

关系数据库的调优



徐辰
华东师范大学
数据科学与工程学院
cxu@dase.ecnu.edu.cn

数据库设计流程

- 需求分析
 - 了解用户需求，确定软件的基本功能。
- 概念模型设计
 - 确定数据库需要记录哪些信息，并通过概念进行描述。
- 逻辑结构设计
 - 确定数据库模式（关系模式）。
- **数据库物理设计**
 - 确定数据的存储方式、索引的使用、系统配置等等。
- 数据库实施
 - 安装数据库、导入数据、调试、运行。

规范逻辑设计的数据模式一定好吗？

客户号	客户名字
007	张三
010	小明
025	大宝
...	...

商品号	商品名称	价格
0815	洗发水	20元
0762	优盘	50元
0453	电吹风	150元
...

订单号	客户号	商品号	数量	日期
00001	007	0815	1	2013-1-12
00002	010	0815	1	2012-12-23
00003	025	0762	10	2012-12-15
00004	007	0453	1	2012-12-1
...

物理设计

基于已有的逻辑设计，结合应用和负载的特征，对逻辑设计进行适当调整，并确定数据库的系统配置、存储方式、索引结构等，使得数据库的设计能够满足应用的性能需求。

案例1

Customer

C_no	C_name
007	张三
...	...

Product

P_no	P_name	Price
0815	洗发水	20元
...

Order

Order_no	C_no	P_no	Quantity	Date
00001	007	0815	1	2013-1-12
...

- 常用查询:

Select Order_no, C_name, P_name From; (80%)

Insert Into Order Values(...); (10%)

物理设计1

Customer

C_no	C_name
007	张三
...	...

Product

P_no	P_name	Price
0815	洗发水	20元
...

Order

Order_no	C_no	C_name	P_no	P_name	Quantity	Date
00001	007	张三	0815	洗发水	1	2013-1-12
...

- 常用查询:

Select Order_no, C_name, P_name From; (80%)

Insert Into Order Values(...); (10%)

案例2

Emp (emp#, empname, phone#, post, ...)

可是，有极少数员工有两个或更多的电话号码，特别是职位较高的员工。

- 方案一：

Emp (emp#, empname, phone#, alt_phone#,
post, ...)

- 方案二：

Emp(emp#, empname, post, ...)

Emp_phone(emp#, phone#)

物理设计2

Emp (emp#, empname, phone,)

Emp_phone_overflow (emp#, overflow-phone#)

函数依赖的两个扩展

- 强函数依赖

- 假设 $X \rightarrow Y$ 是一个完全函数依赖。如果Y中的属性在实际应用中从来不变或者变化很少，我们称 $X \rightarrow Y$ 为一个*强函数依赖*。
- 对于关系
Order(Order_no, C_no, C_name, P_no, P_name, Quantity, Date),
 $C_no \rightarrow C_name$ 和 $P_no \rightarrow P_name$ 为强函数依赖。
- 对于强函数依赖 $X \rightarrow Y$ ，Y在模式中的冗余并不会带来太多的更新异常。

函数依赖的两个扩展

- 弱函数依赖

- 假设 X, Y 是一个关系 R 中的两个属性。如果我们将 R 中的少数几个元组去除之后, $X \rightarrow Y$ 成为一个函数依赖, 我们称 $X \rightarrow Y$ 为一个弱函数依赖。
- 关系 $\text{Emp}(\text{emp\#}, \text{empname}, \text{phone\#}, \text{post}, \dots)$ 中存在弱函数依赖 $\text{emp\#} \rightarrow \text{phone\#}$ 。
- 弱函数依赖涉及的“少数几个元组”可以单独作为一个关系。（一般情况下, 为了优化系统性能才这么做。）

案例3：代理键的使用

Titles (title, type, pubname, price, pub-date, ...)

VS

Titles (title-id, title, type, pubname, price, pub-date, ...)

案例4

- 模式：
Product(P_no, P_name, Price)
Order(Order_no, C_no, P_no, Quantity, date)
- 常用查询：(当前某一产品共卖了多少钱?)
Select sum(Quantity)*price
From Product, Order
Where Product.P_no = Order.P_no
And Product.P_no = ?

