数据库复习笔记@onevfall

ch1

1. DDL和DML的区别
2. SQL is **NOT** a Turing machine equivalent language
3. DBA：database administrator

ch2

1. *R* = (*A*1, *A*2, ..., *An*) is a *relation schema*, A relation instance *r* defined over schema *R* is denoted by *r* (*R*).
2. **Relations are Unordered**: Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
3. Database schema && Database instance
    - Database schema -- is the logical structure of the database;
    - Database instance -- is a snapshot of the data in the database at a given instant in time.
4. keys:
    - *K* is a **superkey** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*
    - Superkey *K* is a **candidate key** if *K* is minimal
    - One of the candidate keys is selected to be the **primary key**.
    - **Foreign key** constraint: Value in one relation must appear in another
5. relational algebra is not Turing-machine equivalent
6. Six basic operators of relational algebra
    - **select: σ**
        - 选出满足该谓语要求的数据
        - 可以使用与∧或∨非 的连接词
        - 这句话等同于SQL中的where语句
    - **project: ∏**
        - 投影，是直接选取想要的列
        - 注意投影时，冗余会被删除，因为这是relations（Duplicate rows removed from result, since relations are sets），此点与后面章节的数据库SQL有区别
        - 这句话等同于SQL中的select语句
    - **union: ∪ 和 set-intersection ∩**
        - 本质是相同表格属性下的tuple的并集或交集
    - **set difference: –**
    - **Cartesian product: x**
        - *instructor* X *teaches* associates every tuple of instructor with every tuple of teaches.
        - 这句话等同于SQL中的from语句（连着罗列几个relation就是在做Cartesian product）
        - Since the instructor *ID* appears in both relations we distinguish between these attribute by attaching to the attribute the name of the relation from which the attribute originally came.即*instructor.ID* 和*teaches.ID*，可见：这里虽然拼接在一起，但相同的那一列依然独立开的，是两个表的各个行全相乘，即所有情况的罗列，没有经过任何拼接（不匹配，有待筛选）！

    - 而筛选是join的操作，如σ 中附有*instructor.id = teaches.id* 的要求
    - Let “theta” be a predicate on attributes in the schema R “union” S.
    - $$r \bowtie_\theta s = \sigma_\theta (r \times s)$$

- **rename:**

ch3

1. DDL
    - create table r(ID char(5), name varchar(20),)
    - integrity constraints：primary key, foreign key, not null
    - Delete VS Drop Table
        - Delete:删除table的tuples，对应于删除data file里的data，而不改变dictionary的结构，即table中的内容。
        - Drop table：既删除datafile里的所有data，也删除dictionary的structure，即整个table没了。
2. the result of an SQL query is a relation.
3. **select** clause：
    - SQL names are case insensitive.(大小写不敏感) Name = NAME = name
    - SQL allows duplicates in relations as well as in query results. To force the elimination of duplicates,insert the keyword **distinct** after select.【这点与关系代数查询不同，关系代数查询后会自动删除所有重复，因为那是集合set】

      ```
      /*删除重复*/
      select distinct dept_name
      from instructor
      /*保留重复*/
      select all dept_name
      from instructor
      ```

    - " * " denotes "all attributes"

      ```
      select * from instructor
      ```

    - An attribute can be a literal
        - An attribute can be a literal with no **from** clause, `select '437'` means that results is a table with one column and a single row with value “437”.
        - An attribute can be a literal with **from** clause, `select 'A'  from instructor` means that Result is a table with one column and *N* rows (number of tuples in the *instructors* table), each row with value “A”.
    - can contain arithmetic expressions involving the operation `+`、`-`、`*`、`/`。
4. **where** clause
    - The **where** clause specifies conditions that the result must satisfy. Corresponds to the selection predicate of the relational algebra.
        - 本质上是谓语筛选，筛选出满足条件的，比如属性等于xx、属性 **in** 嵌套子查询返回的结果（相当于满足了子查询条件的结果）等等。

        - 可以使用逻辑连接词 **and**, **or**, **not**
5. **from** clause
    - The **from** clause lists the relations involved in the query, Corresponds to the **Cartesian product** operation of the relational algebra.
    - **属性自动重命名【按原来的所属关系】**： For common attributes (e.g., *ID*), the attributes in the resulting table are **renamed** using the relation name (e.g., *instructor.ID*)
6. **as** clause:
    - **rename operation**：old-name **as** new-name
    - Keyword **as** is optional and may be omitted
        - *instructor* **as** *T* ≡ *instructor T*
7. **string** operation:
    - The operator **like** uses patterns that are described using two special characters:
        - percent (%). The % character matches **any substring**.
        - underscore (_). The _ character matches **any character**.
    - 例子：Find the names of all instructors whose name includes the substring “dar”.
        - ```
          select name from instructor where name like '%dar%'
          ```

- 既然%有特殊含义，那么就需要使用转义符'\', like '100\%' escape '\'代表 match the string "100%" **【此处不太懂】**

8. Order the display of tuples
    - List in alphabetic order the names of all instructors :

      ```
      select distinct name from instructor order by name
      ```

    - We may specify **desc** for descending（降序） order or **asc** for ascending（升序） order, for each attribute; ascending order is the default.
        - Example: **order by** *name* **desc**
    - Can sort on **multiple attributes** 【多级属性排列，先排前面，再排后面】
        - Example: **order by** *dept_name, name* （优先级不同）
9. **Where Clause Predicates**
    - SQL includes a **between** comparison operator:

