SQL

2025年5月8日 13:57

1. sql目录
   a. 数据定义语言 DDL 关系层面
   b. 数据操控语言 DML 查询修改元组层面
   c. 包括约束完整性
2. DDL
   a. 基本类型
      - **char**(n)：固定长度的字符串，用户指定长度 n。也可以使用全称 **character**。
      - **varchar**(n)：可变长度的字符串，用户指定最大长度 n，等价于全称 **character varying**。
      - **int**：整数类型(和机器相关的整数的有限子集)，等价于全称 **integer**。
      - **smallint**：小整数类型(和机器相关的整数类型的子集)。

      i.
      - **numeric**(p, d)：定点数，精度由用户指定。这个数有 p 位数字(加上一个符号位)，其中 d 位数字在小数点右边。所以在一个这种类型的字段上，**numeric**(3, 1)可以精确储存 44.5，但不能精确存储 444.5 或 0.32 这样的数。
      - **real**, **double precision**：浮点数与双精度浮点数，精度与机器相关。
      - **float**(n)：精度至少为 n 位的浮点数。

   b. 模式定义
      i. **primary key**(A_{j1}, A_{j2}, $\cdots$, A_{jm})：
      ii. **foreign key**(A_{k1}, A_{k2}, $\cdots$, A_{kn}) **references**
      iii. **not null**：一个属性上的 **not null** 约束表明在该属性上不允许空值。
      iv. 插入元组 要符合 主码不空且不重 外码有依赖

      1)
      ```
      insert into instructor
          values (10211, 'Smith', 'Biology', 66000);
      ```

      v. 删除关系：
         1) delete from r 清空全部r中元组
         2) Drop table r 删除关系r

      vi. 增删属性

      我们使用 **alter table** 命令为已有关系增加属性。关系中的所有元组在新属性上的取值将被设为 *null*。**alter table** 命令的格式为：

      1)
      ```
      alter table r add A D;
      ```

      其中 r 是现有关系的名字，A 是待添加属性的名字，D 是待添加属性的域。我们可以通过命令

      ```
      alter table r drop A;
      ```

3. DML
   a. 自然连接

   ```
   select name, course_id
   from instructor, teaches
   where instructor.ID = teaches.ID;
   ```

      i. 该查询可以用 SQL 的自然连接运算更简洁地写作：

   ```
   select name, course_id
   from instructor natural join teaches;
   ```

   b. 重命名
      i. 目的简化
      ii. 目的比较同一元组中的关系：

      假设我们希望写出查询："找出满足下面条件的所有教师的姓名，他们的工资至少比 Biology 系某一个教师的工资要高"，我们可以写出这样的 SQL 表达式：

      1)
      ```
      select distinct T.name
      from instructor as T, instructor as S
      where T.salary > S.salary and S.dept_name = 'Biology';
      ```

   c. 字符串
      i. 单引号 ，如果单引号也是字符串一部分用''
      - 百分号(%)：匹配任意子串。
      - 下划线(_)：匹配任意一个字符。

      模式是大小写敏感的，也就是说，大写字符与小写字符不匹配，反之亦然。为了说明模式匹配，考虑下列例子：

- 百分号(%)：匹配任意子串。
- 下划线(_)：匹配任意一个字符。

模式是大小写敏感的，也就是说，大写字符与小写字符不匹配，反之亦然。为了说明模式匹配，考虑下列例子：

ii.
- 'Intro%'匹配任何以"Intro"打头的字符串。
- '% Comp%'匹配任何包含"Comp"子串的字符串，例如'Intro. to Computer Science'和'Computational Biology'。
- '___'匹配只含三个字符的字符串。
- '___%'匹配至少含三个字符的字符串。

iii. Like 匹配 escape+'\'表示转义

1) • **like** 'ab\%cd%' **escape** '\' 匹配所有以"ab%cd"开头的字符串。

d. 集合运算

i. 并

1)
```
(select course_id
 from section
 where semester = 'Fall' and year = 2009)
 union
(select course_id
 from section
 where semester = 'Spring' and year = 2010);
```

2) Union 自动去重复

3) Union all 不去

ii. 交

INTERSECT

来自 <https://chat.deepseek.com/a/chat/s/2bde5a4e-744b-466b-aee5-120f5807f250>

iii.

iv. 差 except

e. 空

i. Unknown 和null的比较结果、

ii. 比较表

1 or u =1	1 and u = u
0 or u = u	0 and u=0

iii. Is (not) null

iv. 直接输出Null =null ->unknown 但是比较两个元组等不等

默认值null=值null

f. 聚集

i. 五个基本聚集函数 avg max min sum count

1) count（distinct（ID））但count(*)不许distinct

Count(*)包括空值

2) all是默认的 即保留重复

ii. Group by 把相同属性分一组

1) select只能取聚集函数里的和group by的属性

iii. Having

1) 选择满足条件的分组 只能取聚集函数里的和group by的属性

g. 嵌套

i. （not）In +（集合）

1) 枚举集合:

```
select distinct name
from instructor
where name not in ('Mozart', 'Einstein');
```

2) 任意多个查找in

```
select count ( distinct ID)
from takes
where (course_id, sec_id, semester, year) in (select course_id, sec_id, semester, year
                                              from teaches
                                              where teaches. ID = 10101);
```

ii. 比较

1) 前面提过用as 重命名相同属性来比较 where S.a> T.a and T=

2)

>some(集合A)	至少比A中一个大
> all	比所有的都大
<>some	不等于not in
<>any	等于not in
=some	等于in

iii. Exits（集合）

1) 在集合非空为true

2) Not exits相反

iv. unique（集合）集合中无重复 true

v. From 子查询

1)

```
select dept_name, avg_salary
from (select dept_name, avg (salary)
      from instructor
      group by dept_name)
      as dept_avg (dept_name, avg_salary)
where avg_salary > 42000;
```

子查询的结果关系被命名为 *dept_avg*，其属性名是 *dept_name* 和 *avg_salary*。

2) 用来两次聚集函数 比如查平均工资最大的 from里面取出平均，外部再max套

vi. With 临时定义一个 子关系 with+新关系名+as（select出集合（关系））

h. 修改数据库

i. 删除 delete 属性 from 关系 where……

ii. 插入：

1) 可以不指定 （按定义顺序）

2) 可以指定

a)

```
insert into course (course_id, title, dept_name, credits)
    values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
insert into course (title, course_id, credits, dept_name)
    values ('Database Systems', 'CS-437', 4, 'Comp. Sci.');
```

