数据库

概念

数据库：按照数据结构来组织、存储、管理数据的仓库。

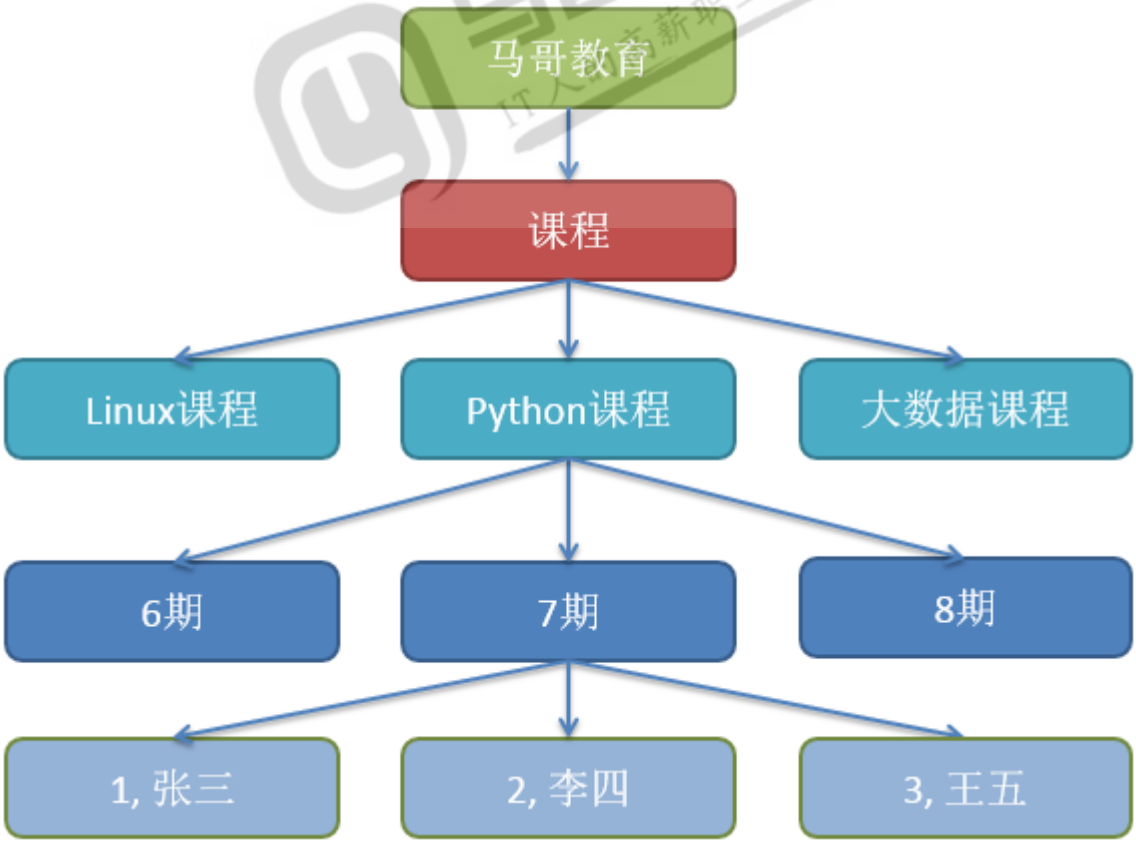
诞生
计算机的发明是为了做科学计算的，而科学计算需要大量的数据输入和输出。
早期，可以使用打孔卡片的孔、灯泡的亮灭来表示数据输入、输出。
后来，数据可以存储在磁带上，顺序的读取、写入磁带。
1956年IBM发明了磁盘驱动器这个革命性产品，支持随机访问。
随着信息化时代的到来，有了硬件存储技术的发展，有大量的数据需要存储和管理，数据库管理系统DBMS就诞生了。
不管使用什么存储介质，数据库的数据模型才是其核心和基础。