      ```
      select name from instructor where salary between 90000 and 100000
      ```

    - Tuple comparison:

      ```
      where (instructor.ID, dept_name) = (teaches.ID, 'Biology');
      ```

10. set operation：
    - **union**, **intersect**, and **except**： Each of the above operations automatically eliminates duplicates
    - **union all**, **intersect all**， **except all** can retain all duplicates.
    - use example:
      ```

```
/*Find courses that ran in Fall 2017 or in Spring 2018*/
(select course_id  from section where sem = 'Fall' and year = 2017)
union
(select course_id  from section where sem = 'Spring' and year = 2018)
/*Find courses that ran in Fall 2017 and in Spring 2018*/
(select course_id  from section where sem = 'Fall' and year = 2017)
intersect
(select course_id  from section where sem = 'Spring' and year = 2018)
/*Find courses that ran in Fall 2017 but not in Spring 2018*/
(select course_id  from section where sem = 'Fall' and year = 2017)
except
(select course_id  from section where sem = 'Spring' and year = 2018)
```

11. null values
    - **null** signifies an unknown value or that a value does not exist.
    - **where** *salary* **is null**、**where** *salary* **is not null** 等等可用于check
    - Null的运算仍为null
12. **Aggregate Functions**:
    - 理解:
        - PPT：These functions operate on the multiset of values of a column of a relation
        - 本人对聚集函数理解：是对某一列的各个元素的统计值
    - 统计值包括如下:

      avg:average value
      min: minimum value
      max: maximum value
      sum: sum of values
      count: number of values
    - 与**group by**的联用:
        - 将某些属性分开，如department分开了不同学院自成一组
        - 谨防误用：Attributes in **select** clause outside of aggregate functions must appear in **group by** list【讲得好！】

          ```
          /* erroneous query */
          select dept_name, ID, avg (salary)
          from instructor
          group by dept_name;
          /*(ID is not fit for group by dept_name,此处ID就没在group by的list
          里)*/
          ```
    - 与**Having Clause**的联用:
        - 理解：是对group by分组后在每组内进行的再筛选
        - 区分**having**与**where**： predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups【详细讲解：https://www.cnblogs.com/fanguangdexiaoyuer/p/6268211.html】
        - “Where” 是一个约束声明，使用Where来约束来之数据库的数据，Where是在结果返回之前起作用的，且Where中不能使用聚合函数。

          “Having”是一个过滤声明，是在查询返回结果集以后对查询结果进行的过滤操作，在Having中可以使用聚合函数。

13. **Subqueries**（含嵌套子查询和关联子查询）
    - 语法适用：
        - **From clause:** *ri* can be replaced by any valid subquery
        - **Where clause:** *P* can be replaced with an expression of the form:

            B (subquery)

            B is an attribute and to be defined later.

            常使用in, not in,exists,not exists等句式
        - **Select clause:**

            Ai can be replaced be a subquery that generates a single value.
    - **in、not in**
        - 是set membership，集合关系
        - 常用嵌套子查询
    - **exists** clause:
        - The **exists** construct returns the value **true** if the argument subquery is nonempty.
        - 开始用**关联子查询**(correlated subquery)
            - in 和 not in常用嵌套子查询
            - exists 和 not exists常用关联子查询

            【嵌套子查询和关联子查询区别：https://www.cnblogs.com/DavidYan/articles/2044743.html】！！！

            嵌套子查询: 1. 内部查询只处理一次; 2. 与null比较,总得到null 3.先进行内部查询,然后再进行外部查询

            关联子查询: 1. 外部查询得到的每行记录传入到内部查询; 2.内部查询基于外部查询传入的值; 3.内部查询从其结果中把值传回到外部查询，外部查询使用这些值来完成其处理。【在关联子查询中是信息流是双向的。】

            （对于外部查询返回的每一行数据，内部查询都要执行一次。外部查询的每行数据传递一个值给子查询，然后子查询为每一行数据执行一次并返回它的记录。然后，外部查询根据返回的记录做出决策。）
    - **unique** clause：
        - The **unique** construct evaluates to “true” if a given subquery contains no duplicates .
14. set comparison 集合的比较
    - **some** clause：
        - 案例:

            Find names of instructors with salary greater than that of some (at least one) instructor in the Biology department.

            ```
            select distinct T.name
            from instructor as T, instructor as S
            where T.salary > S.salary and S.dept name = 'Biology';
            /*等价于*/
            select name
            from instructor
            where salary > some (select salary from instructor where dept name =
            'Biology');
            ```

        - (= **some**) 等价于**in**，(<>**some**) 不等价于**not in** 【此处<>为不等于符号】
            - 5 = some(0,5) returns true
            - 5 <> some(0,5) returns true 【since 0 <> 5】
    - **all** clause：
        - 即满足任意条件
        - (<> all) 等价于 not in, (=all) 不等于 in
            - 5 = all(0, 5) returns false，但5 in (0,5)
            - 5 <> all(4, 6) returns true 【since 5<>4 and 5<>6】
15. **with** clause：
    - The **with** clause provides a way of defining a temporary relation whose definition is available only to the query in which the **with** clause occurs. 作用域是暂时的，只对with语句下生效
    - 因为实质上是先计算并保存结果在一个表里，所以可使语句非常简洁好看！
16. The **select from where** statement is evaluated fully before any of its results are inserted into the relation. 即我们是查完表进行充分的计算后再插入，而非插完一个就更新再新插。否则我们下面的例子就会报错：【实际上没有错误，原因就是先充分计算完再插入】

