第6章: 物理数据库设计

Physical Database Design

邹兆年

哈尔滨工业大学
计算机科学与技术学院
海量数据计算研究中心
电子邮件: znzou@hit.edu.cn

2019年春

教学内容[1]

1. 物理数据库设计概述
2. 索引的概念
3. 索引的功能
4. 索引的设计
5. 物理存储结构的设计

[1]课件更新于2019年3月26日

6.1 物理数据库设计概述

Introduction to Physical Database Design

物理数据库设计概述

- 任务: 在逻辑数据库设计的基础上，为每个关系模式选择合适的存储结构和存取方法，使得数据库上的事务能够高效率地运行
- 设计步骤:
  1. 分析影响物理数据库设计的因素
  2. 为关系模式选择存取方法
  3. 设计关系、索引等数据库文件的物理存储结构

设计步骤1: 分析影响物理数据库设计的因素

- 对于数据库查询事务，需得到如下信息
  - 查询涉及的关系
  - 查询设计的属性: 选择条件涉及的属性、连接条件涉及的属性、投影属性
- 对于数据库更新事务，需得到如下信息
  - 被更新的关系
  - 每个关系上更新操作的类型(增加、删除、修改)
  - 删除和修改操作的条件所涉及的属性
  - 修改操作要修改的属性
- 了解每个事务在各关系上的执行频率
- 了解每个事务的执行时间约束

设计步骤2: 为关系模式选择存取方法

- 存取方法(access methods): 访问数据记录的方法
  - 顺序扫描
  - 利用索引
- 索引(index)是提高数据库系统性能的最重要的手段之一
  - 索引的概念(第6.2节)
  - 索引的功能(第6.3节)
  - 索引的设计(第6.4节): 根据步骤1中获得的影响物理数据库设计的因素，合理地设计索引
    - ★ 减少查询处理的时间
    - ★ 不显著增加索引维护的代价

设计步骤3: 设计关系及索引的物理存储结构

- 确定如何在存储器上存储关系及索引，使得存储空间利用率最大化，数据操作产生的系统开销最小化
- RDBMS使用的物理存储结构取决于存储引擎，用户可以选择合适的存储引擎，但不能改变存储引擎使用的物理存储结构

6.2 索引的概念

Indexes

什么是索引?

- 索引(index): DBMS用于快速查找数据记录的数据结构
- 索引对于改善数据库系统的性能至关重要

Example (索引)

例: 如果Student关系的Sno属性上建有索引，则DBMS将使用该索引快速找到Sno为221101的学生记录

```
SELECT Sname FROM Student WHERE Sno = '221101';
```

- 索引属性(索引键): 索引中用于查找数据记录的属性
  - 索引可以包含1个或多个属性
  - 如果索引包含多个属性，那么属性的顺序也非常重要

索引的结构

- 索引本质上存储了数据记录的索引属性值与数据记录地址之间的映射关系
- 索引中每个(键值，记录地址)对称为一个索引项(entry)
- 注意: 索引中通常存储数据记录的地址，而不是数据记录本身。存储数据记录本身的索引称为聚簇索引(clustered index)，又称索引组织表(index-organized table)

Example (主索引)

主索引

Sno	记录地址
221101	$addr_1$
231101	$addr_2$
231102	$addr_3$
232101	$addr_4$

地址
$addr_1$
$addr_2$
$addr_3$
$addr_4$

Student

Sno	Sname	Ssex	Sage	Sdept
221101	Nick	M	20	Physics
231101	Elsa	F	19	CS
231102	Eric	M	19	CS
232101	Abby	F	18	Math

索引的类型

主索引(primiary index) vs. 二级索引(secondary index)

- 如果索引属性是关系的主键，则该索引为主索引
- 如果索引属性不是关系的主键，则该索引为二级索引

Example (二级索引)

二级索引

Sname	记录地址
Abby	$addr_4$
Elsa	$addr_2$
Eric	$addr_3$
Nick	$addr_1$

Student

地址	Sno	Sname	Ssex	Sage	Sdept
$addr_1$	221101	Nick	M	20	Physics
$addr_2$	231101	Elsa	F	19	CS
$addr_3$	231102	Eric	M	19	CS
$addr_4$	232101	Abby	F	18	Math