3) 可以自己创集合：

a)

```
insert into instructor
    select ID, name, dept_name, 18000
    from student
    where dept_name = 'Music' and tot_cred > 144;
```

4) 只插入部分其余赋值为NULL

iii. 更新

1) Update 关系 set 属性=……where……

iv. case：返回then后面的

1)

```
case
    when pred1 then result1
    when pred2 then result2
    ...
    when predn then resultn
    else result0
end
```

2) 可加在select，=后面，相当于一个值

4. 中级SQL
   a. 链接
      i. Select S.ID as ID 让结果ID只显示一次
      ii. R1 Join r2 using（A）相当于 R1 join r2 on r1.A=r2.A
      iii. 外连接
         1）对不出现的呈现空值
         2）右外保留join右边的关系里的元组 其余同理
   b. 视图
      i. 虚关系 只用查才计算
      ii. Create view +视图名 as +查询表达式

```
create view departments_total_salary(dept_name, total_salary) as
  select dept_name, sum (salary)
  from instructor
  group by dept_name;
```

      iii. 也可以物化视图 类似吧视图结果存下来不用查时计算
      但是空间浪费
      iv. 视图一般不可更新，除非：
         - **from** 子句中只有一个数据库关系。
         - **select** 子句中只包含关系的属性名，不包含任何表达式、聚集或 **distinct** 声明。
         1）
         - 任何没有出现在 **select** 子句中的属性可以取空值；即这些属性上没有 **not null** 约束，也不构成主码的一部分。
         - 查询中不含有 **group by** 或 **having** 子句。

         2）插入时候必须要满足视图where等语句的要求
   c. 完整性约束
      i. 单个关系
         1）Not
         2）Unique（A1，A2.....）组成候选码
         3）check（谓词）
      ii. 参照完整性
         1）Reference 必须要primary key或者unique越苏
   d. 数据类型与模式
      i.
         - **date**：日历日期，包括年(四位)、月和日。
         - **time**：一天中的时间，包括小时、分和秒。可以用变量 **time**(p)来表示秒的小数点后的数字位数(这里默认值为0)。通过指定 **time with timezone**，还可以把时区信息连同时间一起存储。
         - **timestamp**：**date** 和 **time** 的组合。可以用变量 **timestamp**(p)来表示秒的小数点后的数字位数(这里默认值为6)。如果指定 **with timezone**，则时区信息也会被存储。

      日期和时间类型的值可按如下方式说明：

```
date '2001 - 04 - 25'
time '09: 30: 00'
timestamp '2001 - 04 - 25 10: 29: 01. 45'
```

      ii. Cast e as t 把字符串e转变成t类型
      iii. extract（year from d）d为t类型一种
         1）我们可以利用 **extract**(*field* **from** d)，从 **date** 或 **time** 值 d 中提取出单独的域，这里的域可以是 **year**、**month**、**day**、**hour**、**minute** 或者 **second** 中的任意一种。时区信息可以用 **timezone_hour** 和 **timezone_minute** 来提取。
      iv. 默认值：default +默认值
      v. 索引：create index 索引名 on 关系（属性）
5. 高级SQL
   a. 函数
      i. Create function 函数名（参数名 参数类型）
      Returns +返回值类型 表table（表的定义）

Begin
end

ii.
```
create function dept_count(dept_name varchar(20))
    returns integer
    begin
    declare d_count integer;
        select count(*) into d_count
        from instructor
        where instructor.dept_name= dept_name
    return d_count;
    end
```

iii. Return 可以返回表 要return table（关系定义）

iv. Declare +变量名+类型

b. 过程

i. Create procedure 过程名（in+变量名+类型，out+变量名+类型)

ii. 用call 调用

c. While +布尔+do

语句序列

End while

Repeat

语句

Until +bool

End repeat

For r as 关系（每次r取关系一行）

Do 语句 leave可退出=break iterate=continue

End for

d. 触发器 trigger

i. Create trigger + 触发器名 + after/before + insert/delete on / update of+表名

ii. 自动启动特定任务 帮助约束

1)
```
create trigger timeslot_check1 after insert on section
```

2) 指定触发检测事件

3)
```
referencing new row as nrow
for each row
when (nrow. time_slot_id not in (
        select time_slot_id
        from time_slot)) / * time_slot 中不存在该 time_slot_id * /
begin
  rollback
end;
```

4) when是判定条件，成立就做后面的事

5) For each row遍历新加入（insert）或者删除（delete），修改的行

6) Referencing 方便后续引用

iii. After update （of +属性） on grade

iv. After Delete （of +属性） on grade

v. 可以drop丢弃 或者disable

vi. 触发器避免使用： 意外执行的风险

e. 排序 order by 默认升序asc

i. desc降序

ii. Create Index on R（A） order by A 对索引排序

形式化关系查询+关系模型介绍

2025年5月8日 13:58

1. 关系代数
   a. 基本运算
      i. $\sigma_{dept\_name = \text{"Physics"}}(instructor)$ 选择运算 where
      投影 select
      ii. 更名运算
         1) $\rho_{x(A_1, A_2, \cdots, A_n)}(E)$
         返回表达式 E 的结果，并赋给它名字 x，同时将各属性更名为 A_1，A_2，…，A_n。
         2) 相当于as，同理可以进行同属性值比较，求出存在其他元组，比该元组大（小）的
         3) 这样 最大值就是全部 - 元组（存在其他元组，比该元组大（小）的）
   b. 附加运算
      i. 集合交 附加 并非基本
      ii. 自然连接
         1) $r \bowtie s = \prod_{R \cup S}(\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \cdots \wedge r.A_n = s.A_n}(r \times s))$
         散步 先r*s，再选择属性等元组，再投影选 R和S都出现属性
         2) 可结合
         3) $r \bowtie_\theta s = \sigma_\theta(r \times s)$ 选择θ属性连接
      iii. 外连接
         1) $(\supset\!\bowtie\!\subset)$
      iv. 聚集函数
         1)
         select A_1, A_2, **sum**(A_3)
         from $r_1, r_2, \ldots, r_m$
         where P
         group by A_1, A_2

