数据库笔记（BNU MOOC版）

Chapter 2 关系模型

2.1 关系结构与约束

- 同一表不能出现完全相同的行
- 数据库数据应该与现实世界时时保持一致
- 外键：必须匹配另一个关系表中的属性（可以是其中的某个值或者空值），必须是另一个关系中的主键或者复合主键。
- 候选键一定是超键，超键不一定是候选键。一个关系中可能有多个候选键，用且仅用一个作为主键。
- 数据是否是超键由属性本身决定，而不是观察给出的几行数据中有无重复而确定
- 关系数据表中一定有主键，单独一个属性不行的时候可以使用复合主键

2.2 关系操作

基本关系代数运算

- 包含：选择、投影、并、差、笛卡尔积、更名运算
- 关系代数把表看成元组集合，关系运算的参数和结果都是关系，所有关系代数运算都去重
- 并和差运算只能在相容的关系之间进行

附加关系代数运算

- 包含：交（相容）、自然联接、除、赋值
- 定义附加运算没有增加关系代数的表达能力

> 关系代数的基本运算足以表达任何关系代数查询，只是有时候这些查询表达式会很复杂，定义附加运算是为了简化表达式，没有增加表达能力。

扩展关系代数运算

- 外联接：考虑悬浮元组。e.g.未参与阅卷的考官的个人信息也出现
- 聚集运算：$G_{avg(erage)}(examiner)$

数据库笔记（BNU MOOC版）

Chapter 3 PostgreSQL数据定义与操作

3.1 SQL与PostgreSQL

- PostgreSQL（PG）结构化查询语言
- SQL字面含义是“查询语言”，但其功能包括数据定义、查询、修改和保护等许多内容。
- SQL语言是大小写不敏感的。建议SQL保留字用大写，一般所有对象名用小写
- 查询（SELECT）语句中其它子句都可以不出现，但至少要有一个SELECT子句。
- 执行过程不是真实的，实际中数据库管理系统会为了尽可能快地获得等价查询结果而采取完全不同的执行步骤。

3.2 数据定义

- 定义表的属性时必须指明数据类型
- 插入元组可以直接插入常量元组，可以插入查询结果，可以插入表。插入元组的属性值必须在属性域中，属性值空null也在属性域中
- PG中使用单引号做字符串常量的标识，如果要查询带单引号的字符串，需要在其单引号后再加一个单引号，或者在 ' 符号前加一个转义符号 \ ，并在整个字符串前加一个 E 字母。
- WHERE 和 HAVING虽然后面都跟着一个条件表达式，但是它们不可以互换使用。没有FROM就没有WHERE，没有GROUP BY 就没有HAVING
- * 和 @ 可以得到表中的所有列
- 一些SQL语句举例

```
//检索数据
FROM
```

```
//数据查询
SELECT

//查询工资为4000或为8000的所有考官
SELECT * FROM examiner WHERE ersalary IN(4000,8000);
```

```
//数据定义
CREATE
DROP
ALTER

//------------------------------------------------------------------------------------
---

//创建表
CREATE TABLE

//修改表的结构
ALTER TABLE

//把表examiner的名字改为erexamine
ALTER TABLE examiner RENAME TO erexamine;
//向examiner表增加属性，属性名为er_entrance，数据类型为日期型
ALTER TABLE examiner ADD COLUMN er_entrance DATE;
//将erage的数据类型由SMALLINT型改为INT型
ALTER TABLE examiner ALTER COLUMN erage TYPE INT;
//删除examiner中的erage列
ALTER TABLE examiner DROP COLUMN erage;
```

```
//数据操控
UPDATE
INSERT
DELETE

//------------------------------------------------------------------------------------
---

//使examiner表中所有大于30岁考官的年龄+1
UPDATE examiner SET erage=erage+1
WHERE erage>30;


//向表中插入一个常量元组
```

```
INSERT INTO examinee(eedepa, eeage, eeid, eesex, eeename) VALUES ('历史学院', 20,
'218811011016', '男', '张强');
INSERT INTO examinee VALUES ('历史学院', 20, '218811011016', '男', '张强');
```

```
//数据控制
GRANT
DELETE
```

3.3 单表查询

- 投影：投影是指选取表中某些列的列值；广义投影是指在选取属性列时，允许进行适当运算。
- ORDER BY子句，按照后跟属性的出现先后决定优先级排序。升序用ASC，排序列为空值的行最后显示，降序用DESC，排序列为空值的行最先显示（空最大）。默认为升序
- 投影结果中可能出现所有列值均相等的重复行，DBMS采取惰性原则，非特别要求（DISTINCT）时保留重复。在关系代数向SQL语句转化的时候，需要注意加上DISTINCT
- HAVING子句的条件只对GROUP BY子句形成的分组起作用
- 关系代数中的一元运算有选择运算σ，投影运算Π，更名运算ρ三个。选择运算σ的作用相当于SQL语句中的WHERE，投影运算Π相当于SQL语句中的SELECT，更名运算ρ相当于SQL语句中的AS
- 聚集函数不能进行复合运算，聚集中SUM和AVG的输入必须是数值类型，其他聚集可以有非数值类型的输入

3.4 联接查询

- 外联接都要首先计算内联接，然后再在内联接的结果中加上相应的表中的悬浮行。e.g.**右外联接**就是把右侧表中的悬浮行补上空值后加入结果表，这些行中**来自左侧表的属性赋为空值null**
- 关键词顺序 e.g. NATURAL LEFT OUTER JOIN，OUTER可以省略，LEFT，RIGHT，FULL默认包含了OUTER
- 内联接INNER，外联接OUTER，默认为INNER

3.5 嵌套查询

- PG提供嵌套子查询机制，一个SELECT-FROM-WHERE语句是一个查询块
- 不相关嵌套查询：如果内层子查询不依赖于外层查询，称为不相关嵌套查询，可**由内向外逐层**处理。内层子查询的结果用于外层查询。
- 相关嵌套查询：这种情况下，对外层查询表中的**每一行**，根据它与内层子查询相关的列值处理内层查询，直至外层查询处理完为止。

