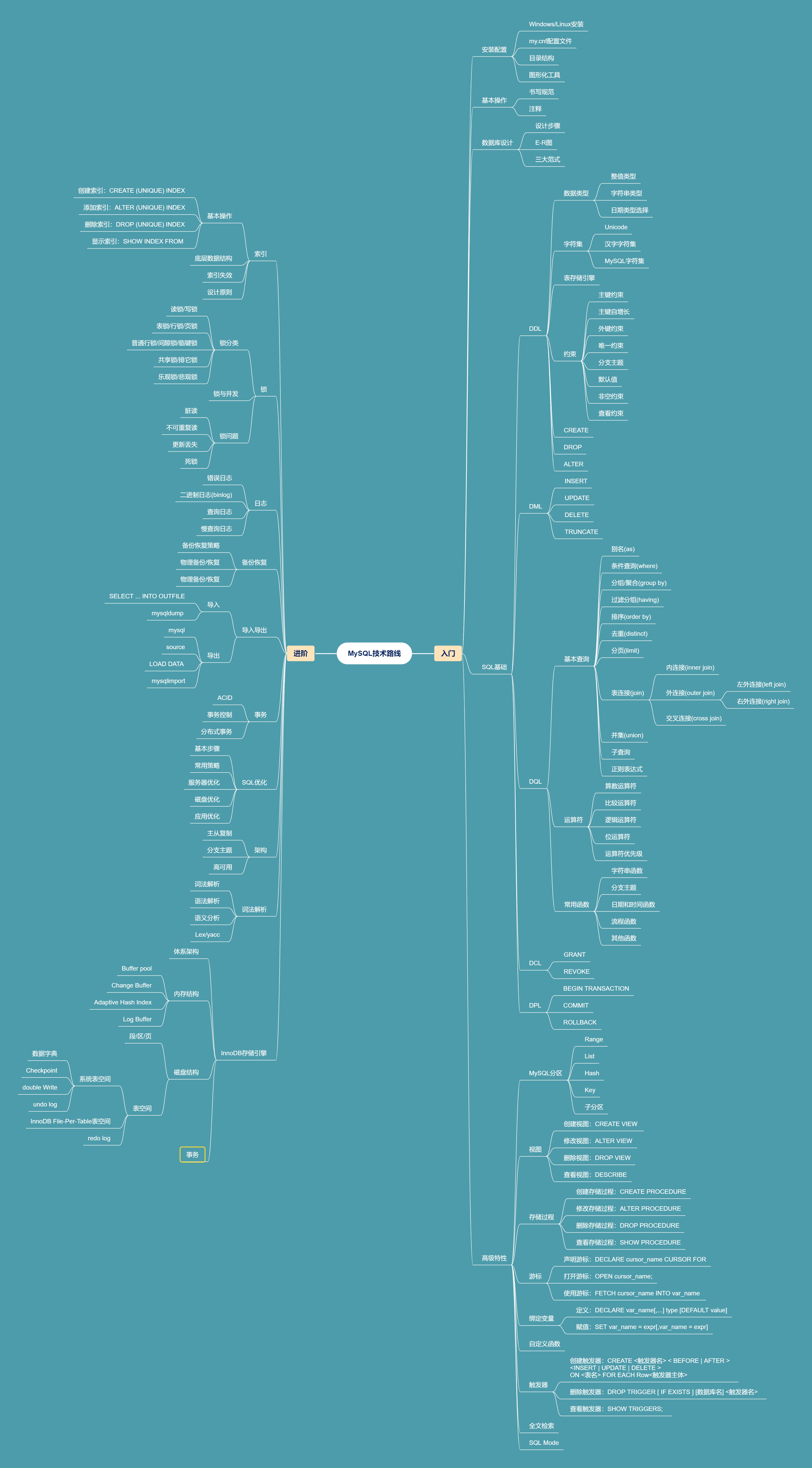
# 思维导图



# 入门

数据库入门首选要学会基本环境搭建，然后选择一款顺手的图形界面工具，学习各种SQL语句，包括基本建库建表的DDL语句，然后增删改查语句，再掌握了基本的操作之后，需要学会使用一些高级特性，比如视图、存储过程、游标、自定义函数等。