         $A_1, A_2 \mathcal{G}_{sum(A_3)}(\prod_{A_1, A_2, \cdots, A_n}(\sigma_P(r_1 \times r_2 \times \cdots \times r_m)))$
2. 元组关系演算
   a. 查询
      i. $\{t \mid t \in instructor \wedge t[salary] > 80000\}$

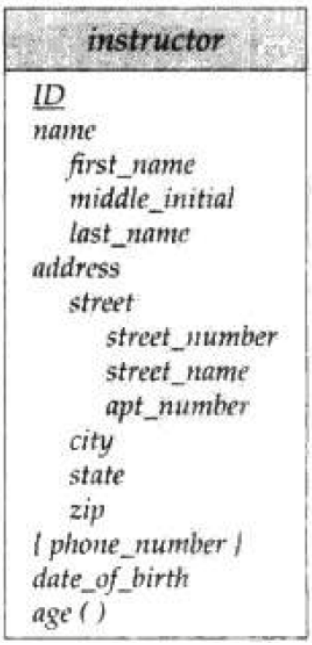
      $\{t \mid \exists s \in instructor(t[ID] = s[ID] \wedge s[salary] > 80000)\}$
      ii. 我们可以这样来读上述表达式："它是所有满足如下条件的元组 t 的集合：在关系 *instructor* 中存在元组 s，使 t 和 s 在属性 *ID* 上的值相等，且 s 在属性 *salary* 上的值大于 80 000 美元"。
      元组变量 t 只定义在 *ID* 属性上，因为这一属性是对 t 进行限制的条件所涉及的唯一属性。因此，结果得到(*ID*)上的关系。
3. 关系模型
   a. 定义
      i. 关系：表
      ii. 元组：行

iii. 关系实例：行
iv. 域：属性域

b. 码
i. 超码：唯一标识元组
ii. 候选码：最小超码（真子集不可，并不一定属性最少）
iii. 主码：选定的候选码
iv. 外码：包含另一个的主码

E-R

2025年5月11日 16:21

1. 实体联系E-R
   a. 实体集
   b. 联系集
      i. 实体集数目称为联系集的度degree 二元度为2
   c. 属性
      i. 简单属性 （原子）和 复合属性（可有层次（
      ii. 单值多值属性（{属性名}）
      iii. 派生属性（不计算 由基属性计算出来）
   d. 映射基数 基数比率 表示一个联系能关联实体与实体 一对一 一对多关系
   e. 联系集的主码由映射基数决定 是一的哪一个实体的主码 多对多主码并 一对一任意
2. 冗余属性
   a. 如果两个实体集都出现一个属性且两个实体集关联，删除不是主码的那个属性
3. ER图
   a. 1对多，多的那一方在联系集中只出现一次，因此约束最多为1
   b. 复杂属性
   c. 

   d. 非二元只准最多一个箭头
   e. 弱实体集：无主码

在图 7-14 中，弱实体集 *section* 通过联系集 *sec_course* 依赖于强实体集 *course*。

i.

双线表示所有section都在联系中必须参与

虚线表示弱实体集标识符

双菱形表示连到弱

4. E-R转换为关系
   a. 强实体 --- 直接成关系，加上有多对一关系中一的主码
   b. 复杂属性拆成若干属性 主码需要每个属性再考虑
   c. 弱实体集----将依赖的强实体集主码加进去 主码再考虑
   d. 联系集属性为 自己属性∪相连的实体集主码并集
5. 实体设计
   a. 把n元变成若干二元联系集组合
6. 扩展E-R
   a. 特化：将一个实体集再加附加属性区分成多种
      i. 重叠特化：实体集可属于多个特化实体集
      ii. 不相交特化：至多一个
   b. 概化：两个实体集具有共性 从底向上
      i. 高层：超类
      ii. 低层：子类
   c. 继承：
      i. 高层属性被低层继承
      ii. 联系集继承
      iii. 单继承 /多继承 一个底层指向多个高层

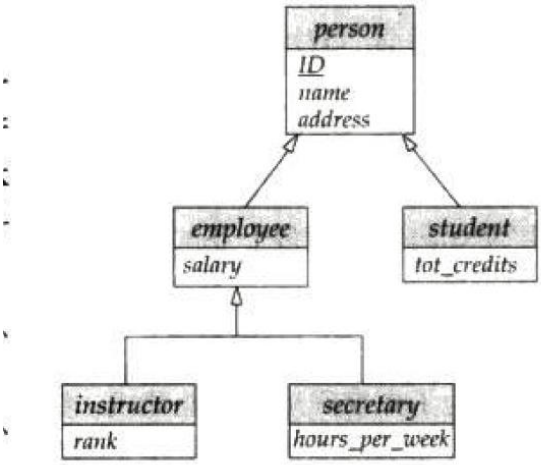


图 7-21 特化和概化

d. 概化约束：
   i. 全部概化 高层所有实体必属于一个子类
   ii. 部分概化 可以不属于子类
e. 聚集
   i. 可以把联系看成高层实体集
   ii.

聚集是一种 **高阶关系**，它将一个 **关系（Relationship）** 和相关的 **实体（Entity）** 组合成一个 **更高层次的抽象单元**，以便于与其他实体或关系建立关联。

适用场景

- 当某个 **关系本身需要参与另一个关系** 时。
- 当需要表示 **“整体-部分”** 结构，并且部分实体需要独立管理时。
- 当模型存在 **多层关系**（如“项目-员工-任务”之间的复杂关联）。

- **员工（Employee）** 和 **项目（Project）** 之间存在 **“分配（Assigns）”** 关系。
- 每个 **“分配”** 关系可以关联多个 **任务（Task）**。

   iii.