表式嵌套

- 出现位置：WITH子句，FROM子句后
- 查询块可以出现在另外一个查询中表名可以出现的任何地方（子块返回一个表）
- **WITH子句** 定义一个临时表，在SELECT子句之前给出

```
//查询学生中平均分>=80的人数
WITH avgach(eeid, avgchieve) AS
          (SELECT eeid, AVG(achieve)
           FROM eeexam
           GROUP BY eeid)
SELECT COUNT(*)
FROM avgach
WHERE avgachieve>=80;
```

- **FROM子句** 中的AS可以省略不写，下面AS的括号表示可选，不是要加括号

```
//查询学生中平均分>=80的人数
SELECT COUNT(*)
FROM (SELECT eeid, AVG(achieve)
      FROM eeexam
      GROUP BY eeid) (AS) avgach(eeid, avgachieve)
WHERE avgachieve>=80;
```

集合式嵌套

- 出现位置：WHERE子句，HAVING子句后
- HAVING只对GROUP BY中的属性做限制，如果没有GROUP BY，则用WHERE即可。
- 当SELECT后面同时出现聚集和不聚集的属性，不聚集的属性一定要出现在GROUP BY子句中。
- 查询块也可以出现在集合能够出现的任何的地方
- **IN关键词**，判断是否属于子集合
- **EXISTS关键词**，如果内层查询的结果为空，则返回true，否则返回false
- **比较运算符**，
  - <>ALL：和所有的都不相等，没有出现
  - <>SOME（ANY）：和部分（某个）不相等，一般用来看两个集合有没有共同元素
  - =SOME（ANY）：和部分（某个）相等，一般用来看有没有交集
  - =ALL：和所有相等，一般用来看两个集合是否相等
- **WHERE子句**

```
//查询218811011013号考生报考的试卷号和试卷名
SELECT eid, ename
FROM exampaper
WHERE eid IN(SELECT eid
             FROM eeexam
             WHERE eeid='218811011013');
```

```
//查询所有报考了0205000002号试卷的考生详细信息
SELECT *
FROM examinee
WHERE EXISTS(SELECT *
             FROM eeexam
             WHERE examinee.eeid=eeid AND eid='0205000002');
```

- **HAVING子句**

```
//查询有名叫刘诗诗的考生的学院的考生平均年龄
SELECT eedepa, AVG(eeage)
FROM examinee
GROUP BY eedepa
HAVING eedepa IN(SELECT eedepa
                 FROM examinee
                 WHERE eename='刘诗诗');
```

标量式嵌套

- 出现位置：如果能确定查询块只返回单行单列的单个值，查询块可以出现在**单个属性名、单个表达式、单个常量，即单值表达式能够出现的任何地方。**
- 举例：SELECT子句，WHERE子句，ORDER BY子句、LIMIT子句（后跟数值，要求子查询结果为数值，限制显示条数）、OFFSET子句（后跟数值，要求子查询结果为数值，限制显示起始位置）后
- **WHERE子句**

```
//查询与218811011028号考生同院系的考生的报考号、姓名
SELECT eeid, eename
FROM examinee
WHERE eedepa = (SELECT eedepa
                FROM examinee
                WHERE eeid='218811011028');
```

- **SELECT子句** 和 **GROUP BY**子句

```
//查询每个院系的平均分
SELECT (SELECT eedepa
        FROM examinee
        WHERE examinee.eeid=eeexam.eeid),
        AVG(achieve)
FROM eeexam
GROUP BY (SELECT eedepa
          FROM examinee
          WHERE examinee.eeid=eeexam.eeid);
```

- **LIMIT子句 和 OFFSET子句**

```
//对examinee表按eeid升序，从考生人数1/4处开始，列出1/4的考生信息
SELECT *
FROM examinee
ORDER BY eeid
LIMIT (SELECT COUNT(*)/4
       FROM examinee)
OFFSET (SELECT COUNT(*)/4
        FROM examinee);
```

- **SELECT子句 和 ORDER BY子句**

```
//查询每个院每个考生得分，列出院名、考生名和分数，按院名升序排列，同院按分数升序排列
SELECT (SELECT eedepa
        FROM examinee
        WHERE examinee.eeid=eeexam.eeid),
       (SELECT eename
        FROM examinee
        WHERE examinee.eeid=eeexam.eeid),
        achieve
FROM eeexam
ORDER BY (SELECT eedepa
          FROM examinee
          WHERE examinee.eeid=eeexam.eeid),
          achieve ASC;
```

数据库笔记（BNU MOOC版）

Chapter 4 PostgreSQL应用

4.1 应用体系结构

数据库系统结构

- 网络上的数据库系统将数据的表示、处理、存储分为两层、三层或多层。常见的有C/S（Client/Server）结构 和 B/S（Browser/Server）结构
- SQL语言连接数据库，高级语言进行数据处理和表示。高级语言可以通过SDBC / JDBC / ADO等调用SQL
- C/S结构，包括服务器、客户机两层。需要针对不同的操作系统开发不同版本的软件，当系统升级时，每一台客户机都需要安装客户端新版本，维护和升级成本高。如腾讯QQ、网络多人游戏等的就是典型的基于C/S体系结构
- B/S结构，多层结构：客户端——Web服务器——应用服务器——数据库服务器——数据库。客户端不需要安装任何软件，只需要一个浏览器就能与服务器进行交互。只需要在服务器端维护升级，一般客户端不需要专门维护。很多采用ASP、PHP、JSP技术的网站就是典型的基于B/S体系结构的

Web与数据库

- Web与数据库是当今互联网上两大无处不在的关键基础技术
- **数据库** 管理具有严格的数据模型与数据模式，有标准查询语言SQL；**Web** 结构松散、随意、访问自由
- 浏览器和Web服务器主要使用HTML语言，应用程序通常使用高级语言编写的，**从HTML到SQL**需要两个桥梁，HTML与高级语言之间的CGI、ASP、JSP等桥梁，高级语言与数据库之间的JDBC、ODBC、ADO等桥梁