物理设计4：推导属性

Product(P_no, P_name, Price, **total_sales**)

Order(Order_no, C_no, P_no, Quantity, date)

每次插入Order表时,

$\text{total_sales} = \text{total_sales} + \text{Quantity} * \text{Price}$

案例5

- 模式：
publisher (pub-id, pubname, city, state)
Titles (title-id, title, type, pub-id, price,
pub-id, ...)
- 常用查询：
Select * From publisher, Titles
Where title = "....."
And publisher.pub-id = Titles.pub-id

物理设计5：物化表

- All_Titles (title-id, title, type, pub-id, price, total-sales, **pubname, city, state,** pubdate)
- 通常在数据库中使用**物化视图** (Materialize View) 来实现
CREATE materialized VIEW all_Titles
AS
Select * From publisher, Titles
Where publisher.pub-id = Titles.pub-id

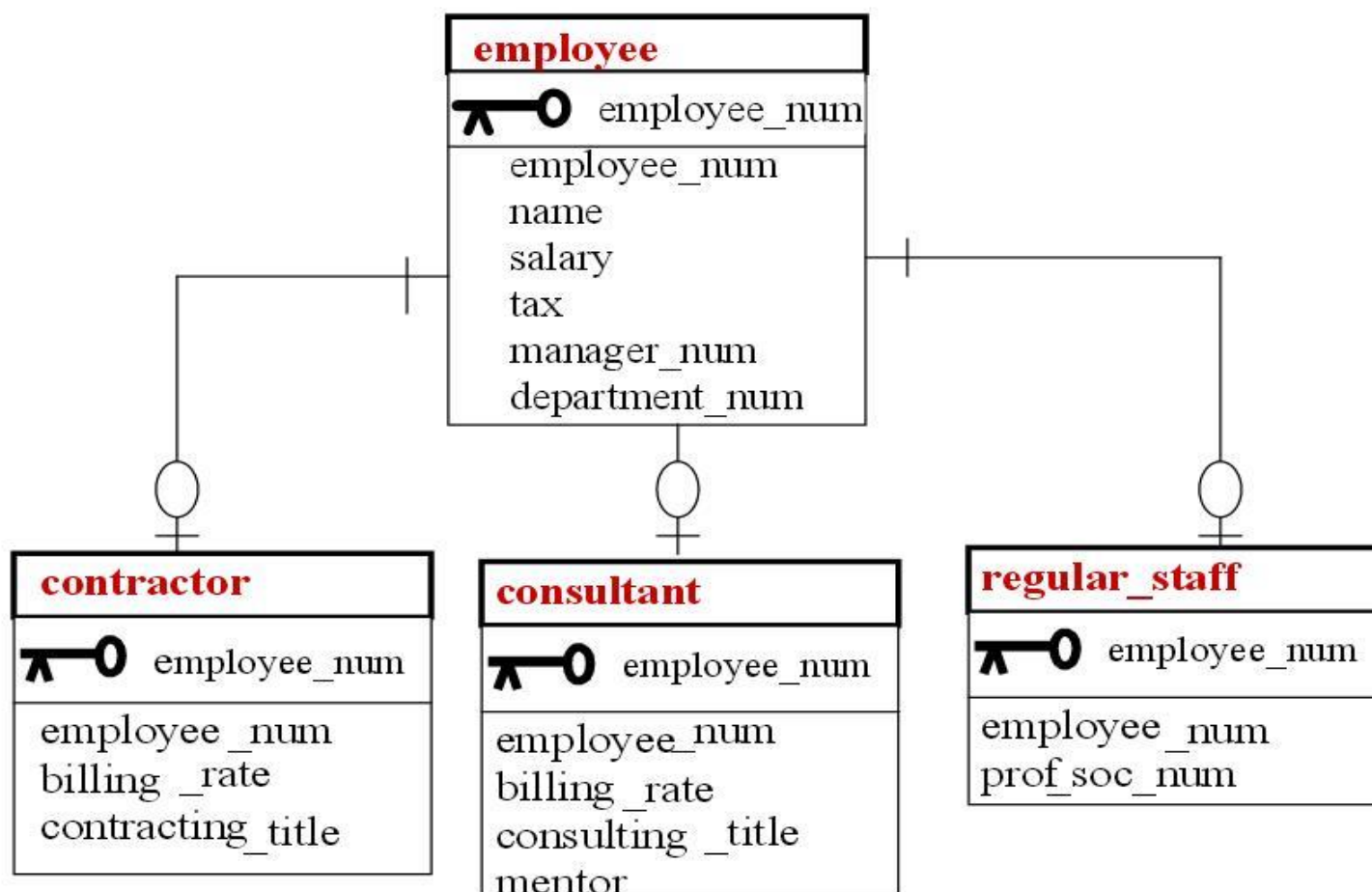
案例6

- 模式：
Emp (Eno, name, salary, tax, mgr#, dept#)
- 大部分的查询只涉及Eno, tax 和 salary

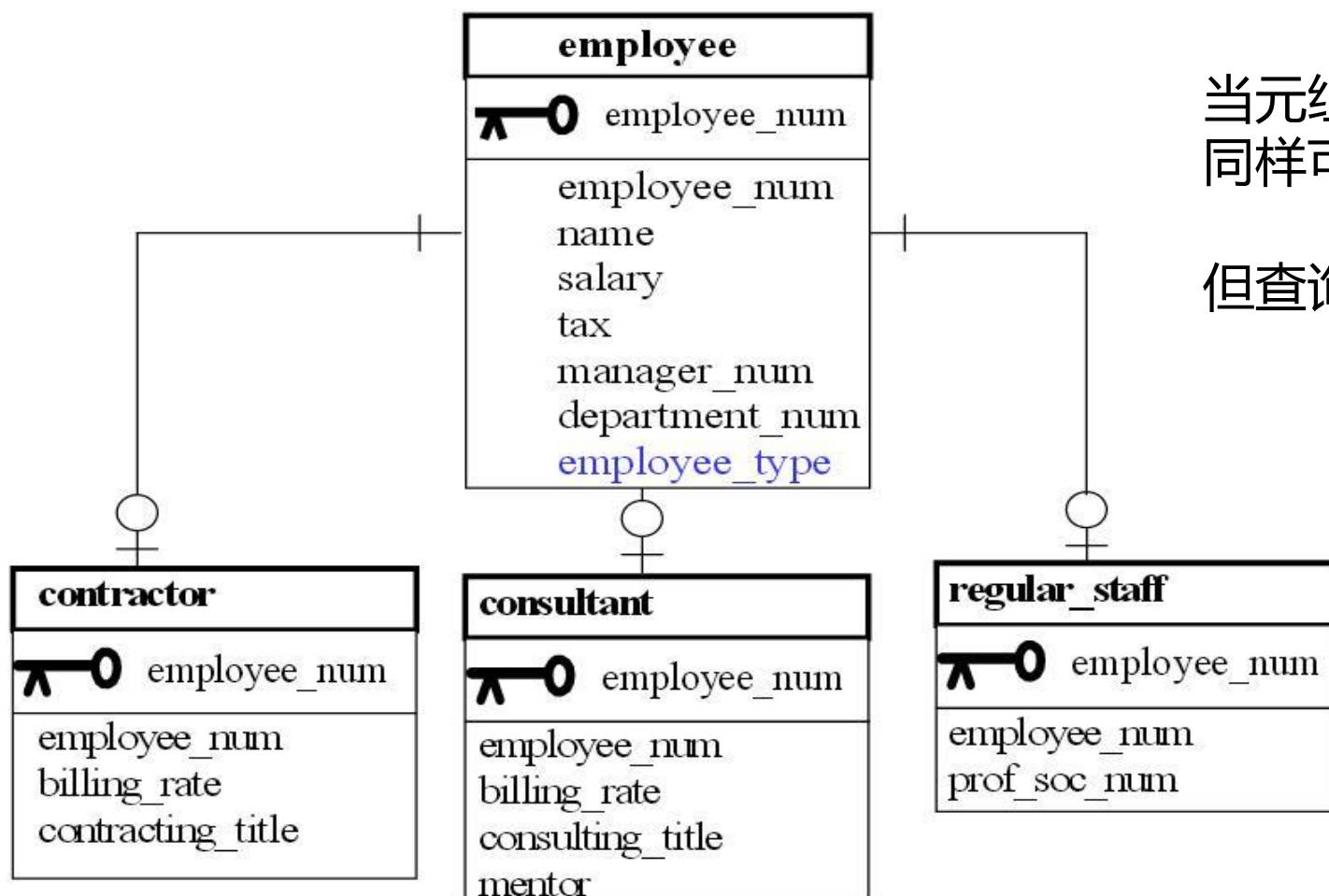
物理设计6：表分裂

- Emp_bio (Eno, name, mgr#, dept#)
- Emp_comp (Eno, salary, tax)