    ```
    insert into table1 select * from table1
    ```

17. 重点：注意把create table的地方，以及对表的更改强化记忆。

ch4

1. 对join语句的理解：
    - 将两个relation合成为一个relation，其实是一个子查询表达式，常用于from语句中
2. natural join：
    - Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column.
    - 危险性：没有关联的属性可能因为名字相同而错误的join在一起。
        - 例如List the names of students instructors along with the titles of courses that they have taken。其中student、takes、course三个直接natural join在一起就会出错，因为student和course有相同的属性——department。

          所以一般连着使用多个natural join需要谨慎一点。
        - 解决方案：为了正确解决可能歧义的问题，
            - 一是使用where语句来进行明确的限制；
            - 二是可以***using***来指明是哪个进行join。

              ```
              select name, title
              from  (student natural join takes)  join course using
              (course_id)
              ```

            - 三是使用**on**语句替代**where**，区别不大，只是使用了on的关键字

              ```
              select *
              from  student join takes on student_ID  = takes_ID
              ```

    - 本质上属于inner join内连接。

3. outer join
    - 与内连接的区别在于，外连接不仅返回匹配的行，也会返回不匹配的行，所以**外连接可以防止信息的丢失。**
        - 而内连接（ [inner] join）是从查询结果表中删除与其他被连接表中没有匹配行的所有行，所以**内连接可能会丢失信息。**注明，[inner] join表示inner可以省略。
    - 三种outer join：
        - **course natural left outer join prereq**：course在left，则匹配时保留course的course_id这列全部信息，而prereq的将可匹配信息写上，其他地方若无则写null
        - **course natural right outer join prereq**：prereq在right，则匹配时保留prereq的course_id这列的全部信息，而course的将可匹配信息写上，其他地方若无则写null
        - **course natural full outer join prereq**：匹配时保留prereq、course两边的course_id这列的全部信息，其他地方若对方没有则都添加null。
4. **join condition**：natural、 on 、using <A1,A2,...,An>
5. **join types**: [inner] join、left outer join、right outer join、full outer join
6. **view**的理解
    - A **view** provides a mechanism to hide certain data from the view of certain users.
    - Any relation that is not of the conceptual model but is made visible to a user as a “virtual relation” is called a **view**.
    - View definition is not the same as creating a new relation by evaluating the query expression。Rather, a view definition causes **the saving of an expression**; the expression is substituted into queries using the view.因为它不是物理存储的，使用时会层层展开到relation的层级。
7. A view is defined using the **create view** statement which has the form：

   create view *v* **as** < query expression >
8. view的创建可以依赖于另一个view。
9. why we do not allow view definition is "recursive"?
    - because when view expansion is executed, as long as the view definitions are not recursive, this loop will terminate.
10. materialized views是实化的视图，进行了物理存储，但这就需要定期更新了。
11. view update时必须带动relation update，因为有的view不包含的属性不知道插入或修改成什么。【未解决，待看书】
12. 大部分SQL只允许更新非常简单的views
13. candidate keys are permitted to be null(in contrast to primary keys)
14. 参照完整性（Referential integrity）存在级联行动（cascading actions）
15. If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation.
16. All privileges that depend on the privilege being revoked are also revoked.
17. **grant on to**
    - privilege list：select、insert、update、delete、all privileges
    - 例：**grant select on** *instructor* **to** *U*1, *U*2, *U*3

ch5

1. SQL注入：Never create a query by concatenating strings，否则可能会遭遇恶意拼接查询； 可以利用参数化查询来应对。
2. SQL function

- 这种参数化使用在形式上非常像关联子查询，从外部调入数据来进行查询。以下面的dept_name的传参来思考，这里若换成关联子查询也是可以的。
- ```
  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

  select dept_name, budget
  from department
  where dept_count (dept_name ) > 12
  ```

3. table function：返回值为table的function
   - ```
     /*returns all instructors in a given department*/
     create function instructor_of(dept_name char(20))
         return table(
             ID varchar(5),
             name varchar(20),
             dept_name varchar(20),
             salary numeric(8,2))
         return table
             (select ID,name,dept_name,salary
             from instructor
             where instructor.dept_name = instructor_of.dept_name)

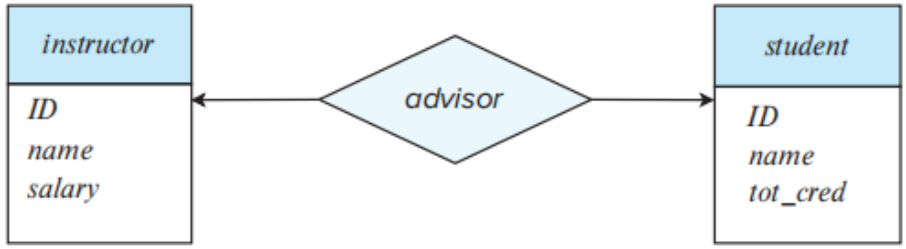
     select *
     from table(instructor_of('Music'))
     ```

4. SQL procedure:
   - 对比：function是直接使用表达式（expression），procedure是显式调用（call）
   - ```
     create procedure dept_count_proc(in dept_name varchar(20),out d_count
     integer)
         begin
             select count(*) into d_count
             from instructor
             where instructor.dept_name = dept_count_proc.dept_name
         end

     declare d_count integer;
     call dept_count_proc('Physics', d_count)
     ```

ch6

1. derived value means that it can be deferred.
2. 如果你被E-R描述方向弄晕了，可以看看这儿。一句话总结E-R图的画法：无论是partial-total、one-many、m..n等描述，E-R图中E与R的连线上的表示总是在描述**一个Entity**和**一个Relationship**之间的关系，谨防掉进一个E-R图中有另一个Entity就影响思考了。
   - 该线为双线，说明该Entity参与该Relationship的方式为total participate。
     ```