4.4 PG中的函数

PL/pgSQL语言与存储函数

- PG允许使用各种不同的程序设计语言来编写函数，特别是**内建了PL/pgSQL**。函数经编译和优化后，存储在数据库服务器中，可以反复被调用
- 存储函数具有以下优点：
  - 不需要额外的语法分析步骤
  - 降低了客户机和服务器之间的通信量
  - 便于实施业务规则
- PL/pgSQL语言与高级语言在功能上十分类似，但存在一些格式上的不同
  - pgSQL是非过程化的
  - PL/pgSQL是为PG扩展的过程语言
  - 数据操作和流程控制
  - 用于创建存储函数
  - 基于块结构，块中的每个声明和每条语句都是以分号结束，如果某子块在另一块中，那么该子块的END保留字后面必须以分号结束，函数体的最后一个END保留字后面可以省略分号

语句规则

- PL/pgSQL可以使用pgSQL的所有数据类型来声明变量，比如integer、varchar、char等

```
//定义一个整型变量uid
uid integer;
```

```
//变量achieve是小数点两边总共五个数字，后含两位小数的数值型
achieve numeric(5,2);
```

- 变量名称: = 表达式 ——赋值语句
- ELSIF和ELSEIF等价
- 使用pgSQL语言创建存储函数

```
//创建存储函数
CREATE OR REPLACE FUNCTION add(INT, INT)
  RETURNS INT
  AS
  $ $
  DECLARE intsum INT;
  BEGIN
    SELECT $1+$2 INTO intsum;
    RETURN intsum;
  END;
  $ $
  LANGUAGE plpgsql
```

- 执行存储函数
  - **方法1**: SELECT 函数名([参数1，参数2……]);，会存储函数执行返回的结果
  - **方法2**: PERFORM 函数名([参数1，参数2……]);，调用执行存储函数，执行结果的内容无关紧要可以丢弃
- 删除存储函数
  - DROP FUNCTION 函数名([参数1，参数2……]);

数据库笔记（BNU MOOC版）

Chapter 5 PostgreSQL数据保护

5.1 数据保护

数据安全性

1、数据保护保密性：指仅允许经授权地读数据

- 数据值的保密
- 数据存在性的保密

2、数据完整性：指数据的可信度。通常保护完整性就是指仅允许合法授权的数据修改

- 数据值的完整性
- 数据来源的完整性

3、数据可用性：是指对数据的期望访问能力。保护数据可用性通常指减少数据库系统停工时间，保持数据持续可访问。

数据保护

- 安全策略：允许什么，禁止什么的说明
- 安全机制：实施策略
- **访问控制矩阵**：描述系统访问策略的最简单模型，矩阵的行对应用户或角色，列对应数据库对象，矩阵中的元素表示相应用户对相应数据库对象的访问权限，访问控制矩阵通常依据用户在系统应用中担任的角色确定。
- PG中的授权和收权语句可以赋予或撤销用户相应的访问权限，数据库管理系统确保只有获得授权的用户有资格访问数据库对象，令所有未授权人员无法接近，从而保护数据保密性和完整性。

5.2 视图

视图和表的异同

- 视图和表都是关系，都可在查询中直接应用
- DB中存储表的模式定义和数据；但只存储试图的定义，不存视图的数据，视图数据在**使用视图时临时计算**。

视图的使用机制

- **数据的逻辑独立性**：视图定义中引用的表成为基表，**当基表中的数据发生变化时，相应的视图数据也随之改变**。视图可以把基表结构的细节封装起来，表可以随应用进化而变化，但视图以及基于视图的应用程序，可以尽可能少地受表变化的影响，
- **数据存在性的保密性保护**：用户只能查询和修改视图中见得到的数据。

相关SQL语句

- 创建视图

```
//基于表创建视图
CREATE VIEW avgachieve(eeid, average) AS
  SELECT eeid, AVG(achieve)
  FROM eeexam
  GROUP BY eeid;
```

```
//使用视图创建视图
//首先基于表创建视图eeexamv1
CREATE VIEW eeexamv1 AS
  SELECT examinee.eeid, examinee.eedepa,
         exampaper.ename, eeexam.achieve
  FROM examinee, exampaper, eeexam
  WHERE examinee.eeid=eeexam.eeid
        AND eeexam.eid=exampaper.eid;
```

```
//接着使用eeexamv1创建视图eeexamv2
CREATE VIEW eeexamv2 AS
  SELECT eeid, ename, achieve
  FROM eeexamv1
  WHERE eedepa='历史学院';
```

- 修改视图

  PG只允许对可更新视图进行修改操作，可更新视图需要满足如下条件：

  - 视图是从单个关系只使用投影、选择操作导出
  - SELECT子句中只包含属性名，不包含其它表达式、聚集、DISTINCT声明等，查询中没有GROUP BY或HAVING子句，WHERE子句子查询不出现基表名。并且对视图的更新操作符合一般更新语句的规则，比如插入时主键不能为空，执行插入操作的视图的投影列需包含基础关系的键
  - 比如，对视图进行INSERT操作，实际是对基表的对应位置进行了对应属性的插入（元组的其他位置为空）；如果在视图定义的末尾包含**WITH CHECK OPTION**，数据库管理系统自动检查对视图的更新应满足视图定义中**WHERE**的条件，如果不满足此插入操作便会被拒绝执行。

5.3 访问控制

- 给定用户拥有在给定数据库对象上的给定操作权限
  - 用户
  - 数据对象
  - 操作权限

用户

- PG使用角色来统一管理用户，分登陆角色和组角色。登陆角色就是具有登陆权限的角色，相当于用户。组角色就是作为组使用的角色，一般不应当具有LOGIN属性。
- 默认情况下，新建立的数据库总是包含一个超级用户角色，并且默认这个角色名是postgres
- 创建角色

```
//创建组角色yanni
CREATE ROLE yanni;

//创建具有登陆权限的组角色yuxiaotong
CREATE ROLE yuxiaotong LOGIN;

//创建可创建数据库的组角色masu
CREATE ROLE masu CREATEDB;

//创建可创建角色的组角色lichen
CREATE ROLE lichen CREATEROLE;
```

```
//创建具有口令的组角色wangxi
CREATE ROLE wangxi PASSWORD ‘654321’;

//创建数据库超级用户角色nini
CREATE ROLE nini SUPERUSER;
```