## 安装配置

### 安装配置

1. Windows和Linux系统MySQL安装
2. MySQL基本配置
3. 启动和关闭MySQL

### my.cnf配置文件

### 目录结构

1）bin 目录

用于放置一些可执行文件，如 mysql.exe、mysqld.exe、mysqlshow.exe 等。

2）docs 目录

存放一些文档

3）Data 目录

登录数据库后，可使用 SHOW GLOBAL VARIABLES LIKE "%Datadir%"; 命令查看 Data 目录位置。

4）include 目录

用于放置一些头文件，如：mysql.h、mysql\_ername.h 等。

5）lib 目录

用于放置一系列库文件

6）share 目录

用于存放字符集、语言等信息

7）my.ini 文件

### 图形化工具

#### MySQL Workbench

#### phpMyAdmin

#### Navicat

#### SQLyog

#### MySQLDumper

## 基本操作

### SQL基本书写规则

### 注释

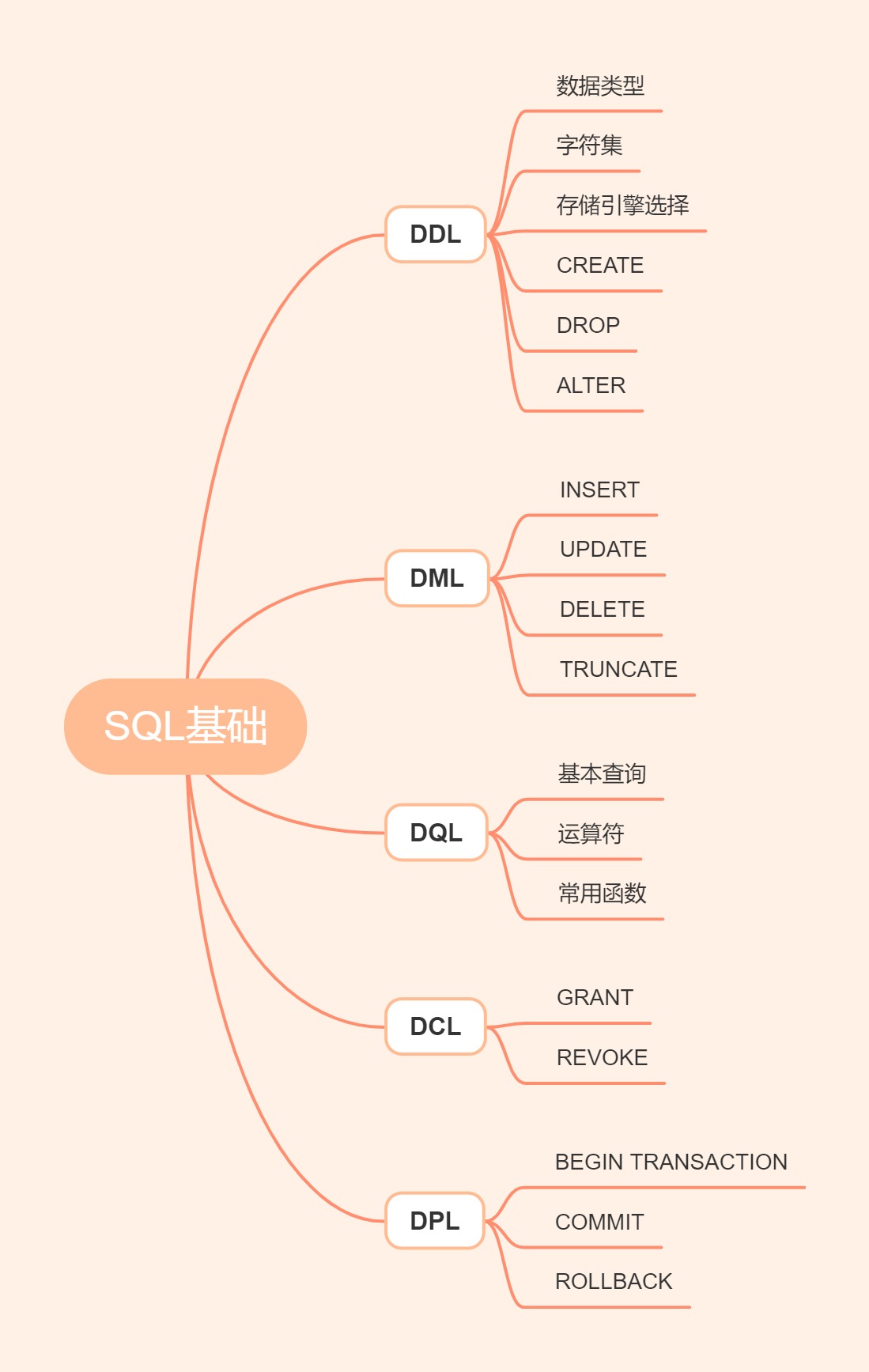
## 数据库设计

### 设计步骤

### E-R图

### 三大范式

## SQL基础



### DDL

#### 数据类型

1. 整值类型：

整数类型：TINYINT、SMALLINT、MEDIUMINT、INT、INTEGER、BIGINT

浮点数类型：FLOAT、DOUBLE

定点数类型：DEC、DECIMAL

位类型：BIT

1. 字符串类型：

CHAR/VARCHAR

BINARY、VARBINARY

TEXT、BLOB

ENUM、SET

1. 日期类型选择

DATE、DATETIME、TIMESTAMP、TIME、YEAR

#### 字符集

1. 概述
2. Unicode
3. 汉字编码
4. 字符集选择
5. MySQL字符集

#### 表类型选择

1. 存储引擎概述
2. 各种存储引擎基本特性
3. 如何选择存储引擎

#### 约束

##### 主键约束

<字段名> <数据类型> PRIMARY KEY [默认值]

##### 主键自增长

字段名 数据类型 AUTO\_INCREMENT

##### 外键约束

[CONSTRAINT <外键名>] FOREIGN KEY 字段名 [，字段名2，…]  
REFERENCES <主表名> 主键列1 [，主键列2，…]

##### 唯一约束

<字段名> <数据类型> UNIQUE

##### 检查约束

CHECK <表达式>

##### 默认值

<字段名> <数据类型> DEFAULT <默认值>;

##### 非空约束

<字段名> <数据类型> NOT NULL;

##### 查看约束

SHOW CREATE TABLE <数据表名>;