案例7：子类的处理




物理设计7-1：原始设计



当元组的子类未知时，
同样可以被插入数据库。

但查询可能涉及链接操作。

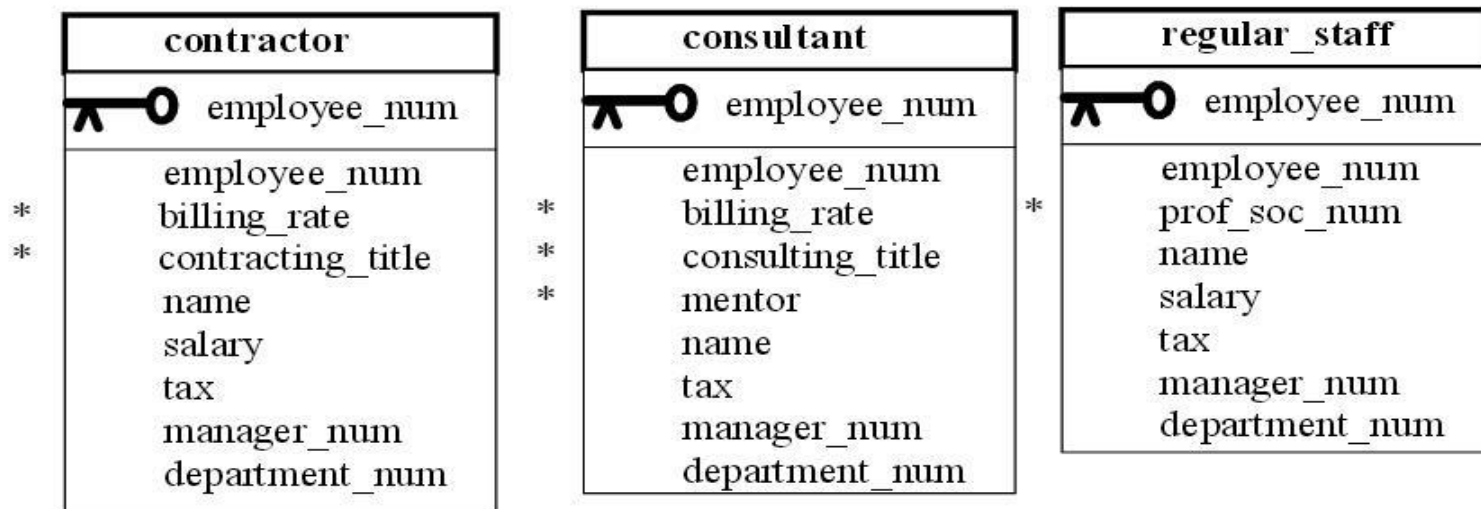
物理设计7-2： 单个表

employee	
	employee_num
<hr/>	
	employee_num
	name
	salary
	tax
	manager_num
	department_num
	consulting_title
	contracting_title
	billing_rate
	mentor
	prof_soc_num
	employee_type