- 变更角色

```
//修改组角色名yanni为newyanni
ALTER ROLE yanni RENAME TO newyanni;

//给组角色yangchen添加创建角色和数据库的这个权限
ALTER ROLE yangchen CREATEROLE CREATEDB;

//收回组角色yangchen创建角色和数据库的权限
ALTER ROLE yangchen NOCREATEROLE NOCREATEDB;
```

为了与SQL标准兼容，PostgreSQL也有用户管理命令，该角色管理的命令很相似。

```
//创建用户lini
CREATE USER nini;
和 CREATE ROLE nini LOGIN; 等价

//删除组角色yangni
DROP ROLE yangni;

//删除用户wangni
DROP USER wangni;
```

- **组角色**权限

```
CREATE ROLE Alice LOGIN INHERT
CREATE ROLE Bob NOINHERIT
CREATE ROLE Rose NOINHERIT
GRANT Bob TO Alice
GRANT Rose TO Bob
```

  - 在以角色Alice连接之后，该数据库会话将立刻拥有直接赋予Alice的权限加上任何赋予Bob的权限，因为Alice“继承”Bob的权限。不过属于Rose的权限不可用，因为即使Alice是Rose的一个间接成员，但该成员关系是通过Bob过来的，而该组有NOINHERIT属性
  - 在SET ROLE Bob之后，该会话将只拥有那些已经赋予Bob的权力，而不包含那些已经赋予Alice的权限。在CREATE ROLE Rose之后，该会话将只能使用已赋予Rose的权限，而不包括已赋予Alice或Bob的权限
- PG提供**GRANT语句**和**REVOKE语句**来给角色**授予或撤销数据库操作权限**

```
GRANT UPDATE(erid), SELECT
ON TABLE examiner
TO uYing;
```

```
REVOKE SELECT
ON TABLE examiner
FROM uYing
```

5.4 完整性约束

- DBMS无法保证数据始终与其对应的现实世界状态一致，但可以保证数据始终与系统中明确定义的约束一致。这种约束通常称为完整性约束。

分类

- 主键约束（实体完整性）
  - 不重复，不为空。使用PRIMARY KEY关键词
  - 单属性构成的主键——属性级/元组级约束，多个属性构成的主键——元组级约束，即在所有属性声明的后面PRIMARY KEY(eeid)。
  - 只有对关系进行插入或修改时，系统才检查主键约束。删除时不检查主键约束。
- 外键约束（引用完整性）：
  - REFERENCES后跟着另一个表中的属性

    ```
    CREATE TABLE eeexam{
      eeid CHAR(9) REFERENCES examinee(eeid),
      achieve SMALLINT
    };
    ```

  - 引用表插入元组或对外键列进行修改时，或被引用表进行删除或修改时，系统自动检查是否违背外键约束。
  - **违背外键约束**，有以下几种**处理策略**：
    - NO ACTION-拒绝相应操作；
    - RESTRICT-不允许事物检查推迟到晚些时候；
    - CASCADE-级联，删除被引用行的属性值或分别把被引用行的属性值更新为引用属性的新数值；
    - 设为空值或者默认值
- 非空约束

  NOT NULL
- 唯一值约束

  UNIQUE <候选键>，与主键约束不同，可为空值
- CHECK约束

- CHECK(P)子句指定一个谓词P，关系中的每一个元组都必须满足谓词P
- PostgreSQL目前仅支持属性级（插入元组或修改此属性值时）和元组级CHECK约束（涉及多个属性时使用元组级，插入元组或修改一行中任意属性时），目前尚不允许CHECK约束出现子查询
- 新值与约束相违背时拒绝更新，修改时不满足条件则拒绝执行

给约束命名

- 可以给约束起名便于更改，如果没有为约束命名，PG会自动命名
- 声明一个命名约束
  - 使用CONSTRAINT <约束名> UNIQUE(属性) 的格式。
  - 或者

```
ALTER TABLE department ADD CONSTRAINT dun UNIQUE(dloca);
ALTER TABLE department DROP CONSTRAINT dun;
```

5.5 触发器

- 触发器Trigger是用户定义在关系表上的由时间驱动调用函数的机制。比CHECK约束更灵活。

格式

```
//声明触发器函数
CREATE FUNCTION function_name()
RETURNS TRIGGER AS $<name>$
DECLARE 变量声明; //DECLARE部分可选
BEGIN
  函数执行代码;
END;
$$ LANGUAGE plpgsql; //对函数编写语言的说明
```

```
//创建触发器
CREATE TRIGGER name {BEFORE|AFTER} INSERT ON examinee
FOR EACH {ROW|STATEMENT}
[WHEN (condition)]
EXECUTE PROCEDURE function_name();
```

- WHEN子句（可选）的condition是个布尔表达式，指明**触发条件**：触发器被激活时，只有触发条件为真触发器函数才执行；否则触发器函数不执行；如果没有WHEN子句，则触发器函数在触发器激活后立即执行

行级触发器和语句级触发器

- 触发器函数必须返回一个NULL或者一个元组类型的变量
- 行级触发器的触发器函数为触发语句影响的每一行执行一次；语句级触发器的触发器函数为每条触发语句执行一次
    - 假设在examiner表上创建了一个AFTER UPDATE触发器，如果表examiner有10000行，执行如下这样一个语句UPDATE examiner SET erage=erage+1
    - 如果该触发器为语句级触发器，那么执行完该语句后，触发动作只发生一次，如果是行级触发器，触发动作将执行10000次
- 行级触发器函数，可以使用NEW或者OLD进行引用；语句级触发器函数，不能使用NEW或者OLD进行引用
- 行级AFTER触发器的值总是被忽略，可以返回NULL；行级BEFORE触发器的返回值不同，对触发器操作的影响也不同，如果返回NULL则忽略该触发器的行级别操作，其后的触发器也不会被执行，如果非NULL则返回的行将成为被插入或者更新的行。
- 通常，用行级BEFORE触发器检查或修改将要插入或者更新的数据。