#### CREATE

##### 库

建库：

CREATE DATABASE [IF NOT EXISTS] <数据库名>

[[DEFAULT] CHARACTER SET <字符集名>]

[[DEFAULT] COLLATE <校对规则名>];

##### 表

建表：

CREATE TABLE <表名> ([表定义选项])[表选项][分区选项];

#### DROP

##### 库

DROP DATABASE [ IF EXISTS ] <数据库名>

##### 表

DROP TABLE [IF EXISTS] 表名1 [ ,表名2, 表名3 ...]

##### 字段

#### ALTER

##### 库

修改库：

ALTER DATABASE [数据库名] {

[ DEFAULT ] CHARACTER SET <字符集名> |

[ DEFAULT ] COLLATE <校对规则名>}

##### 表

修改表：

ALTER TABLE <表名> [修改选项]

##### 字段

ALTER TABLE <表名> CHANGE <旧字段名> <新字段名> <新数据类型>；

### DML

#### INSERT

INSERT VALUES 的语法格式为：

INSERT INTO <表名> [ <列名1> [ , … <列名n>] ]  
 VALUES (值1) [… , (值n) ];

INSERT…SELECT 语句向表中插入其他表的数据。

#### UPDATE

UPDATE <表名> SET 字段 1=值 1 [,字段 2=值 2… ] [WHERE 子句 ]  
[ORDER BY 子句] [LIMIT 子句]

#### DELETE

DELETE FROM <表名> [WHERE 子句] [ORDER BY 子句] [LIMIT 子句]

#### TRUNCATE

TRUNCATE [TABLE] 表名

### DQL

#### 基本查询

查询语句：

SELECT

{\* | <字段列名>}

[

FROM <表 1>, <表 2>…

[WHERE <表达式>

[GROUP BY <group by definition>

[HAVING <expression> [{<operator> <expression>}…]]

[ORDER BY <order by definition>]

[LIMIT[<offset>,] <row count>]

]

##### 别名

<表名> [AS] <别名>

##### 条件查询

WHERE 查询条件

查询条件可以是：

带比较运算符和逻辑运算符的查询条件

带 BETWEEN AND 关键字的查询条件

带 IS NULL 关键字的查询条件

带 IN 关键字的查询条件

带 LIKE 关键字的查询条件

###### 模糊查询

[NOT] LIKE  '字符串'