传统E-R模型（无聚集）

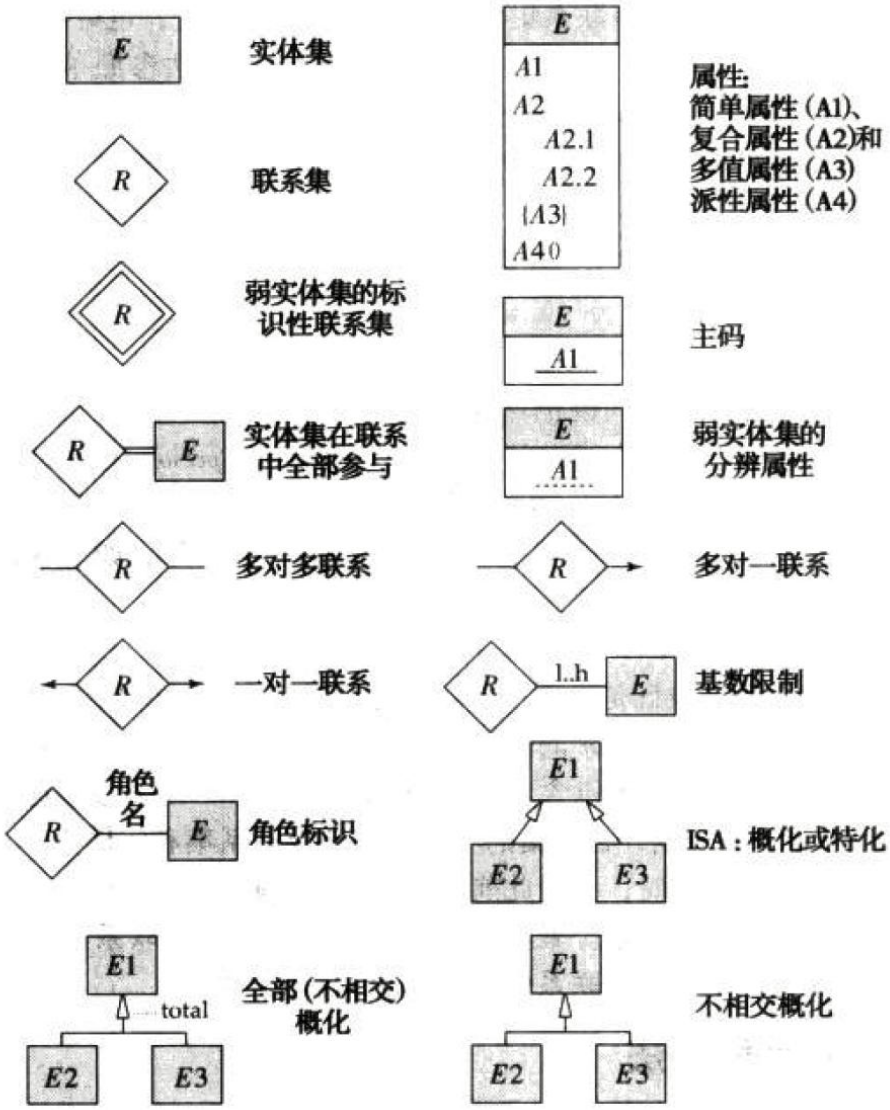
- `Employee` ——（`Assigns`）→ `Project`
- `Assigns` ——（`Has`）→ `Task`
- 但 `Assigns` 本身是一个关系，无法直接与 `Task` 关联。

使用聚集后的E-R模型

1. 将 `Employee` 和 `Project` 之间的 `Assigns` 关系封装成一个 **聚集（Aggregate）**，称为 `Assignment`。
2. 然后，`Assignment` 可以与 `Task` 建立新的关系 `Performs`。

   iv. 聚集的主码就是联系的主码
7. 表示

a.


E
实体集
R
联系集
R
弱实体集的标识性联系集
R
E
实体集在联系中全部参与
R
多对多联系
R
一对一联系
R
角色名
E
角色标识
E1
total
E2
E3
全部(不相交)概化
E
A1
A2
A2.1
A2.2
{A3}
A4()
属性:
简单属性(A1)、
复合属性(A2)和
多值属性(A3)
派生属性(A4)
E
A1
主码
E
A1
弱实体集的分辨属性
R
多对一联系
R
1..h
E
基数限制
E1
E2
E3
ISA:概化或特化
E1
E2
E3
不相交概化

关系数据库设计

2025年5月12日 21:14

1. 更小模式
   a. 函数依赖：
   b. 能定义如“*dept_name* 的每个特定的值对应至多一个 *budget*”这样的规则。换句话说，我们需要写这样一条规则“如果存在模式（*dept_name*，*budget*），则 *dept_name* 可以作为主码”。这条规则被定义为**函数依赖**（functional dependency）

$$dept\_name \rightarrow budget$$

   c. 有损分解：
   d. 无损分解：可以通过自然连接恢复
2. 原子域第一范式
   a. 定义
      i. 原子：属性无子结构 元素不可再分
      ii. 1NF：所有属性都原子
3. 函数依赖进行分解：
   a. 定义 也可以AB表示{A,B}，或者小写希腊字母α

i.

概念	表示方法	示例
属性（Attribute）	大写字母 A, B, C	A 是一个属性
属性集（Set of Attributes）	花括号 $\{A, B\}$	$X = \{A, B\}$
关系模式（Relation Schema）	$R(A, B, C)$ 或 $R = \{A, B, C\}$	$R(A, B, C)$
超码（Superkey）	$K = \{A, B\}$	K 是超码
候选码（Candidate Key）	$\underline{A}$ 或 **A**	$\underline{A}$ 是码
函数依赖（FD）	$A \rightarrow B$	A 决定 B
关系实例（Instance）	$r(R)$	r 是 R 的数据表

   b. 依赖

i.

