序
    - 字符集　字母和符号的集合
    - 编码　某个字符集成员的的内部表示
    - 校对　规定字符如何比较的指令　大小写，不同语言的字符集
  - 使用字符集和校对顺序
    - show character set;
    - show variables like 'collation%';
  - 作用范围
    - 字符集很少是服务器范围的设置，不同的表甚至不同的列都有可能需要不同的字符集，而且两者都可以在建表的时候指定。
    - 表　create table mytable (column1 INT, column2 varchar(10))DEFAULT CHARACTER SET hebrew COLLATE hebrew_general_ci;　对整个表指定给了字符集和校对
    - 列　create table mytable (column1 INT, column2 varchar(10) column3 varchar(10) character set latin1 collate latin1_general_ci)DEFAULT CHARACTER SET hebrew COLLATE hebrew_general_ci;　对整个表和特定的列指定了字符集和校对
  - 在select语句中设定　select * from customers order by cust_city,cust_name collate latin1_general_cs;　临时指定，区分大小写
  - 串的字符集转换　如果绝对需要，串可以以在字符集之间进行转换，可以用cast()或者convert函数
- 安全管理
  - 访问控制
    - 不要使用root
    - 通常防止的是无意的错误
    - 不同的需求不同的账号权限
  - 管理用户
    - 创建用户账号　use mysql; --一个默认数据库 select user from user;--user用来存放用户信息 create user ben identified by 'passwd123';--使用grant或者insert grant也可以创建用户，但是create user是最简单清晰的 rename user ben to max;--重命名一个用户账号
    - 删除用户账号　drop user max;--删除用户账号即想干权限
    - 设置访问权限　show grants for max; --显示账号max 的权限 grant select on crashcourse.* to max; --为账号max赋予指定数据库的访问权限（只读） show grants for max;--确认这个更改 revoke select on crashcourse.* from max; --grant 授予 --revoke 废除，取消，撤回
      - grant和revoke 在几个层次上控制访问权限　≡
      - 权限列表　略
    - 更改口令　set password for max = password("password") --必须传递到函数中
      - set password = password('newpasswd123')--重设自己的密码
- 数据库维护
  - 备份数据
    - 刷新未写数据　flush tables;--保证所有数据都被写到磁盘（包括索引数据）
    - 方案1　mysqldump，转储所有数据库内容到某个外部文件
    - 方案2　mysqlhotcopy从一个数据库复制所有数据
    - 方案3　backup table或者select into outfile转储所有数据到某个外部文件
  - 进行数据库维护
    - analyze table --用来检查表键是否正确
    - check table --用来针对许多问题对表进行检查
    - repair table --修复有问题的表
    - optimize table --删除大量数据后收回所用的空间
  - 诊断启动问题
    - mysql服务器自身通过在命令行上执行mysqld启动
    - 重要命令选项
      - --help 显示帮助
      - --safe-mode 装载减去某些最佳配置的服务器
      - --verbose显示全文本消息
      - --version显示版本信息
  - 查看日志文件
    - 错误日志　包含启动和关闭问题以及任意关键错误的细节 通常名为hostname.err位于data目录中
    - 查询日志　记录所有mysql活动，在 诊断问题时非常有用， 通常名为hostname.log位于data目录中
    - 二进制日志　记录更新过数据的所有语句，通常名为hostname-bin位于data目录内
    - 缓慢查询日志　记录执行缓慢的任何查询，往往用来协助优化数据库。通常名为hostname-slow.log,位于data目录中
    - 备注　使用flush logs语句刷新和重新开始所有日志文件
- 改善性能

XMind | 试用模式