###### 范围查找

[NOT] BETWEEN 取值1 AND 取值2

###### 空值查询

IS [NOT] NULL

##### 分组/聚合

GROUP BY  <字段名>

##### 过滤分组

HAVING <查询条件>

##### 排序

ORDER BY <字段名> [ASC|DESC]

##### 去重

SELECT DISTINCT <字段名> FROM <表名>;

##### 分页

LIMIT 初始位置，记录数

##### 表连接

###### 内连接

SELECT <字段名> FROM <表1> INNER JOIN <表2> [ON子句]

###### 外连接

左外连接

SELECT <字段名> FROM <表1> LEFT OUTER JOIN <表2> <ON子句>

右外连接

SELECT <字段名> FROM <表1> RIGHT OUTER JOIN <表2> <ON子句>

###### 交叉连接

SELECT <字段名> FROM <表1> CROSS JOIN <表2> [WHERE子句]

或

SELECT <字段名> FROM <表1>, <表2> [WHERE子句]

##### 并集

##### 子查询

WHERE <表达式> <操作符> (子查询)

其中，操作符可以是比较运算符和 IN、NOT IN、EXISTS、NOT EXISTS 等关键字。

1）IN | NOT IN

当表达式与子查询返回的结果集中的某个值相等时，返回 TRUE，否则返回 FALSE；若使用关键字 NOT，则返回值正好相反。

2）EXISTS | NOT EXISTS

用于判断子查询的结果集是否为空，若子查询的结果集不为空，返回 TRUE，否则返回 FALSE；若使用关键字 NOT，则返回的值正好相反。

##### 正则表达式

#### 运算符

##### 算数运算符

加（+）、减（-）、乘（\*）、除（/，DIV）和模运算（%，MOD）

##### 比较运算符

=、<>/!=、<=>、<、<=、>、>=、BETWEEN、IN、IS NULL、IS NOT NULL、LIKE、REGEXP、RLIKE

##### 逻辑运算符

NOT/!、AND/&&、OR/||、XOR

##### 位运算符

&、|、^、~、>>、<<

##### 运算符优先级

#### 常用函数

##### 字符串函数

CONCAT()、INSERT()、LOWER()、UPPER()、LEFT()、RIGHT()、LPAD()、RPAD()、LTRIM()、RTRIM()、REPEAT()、REPLACE()、STRCMP()、TRIM()、SUBSTRING()

##### 数值函数

ABS()、CEIL()、FLOOR()、MOD()、RAND()、ROUND()、TRUNCATE()

##### 日期和时间函数

CURDATE()、CURTIME()、NOW()、UNIX\_TIMESTAMP()、FROM\_UNIXTIME()、WEEK()、YEAR()、HOUR()、MINUTE()、MONTHNAME()、DATE\_FORMAT()、DATE\_ADD()、DATEDIFF()

##### 流程函数

IF、IFNULL、CASE WHEN

##### 其他函数

DATABASE()、VERSION()、USER()、INET\_ATON()、INET\_NTOA()、PASSWORD()、MD5()

### DCL

#### 用户管理

##### 创建用户

CREATE USER <用户> [ IDENTIFIED BY [ PASSWORD ] 'password' ] [ ,用户 [ IDENTIFIED BY [ PASSWORD ] 'password' ]]

##### 修改用户

RENAME USER <旧用户> TO <新用户>

其中：

<旧用户>：系统中已经存在的 MySQL 用户账号。

<新用户>：新的 MySQL 用户账号。

##### 删除用户

DROP USER <用户1> [ , <用户2> ]…

其中，用户用来指定需要删除的用户账号。

##### 查看用户权限

在 MySQL 中，可以通过查看 mysql.user 表中的数据记录来查看相应的用户权限，也可以使用 SHOW GRANTS 语句查询用户的权限。  
 mysql 数据库下的 user 表中存储着用户的基本权限，可以使用 SELECT 语句来查看。SELECT 语句的代码如下：

SELECT \* FROM mysql.user;

除了使用 SELECT 语句之外，还可以使用 SHOW GRANTS FOR 语句查看权限。其语法格式如下：

SHOW GRANTS FOR 'username'@'hostname';

其中，username 表示用户名，hostname 表示主机名或主机 IP。

#### GRANT

GRANT priv\_type [(column\_list)] ON database.table

TO user [IDENTIFIED BY [PASSWORD] 'password']

[, user[IDENTIFIED BY [PASSWORD] 'password']] ...