索引的类型(续)

唯一索引(unique index) vs. 非唯一索引(non-unique index)

- 如果索引属性值不可以重复，则该索引为唯一索引
- 如果索引属性值可重复，则该索引为非唯一索引

聚簇索引(clustered index) vs. 非聚簇索引(non-clustered index)

- 如果索引里存储的是数据记录本身，则该索引为聚簇索引
- 如果索引里存储的是数据记录的地址，则该索引为非聚簇索引
- 一个关系上只有一个聚簇索引

Example (聚簇索引)

聚簇索引

地址	Sno	Sname	Ssex	Sage	Sdept
$addr_1$	221101	Nick	M	20	Physics
$addr_2$	231101	Elsa	F	19	CS
$addr_3$	231102	Eric	M	19	CS
$addr_4$	232101	Abby	F	18	Math

索引的类型(续)

MySQL的聚簇索引与二级索引

- 在MySQL的InnoDB存储引擎中，主索引采用聚簇索引，其他索引都是二级索引，而且都是非聚簇索引
- 二级索引存储的不是数据记录的地址，而是主键值
- 好处？坏处？

Example (MySQL的聚簇索引与二级索引)

二级索引

Sname	Sno
Abby	232101
Elsa	231101
Eric	231102
Nick	221101

Student (聚簇索引)

Sno	Sname	Ssex	Sage	Sdept
221101	Nick	M	20	Physics
231101	Elsa	F	19	CS
231102	Eric	M	19	CS
232101	Abby	F	18	Math

创建索引

创建主索引

- 在CREATE TABLE或ALTER TABLE语句中使用PRIMARY KEY声明主键时，自动建立主索引
- 可以用ASC或DESC声明主索引中属性按升序还是降序排列
- 在MySQL中，主索引的名称就是PRIMARY
- 只能在CREATE TABLE或ALTER TABLE语句中声明主键

Example (创建主索引)

```
CREATE TABLE Student (
Sno CHAR(6),
Sname VARCHAR(10),
Ssex ENUM('M', 'F'),
Sage INT,
Sdept VARCHAR(20),
PRIMIARY KEY (Sno ASC));
```

创建索引(续)

创建二级索引

- 在CREATE TABLE或ALTER TABLE语句中，使用KEY|INDEX [索引名]声明二级索引
- 可以为二级索引取名字

Example (创建二级索引)

```
CREATE TABLE Student (
Sno CHAR(6) PRIMARY KEY,
Sname VARCHAR(10),
Ssex ENUM('M', 'F'),
Sage INT,
Sdept VARCHAR(20),
KEY (Sname, Sage DESC) key_sname_sage);
```


创建索引(续)

创建唯一索引

- 在CREATE TABLE或ALTER TABLE语句中，使用UNIQUE [KEY|INDEX] [索引名]声明唯一索引

Example (创建唯一索引)

```
CREATE TABLE Student (
Sno CHAR(6) PRIMARY KEY,
Sname VARCHAR(10),
Ssex ENUM('M', 'F'),
Sage INT,
Sdept VARCHAR(20),
UNIQUE (Sname) uk_sname);
```

创建外键索引

- 在CREATE TABLE或ALTER TABLE语句中，使用FOREIGN KEY [索引名]声明外键时，也会为外键创建索引

创建索引(续)

- 尽量在CREATE TABLE语句中将一个关系上的所有索引建好
- 可以使用ALTER TABLE语句增加索引
- 也可以使用CREATE INDEX语句增加除主键外的其他索引

```
CREATE [UNIQUE] INDEX 索引名
ON 关系名 (属性名1 [ASC|DESC], ..., 属性名k [ASC|DESC]);
```