分类

按照数据模型分类：网状数据库、层次数据库、关系型数据库

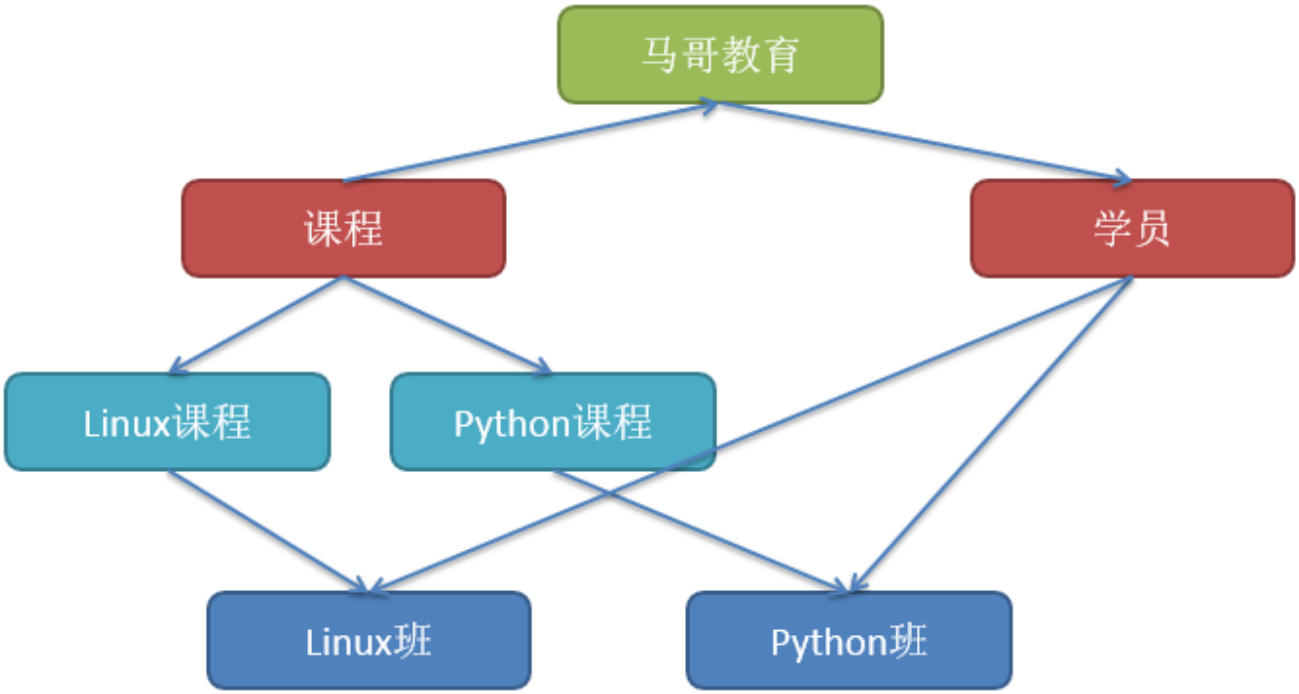
层次数据库

以树型结构表示实体及其之间的联系。关系只支持一对多。代表数据库IBM IMS。



网状数据库

通用电气最早在1964年开发出网状数据库IDS，只能运行在GE自家的主机上。

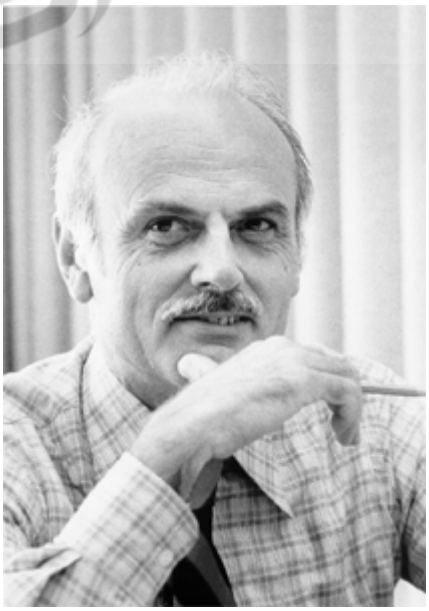


结点描述数据，结点的联系就是数据的关系。

能够直接描述客观世界，可以表示实体间多种复杂关系，而这是层次数据模型无法做到的。比如，一个结点可以有多个父结点，结点之间支持可以多对多关联。

关系数据库

使用行、列组成的二维表来组织数据和关系，表中行（记录）即可以描述数据实体，也可以描述实体间关系。
关系模型比网状模型、层次模型更简单，不需要关系数存储的物理细节，专心于数据的逻辑构建，而且关系模型有论文的严格的数学理论基础支撑。



1970年，IBM的研究员E.F.Codd发表了名为“A Relational Model of Data for Large Shared Data Banks”的论文，提出了关系模型的概念，奠定了关系模型的理论基础。
关系模型，有严格的数学基础，抽象级别较高，简单清晰，便于理解和使用。
经过几十年的发展，关系数据库百花齐放，技术日臻成熟和完善。

基于关系模型构建的数据库系统成为RDBMS(Relational DataBase System)。
IBM DB2、Oracle的Oracle和Mysql、微软的MS SQL。以前的infomix、Sybase等。