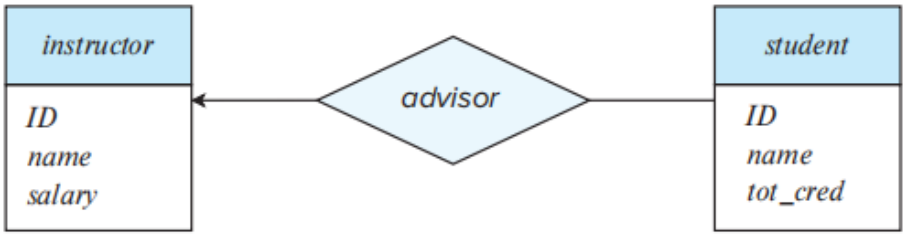
- 该线为有箭头，说明该Entity A为one，在该Relationship关系中另一个Entity B最多只能获取到一个A。
- 该线为m..n，说明该Entity参与的数量为[m,n],某些特殊值可以替代前两种画法。如1..*就代表total participate，0..1代表最多参与1个，等同于有箭头指向。

3. 关于one与many的问题

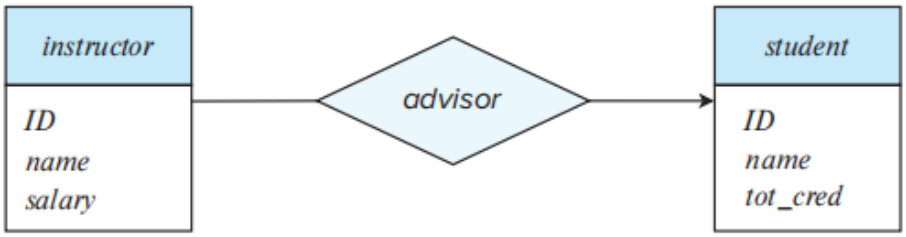
- 有指向箭头的代表one；无指向箭头代表many
- **One-to-one**. We draw a directed line from the relationship set to both entity sets. For example, in Figure 6.11a, the directed lines to *instructor* and *student* indicate that an instructor may advise at most one student, and a student may have at most one advisor.
- **One-to-many**. We draw a directed line from the relationship set to the "one" side of the relationship. Thus, in Figure 6.11b, there is a directed line from relationship set *advisor* to the entity set *instructor*, and an undirected line to the entity set *student*. This indicates that an instructor may advise many students, but a student may have at most one advisor.
- **Many-to-one**. We draw a directed line from the relationship set to the "one" side of the relationship. Thus, in Figure 6.11c, there is an undirected line from the relationship set *advisor* to the entity set *instructor* and a directed line to the entity set *student*. This indicates that an instructor may advise at most one student, but a student may have many advisors.
- **Many-to-many**. We draw an undirected line from the relationship set to both entity sets. Thus, in Figure 6.11d, there are undirected lines from the relationship set *advisor* to both entity sets *instructor* and *student*. This indicates that an instructor may advise many students, and a student may have many advisors.



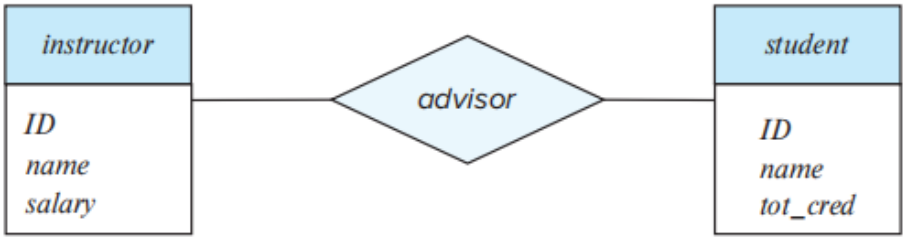
(a) One-to-one



(b) One-to-many



(c) Many-to-one



(d) Many-to-many

3. 关于partial和total的问题:

- The participation of an entity set *E* in a relationship set *R* is said to be **total** if every entity in *E* must participate in at least one relationship in *R*.
- If it is possible that some entities in *E* do not participate in relationships in *R*, the participation of entity set *E* in relationship *R* is said to be **partial**.
- **双线为total，单线为partial。**We indicate total participation of an entity in a relationship set using double lines. Figure 6.12 shows an example of the *advisor* relationship set where the double line indicates that a student must have an advisor.

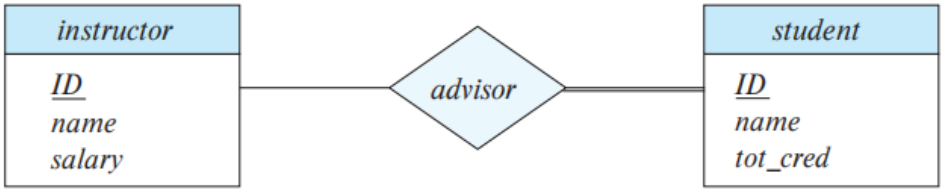
- 

Figure 6.12 E-R diagram showing total participation.