删除索引

- 删除二级索引

```
DROP INDEX 索引名 ON 关系名;
```

- 删除主索引

```
DROP INDEX 'PRIMARY' ON 关系名;
```

6.3 索引的功能

Functions of Indexes

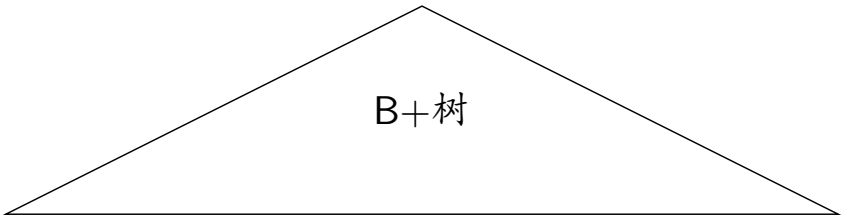
索引结构的类型

索引的功能取决于索引的数据结构

- B+树索引
- 哈希索引(hash index)
- 空间索引(spatial index)

B+树索引

- B+树是大多数RDBMS所使用的索引结构
- B+树是一棵平衡多叉树，所有叶子节点的深度都相同
- 索引项全部存储在B+树的叶子节点中，并按索引键值排序存储



221101	$addr_1$	231101	$addr_2$	231102	$addr_3$	232101	$addr_4$

Student

地址	Sno	Sname	Ssex	Sage	Sdept
$addr_1$	221101	Nick	M	20	Physics
$addr_2$	231101	Elsa	F	19	CS
$addr_3$	231102	Eric	M	19	CS
$addr_4$	232101	Abby	F	18	Math

B+树索引支持的查询类型

```
CREATE INDEX idx ON Student (Sname, Sage, Ssex) USING BTREE;
```

1. 全值匹配: 和所有索引属性进行匹配

```
SELECT * FROM Student WHERE Sname = 'Elsa' AND Sage = 19;
```

2. 匹配最左前缀: 和最前面几个索引属性进行匹配

```
SELECT * FROM Student WHERE Sname = 'Elsa';
```

3. 匹配属性前缀: 只匹配某属性的开头部分

```
SELECT * FROM Student WHERE Sname LIKE 'E%';
```

4. 范围匹配: 在给定范围内对某属性进行匹配

```
SELECT * FROM Student
WHERE Sname BETWEEN 'Ella' AND 'Emma';
```

5. 精确匹配某一列并范围匹配另一列

```
SELECT * FROM Student
WHERE Sname = 'Elsa' AND Sage BETWEEN 18 AND 20;
```

6. B+树索引还支持按索引属性进行排序

B+树索引支持的限制

```
CREATE INDEX idx ON Student (Sname, Sage, Sex) USING BTREE;
```

1. 必须从索引的最左属性开始查找

   SELECT * FROM Student WHERE Sage = 19; (不能使用索引)
2. 不能跳过索引中的属性

   SELECT * FROM Student
   WHERE Sname = 'Elsa' AND Ssex = 'F'; (不能使用索引)
3. 如果查询中有关于某个属性的范围查询，则其右边所有属性都无法使用索引查找

   SELECT * FROM Student
   WHERE Sname = 'Elsa' AND Sage BETWEEN 18 AND 20
   AND Ssex = 'F'; (不能使用索引查找Ssex)

哈希索引

- 哈希索引是基于哈希表(hash table)实现的[2]
- 哈希表中的索引项是(索引键值的哈希值, 数据记录地址)
- 哈希索引只支持对所有索引属性的精确匹配

Example (哈希索引)

哈希索引

h(Sname)	地址
2323	$addr_2$
2458	$addr_3$
7437	$addr_1$
8784	$addr_4$

地址
$addr_1$
$addr_2$
$addr_3$
$addr_4$

Student

Sno	Sname	Ssex	Sage	Sdept
221101	Nick	M	20	Physics
231101	Elsa	F	19	CS
231102	Eric	M	19	CS
232101	Abby	F	18	Math

- h('Abby') = 7437
- h('Elsa') = 2323
- h('Eric') = 2458
- h('Nick') = 8784

[2]MySQL中只有MEMORY存储引擎支持哈希索引，创建哈希索引时使用USING HASH

哈希索引的限制

```
CREATE INDEX idx ON Student (Sname, Sage, Sex) USING HASH;
```

- 哈希索引不支持部分索引属性匹配

  SELECT * FROM Student WHERE Sage = 19; (不能使用索引)
- 哈希索引只支持等值比较查询(=, IN)，不支持范围查询

  SELECT * FROM Student WHERE Sname = 'Elsa' AND Sage < 19 AND Ssex = 'F'; (不能使用索引)
- 哈希索引并不是按照索引值排序存储的，所以无法用于排序
- 哈希索引存在冲突问题