概念	表示	示例
单属性依赖	$A \rightarrow B$	学号 → 姓名
属性集依赖	$\{A, B\} \rightarrow C$	{学号, 课程号} → 成绩
平凡依赖	$\{A, B\} \rightarrow A$	右边是左边的子集
完全依赖	$\{A, B\} \rightarrow C$，且 $A \not\rightarrow C, B \not\rightarrow C$	成绩完全依赖学号和课程号
部分依赖	$\{A, B\} \rightarrow C$，但 $A \rightarrow C$ 也成立	成绩部分依赖学号
传递依赖	$A \rightarrow B, B \rightarrow C \Rightarrow A \rightarrow C$	学号 → 院系 → 院长

结论：

- **函数依赖可以描述属性集之间的关系**，而不仅仅是单属性依赖。
- **数据库规范化**（2NF, 3NF, BCNF）的核心就是分析并优化这些依赖关系。
- **Armstrong 公理** 用于推导隐含的函数依赖。

   ii. 平凡的函数依赖

给定一个关系模式 R(U)$R(U)$（UU 是属性全集），如果 **Y⊆X**（即 Y是 X 的子集），则函数依赖：X→Y **必定成立**，称为**平凡的函数依赖**

   iii. 闭包

c. Boyce-CODD BCNF

i. 着。具有函数依赖集 F 的关系模式 R 属于 BCNF 的条件是，对 F^+ 中所有形如 $\alpha \rightarrow \beta$ 的函数依赖（其中 $\alpha \subseteq R$ 且 $\beta \subseteq R$），下面至少有一项成立：
   - $\alpha \rightarrow \beta$ 是平凡的函数依赖（即，$\beta \subseteq \alpha$）。
   - α 是模式 R 的一个超码。

   一个数据库设计属于 BCNF 的条件是，构成该设计的关系模式集中的每个模式都属于 BCNF。

ii. 分解规律：

1)
   - $(\alpha \cup \beta)$
   - $(R - (\beta - \alpha))$

2) 检查步骤：
   1. 找出关系中的所有候选键
   2. 列出所有非平凡函数依赖
   3. 检查每个函数依赖的左侧是否是超键
      - 如果是，则满足BCNF
      - 如果不是，则不满足BCNF

d. 3NF

i. 具有函数依赖集 F 的关系模式 R 属于**第三范式**（third normal form）的条件是：对于 F^+ 中所有形如 $\alpha \rightarrow \beta$ 的函数依赖（其中 $\alpha \subseteq R$ 且 $\beta \subseteq R$），以下至少一项成立：
   - $\alpha \rightarrow \beta$ 是一个平凡的函数依赖。
   - α 是 R 的一个超码。
   - $\beta - \alpha$ 中的每个属性 A 都包含于 R 的一个候选码中。

4. 函数依赖

a.
   - **合并律**（union rule）。若 $\alpha \rightarrow \beta$ 和 $\alpha \rightarrow \gamma$ 成立，则 $\alpha \rightarrow \beta\gamma$ 成立。
   - **分解律**（decomposition）。若 $\alpha \rightarrow \beta\gamma$ 成立，则 $\alpha \rightarrow \beta$ 和 $\alpha \rightarrow \gamma$ 成立。
   - **伪传递律**（pseudotransitivity rule）。若 $\alpha \rightarrow \beta$ 和 $\gamma\beta \rightarrow \delta$ 成立，则 $\alpha\gamma \rightarrow \delta$ 成立。

   - **自反律**（reflexivity rule）。若 α 为一属性集且 $\beta \subseteq \alpha$，则 $\alpha \rightarrow \beta$ 成立。
   - **增补律**（augmentation rule）。若 $\alpha \rightarrow \beta$ 成立且 γ 为一属性集，则 $\gamma\alpha \rightarrow \gamma\beta$ 成立。
   - **传递律**（transitivity rule）。若 $\alpha \rightarrow \beta$ 和 $\beta \rightarrow \gamma$ 成立，则 $\alpha \rightarrow \gamma$ 成立。

b. 判断集合超码：集合可以推出所有属性

c. 属性集的闭包：属性集可以推出全部属性

d. 正则覆盖

i. 无关属性：删除函数依赖一个不改变依赖集闭包

1)
   - 如果 $A \in \alpha$ 并且 F 逻辑蕴含 $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$，则属性 A 在 α 中是无关的。
   - 如果 $A \in \beta$ 并且函数依赖集 $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ 逻辑蕴含 F，则属性 A 在 β 中是无关的。

   属于α可以直接α中去掉A看还能不能推出β

ii. 正则覆盖：不含无关属性 左半部唯一

iii. 无损分解：若R分成R1,R2 R1交R2是R1或R2超码

iv. 保持依赖

1) 限定：F在R上限定是F+中只包含Ri的

2) 是不是最后保持开始全部函数依赖

5. 分解算法：

a. BCNF

b. 3NF

6. 多值依赖

令 $r(R)$ 为一关系模式，并令 $\alpha \subseteq R$ 且 $\beta \subseteq R$。**多值依赖**（multivalued dependency）

$$\alpha \twoheadrightarrow \beta$$

在 R 上成立的条件是，在关系 $r(R)$ 的任意合法实例中，对于 r 中任意一对满足 $t_1[\alpha] = t_2[\alpha]$ 的元组对 t_1 和 t_2，r 中都存在元组 t_3 和 t_4，使得

a.

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$
$$t_3[\beta] = t_1[\beta]$$
$$t_3[R - \beta] = t_2[R - \beta]$$
$$t_4[\beta] = t_2[\beta]$$
$$t_4[R - \beta] = t_1[R - \beta]$$

356

由多值依赖的定义，我们可以得出以下规则，对于 α，$\beta \subseteq R$：

b.

- 若 $\alpha \rightarrow \beta$，则 $\alpha \twoheadrightarrow \beta$。换句话说，每一个函数依赖也是一个多值依赖。
- 若 $\alpha \twoheadrightarrow \beta$，则 $\alpha \twoheadrightarrow R - \alpha - \beta$。

c. 平凡的多值依赖：a包含b 或者a∪b=R

d. 4NF

引言

2025年5月13日 10:52

1. 定义：
    a. DBMS
    b. DBAP
    c. DBA
2. 弊端：
    a. 冗余不一致
    b. 数据访问困难
    c. 数据孤立
    d. 完整性问题 约束
    e. 原子性 操作必须完成发生或者不发生
    f. 并发访问异常
    g. 安全性
3. 数据视图
    a. 抽象
        i. 物理层 最底层 描述实际存储数据
        ii. 逻辑层
            1) 管理员使用
            2) 描述怎么存储数据以及数据之间关系
            3) 物理数据独立性：不知道物理层
        iii. 视图层 只描述数据库某个部分 可以多个视图
    b. 实例与模式
        i. 信息集合称为实例
        ii. 总体设计称为模式（对应上面抽象）
            1) 物理模式 内模式
            2) 逻辑模式 全局模式
            3) 子模式 外模式
        iii. 两层映像：E-C C-I`\数据模型
        iv. 关系模型
        v. 实体联系模型
        vi. 基于对象的数据模型
        vii. 半结构化 xml
    c. 语言
        i. Dml
            1) 过程化 要求数据以及如何获得数据
            2) 声明式 只要求数据
        ii. Ddl 定义

1) 约束
2) 参照完整性
3) 断言
4) 授权

d. 设计数据库
   i. 逻辑设计
   ii. 物理设计

e. 事务管理
   i. 原子性
   ii. 一致性：总量不变
   iii. 持久性：系统故障也能保持新值

iv.