激活触发器的顺序

执行该表上**语句级BEFORE**触发器

—>执行该表上**行级BEFORE**触发器

—>执行激活触发器的**SQL语句**

—>执行该表上的**行级AFTER**触发器

—>执行该表上**语句级AFTER**触发器

删除触发器

```
DROP TRIGGER <触发器名> ON <表名>
```

5.6 事务

- 事务是对数据库进行操作的程序单位。

ACID特性

- 原子性（**事务中包含的所有操作要么都做要么都不做**）
- 一致性（事务单独成功执行，使数据库**从一个一致性状态转换到另一个一致性状态**）
- 隔离性（**一个事务的执行不能被其他事务干扰**）
- 持久性（一个事务**一旦被提交，它对数据库中数据的改变就应该是持久性的**，接下来的其他操作或故障不应该对其结果有影响）

事务实现

事务声明

- PG用BEGIN和COMMIT（或ROLLBACK）将数据库访问操作指令序列包围以声明一个事务。如果没有显示的BEGIN命令，PG把每个SQL语句当作一个事务来看待。

事务隔离级别

1. **读已提交 READ COMMITED**（默认）：在这个隔离级别事务中的语句，看到的是其开始执行时瞬间数据库的一个快照，在同一个事务里两个相邻的SELECT命令可能看到不同的快照，因为其他事务会在第一个SELECT执行期间提交，读已提交提供的这种部分隔离对于许多应用而言就足够，并且这个模式速度快，使用简答。
2. **可重复读 REPEATABLE READ**：如果一个事务需要连续做若干个命令，而这几个命令必须看到完全相同的数据库视图，可以选择可重复读隔离级别，同一个事务内部后面的SELECT命令总是看到同样的数据。

   ```
   //设置隔离级别为REPEATABLE READ
   SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
   ```

3. **可串行化 SERIALIZABLE**：看到的是该事务开始时的快照，执行最严格的隔离级别。

- 可重复度和可串行化隔离级别事务中的语句，看到的是该事务开始时的快照，而不是该事务内部当前查询开始时的快照，这一点和读已提交不一样。REPEATABLE READ和SERIALIZABLE隔离级别的事务里面的SELECT命令总是看到相同的数据状态。可串行化级别提供最严格的事务隔离，就像（实际不是）事务是一个接着一个那样串行执行似的。

5.7 加密

- 数据被称为**明文**，用某种方法伪装数据以隐藏它的内容的过程称为**加密**。加密所用的方法称作**加密算法**，数据被加密后的结果被称为**密文**，把密文还原为明文的过程称为**解密**，解密所用的算法称为**解密算法**。
- 加密体系中**最核心**的是用于加密解密的算法和密钥。
- 现代加密体系中算法通常是公开的，密钥是保密的并且需要向可信权威机构申请，**安全性完全取决于密钥的保密性**。

加密算法的分类

加密算法一般可分为加密解密密钥相同的**对称加密**、加密解密密钥不同的**非对称加密**、**单向加密**三种。

- **对称加密**体系当中代表性的算法有DES算法、三重DES、AES算法等等。
- **非对称加密**体系当中的代表性算法有RSA算法、DSA算法。
- **单向加密**体系当中的代表性算法有MD5、SHA算法等等。

pgcrypto扩展包

- 在PG中使用加密技术，首先需要通过**CREATE EXTENSIOIN pgcrypto**创建pgcrypto扩展包。
- 该扩展包包含了当前主流加密解密算法的函数定义。

数据库笔记（BNU MOOC版）

Chapter 6 数据库设计：实体-联系方法

6.1 数据库设计方法和生命周期

数据库设计方法分类

- 数据库设计方法主要包括实体-联系方法和属性-联系方法两种

  【宏观 把握关键】实体-联系方法以**实体**为中心，着重于一个关系模式基本对应一个实体或联系，即关系模式与实体或联系之间基本是一一对应的。

  【微观 进行优化】属性-联系方法以**属性**为中心，着重于属性之间的**依赖关系**。把需要数据库保存的所有属性放在一张关系表中，进而通过属性之间的联系来优化这个关系模式。属性-联系方法没有概念设计阶段。

实体-联系设计方法的主要过程

1. 分析数据需求，有了数据需求，就可以进行概念设计，即设计概念模式，概念模式与具体DBMS无关，通常使用实体–联系图表示，也叫E-R图。
2. 在概念模式的基础上，进行逻辑设计，即将概念模式转换成相应的逻辑模式，获得符合选定DBMS数据模型的逻辑结构，比如关系模式。
3. 物理设计，在需要的时候进行一些参数的设置和选择，比如索引机制、块大小，由于关系数据库系统的物理数据独立性，在逻辑模式确定后伴随着物理设计，就可以进行应用程序的设计
4. 物理设计完成后，就可以开始实现数据库，用DDL/DPL定义数据库结构、把数据入库、编制与调试应用程序并进行数据库试运行，如果试运行中**功能指标和性能指标**都达到了设计目标，数据库系统就可以正式投入运行。
5. 在运行过程当中，还要经常对数据库进行维护
   - 定期对数据库和事务日志进行备份，保证发生故障时，能利用这些备份，尽快将数据库恢复到一个一致的状

态
    - 当应用环境、用户、完整性约束等出现变化时往往需要根据实际情况调整原有安全策略、完善原有安全机制
    - 利用DBMS提供的系统性能监测工具监督系统运行，必要时调整某些参数，改进系统性能
    - 针对数据库性能随着数据库运行逐渐下降问题，必要时重组数据库，回收垃圾，减少指针链，提高系统性能
    - 针对应用需求的变化，适当调整数据库模式也叫重构数据库
6. 如果实际应用发生了根本性的变化，重构数据库的代价就会非常大，那么这个时候，我们就应该终止现有的数据库应用系统生命周期，重新建立新的数据库系统，开始新的生命周期