[WITH with\_option [with\_option]...]

#### REVOKE

REVOKE priv\_type [(column\_list)]...  
 ON database.table  
 FROM user [, user]...

#### 密码管理

##### 修改密码

SET PASSWORD FOR 'username'@'hostname' = PASSWORD ('newpwd');

##### 重置密码

### DPL

#### BEGIN TRANSACTION

#### COMMIT

#### ROLLBACK

## 高级特性



### MySQL分区

1. 什么是分区？
2. 分区类型：Range、List、Columns、Hash、Key、子分区
3. 分区管理
4. 分区与性能

### 视图

1. 什么是视图？
2. 视图基本操作：

#### 创建视图

CREATE VIEW <视图名> AS <SELECT语句>

#### 修改视图

ALTER VIEW <视图名> AS <SELECT语句>

#### 删除视图

DROP VIEW <视图名1> [ , <视图名2> …]

#### 查看视图

DESCRIBE 视图名;

或简写成：

DESC 视图名;

### 存储过程

1. 什么是存储过程
2. 存储过程基本操作

#### 创建存储过程

CREATE PROCEDURE <过程名> ( [过程参数[,…] ] ) <过程体>  
 [过程参数[,…] ] 格式  
 [ IN | OUT | INOUT ] <参数名> <类型>

#### 修改存储过程

ALTER PROCEDURE 存储过程名 [ 特征 ... ]

#### 删除存储过程

DROP PROCEDURE [ IF EXISTS ] <过程名>

#### 查看存储过程

SHOW PROCEDURE STATUS LIKE 存储过程名;

### 游标

#### 声明游标

DECLARE cursor\_name CURSOR FOR select\_statement;

#### 打开游标

OPEN cursor\_name;

#### 使用游标

FETCH cursor\_name INTO var\_name [,var\_name]...

#### 关闭游标

CLOSE cursor\_name;

### 绑定变量

#### 定义变量

DECLARE var\_name[,...] type [DEFAULT value]

#### 为变量赋值

SET var\_name = expr[,var\_name = expr]...

### 自定义函数

CREATE FUNCTION sp\_name ([func\_parameter[...]])  
 RETURNS type  
 [characteristic ...] routine\_body

### 触发器

1. 什么是触发器
2. 触发器基本操作：

#### 创建触发器

CREATE <触发器名> < BEFORE | AFTER >  
 <INSERT | UPDATE | DELETE >  
 ON <表名> FOR EACH Row<触发器主体>

#### 删除触发器

DROP TRIGGER [ IF EXISTS ] [数据库名] <触发器名>

#### 查看触发器

SHOW TRIGGERS;