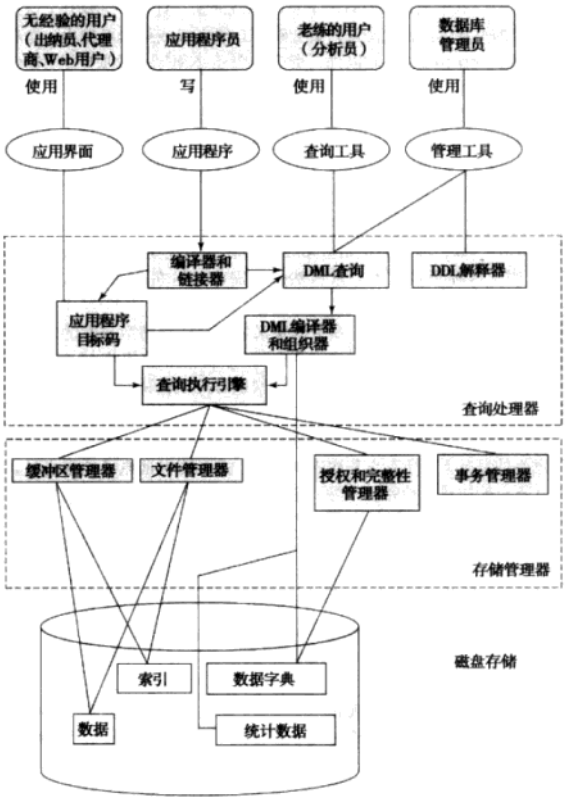


图 1-5 系统体系结构

v. 数据库系统

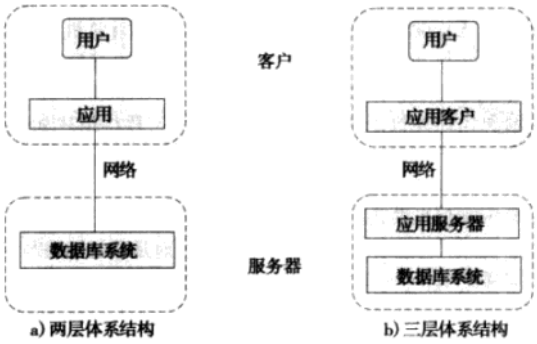


图 1-6 两层和三层体系结构

存储和文件

2025年6月14日 21:43

1. 物理介质
2. 磁盘和快闪
   a. 两面都有盘片
      i. 盘片划分磁道
      ii. 刺刀划分沙区
   b. 磁盘性能
3. 文件组织
   a. 文件逻辑上组织序列，映射在磁盘上，分成定长的存储单元，称为块，块是基本单元
      - 一个块可以包含多条记录。
      - 没有记录的大小超过块的大小。
      - 每条记录完整地包含在单个块中。
   b. 数据库映射为文件
      i. 用多个文件，定长记录
         1) 每个块取最大记录大小
         2) 删除留空，用空闲列表
            a) 文件头存第一个空地址
            b) 每个空地址存下个空地址
            c) 插入时插到文件头指向的第一个空地址，然后改文件头指向
      ii. 结构化自己的文件，以变长记录
         1) 多种类型记在一个文件里面
         2) 由固定长度属性和变属性
            a) 固定直接分配
            b) 变 由偏移量+长度表示
         3) 数位图 空值为1

索引和散列

2025年6月14日 21:56

【第1-6章】SQL、关系代数
【第7章】E-R图
【第8章】函数依赖、分解算法
【第11章】B+树索引、可扩充散列
【第12章】外部归并排序算法和代价计算；连接、查询、选择运算代价计算
【第13章】连接策略选择、连接结果大小计算
【第14章】事务调度（可串行化、可恢复、无级联）
【第15章】并发控制
【第16章】故障恢复（特别是Recovery After a System Crash的具体步骤和过程）

1. 索引类型
   a. 顺序索引
   b. 散列

索引评估指标

- 访问类型：支持高效查询指定值或范围内的记录。
- 访问时间：找到目标项的时间。
- 插入 / 删除时间：包括定位和更新索引结构的时间。
- 空间开销：索引占用的额外空间。

2. 顺序
   a. 聚簇索引：搜索键对应顺序和文件一样
   b. 非聚簇索引（Non-clustering Index）：搜索键指定的顺序与文件的顺序不同。

（一）稠密索引与稀疏索引

1. 稠密索引（Dense Index）：文件中每个搜索键值都有索引项。
   - 聚簇稠密索引：索引项包含搜索码和指向首条对应数据记录的指针。
   - 非聚簇稠密索引：需存储指向所有相同搜索键值记录的指针列表。
2. 稀疏索引（Sparse Index）：仅部分搜索键值有索引项，仅适用于文件按搜索键排序的聚簇索引场景。
   索引项包含搜索码和指向首条对应数据记录的指针。查找最大值大于搜索码的位置，然后沿着指针查找

3. B+树

a. 特征

1. 平衡树：所有从根到叶子的路径长度相同。
2. 节点限制：
   - 非根非叶节点（内部节点）：子节点数(指针数)在$\lceil n/2 \rceil$ 到n之间（n为固定值）。
   - 叶节点：值的数量在$\lceil (n-1)/2 \rceil$ 到n－1之间。
   - 根节点：若非叶节点，至少 2 个子节点；若是叶节点，可含 0 到n－1个值。
   -

b. B + 树节点结构

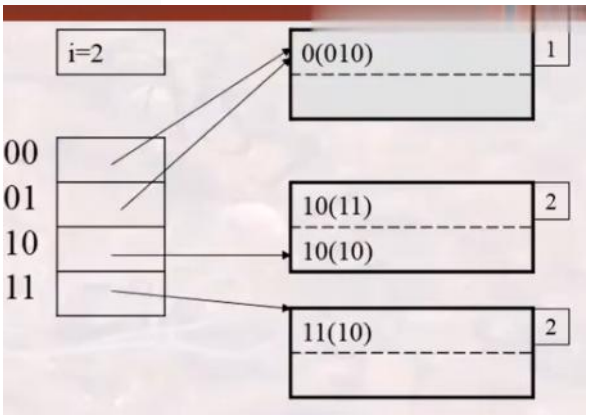
节点格式示例

1.
   - K_i：搜索键值，按顺序排列（$K_1 < K_2 < \cdots < K_{n-1}$）。
   - P_i：非叶节点中指向子节点，叶节点中指向记录或记录桶。

c. 操作（时刻保持利用率50%-100%）
   1. 插入时满了需要分裂，从下往上，不断分裂
   2. 删除时指针或者叶子结点值必须在范围，否则合并
      1）合并可能合并索引到另一块
      2）也可能合并两块
   3. 指针指向要改