6.2 E-R模型的基本元素

E-R模型的基本元素

- E-R图主要包含实体和联系，以及它们各自的属性。
- 实体【方框】和实体集统称为实体。实体之间不是孤立的，总是存在一些联系。
- 联系【菱形框】就是一个或多个实体之间的关联关系。同类联系组成的集合称为联系集。习惯上，把联系和联系集统称为联系。
- 属性域：属性【椭圆】可能取值的范围。
- 主键（标识符）要加下划线，指能够并且用以区分一个实体集中不同实体的最小的属性集，组成主键的属性称为标识属性。

6.3 基本E-R图设计

联系元数

- 联系关联的实体个数称为该联系的元数或度数
    1. 同类实体集内部实体与实体之间的联系，称为一元联系。（e.g.一个人会喜欢另一个人，人与人之间的“喜欢”是一个一元联系。）
    2. 两个不同实体集中实体之间的联系，称为二元联系。
    3. 三个不同实体集之间的联系称为三元联系。
    4. 如果实体集E1中，每个实体可以与实体集E2中任意个（零个或多个）实体之间具有联系，并且E2中每个实体至多和E1中一个实体有联系，那么我们就把E1对E2的联系称为“一对多联系”。

映射基数

- 如果实体集E1和E2之间有二元联系，则参与该联系的实体数目称为映射基数。（一对一 / 一对多 / 多对多）注：这里的“多”指0个或多个，即任意个。
    - 一对一实例：一个学生只有个身份证编号。
    - 一对多实例：一个班级有多个学生。
    - 多对多实例：多对多就是双向一对多，一个学生可以选择多门课，一门课也有多名学生。

属性分类

- 复合属性：可以按组成分解成一系列子属性。
- 单值属性：同一个实体在这个属性上只能取一个值。
- 多值属性：同一个实体的某些属性可能取多个值。如，一个考官可能有多个联系方式。可以用多个单值属性代替单个多值属性。
- 派生属性：能从其他属性值推导出值的属性。

完全参与和部分参与

- 如果实体集S中的每个实体都参与联系集L的至少一个联系，称实体集S“完全参与”【双线】联系集L。
- 如果实体集S中只有部分实体参与联系集L的联系，称实体集S“部分参与”【单线】联系集L。

6.4 基本E-R图转换为关系模式

1. 一个实体转化成一个关系模式
    - 实体的属性-表的列
    - 实体的主键-表的主键
2. 一个联系转化成一个关系模式
    - 如果联系M:N，主键是所有参与联系的各实体主键的并集加上联系自己特有的属性；
    - 如果联系1:N，主键是多端实体主键；
    - 如果联系1:1，主键是任一端实体主键。
3. 主键相同的关系模式可以**合并（关系的精简）**
    - 如果联系是1:N，那么联系表可合并到多端实体的表
    - 如果联系是1:1，那么联系表可与任一端实体对应的表合并在一起

- 由联系转换来的表的主键与任一端实体主键相同。❌
- e.g. 借还书数据库
    - 图书（ISBN, 书名，作者，出版社，出版时间，图书状态）
    - 读者（读者号，姓名，类型）
    - 借阅（读者号，ISBN，借书时间）

数据库笔记（BNU MOOC版）

Chapter 7 数据库设计：属性-联系方法

7.1 数据依赖

- 数据库中所保存的数据值是对现实世界状态的反映，无论现实世界的状态如何变化，一个关系模式中不同属性在取值上总会存在相互依赖又相互制约，这种属性与属性之间的联系，称为数据依赖。
- 数据依赖是**属性-联系数据库**设计方法的基础

数据依赖种类

- 函数依赖
  - 具有实用价值，最重要的数据依赖。
  - 属性A_i决定属性A_j，$A_i \rightarrow A_j$
- 多值依赖
- 联接依赖

逻辑蕴涵

- 定义：给定关系模式S的**函数依赖集D**，可以证明其它一些函数依赖也成立，就称这些**被证明成立的函数依赖是被D逻辑蕴涵**
- 闭包：给定关系模式S的函数依赖集D，D**逻辑蕴涵的所有函数依赖的集合称为D的闭包**，记作D^+

Armstrong公理

- 反射律：若$A_j \subseteq A_i$，则 $A_i \to A_j$
- 增广律：若$A_i \to A_j$，则$A_iA_k \to A_jA_k$
- 传递律：若$A_i \to A_j$，$A_j \to A_k$，则$A_i \to A_k$，

三条重要的规则（基于Armstrong公理推出）

- 合并规则：若$A_i \to A_j, A_i \to A_k$，则 $A_i \to A_jA_k$
- 分解规则：若$A_i \to A_jA_k$，则$A_i \to A_j$，　$A_i \to A_k$，
- 伪传递规则：若$A_i \to A_j, A_jA_l \to A_k$，则$A_iA_l \to A_k$

最小函数依赖集

- 不同函数依赖集的闭包可能是相等的。$G^+ = H^+ \Rightarrow$H与G等价，H中的每个函数依赖属于G的闭包，G的每个函数的依赖属于H的闭包。因此可以只关注函数依赖集中最小函数依赖集。
- 最小/极小函数依赖集条件：
  1. D中任一函数依赖的右部仅含有一个属性
  2. D中每一个函数依赖的左部都不包含多余的属性
  3. D中不包含多余的依赖（可以由其他依赖推导出就是多余的）
- 每一个函数依赖集**都有**等价的极小函数依赖集D_m，一个函数依赖集的极小依赖集**不唯一**

平凡/非平凡函数依赖

- 若$A_i \to A_j$，但A_j不是A_i的子集，则称$A_i \to A_j$ 是**非平凡的函数依赖**
- 若$A_i \to A_j$，A_j是A_i的子集，则称$A_i \to A_j$ 是**平凡的函数依赖**
- 除非特别说明，**一般只考虑非平凡的函数依赖**。

7.2 模式分解

- 关系数据库设计的**属性-联系**方法，就是把需要数据库保存的所有属性放在一张关系表中，进而给予数据依赖来优化这个模式，得到期望的结果，这一过程的基本操作就是模式分解。