6.4 索引的设计

Design of Indexes

设计策略1: 伪哈希索引

- 尽管有些存储引擎不支持哈希索引，但我们可以模拟哈希索引

Example (伪哈希索引)

在Student关系上创建Sname属性上的伪哈希索引

1. 在Student中增加SnameHash属性，存储Sname属性的哈希值CRC32(Sname)
2. 删除Sname上的索引，创建SnameHash上的索引
3. 在查询时，对查询语句进行修改

```
SELECT * FROM Student
WHERE Sname = 'Elsa' AND SnameHash = CRC32('Elsa');
```

伪哈希索引

S.H.	地址
2323	$addr_2$
2458	$addr_3$
7437	$addr_1$
8784	$addr_4$

Student

地址	Sno	Sname	Ssex	Sage	Sdept	S.H.
$addr_1$	221101	Nick	M	20	Physics	7437
$addr_2$	231101	Elsa	F	19	CS	2323
$addr_3$	231102	Eric	M	19	CS	2458
$addr_4$	232101	Abby	F	18	Math	8784

设计策略1: 伪哈希索引(续)

- 优点: 查询速度快
- 缺点:
  - 仅支持等值查询，不支持范围查询
  - 需要改写查询
  - 需要在数据更新时维护哈希值属性

Example (伪哈希索引)

```
SELECT * FROM Student
WHERE Sname = 'Elsa' AND SnameHash = CRC32('Elsa');
```

伪哈希索引

S.H.	地址
2323	$addr_2$
2458	$addr_3$
7437	$addr_1$
8784	$addr_4$

Student

地址	Sno	Sname	Ssex	Sage	Sdept	S.H.
$addr_1$	221101	Nick	M	20	Physics	7437
$addr_2$	231101	Elsa	F	19	CS	2323
$addr_3$	231102	Eric	M	19	CS	2458
$addr_4$	232101	Abby	F	18	Math	8784

设计策略2: 前缀索引

- 当索引很长的字符串时，索引会变得很大，而且很慢
- 当字符串的前缀(prefix)具有较好的选择性时，可以只索引字符串的前缀
  - 例: 'E', 'El', 'Els'都是'Elsa'的前缀

Definition (索引的选择性(selectivity))

不重复的索引键值和数据记录数量N的比值，即$\frac{\texttt{COUNT(DISTINCT A)}}{\texttt{COUNT}(*)}$，范围在$1/N$和1之间。选择性越高，索引的过滤能力越强

Example (前缀索引，前缀长度= 1，选择性= 0.75)

```
CREATE INDEX idx1 ON Student (Sname(1));
```

前缀索引idx1

Sname(1)	地址
A	$addr_4$
E	$addr_2$
E	$addr_3$
N	$addr_1$

Student

地址	Sno	Sname	Ssex	Sage	Sdept
$addr_1$	221101	Nick	M	20	Physics
$addr_2$	231101	Elsa	F	19	CS
$addr_3$	231102	Eric	M	19	CS
$addr_4$	232101	Abby	F	18	Math

设计策略2: 前缀索引(续)

Example (前缀索引，前缀长度= 2，选择性= 1 $\star$)

```
CREATE INDEX idx2 ON Student (Sname(2));
```

前缀索引idx2

Sname(2)	地址
Ab	$addr_4$
El	$addr_2$
Er	$addr_3$
Ni	$addr_1$

Student

地址	Sno	Sname	Ssex	Sage	Sdept
$addr_1$	221101	Nick	M	20	Physics
$addr_2$	231101	Elsa	F	19	CS
$addr_3$	231102	Eric	M	19	CS
$addr_4$	232101	Abby	F	18	Math