4. B+树扩展
    a. 叶节点不存指向文件的指针，直接存文件记录
    b. 字符串索引，用前缀研所，非叶节点存可以区分左右子树搜索吗的前缀
5. 动态散列
    a. 数据结构
        1. 用指向块的指针表表示桶
        2. 散列函数计算一个k位的二进制函数，k表示所使用的最多位数，2^k最多桶数，i为当前的使用位数，2^i为桶个数
        3. 需要指定每个桶记录个数
    b. 操作
        1. 插入，根据指针表找到要存的桶，一旦满就会分裂，重新散列该块到两块中，指针表大小变成两倍，其余没满的不分裂，多余的指针指向同一个桶

1)



查询处理

2025年6月14日 22:51

1. 外部归并排序
   a. 第一阶段，建立多个排好序的段
   b. 第二阶段，对归并段进行排序，把归并段导入内存，并且分配一个结果空间 内存容纳M块，br为包含r记录的磁盘块数
      下面都是上取整：
   c. 归并趟数：log(M-1)[br/M]
   d. 磁盘块传输总数 br(2[logM-1(br/M)]+1)
   e. 一个归并段bb块
   f. 磁盘搜索：2[br/M]+[br/bb]*(2[log[M/bb]-1(br/M)]-1)
2. 嵌套连接
   a. 外层br块 nr组
   b. 内层bs块，ns组
   c. 最坏 nr*bs+br次块传输 nr+br次搜索
   d. 最好 br+bs 次传输 2次 搜索
3. 块嵌套
   a. 先循环块，形成块对，才搜索元组队
   b. 最坏 br*bs+br快传输，2br次磁盘搜搜
   c. 最好 br+bs 次块传输，2次搜索
   d. 若可用M块
   e. 传输 br+[br/(M-2)]*bs
   f. 搜索 2[br/(M-2)]
4. 归并连接：
   a. 假如排序过
      i. 传输 br+bs
      ii. 每个关系bb个缓冲块 搜索[br/bb]+[bs/bb]
   b. 没有排序
      i. 传输+br（2logM-1（br/M)+1）以及bs对应排序
      ii. 搜索 2*（br/M)+br(2logM/bb-1(br/M)-1)
5. 散列连接
   a. 原表太大要分成小子表存
   b. 第一趟：原始关系通过hp散列成m-1子表 散列大小nh
   c. 不需要递归划分 M>nh+1或者M>(bs/M)+1,M内存块数
      i. 传输 3（br+bs)+4nh
      ii. 搜索：2[br/bb]+[bs/bb]+2nh
   d. 需要划分，每趟划分大小减小为原来的1/（M-1）递归[log(M-1)(bs)-1]趟
      i. 传输 （2br+2bs)(logm-1(bs)-1]+br+bs

ii. 搜索 2([br/bb]+[bs/bb])[logM-1(bs)-1]

e. 如果内存可以容纳整个S关系

i. 传输 br+bs

ii. 搜索2

查询优化

2025年6月17日 16:05

1. 等价
   a. 投影运算只有最后一个是必须的
   b. 选择可以和笛卡尔积以及theta连接合起来
      a. $\sigma_\theta(E_1 \times E_2) = E_1 \bowtie_\theta E_2$。
      i. 该表达式就是 θ 连接的定义。
      b. $\sigma_{\theta 1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$
   c. 分配律：
      a. 当选择条件 θ_0 中的所有属性只涉及参与连接运算的表达式之一（比如 E_1）时，满足分配律：
      $$\sigma_{\theta_0}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_\theta E_2$$
      i. b. 当选择条件 θ_1 只涉及 E_1 的属性，选择条件 θ_2 只涉及 E_2 的属性时，满足分配律：
      $$\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_\theta (\sigma_{\theta_2}(E_2))$$
      ii. 投影也具有
2. 选择结果的统计
   a. 值：
      i. nr： 关系r的元组数
      ii. br：包含关系r中元组的磁盘块数
      iii. lr：关系r中每个元组的字节数
      iv. fr：关系r的块因子——一个磁盘块能容纳关系r中元组的个数
      v. V（A,r），非重复值数 和A的投影一样
      vi. Br=[nr/fr]
   b. 大小估计（按平均，否则由直方图得到）
      i. A=a
         1) 平均 nr/V（A，r）
      ii. A<=v 存min（A，r），r中A属性最小值
         1) 平均 nr*(v-min)/(max-min)
         2) 若v<min,0
         3) v>=max nr
   c. 合取
      i. si：条件i查询出来的选择大小
      ii. 中选率：si/nr
      iii. 个数=nr*（s1*s2*…..*sn)/(nr^n)
   d. 析取
      i. 个数=nr*(1-(1-s1/nr)*(1-s2/nr)*….*(1-sn/nr))
   e. 取反
      i. Nr-查询值
3. 连接结果的统计
   a. 笛卡尔：nr*ns*(ls+lr)
   b. R与S无交集 等于笛卡尔
   c. R与s交集是R主码 小于等于s个数，反之一样
      i. 构成S外码，等于s个数
   d. R与s交集不是任何主码 ns*nr/V（A，s），如果是S连接R ns*nr/V(A,r) 取这两个的小值
   e. 外连接还要加上关系大小
      i. $r \overrightarrow{\bowtie} s = size\ of\ r \bowtie s + size\ of\ r$
   f. 投影 A（r）=V（A，r）
   g. 聚集A=V（A，r）
   h. r-s=size(r)
4. V的计算
   a. A的属性都在r
      i. estimated $V(A, r \bowtie s) = \min(V(A, r), n_{r \bowtie s})$