无损联接与保持依赖

- 是否无损联接：元组的增多意味着信息的丢失；分解后自然联接的结果与原表一致则是无损联接。
  - 方法1：使用 **追赶算法** 检验，在过程中，某一行完全变x，则分解具有无损联接性。
  - 方法2：把一个关系模式分解为两个关系模式时，**分解具有无损联接性当且仅当两个关系模式的公共属性是其中一个模式的键**

    - 方法3：分解后两个表的**自然联接**的结果都与原来表中的内容保持一致
- 是否保持依赖：如果某个分解能保持函数依赖，那么就可以在分解后的模式上定义等价的完整性约束，在数据输入或更新时，要求每个函数依赖被满足，就可保证数据库中数据的语义完整性。否则，完整性约束会被削弱。
    - 方法：判断**函数依赖的闭包相等**
- 无损联接和保持依赖二者是相互独立的标准，不能互相推出。（保持/不保持共有4种组合方式）
- 一般希望模式分解至少具有无损联接性，才有实际意义，最好也能保持函数依赖。
- 保持函数依赖的分解是关系数据库模式S分解的一个性质，这里S_i中的每个函数依赖：直接出现在S分解得到的一个关系模式S_i中的D_i中，或者由S_i中的函数依赖D_i推出。

举例（判断函数依赖的个数）

- 设有关系模式$R(A,B,C,D)$

  $F=\{B\to A,D\to C\}$，则 F^+中左部为 (BC) 的函数依赖有 8 个

  因为：$(BC)^+=\{B,C,A\}$，$2^3=8$。

7.3 范式

分类

- 完全依赖（F）：左侧已经是最小集合
- 部分依赖（P）：存在能达到相同效果的真子集
- 传递依赖（T）：$A\to B,B\to C$，则$A\to C$是传递依赖
- 直接依赖（D）：不存在传递依赖中的B

  完全和直接意味着属性之间的依赖和制约性更强，属性之间的联系更紧密、更亲近；部分和间接则相反。

键基于函数依赖的定义

- **超键**：设K为S<A,D>的属性或属性组，若K决定A，则称K为S的超键
- **候选键**：设K为S<A,D>的超键，若K完全决定A，则称K为S的候选键
- **主键**：若S<A,D>有多个候选键，则可以从中选定一个作为S的主键
- **候选键中的属性，称作主属性**；不包含在任何候选键中的属性称为非主属性
- 实际当中常用的是3NF和BCNF

1～3范式（1～3NF）

- 第1范式：如果关系模式S的每个关系的**每个属性值都是不可分的原子值**，称S是第一范式的模式。**1范式是关系模式起码的条件**。
- 第2范式：如果关系模式S是1NF，且每一个非主属性都不部分依赖于S的任何候选键，则S属于2NF。（即非主属性不存在部分依赖关系）**允许存在传递依赖**，是第1和第3中的过度阶段
- 第3范式：如果关系模式S<A,D>是1NF，且每个非主属性都既不部分也不传递依赖于S的任何候选键，那么称S是第3范式的模式。

BC范式（BCNF）

- 说法一：如果关系模式S<A,D>是第三范式，它的任何一个主属性都既不部分也不传递依赖于S的任何候选键，则称S属于BCNF
- 说法二：如果关系模式S<A,D>（则一定是1NF），它的任何一个（非主、主）属性既不部分也不传递依赖于任何候选键，则称S属于BCNF
- 说法三：如果关系模式S<A,D>属于1NF，其D中任意一个非平凡函数依赖的决定因素都包含（候选）键，则S属于BCNF

7.4 规范化

- 规范化：

  一个**较低范式**的关系模式，依据其中的数据依赖、通过**模式分解**可以转换为**高范式**关系模式的集合，这个过程称为规范化。

- 关系模式规范化实际上就是一个模式分解过程：把逻辑上相对独立的信息放在独立的关系模式中。
- 属性分组使**候选键的决定力越纯粹统一**，模式达到的**范式越高**，则对事务处理的支持越强。（BCNF的要求比3NF更高，范式越高，分得越开，保持依赖性越不容易满足）。
- 如果**仅仅要求分解具有无损联接性**，那么**一定能够达到BCNF**。(即不保证保持依赖)。
- 如果要求分解**既具有无损联接性，又具有保持依赖性**，则**一定能够达到3NF**，但**不一定能够达到BCNF**。
- 例子
  - 无损联接分解关系模式达到BCNF

    模式：研究生导师（研究生号，导师号，院系名）

    - **说明**：每个导师可以指导多名研究生但只能在一个院系工作；每位研究生可以有多位导师，但是在每个院系只能有一个
    - **函数依赖**：(研究生号，导师号)->院系名

      导师号->院系名

      (研究生号，院系名)->导师号
    - **候选键**：(研究生号，导师号)、(研究生号，院系名)
    - **解**：研究生号、导师号、院系名都是主属性，存在“导师号——>院系名”，所以“（研究生号，导师号）——>院系名”是一个主属性对候选键的部分依赖，这说明研究生导师模式不是BCNF

      为了将该模式无损联接地分解为BCNF，首先要初始化，之所以该模式不是BCNF，是因为D中函数依赖“导师号——>院系名”的左部没有包含候选键“（研究生号，导师号）”或“（研究生号，院系名）”。

      将该函数依赖左右两端的属性一起作为一个新的模式并给起名“导师院系”，将研究生导师模式中的属性“院系名”移去后剩下的属性成为一个模式并给起个名字“师生”。这样分解后包括导师院系模式和师生模式，导师院系有函数依赖，师生模式的函数依赖集中没有非平凡的函数依赖，此时导师院系和师生两个关系模式都已经达到BCNF，分解结束。
  - 无损联接且保持依赖地分解成3NF

【例1】**模式：考官院系（考官号，姓名，院系名，院系总人数）**

- {考官号->姓名，考官号->院系名，院系名->院系总人数}
- {考官号->(姓名，院系名)，院系名->院系总人数}
- 考官(考官号，姓名，院系名)、院系(院系名，院系总人数)