### 全文索引

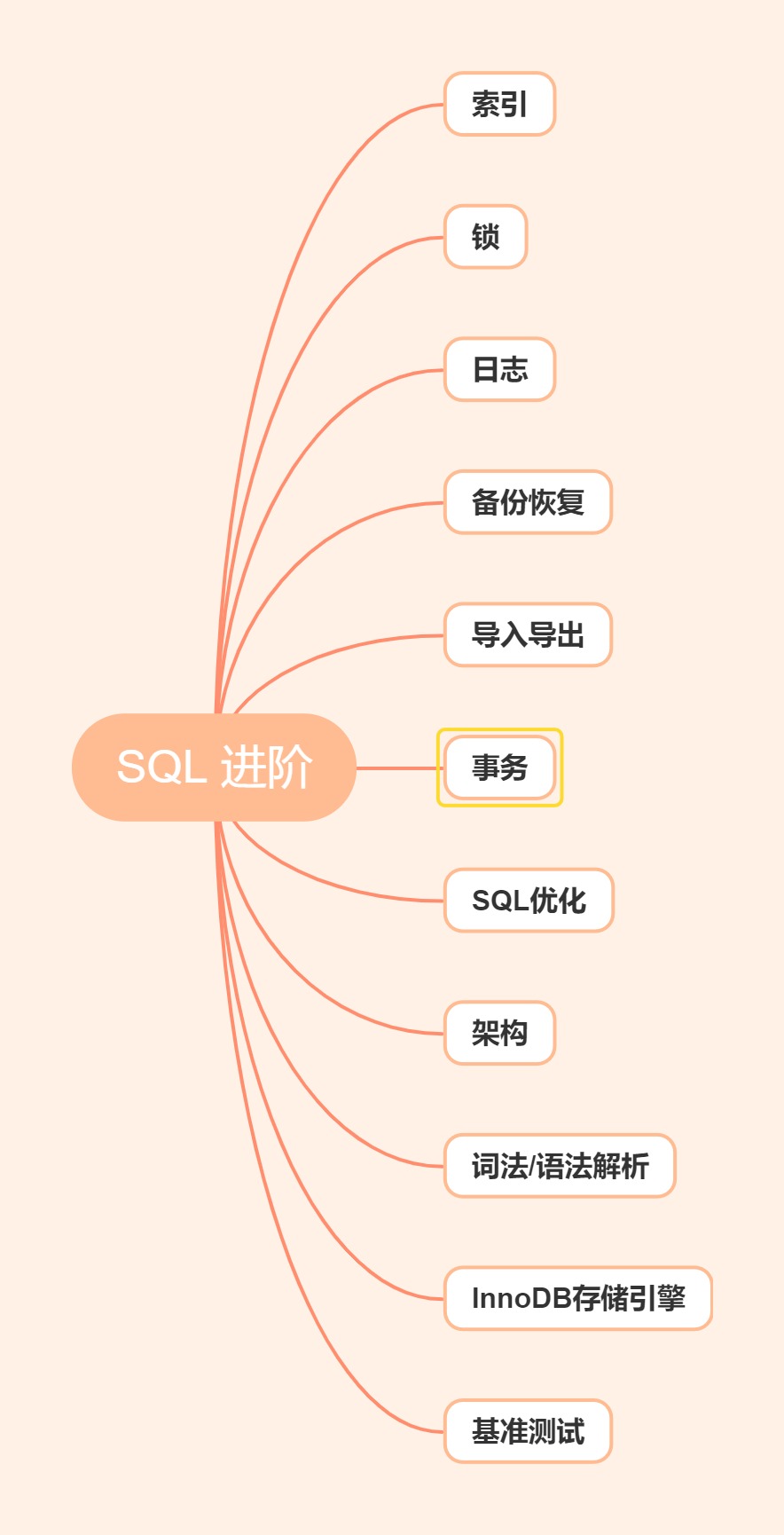
### SQLMode

1. 什么是SQL Mode？
2. 常见SQL Mode
3. 数据迁移中使用SQL Mode

# 进阶

了解基本SQL操作和高级特性可以基本满足平时开发的需要，但是如果真正从事数据库底层开发，还需要了解这些SQL语句背后的算法，这就涉及索引实现及优化，锁的使用，事务控制，SQL优化。

如果需要掌握一个庞大的数据库系统的操作和运维，还需要掌握主从复制、备份恢复、高可用、基准测试。



## 索引

### 基本操作

#### 创建索引

CREATE (UNIQUE) INDEX indexName ON table\_name (column\_name)

#### 添加索引

ALTER table tableName ADD (UNIQUE) INDEX indexName(columnName)

#### 删除索引

DROP (UNIQUE) INDEX [indexName] ON mytable;

#### 显示索引

SHOW INDEX FROM table\_name; \G........

### 底层数据结构

#### 二叉树

#### B树

#### B+树

### 索引失效

### 索引设计原则及优化

## 锁

### 锁分类

#### 读锁/写锁

#### 表锁/行锁/页锁

#### 共享锁/排它锁

#### 乐观锁/悲观锁

### 锁与并发控制

#### 锁

#### 无锁：MVCC

### 死锁

## 日志

### 错误日志

### 二进制日志

### 查询日志

### 慢查询日志

## 备份恢复

### 备份恢复策略

### 逻辑备份与恢复

### 物理备份与恢复

## 导入导出

### 导入

#### SELECT ... INTO OUTFILE

#### mysqldump

### 导出

#### mysql

使用 mysql 命令导入语法格式为：

mysql -u用户名 -p密码 < 要导入的数据库数据(runoob.sql)