Example (前缀索引，前缀长度= 3，选择性= 1)

```
CREATE INDEX idx3 ON Student (Sname(3));
```

前缀索引idx3

Sname(3)	地址
Abb	$addr_4$
Els	$addr_2$
Eri	$addr_3$
Nic	$addr_1$

Student

地址	Sno	Sname	Ssex	Sage	Sdept
$addr_1$	221101	Nick	M	20	Physics
$addr_2$	231101	Elsa	F	19	CS
$addr_3$	231102	Eric	M	19	CS
$addr_4$	232101	Abby	F	18	Math

设计策略2: 前缀索引(续)

前缀索引的缺点

- 前缀索引不支持排序(ORDER BY)
- 前缀索引不支持分组查询(GROUP BY)

Example (前缀索引，前缀长度= 1，选择性= 0.75)

```
CREATE INDEX idx1 ON Student (Sname(1));
```

前缀索引idx1

Sname(1)	地址
A	$addr_4$
E	$addr_2$
E	$addr_3$
N	$addr_1$

Student

地址	Sno	Sname	Ssex	Sage	Sdept
$addr_1$	221101	Nick	M	20	Physics
$addr_2$	231101	Elsa	F	19	CS
$addr_3$	231102	Eric	M	19	CS
$addr_4$	232101	Abby	F	18	Math

Question

*Sno*属性上适合建前缀索引吗?

设计策略3: 单个多属性索引 vs. 多个单属性索引

```
SELECT * FROM Student
WHERE Sname = 'Elsa' AND Sage = 19 AND Ssex = 'F';
```

单个多属性索引

```
CREATE INDEX idx ON Student (Sname, Sage, Sex);
```

该索引能直接支持上面的查询

多个单属性索引

```
CREATE INDEX idx1 ON Student (Sname);
CREATE INDEX idx2 ON Student (Sage);
CREATE INDEX idx3 ON Student (Sex);
```

1. 在idx1上查找Sname = 'Elsa'的记录地址
2. 在idx2上查找Sage = 19的记录地址
3. 在idx3上查找Ssex = 'F'的记录地址
4. 索引合并(index merge): 对上述3个记录地址集合取交集，再取回数据记录

设计策略3: 单个多属性索引 vs. 多个单属性索引(续)

多个单属性索引的缺点

- 效率没有在单个多属性索引上做查询高
- 索引合并涉及排序，要消耗大量计算和存储资源
- 在查询优化时，索引合并的代价并不被计入，故“低估”了查询代价

设计策略4: 索引属性的顺序

在属性Sname, Sage, Ssex上可以建立6种不同顺序的索引，哪种好?

1. `CREATE INDEX idx1 ON Student (Sname, Sage, Sex);`
2. `CREATE INDEX idx2 ON Student (Sname, Ssex, Sage);`
3. `CREATE INDEX idx3 ON Student (Sage, Sname, Sex);`
4. `CREATE INDEX idx4 ON Student (Sage, Ssex, Sname);`
5. `CREATE INDEX idx5 ON Student (Ssex, Sname, Sage);`
6. `CREATE INDEX idx6 ON Student (Ssex, Sage, Sname);`

经验法则

当不考虑排序(ORDER BY)和分组(GROUP BY)时，将选择性最高的属性放在最前面通常是好的，可以更快地过滤掉不满足条件的记录

设计策略5: 聚簇索引

优点

- 相关数据保存在一起，可以减少磁盘I/O
- 无需“回表”，数据访问更快

缺点

- 设计聚簇索引是为了提高I/O密集型应用的性能，如果数据全部在内存中，那么聚簇索引就没什么优势了
- 聚簇索引上记录插入的速度严重依赖于记录插入的顺序
- 更新聚簇索引属性值的代价很高，需要将每个被更新的记录移动到新的位置
- 因为每条记录都很大，需要占用更多的存储空间，全表扫描变慢

Example (聚簇索引)

聚簇索引

Sno	Sname	Ssex	Sage	Sdept
221101	Nick	M	20	Physics
221102	Mike	M	19	Physics
231101	Elsa	F	19	CS
231102	Eric	M	19	CS
232101	Abby	F	18	Math

设计策略5: 聚簇索引(续)