    - 替代方式：也可在直线上写最小值与最大值, m..n.
4. 三元关系注意再看下画法
    - An instance of *proj guide* indicates that a particular student is guided by a particular instructor on a particular project. Note that a student could have different instructors as guides for different projects, which cannot be captured by a binary relationship between students and instructors.

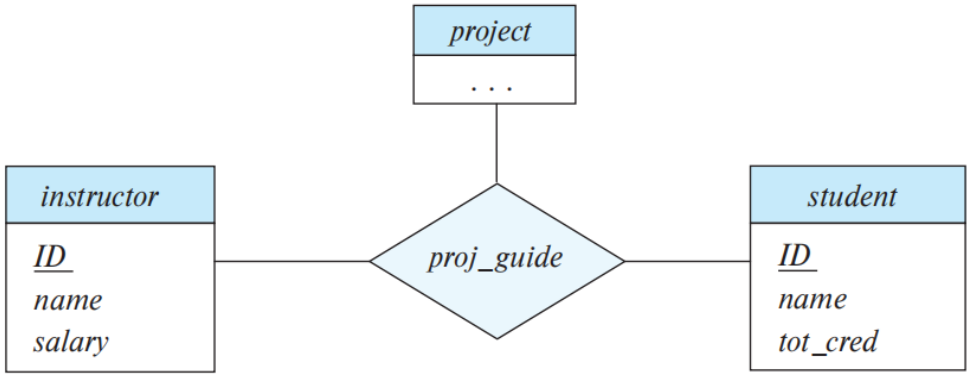
- 

Figure 6.6 E-R diagram with a ternary relationship *proj_guide*.

5. **weak entity set** 弱实体集
    - 定义：A **weak entity set** is one whose existence is dependent on another entity set, called its **identifying entity set**; instead of associating a primary key with a weak entity, we use the primary key of the identifying entity, along with extra attributes, called **discriminator attributes** to uniquely identify a weak entity. An entity set that is not a weak entity set is termed a **strong entity set**.
    - 表现：一个实体的主键依赖于另一个实体的主键
    - 案例：course and section——section一定是某门课的section、问题与回答——回答一定是针对某个问题的回答
    - 画法：

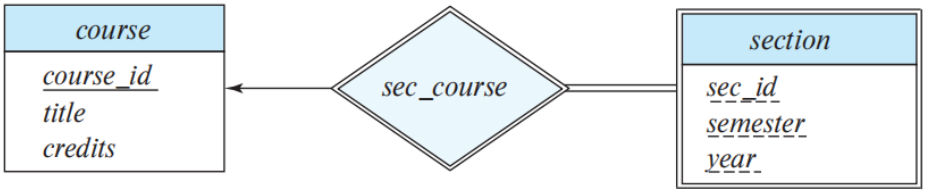


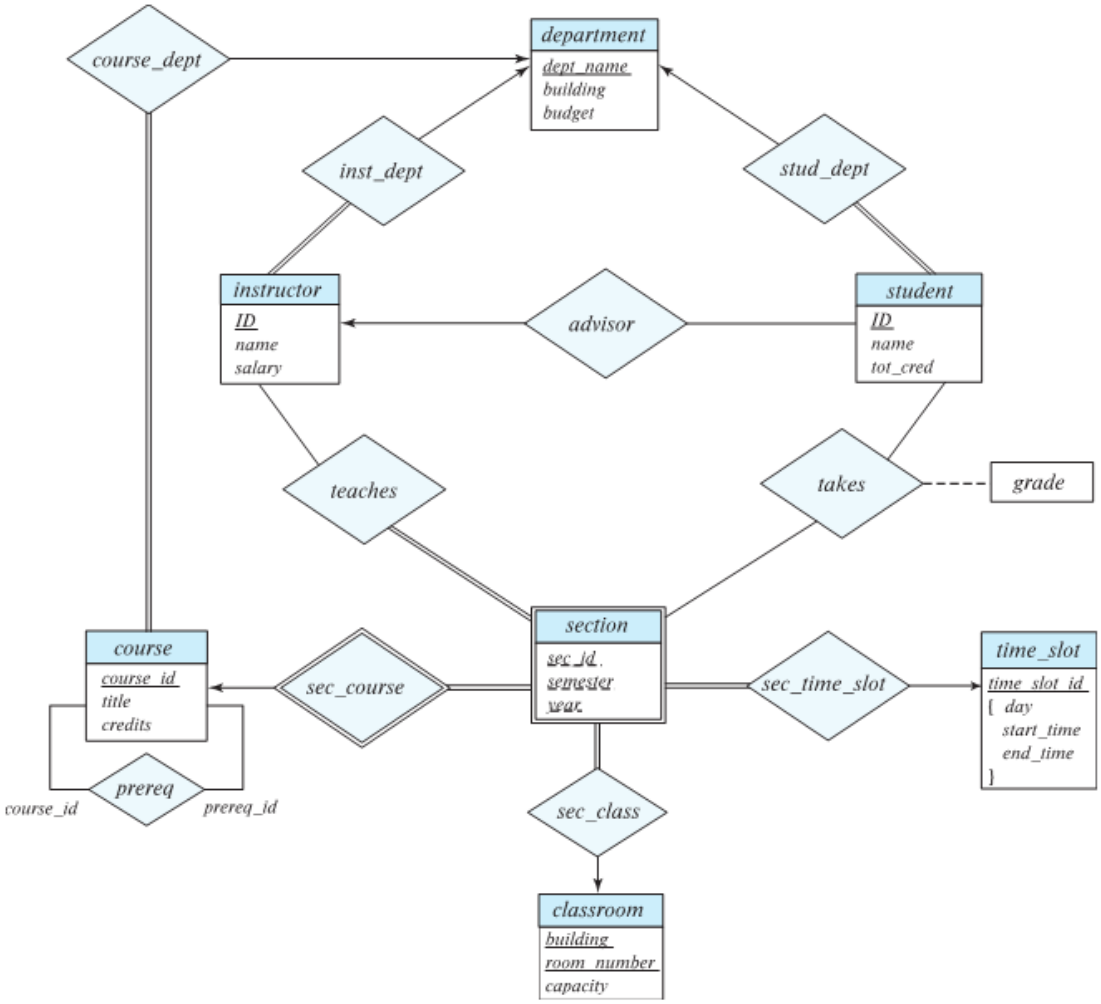
Figure 6.14 E-R diagram with a weak entity set.