#### source

source 命令导入数据库需要先登录到数库终端：

mysql> create database abc; # 创建数据库

mysql> use abc; # 使用已创建的数据库

mysql> set names utf8; # 设置编码

mysql> source /home/abc/abc.sql # 导入备份数据库

#### LOAD DATA

mysql> LOAD DATA LOCAL INFILE 'dump.txt' INTO TABLE mytbl;

#### mysqlimport

$ mysqlimport -u root -p --local mytbl dump.txt

## 事务

### 什么是事务？

### ACID实现原理

### 事务控制

### 分布式事务

## SQL优化

### SQL优化的基本步骤

### 常用优化策略

### 服务器优化

### 磁盘存储优化

### 应用优化

## 架构

### 主从复制

1. 主从复制原理
2. 启动参数
3. 主从切换

### MySQL Cluster

### 高可用

1、高可用架构

## 词法解析/语法解析

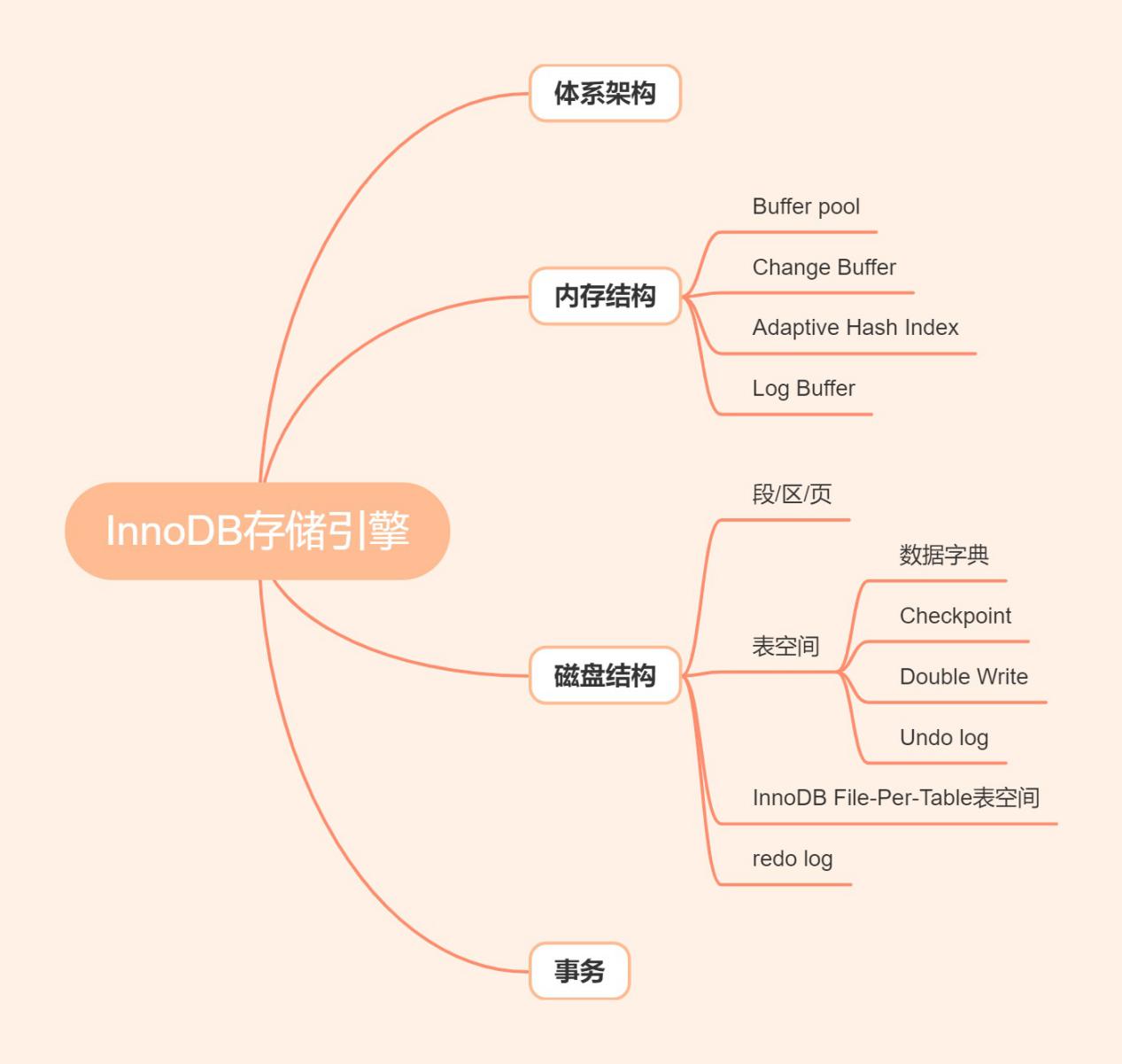
### 词语解析

### 语法解析

### 语义分析

### Lex/yacc

## InnoDB存储引擎



### 体系架构

### 内存结构

<https://www.jianshu.com/p/d4cc0ea9d097>

#### Buffer pool

缓冲池缓存的数据包括Page Cache、Change Buffer、Data Dictionary Cache等，通常 MySQL 服务器的 80% 的物理内存会分配给 Buffer Pool。

#### Change Buffer

通常来说，InnoDB辅助索引不同于聚集索引的顺序插入，如果每次修改二级索引都直接写入磁盘，则会有大量频繁的随机IO。Change buffer 的主要目的是将对 非唯一 辅助索引页的操作缓存下来，以此减少辅助索引的随机IO，并达到操作合并的效果。它会占用部分Buffer Pool 的内存空间。在 MySQL5.5 之前 Change Buffer其实叫 Insert Buffer，最初只支持 insert 操作的缓存，随着支持操作类型的增加，改名为 Change Buffer。如果辅助索引页已经在缓冲区了，则直接修改即可；如果不在，则先将修改保存到 Change Buffer。Change Buffer的数据在对应辅助索引页读取到缓冲区时合并到真正的辅助索引页中。Change Buffer 内部实现也是使用的 B+树。