主键上建立聚簇索引

```
CREATE TABLE Student (
Sno CHAR(16) PRIMARY KEY,
Sname VARCHAR(10),
Ssex ENUM('M', 'F'),
Sage INT,
Sdept VARCHAR(20));
```

- 优点: 记录按Sno属性聚集存储
- 缺点:
  - 若不按Sno递增顺序插入记录，速度会很慢
  - 二级索引的索引项中存储Sno的值，导致二级索引很大

Example (主键上的聚簇索引)

二级索引

Sname	主键值
Abby	232101
Elsa	231101
Eric	231102
Nick	221101

Student (聚簇索引)

Sno	Sname	Ssex	Sage	Sdept
221101	Nick	M	20	Physics
231101	Elsa	F	19	CS
231102	Eric	M	19	CS
232101	Abby	F	18	Math

设计策略5: 聚簇索引(续)

代理键(surrogate key)上建立聚簇索引

```
CREATE TABLE Student (
Sno CHAR(16),
Sname VARCHAR(10),
Ssex ENUM('M', 'F'),
Sage INT,
Sdept VARCHAR(20),
ID INT AUTO_INCREMENT,
PRIMARY KEY (ID));
```

- 主键ID与应用无关，称为代理键
- 优点:
  - 一定按ID递增顺序插入记录，插入速度快
  - 二级索引的索引项中存储ID的值，二级索引更小
- 缺点: 记录不按Sno属性聚集存储

Example (代理键上的聚簇索引)

二级索引

Sname	代理键值
Abby	4
Elsa	2
Eric	3
Nick	1

Student (聚簇索引)

ID	Sno	Sname	Ssex	Sage	Sdept
1	221101	Nick	M	20	Physics
2	231101	Elsa	F	19	CS
3	231102	Eric	M	19	CS
4	232101	Abby	F	18	Math

设计策略6: 覆盖索引

Definition (覆盖索引)

如果一个索引包含(覆盖)一个查询需要用到的所有属性，则称该索引为覆盖索引

Example (覆盖索引)

- 查询: `SELECT Sno FROM Student WHERE Sname = 'Elsa';`
- 索引: `CREATE INDEX idx ON Student (Sname, Sno);`
- 索引idx能够覆盖上述查询

覆盖索引

Sname	Sno	地址
Abby	232101	$addr_4$
Elsa	231101	$addr_2$
Eric	231102	$addr_3$
Nick	221101	$addr_1$

地址
$addr_1$
$addr_2$
$addr_3$
$addr_4$

Student

Sno	Sname	Ssex	Sage	Sdept
221101	Nick	M	20	Physics
231101	Elsa	F	19	CS
231102	Eric	M	19	CS
232101	Abby	F	18	Math

设计策略6: 覆盖索引(续)

- 索引项通常远小于数据记录，如果只需访问覆盖索引，则可以极大减少数据访问量
- 覆盖索引中属性值是顺序存储的，能更快找到满足条件的属性值
- 使用覆盖索引，则无需回表

Example (覆盖索引)

- 查询: `SELECT Sno FROM Student WHERE Sname = 'Elsa';`
- 索引: `CREATE INDEX idx ON Student (Sname, Sno);`
- 索引idx能够覆盖上述查询

覆盖索引

Sname	Sno	地址	地址
Abby	232101	$addr_4$	$addr_1$
Elsa	231101	$addr_2$	$addr_2$
Eric	231102	$addr_3$	$addr_3$
Nick	221101	$addr_1$	$addr_4$

Student

Sno	Sname	Ssex	Sage	Sdept
221101	Nick	M	20	Physics
231101	Elsa	F	19	CS
231102	Eric	M	19	CS
232101	Abby	F	18	Math

设计策略6: 覆盖索引(续)

- 即使一个索引不能覆盖某个查询，我们也可以将该索引用作覆盖索引，并采用延迟连接(deferred join)方式改写查询

Example (延迟连接)

- 查询: `SELECT * FROM Student WHERE Sname = 'Elsa';`
- 索引: `CREATE INDEX idx ON Student (Sname, Sno);`
- 索引idx不能覆盖上述查询，但我们可以将该查询改写为

```
SELECT * FROM Student NATURAL JOIN
(SELECT Sno FROM Student WHERE Sname = 'Elsa');
```

- 可以利用覆盖索引快速执行子查询(延迟连接什么情况下有用?)