6. **Removing Redundant Attributes in Entity Sets**【注意，现在规定的是实体集之间的冗余，这个判断不是基于某一实体集和关系集的；进一步思考，关系集的主键定义本来就基于实体集，先入手分析关系集用处不大，所以在分析的时候还得先思考实体集之间本来的关系：如分析出两实体集之间有依赖关系则为强弱实体集，这才定义了**标识性联系identifying relationship。**所以此处同

理，对比分析千万不要弄成与实体集—联系集进行比较了】

- 建立好的每个entity之间可能存在冗余
- 比如department和instructor两个实体集之间，都有dept_name属性，但dept_name是department的主键，就重复了，那么画E-R图的时候就要省略instructor的dept_name属性。【这不是某老师教书那种:instructor 和teach 中属性重复，这是实体集和关系集】

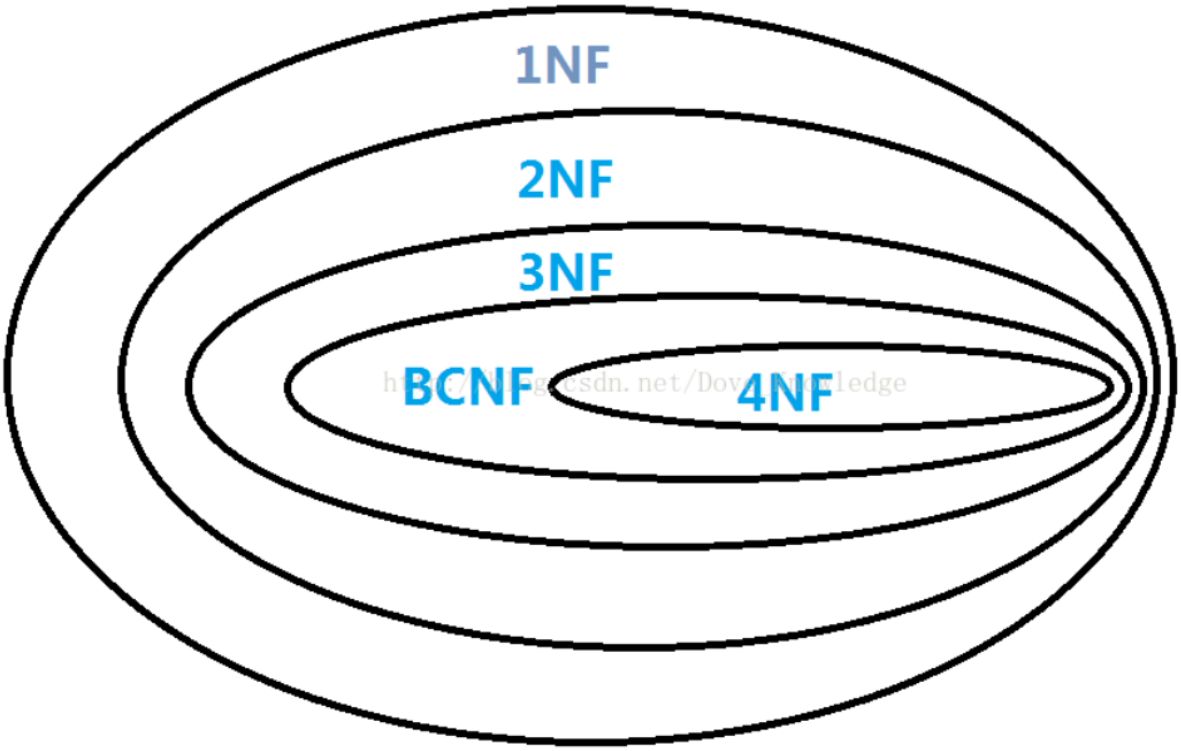
7. **Reducing E-R Diagrams to Relational Schemas**【从E-R图到关系模式】
   - 多值属性单独与主键写
   - 弱实体集获得依赖的强实体集的主键
8. **对于这一章告诉我们的关系型数据库整个的建模过程的思考【可结合下面大学数据库图思考】**
   - 全流程为【自己根据看到的实体先手写一堆带属性的Entity——>画E-R图——>relation schema】
   - 从手写一堆Entity到画E-R图
     - 首先要分析实体集之间的冗余关系，画图时去掉某些属性，比如同样重复的属性有一个是人家的主键；甚至此时可能为弱实体集，需要我们对图进行进一步处理。
     - 然后直接补充一个relationship联系集【一定存在，所以可以直接放】
     - 然后开始连E-R之间（即实体集与联系集之间）的线，标清这个实体集对于这个关系的参与程度,如双线、箭头、m..n等
     - 再整理一下，如果没问题的话，我们的E-R图其实基本上就画好了
   - 从E-R图再到relational schemas
     - 实体集的画法都比较自然易懂，暂不赘述；
     - 主要是对relationship的分析，是否保留建表和怎么做也会影响实体集的进一步改变。
       1. 特别是要分清one-many且total participate这种类型，这种relationship是不需要建的，且这时候many方的实体集属性，再拉入one side的主键作为自己的属性即可；其实弱实体集与之有点类似。
       2. 而对于one-many且partial participate的这种，需要建立relationship的表。
       3. 再者，是one-one的类型的理解：书上表述是把任意一边作为many side，拉入另一个one side的主键作为自己的属性即可。【思考：这其实是因为**“一定有”**，假设一个丈夫有且只有一个妻子的话，那么对于丈夫而言，**一定有**妻子可以作为自己的属性，对于妻子而言，**一定有**丈夫可以作为自己的属性，那么我们就不需要再为之建立relationship的表了（虽然我们自己罗列属性时不会专门指出这一点）。再类比上面第1点所言，其实也是many方**一定有**one方的主键作为自己的属性，就像student**一定有**department的dept_name这个属性一样，此时就不需要再建立relationship的表了】