Oracle的发展

拉里·埃里森(Larry Ellison)仔细阅读了IBM的关系数据库的论文，敏锐意识到在这个研究基础上可以开发商用软件系统。他们决定开发通用商用数据库系统Oracle，这个名字来源于他们曾给中央情报局做过的项目名。几个月后，他们就开发了Oracle 1.0。Oracle快速的被推销，但是很不稳定。直到1992年的时候，Oracle7才逐渐稳定下来，并取得巨大成功。2001年的9i版本被广泛应用。
2009年4月20日，甲骨文公司宣布将以每股9.50美元，总计74亿美金收购sun（计算机系统）公司。2010年1月成功收购。
2013年，甲骨文超过IBM，成为继微软之后的全球第二大软件公司。

Mysql发展

1985年几个瑞典人为大型零售商的项目设计了一种利用索引顺序存取数据的软件，这就是MyISAM的前身。1996年，MySQL 1.0发布，随后发布了3.11.1版本，并开始往其它平台移植。2000年MySQL采用GPL协议开源。
MySQL 4.0开始支持MyISAM、InnoDB引擎。2005年10月，MySQL 5.0成为里程碑版本。
2008年1月被Sun公司收购。
2009年1月，在Oracle收购MySQL之前，Monty Widenius担心收购，就从MySQL Server 5.5开始一条新的GPL分支，起名MariaDB。
MySQL的引擎是插件化的，可以支持很多种引擎：
MyISASM，不支持事务，插入、查询速度快。
InnoDB，支持事务，行级锁，MySQL 5.5起的默认引擎

去IOE

它是阿里巴巴造出的概念。其本意是，在阿里巴巴的IT架构中，去掉IBM的小型机、Oracle数据库、EMC存储设备，取而代之使用自己在开源软件基础上开发的系统。传统上，一个高端大气的数据中心，IBM小型机、Oracle数据库、EMC存储设备，可以说缺一不可。而使用这些架构的企业，不但采购、维护成本极高，核心架构还掌握在他人手中。
对于阿里巴巴这样大规模的互联网应用，应该采用开源、开放的系统架构。这种思路并不是阿里巴巴的新发明，国外的谷歌、Facebook、亚马逊等早已为之。只不过它们几乎一开始就有没有采用IT商业公司的架构，所以他们也不用“去IOE”。
去IOE，转而使用廉价的架构，稳定性一定下降，需要较高的运维水平解决。

NoSQL

NoSQL是对非SQL、非传统关系型数据库的统称。
NoSQL一词诞生于1998年，2009年这个词汇被再次提出指非关系型、分布式、不提供ACID的数据库设计模式。

随着互联网时代的到来，数据爆发式增长，数据库技术发展日新月异，要适应新的业务需求。
随着移动互联网、物联网的到来，大数据的技术中NoSQL也同样重要。

数据库流行度排名 2017.12

Rank			DBMS	Database Model	Score		
Dec 2017	Nov 2017	Dec 2016			Dec 2017	Nov 2017	Dec 2016
1.	1.	1.	Oracle	Relational DBMS	1341.54	-18.51	-62.86
2.	2.	2.	MySQL	Relational DBMS	1318.07	-3.96	-56.34
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1172.48	-42.59	-54.17
4.	4.	4.	PostgreSQL	Relational DBMS	385.43	+5.51	+55.41
5.	5.	5.	MongoDB	Document store	330.77	+0.29	+2.09
6.	6.	6.	DB2	Relational DBMS	189.58	-4.48	+5.24
7.	7.	↑ 8.	Microsoft Access	Relational DBMS	125.88	-7.43	+1.18
8.	↑ 9.	↑ 9.	Redis	Key-value store	123.24	+2.05	+3.34
9.	↓ 8.	↓ 7.	Cassandra	Wide column store	123.21	-1.00	-11.07
10.	10.	↑ 11.	Elasticsearch	Search engine	119.78	+0.37	+16.51
11.	11.	↓ 10.	SQLite	Relational DBMS	115.19	+2.44	+4.36
12.	12.	12.	Teradata	Relational DBMS	74.74	-3.49	+1.37
13.	13.	↑ 14.	Solr	Search engine	66.30	-2.86	-2.70
14.	14.	↓ 13.	SAP Adaptive Server	Relational DBMS	65.68	-1.35	-4.74
15.	15.	↑ 16.	Splunk	Search engine	63.79	-1.08	+8.87
16.	16.	↓ 15.	HBase	Wide column store	63.41	-0.15	+4.79
17.	↑ 18.	↑ 20.	MariaDB	Relational DBMS	56.73	+1.44	+12.64
18.	↓ 17.	↓ 17.	FileMaker	Relational DBMS	55.20	-3.64	+1.08
19.	19.	19.	Hive	Relational DBMS	54.67	+1.42	+5.27