覆盖索引

Sname	Sno	地址	地址
Abby	232101	$addr_4$	$addr_1$
Elsa	231101	$addr_2$	$addr_2$
Eric	231102	$addr_3$	$addr_3$
Nick	221101	$addr_1$	$addr_4$

Student

Sno	Sname	Ssex	Sage	Sdept
221101	Nick	M	20	Physics
231101	Elsa	F	19	CS
231102	Eric	M	19	CS
232101	Abby	F	18	Math

6.5 物理存储结构的设计

Design of Physical Storage Structures

数据类型的选择

- 选择合理的数据类型对于提高数据库系统的性能非常重要

选择数据类型的原则

1. 尽量使用可以正确存储数据的最小数据类型
   - 原因: 最小数据类型占用空间更少，处理速度更快
   - 例: INTEGER或INT占4字节；SMALLINT占2字节；TINYINT占1字节；MEDIUMINT占3字节；BIGINT占8字节
   - 例: 使用VARCHAR(5)和VARCHAR(100)来存储'hello'的空间开销是一样的；但在排序时，MySQL通常会按照类型分配固定大小的内存块
2. 尽量选择简单的数据类型
   - 原因: 简单数据类型的处理速度更快
   - 例: 用DATE、DATETIME等类型来存储日期和时间，不要用字符串
   - 例: 用整型来存储IP地址，而不是用字符串
3. 若无需存储空值，则最好将属性声明为NOT NULL
   - 原因: 含空值的属性使得索引、统计、比较都更复杂

标识符类型的选择

- 为标识符属性选择合适的数据类型非常重要
  - 标识符属性通常会被当作索引属性，频繁地进行比较
  - 标识符属性通常会被当作主键或外键，频繁地进行连接
  - 在设计时，既要考虑标识符属性类型占用的空间，还要考虑比较的效率
- 整型：最好的选择
  - 占用空间少
  - 比较速度快
  - 可声明为AUTO_INCREMENT，为应用提供便利
- ENUM和SET类型：糟糕的选择
  - MySQL内部用整型来存储ENUM和SET类型的值，占用空间少
  - 在比较时会被转换为字符串，比较速度慢
- 字符串型：糟糕的选择
  - 占用空间大
  - 比较速度慢

关系模式的设计

- 一个关系不要包含太多属性，可以纵向拆分为多个关系

$$R(ID, A_1, A_2, \ldots, A_{1000})$$

- 不要使用“实体–属性–值(EAV)”设计模式(即为实体的每个属性建立一个关系)，因为在这种关系数据库模式上要进行大量连接

$$R_1(ID, A_1)$$
$$R_2(ID, A_2)$$
$$\ldots$$
$$R_{1000}(ID, A_{1000})$$

- 可以进行合理的纵向拆分

$$R_1(ID, A_1, A_2, \ldots, A_{50})$$
$$R_2(ID, A_{51}, A_{52}, \ldots, A_{100})$$
$$\ldots$$
$$R_{20}(ID, A_{951}, A_{952}, \ldots, A_{1000})$$

总结

1. 物理数据库设计的步骤
2. 索引的类型
   - 主索引、二级索引
   - 唯一索引、非唯一索引
   - 聚集索引、非聚集索引
3. 索引的功能
   - B+树索引的功能
   - 哈希索引的功能
4. 索引的设计：各种设计策略及其优缺点
   - 伪哈希索引
   - 前缀索引
   - 单个多属性索引vs多个单属性索引
   - 索引属性的顺序
   - 聚簇索引
   - 覆盖索引
5. 物理存储结构的设计

习题

1. 前缀索引可以被用作覆盖索引吗？
2. 唯一索引和非唯一索引的查询效率哪个更高？更新效率哪个更高？
3. 身份证号码上适合建前缀索引吗？

参考资料

- Baron Schwartz, Peter Zaitsev, Vadim Tkachenko. High Performance MySQL.