#### Adaptive Hash Index

自适应哈希索引(AHI)查询非常快，一般时间复杂度为 O(1)，相比 B+ 树通常要查询 3~4次，效率会有很大提升。innodb 通过观察索引页上的查询次数，如果发现建立哈希索引可以提升查询效率，则会自动建立哈希索引，称之为自适应哈希索引，不需要人工干预，可以通过 innodb\_adaptive\_hash\_index 开启，MySQL5.7 默认开启。

考虑到不同系统的差异，有些系统开启自适应哈希索引可能会导致性能提升不明显，而且为监控索引页查询次数增加了多余的性能损耗， MySQL5.7 更改了 AHI 实现机制，每个 AHI 都分配了专门分区，通过 innodb\_adaptive\_hash\_index\_parts配置分区数目，默认是8个，如前一节命令列出所示。

#### Log Buffer

Log Buffer是 重做日志在内存中的缓冲区，大小由 innodb\_log\_buffer\_size 定义，默认是 16M。一个大的 Log Buffer可以让大事务在提交前不必将日志中途刷到磁盘，可以提高效率。如果你的系统有很多修改很多行记录的大事务，可以增大该值。

配置项 innodb\_flush\_log\_at\_trx\_commit 用于控制 Log Buffer 如何写入和刷到磁盘。注意，除了 MySQL 的缓冲区，操作系统本身也有内核缓冲区。

默认为1，表示每次事务提交都会将 Log Buffer 写入操作系统缓存，并调用配置的 "flush" 方法将数据写到磁盘。设置为 1 因为频繁刷磁盘效率会偏低，但是安全性高，最多丢失 1个 事务数据。而设置为 0 和 2 则可能丢失 1秒以上 的事务数据。

为 0 则表示每秒才将 Log Buffer 写入内核缓冲区并调用 "flush" 方法将数据写到磁盘。

为 2 则是每次事务提交都将 Log Buffer写入内核缓冲区，但是每秒才调用 "flush" 将内核缓冲区的数据刷到磁盘。

### 磁盘结构

#### 段/区/页

#### 表空间

##### 系统表空间

###### 数据字典

###### Checkpoint

###### Double Write

###### Undo

#### InnoDB File-Per-Table表空间

#### redo log

### 事务

#### 隔离级别

#### ACID

## 测试

### 基准测试

### 测试工具

# 拓展

## Redis

### 什么是NoSQL？

### 数据类型

### 基本操作

### 持久化

### 事务

### 主从复制

### 集群

## MongoDB

### 索引

### 事务

## Hive SQL

# 学习资料

## 推荐书籍

《SQL必知必会》

## 技术博客

## 教学视频