数据库流行度排名 2018.6

Rank			DBMS	Database Model	Score		
Jun 2018	May 2018	Jun 2017			Jun 2018	May 2018	Jun 2017
1.	1.	1.	Oracle	Relational DBMS	1311.25	+20.84	-40.51
2.	2.	2.	MySQL	Relational DBMS	1233.69	+10.35	-111.62
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1087.73	+1.89	-111.23
4.	4.	4.	PostgreSQL	Relational DBMS	410.67	+9.77	+42.13
5.	5.	5.	MongoDB	Document store	343.79	+1.67	+8.79
6.	6.	6.	DB2	Relational DBMS	185.64	+0.03	-1.86
7.	7.	↑ 9.	Redis	Key-value store	136.30	+0.95	+17.42
8.	↑ 9.	↑ 11.	Elasticsearch	Search engine	131.04	+0.60	+19.48
9.	↓ 8.	↓ 7.	Microsoft Access	Relational DBMS	130.99	-2.12	+4.44
10.	10.	↓ 8.	Cassandra	Wide column store	119.21	+1.38	-4.91
11.	11.	↓ 10.	SQLite	Relational DBMS	114.26	-1.19	-2.44
12.	12.	12.	Teradata	Relational DBMS	75.77	+1.36	-1.55
13.	↑ 14.	↑ 18.	MariaDB	Relational DBMS	65.85	+0.85	+12.95
14.	↓ 13.	↑ 16.	Splunk	Search engine	65.78	+0.68	+8.26
15.	15.	↓ 14.	Solr	Search engine	62.06	+0.55	-1.55
16.	16.	↓ 13.	SAP Adaptive Server	Relational DBMS	61.49	-0.02	-6.04
17.	17.	↓ 15.	HBase	Wide column store	59.70	-0.25	-2.17
18.	18.	↑ 20.	Hive	Relational DBMS	57.33	+0.36	+12.95
19.	19.	↓ 17.	FileMaker	Relational DBMS	56.18	+1.51	-0.90

MySQL

MySQL是一种关系型数据库管理软件，支持网络访问，默认服务端口3306。
MySQL通信使用mysql协议。

问题
MySQL应该基于什么网络协议通信？

连接字符串参考
https://dev.mysql.com/doc/connector-net/en/connector-net-connection-options.html
"server=127.0.0.1;uid=root;pwd=12345;database=test"

安装

使用yum安装rpm包，Centos自带的版本太低，建议官方下载安装5.5以上版本。
推荐安装MariaDB，或者Percona版本。
Percona安装在第十章介绍过，这里不再赘述。
找一台虚拟机，安装好MySQL，充当数据库服务器。

```
# yum install Percona-Server-shared-55-5.5.45-rel37.4.el6.x86_64.rpm Percona-Server-client-55-5.5.45-rel37.4.el6.x86_64.rpm Percona-Server-server-55-5.5.45-rel37.4.el6.x86_64.rpm
# service mysql start
# mysql_secure_installation
# mysql -uroot -p < test.sql
```

test.sql 为测试用脚本

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

SQL语句大小写不敏感。
SQL语句末尾应该使用分号结束。

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 gogs CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;
```

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。

```
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，被认为是非关键字。

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) DEFAULT NULL,
  `name` varchar(64) DEFAULT NULL,
  `password` varchar(128) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;
```

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。
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。

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

UNIQUE约束（唯一键约束）
定义了唯一键索引，就定义了唯一键约束。

PRIMARY KEY约束

定义了主键，就定义了主键约束。

外键约束Foreign Key

外键，在表B中的列，关联表A中的主键，表B中的列就是外键。
如果在表B插入一条数据，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。
修改表B的外键值也是同样，外键值同样要在表A中存在。
如果表A要删除一条记录，那么就等于删除一个主键，那么如果表B中引用到了这个主键，就必须先删除表B中引用这个主键的记录，然后才能删除表A的记录，否则删除失败。
修改表A的主键，由于主键的唯一性，修改的主键相当于插入新主键，那么表B引用过这个主键，将阻止表A的主键修改，必须删除表B的相关记录后，才可修改表A的主键。
外键约束，是为了保证数据完整性、一致性，杜绝数冗余、数据讹误。

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-9223372036854775808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。

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;

Delete语句

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

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;

-- 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。

```
-- 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

```
-- 工资40行
SELECT * FROM salaries;
-- 20行
SELECT * FROM employees;
--- 800行
SELECT * from employees CROSS JOIN salaries;
```

注意：departments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。

```
-- 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
看表的数据的方向，谁是主表，谁的所有数据都显示

自连接
表自己和自己连接

```sql
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;
```

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，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;
```

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。

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。
触发器功能虽然强大，但是会有性能问题。

这两种技术，虽然是数据库高级内容，但基本很少用了。