  解：考官院系不是BCNF，为了保证分解无损联接和保持函数依赖，
- 首先求解该模式上的函数依赖集的极小依赖集（一共三个依赖）
- 合并左部相同的函数依赖得到第三行；
- 分别将这两个函数依赖的两端出现的属性作为单独的模式，并给起上合适的名字就得到两个模式（第四行），原模式“考官院系”的候选键“考官号”已经出现在刚刚产生的“考官”模式中。
- 总共分解为两个模式，分解结束。

【例2】**模式：报考（报考号，姓名，试卷号，试卷名）**

- {报考号->姓名，报考号->试卷名}
- 考生(报考号，姓名)、试卷(报考号，试卷名)、报考(报考号，试卷号)

  解：有关系模式报考，它仅仅是1NF，为了将该模式无损联接且保持依赖地分解以达到3NF
  首先求解该模式上的函数依赖集的极小依赖集，并合并左部相同的函数依赖，得到第二行；
  分别将这两个函数依赖两端出现的属性作为单独的模式并给起上合适的名字，得到第三行；
  原模式“报考”的候选键“（报考号，试卷号）”没有出现在改刚刚产生的两个模式当中，所以单独作为一个模式并且一个合适的名字，总共分为三个模式（第三行）；
  分解结束。

- 不能简单地说数据冗余就不好，因为数据冗余有利有弊。不能简单地说关系模式满足的范式级别越高越好，因为高范式和低范式各有千秋。在设计数据库模式结构时，必须对现实世界的实际情况和用户应用需求作进一步分析，以选择一个合适的规范化和冗余的折中处理。

数据库笔记（面授补充章节）

Chapter 9 查询处理与优化

9.1 问题提出

一个查询往往有多种执行方案，每种方案的效率不同，差别很大。分布式系统中，累积的差距更大，将直接影响系统整体性能。找代价最小的执行方案——查询优化。

- 一个查询：多种SQL表达方式
- 一个SQL语句：多种等价的关系代数表达方式——运算顺序
- 一个表达式：每个运算多种不同实现算法——e.g.是否使用索引
- 两个相邻运算：多种不同传递数据的方式——是否使用流水线

9.2 定性优化——启发（经验）式

中间结果最小

- 尽可能**早**地执行**选择**运算
- 尽可能**早**地执行**投影**运算
- 将**选择和笛卡尔积**运算**组成自然联接**
- **较小的**关系作为**联接运算的左参数**
- 举例：查询报考'6'号试卷的考生姓名
  - 优化前：$Q_1 = \pi_{eename}(\sigma_{examinee.eeid=eeexam.eeid \quad AND \quad eid='6'}(examinee \times eeexam))$

- 优化后：$Q_1 = \pi_{eename}((\pi_{eeid}(\sigma_{eid='6'}(eeexam)))\infty(\pi_{eeid,eename}(examinee)))$

9.3 定量优化——代价估计

- 利用这些统计信息来估计实现各种关系代数运算的算法花费
- 这里提到的统计信息是经过简化的，实际系统的查询优化器通常包含更多的统计信息
- 查询树的根节点表达查询的结果

Chapter 10 事务处理

10.1 数据管理

- 涉及的存储器
  - 易失性存储器：内存、cache存储器
  - 非易失性存储器：磁盘
  - 稳定存储器：存储在稳定存储器中的信息是绝不会丢失的
- 保护管理

  数据库的程序都是以事务为单位运行的

10.2 事务故障

故障类型

- 事务故障：e.g. 运算溢出，转账时发现账面金额不足
- 系统故障：内存信息丢失，但未破坏外存中数据。e.g. CPU故障，突然停电
- 磁盘故障：e.g. 磁盘磁头碰撞、瞬时的强磁场干扰

10.3 事务恢复

基本恢复技术

- 支撑材料
  - 备份：将数据库复制到磁带或者另一个磁盘上。备用数据称后备（后援）副本
  - 日志：系统自动记录数据库每一次更新活动的文件
- 故障后采取措施，将数据库恢复到一个一致性的状态
- 原则：先写日志，日志记录**将要**发生何种修改，DB表示**实际**发生何种修改

- 事务分类：
  - 圆满事务（有commit标识）
  - 夭折事务（只有begin transaction标识，无commit标识）
  - 举例：τ_0圆满事务，τ_1夭折事务

    $< \tau_0 \quad start >$

    $< \tau_0, \quad x, \quad 1000, \quad 950, \quad U >$ //记录名，id，更新前值，更新后值，更新操作

    $< \tau_0, \quad y, \quad 2000, \quad 2050, \quad U >$

    $< \tau_0 \quad commit >$

    $< \tau_1 \quad start >$

    $< \tau_1, \quad z, \quad 700, \quad 600, \quad U >$
- 恢复的基本操作：
  - 对圆满事务所做过的修改操作应执行redo，修改对象被赋予新记录值
  - 对夭折事务所做过的修改操作应进行undo

事务故障恢复

- 撤销事务已对数据库所做的更改
- 措施
  - 反向扫描日志文件，查找该事务的更新操作
  - 对该事务的更新操作执行逆操作，即将事务更新前的旧值写入数据库
  - 继续反向扫描日志文件，查找该事务的其他更新操作，并做同样处理
  - 如此直至读到该事务的开始标识，事务的故障恢复就完成了

系统故障恢复

- 不一致状态原因
  - 未完成事务对数据库的更新已写入数据库
  - 已提交事务对数据库的更新未写入数据库
- 措施
  - 正向扫描日志文件，找出圆满事务，记入重做队列；找出夭折事务，记入撤销队列
  - 反向扫描日志，对撤销队列中事务τ_i的每一个日志记录执行undo操作
  - 正向扫描日志文件，对重做队列中事务τ_i的每一个日志记录执行redo操作
- Checkpoint检查点：只做最后一个检查点到故障时刻的undo和redo操作即可

介质故障恢复

- 磁盘上数据文件和日志文件遭到破坏
- 措施
  - 装入最新的数据库后备副本，使数据库恢复到最近一次备份时的一致性状态
  - 装入相应的日志文件副本，重做已完成的事务