免除了链接操作。

但耗费空间，查询时间增加。

物理设计7-3：子类表



免除了链接操作，且节省空间。

但要求事先知道元组的子类。子类太多也会造成使用麻烦。

性能调优

案例1：多值辅助索引

- 假设A并不是R的主码，如何加速下面的查询？

SELECT B, C

FROM R

WHERE A = 5

- 在以下哪一组属性上建辅助索引？

A

A,B

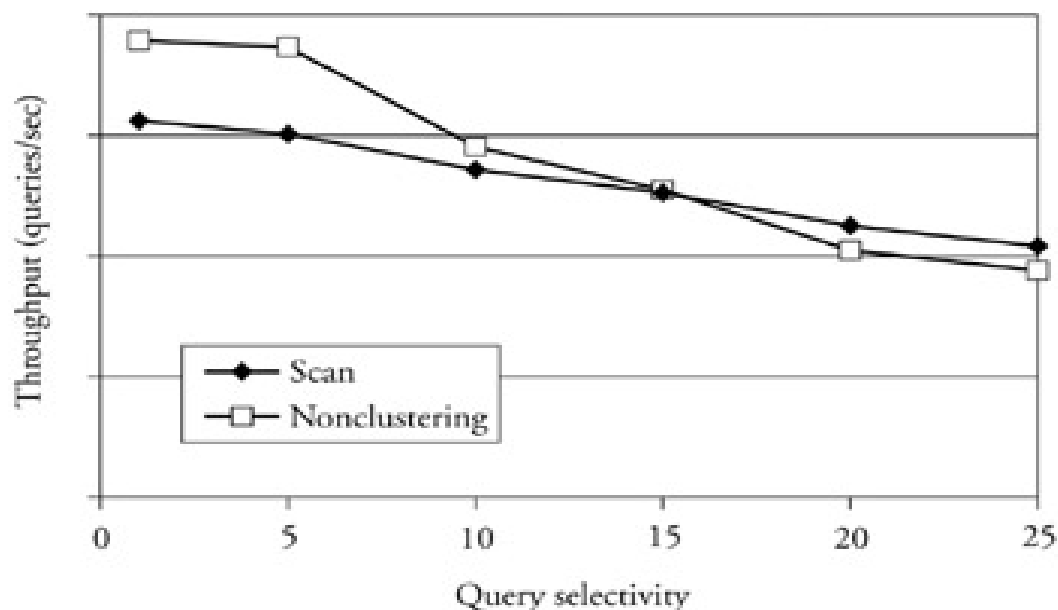
B,A

A,B,C

Cover了查询中的
所有属性：
只需索引就可
以回答查询，
无需再访问数
据

辅助索引的效率与查询选择率有关

- 假设关系表中有 N 个记录，其中 K 个记录满足查询 Q 的条件，那么 K/N 为查询 Q 的选择率 (Selectivity)。
- 假设 N 个记录存放在 M 个数据页中，如果 $M < K$ ，辅助索引对 Q 则毫无意义。



案例2：多次细粒度查询

- 首先获取用户列表
 - rs = Execute (“Select UID
From Customer
Where Country = ‘China’”);
- 再通过每个用户ID获取用户数据
 - For (i in rs){
 Execute(“Select *
 From Document
 Where UID = #i”);
}

避免多次细粒度查询

- 一次性获得所需结果
 - rs = Execute (“Select *
From Customer, Document
Where Customer.UID = Document.UID
And Country = ‘China’”);
- 再逐个处理结果
 - For (i in rs){
.....
}

案例3：Distinct

```
SELECT DISTINCT ssnun /*身份证号*/  
FROM Employee  
WHERE dept=' Efficient Computation'
```

- DISTINCT是多余的，去重代价大。
- 改写：

```
SELECT ssnun  
FROM Employee  
WHERE dept=' Efficient Computation'
```

案例4：子查询

```
SELECT ssn  
FROM Employee  
WHERE dept IN  
(SELECT dept FROM ResearchDept)
```

- 子查询通常性能较低。尽量使用连接：

```
SELECT ssn  
FROM Employee E, ResearchDept D  
WHERE E.dept=D.dept
```

案例5：相关子查询

```
SELECT ssn  
FROM Employee E1  
WHERE salary =  
(SELECT max(salary)  
FROM Employee E2  
WHERE E1.dept=E2.dept)
```

- 相关子查询常常需要被重复执行。

相关子查询改写

改写为:

```
SELECT snum  
FROM Employee E,  
      (SELECT Dept, max(salary) as m  
       FROM Employee  
       GROUP BY dept) maxsal  
WHERE salary=m  
AND E.dept=maxsal.dept
```

案例6：中间表

```
SELECT * INTO temp  
FROM Employee  
WHERE salary > 300000;
```

```
SELECT ssn  
FROM Temp  
WHERE Temp.dept = ' study admin'
```

- Temp用于存放中间结果，Employee.dept上的索引无法使用。
- 尽量避免使用存放中间结果的表。

案例7：部分结果

```
SELECT EMPNO,SALARY  
FROM EMP E1  
WHERE 20<=  
(SELECT COUNT(*) FROM EMP E2  
WHERE E2.salary>=E1.salary)
```

数据库中的Top-K查询

```
SELECT EMPNO,SALARY  
FROM EMP  
ORDER BY SALARY DESC  
OPTIMIZE FOR 20 ROWS (有的数据库使用limit)
```


总结

- 关系数据库的使用并不容易
 - 模式设计需要考虑众多因素
 - SQL的书写必须讲究性能
 - 众多的调优手段：索引、物化视图、硬件
 - 在分布式环境下，设计和优化需考虑的问题更多

Credits

- Slides from
 - 周烜