b. A的元组都在s

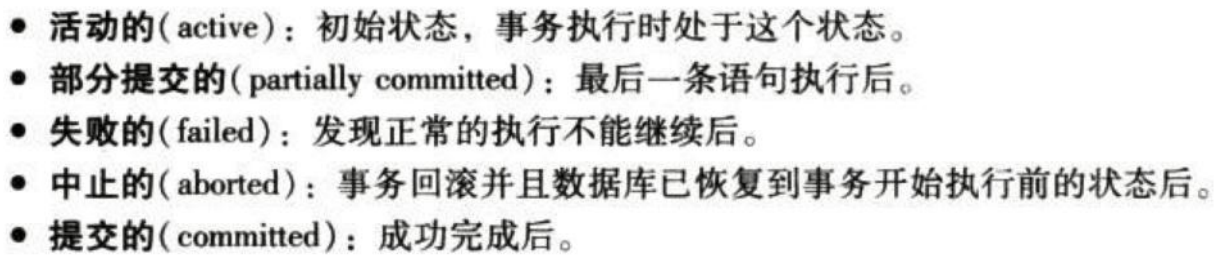
   i. estimated $V(A, r \bowtie s) = \min (V(A, s), n_{r \bowtie s})$

c. A1元组在r，A2属性都在s

   i. estimated $V(A, r \bowtie s) =$
   $\min(V(A1, r)*V(A2-A1, s), V(A1-A2, r)*V(A2, s), n_{r \bowtie s})$

事务

2025年6月17日 19:40

1. 特性、
   a. 一致性
   b. 原子性 保障一致性
   c. 隔离性
   d. 持久性 稳定性存储器
2. 状态
   a. 

   b. 只有在中止或者提交了之后叫做已经结束的
3. 隔离性
   a. 调度：指令时间顺序
   b. 串行（序列化）调度：同一个事物的指令在一起
   c. 等价调度：事务完成后状态相同的调度
   d. 调度必须保证特性
4. 可串行化
   a. 冲突指令：在不同事物上操作同一个数据项的两个指令，且至少有一个是write
   b. 全程用非冲突指令交换转换　冲突等价
   c. 交换操作只能不同事务之间换顺序，同一事务之间顺序不能变
   d. 冲突可串行化：一个调度和串行调度冲突等价
   e. 优先图：Ti到Tj有一个边
      i. Ti write(Q),Tj read(Q)
      ii. Ti read(Q),Tj write(Q)
      iii. Ti write(Q),Tj write(Q)

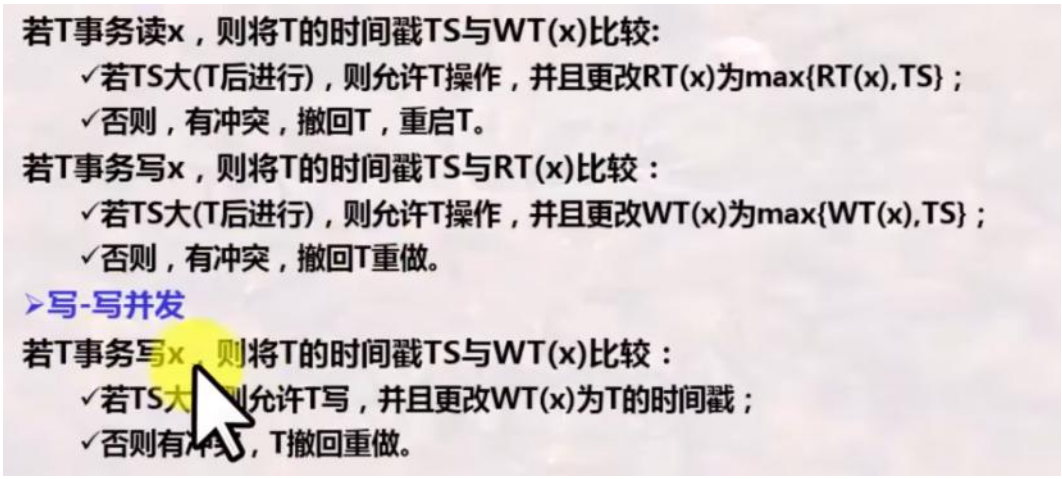
5. 原子性
   a. 可恢复调度：Tj读取了Ti所写的数据项 则Ti先于Tj提交
   b. 无级联：Tj读取了Ti所写的数据项，Ti必须要在Tj读操作之前提交

6. 事物隔离性级别：
   a. 可串行
   b. 可重复：只允许已提交数据，但是一个事务两次读一个数据项期间其余事务不可更新
   c. 已提交读：只允许已提交数据
   d. 未提交读

并发控制

2025年6月17日 20:59

1. 锁
   a. 协议 ti获得Q的
      i. S共享的 Ti可读但不写Q
      ii. X排他：可读可写
2. 两段锁协议
   a. 加锁阶段
   b. 解锁阶段
   c. 读写之前都要获得锁，加锁阶段不能解锁，接缩段不能加锁
   d. 一定冲突可串行
   e. 可能会死锁：两种事物都在等待释放
   f. 强两阶段 封锁协议 事务提交前不可释放锁
   g. 等待图

3. 时间cuo
   a. 表示事务T启动时刻，不用锁
   b. 并发事务的交叉执行，等价于特定顺序的串行执行
   c. 保证先执行的先操作
   d. 

      托马斯归责,写x，若TS小直接跳过wt

恢复

2025年6月18日 1:30

1. 事务故障
    a. Redo
    b. Undo
2. 系统故障
    a. 重做redo阶段
        i. 从上往下扫描日志，重做日志中的事务的操作
        ii. 遇到checkpoint L，将L加到undo list
        iii. 遇到正常操作就操作
        iv. 发现Ti start 把Ti加到undo list
        v. 发现Ti abort或者commit 从undo list去掉
    b. 撤销undo阶段
        i. 根据undo list从下往上回滚
        ii. 遇到ti 就输出ti 旧值 ti abort
        iii. 指导undo list空停止