ch7

1. 解决信息冗余问题——>分解
2. 乱分解可能造成分解丢失信息，即lossy decomposition。
3. 函数依赖是我们进行范式分析的手段
4. Armstrong's Axioms包括Reflexive rule,Augmentation rule,transitivity rule,union rule, decomposition rule, pseudotransitivity rule，其中Decomposition Rule最常用到。
5. 无关属性：An attribute of a functional dependency in F is extraneous if we can remove it without changing F+.
6. 正则覆盖Fc的任何一个函数依赖的左值都是唯一的，即 Fc不存在a1 -> b1 and a2 -> b2 such that a1 = a2。
7. 高效检查依赖保持，使用的是F而非F+【问：1. 此处是专查的BCNF的方法吗？既然3NF本来就是dependency preserving的，能在耽搁的relation上被check，而无需computing a join 2. 如果是针对BCNF做的，那本方法有join的存在吗？以前开始对比BCNF和3NF
时就特意强调了BCNF检查需要join，而这儿似乎不明显了，问问？】
8. 使用函数依赖进行范式分解
    - BCNF的test不能只使用F，因为分解后F中的函数依赖无法（像3NF那样）有效对应每个Ri的信息，所以我们可选择使用F中落在每个Ri上的左值，求他们的属性闭包的方法（另一种方法是test Ri for BCNF with respect to the **restriction** of F+ to Ri）。若该属性闭包包含Ri或只包含自身，都满足BCNF，没问题；但如果包含了超出自身一点点，却又每包含完Ri，就不符合BCNF，必须得分解了。

    - 3NF的分解过程中就直接用的Fc，分解结果中每个Ri又一定包含所有的函数依赖，所以查验确实不需要再join各个Ri
    - 是多项式时间，polynomial time
9. 多值依赖的定义：设R(U)是一个属性集合U上的一个关系模式，X, Y, 和Z是U的子集，并且Z=U-X-Y，多值依赖X->->Y成立当且仅当对R的任一个关系r，r在(X,Z)上的每个值对应一组Y的值，这组值仅仅决定于X值而与Z值无关。若X->->Y，而Z=空集，则称X->->Y为平凡的多值依赖。否则，称X->->Y为非平凡的多值依赖。 【多值依赖强调了独立性】
10. 第四范式：
    - X->->Y中要么它是平凡的多值依赖【Y是X的子集或X∪Y=R时，X->->Y是平凡的】；要么X是一个超码。
    - 对于每一个非平凡多值依赖X->->Y，X若含有候选码，也就是X->Y，所以**4NF所允许的非平凡多值依赖是函数依赖**
    - 第四范式，对于候选键只能存在不超过1个多值属性。要求把同一表内的多对多关系删除。
11. 关系图



ch14

1. **primary index**（clustering index）：in a sequentially ordered file，the index whose search key specifies the sequential order of the file.
    - The search key of a primary index is usually but not necessarily the primary key.
2. **secondary index**（nonclustering index）：an index whose search key specifies an order different from the sequential order of the file.
3. **dense index**: index record appears for **every search-key value** in the index file.
4. **sparse index**:contains index records for **only some** search-key values.
5. 注意:the reason why secondary indices have to be dense:
    - If we use sparse index, we can't find all the data by the order of sparse index because the orders in two files are different.

6. index的缺点：
    - They impose serious overhead on database modification: whenever a file is updated, every index must be updated.【它们对数据库修改造成了严重的开销：每当更新文件时，都必须更新每个索引。】
7. sequential scan using clustering index is efficient, but a sequential scan using a secondary index is expensive on magnetic disk.
    - every record access may fetch a new block from disk.
8. why we use multilevel index?
    - If index does not fit in memory, access becomes expensive.
    - **Solution**:treat index kept on disk as a sequential file and construct a sparse index on it.
9.

注意区

1. 在ch3，注意把create table的地方，以及对表的更改(insert、delete等等)强化记忆。【可以手打一遍】
2. 注意大学数据库模式可以再看一遍。

   classroom(*building*, *room number*, *capacity*)

   department(*dept_name*, *building*, *budget*)

   course(*course_id*, *title*, *dept_name*, *credits*)

   instructor(*ID*, *name*, *dept_name*, *salary*)

   section(*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)

   teaches(*ID*, *course_id*, *sec_id*, *semester*, *year*)

   student(*ID*, *name*, *dept_name*, *tot_cred*)

   takes(*ID*, *course_id*, *sec_id*, *semester*, *year*, *grade*)

   advisor(*s_ID*, *i_ID*)

   time slot(*time_slot_id*, *day*, *start_time*, *end_time*)

   prereq(*course_id*, *prereq_id*)

   ---

   classroom(*building*, *room_number*, *capacity*)
   department(*dept_name*, *building*, *budget*)
   course(*course_id*, *title*, *dept_name*, *credits*)
   instructor(*ID*, *name*, *dept_name*, *salary*)
   section(*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)
   teaches(*ID*, *course_id*, *sec_id*, *semester*, *year*)
   student(*ID*, *name*, *dept_name*, *tot_cred*)
   takes(*ID*, *course_id*, *sec_id*, *semester*, *year*, *grade*)
   advisor(*s_ID*, *i_ID*)
   time_slot(*time_slot_id*, *day*, *start_time*, *end_time*)
   prereq(*course_id*, *prereq_id*)

   ---

3. 对于ch6的PPT的P57-80还没看

4. 正则覆盖Fc的任何一个函数依赖的左值都是唯一的，即 Fc不存在a1 -> b1 and a2 -> b2 such that a1 = a2。
5. 降序排列后面加关键字 desc
6. data dictionary 是system catalog，存放metadata的，所以总是放在主存中。
7. B+树的几大限制：根节点、叶节点的value值个数、非叶节点的children个数、树的高度范围
8. A **weak entity set** is one whose existence is dependent on another entity, called its **identifying entity**。An entity set that is not a weak entity set is termed a **strong entity set**.
9. 正则覆盖中左值唯一

疑问区

1. ch4的P29页的view和relation的更新问题
2. ~~为什么X∪Y=R时，X->->Y是平凡的?~~
3. ~~8.10那道题真的按照第四范式算法依次分解的话，不会漏掉ename吗？为什么能够想到留下这一个属性创建一个表呢?~~

SQL好题

1. 【self join类型】

   Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.

```
select distinct T.name
from instructor as T,instructor as S
where T.salart > S.salary and S.dept_name ="Comp.Sci"
```

2. 【self join类型】

   Find the supervisor of “Bob” in Relation *emp-super*

```
select b.supervisor
from emp-super a, emp-super b
where a.person='Bob' and a.supervisor = b.person
```

引申思考：Can you find ALL the supervisors (direct and indirect) of “Bob”?【多次self join】【忽然想到某项目对评论的评论】

3. 【小心distinct】

   Find the total number of instructors who teach a course in the Spring 2018 semester

```
select count (distinct ID)
from teaches
where semester = 'Spring' and year = 2018;
```

(why we use distinct? because a teacher may teach several courses)

4. 【关联子查询、对全称命题的逻辑性翻译】

   Find all students who have taken all courses offered in the Biology department.

```
select distinct S.ID, S.name
from student as S
where not exists ( (select course_id
                    from course
                    where dept_name = 'Biology')
                    except
                    (select T.course_id
                    from takes as T
                    where S.ID = T.ID));
```

   注：Cannot write this query using = all and its variants

   分析：这里是包含关系，学生选的课包含了生物学院开的所有课程。

5. 【关联子查询、unique】

   Find all courses that were offered **at most once** in 2017

```
select T.course_id
from course as T
where unique ( select R.course_id
               from section as R
               where T.course_id= R.course_id and R.year = 2017);
```

6. 【在from处的子查询】

Find the average instructors' salaries of those departments where the average salary is greater than $42,000."

```
select dept_name, avg_salary
from ( select dept_name, avg (salary) as avg_salary
      from instructor
      group by dept_name)
where avg_salary > 42000;
/*等同于如下写法*/
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;
```

说明：Note that we do not need to use the **having** clause，因为这里我们子查询返回的是一个新表!

7. 【with的灵活使用，连用版】

Find all departments where the total salary is greater than the average of the total salary at all departments

```
with dept _total (dept_name, value) as
        (select dept_name, sum(salary)
         from instructor
         group by dept_name),
dept_total_avg(value) as
       (select avg(value)
       from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value > dept_total_avg.value;
```

8. Recompute and update tot_creds value for all students

```
update student S
set tot_cred = (select sum(credits)
                from takes natural join course
                where S.ID = takes.ID and takes.grade<>'F' and takes.grade is not
null
                )
```

【析：成绩非空，且没挂科】

9. 【some的写法】

```
select name
from instructor
where salary > some (select salary
                     from instructor
                     where dept name = 'Biology');
```

10. 【view的创建】

```
 create view faculty as
    select ID, name, dept_name
    from instructor
```

复习剩余区：

1. ~~第10章、第11章~~
2. ~~第17章~~
3. ~~对于ch6的PPT的P57-80还没看~~
4. 06年A卷有个概念题还没看
5. ~~关系代数必须做点题~~

考前晚上必温故

1. SQL的创建 create table、更新（update）、插入、删除等
2. 大学数据库模式

```
3.11
/*评：其实可以直接takes与course natural join而不是与section，毕竟都有course_id*/
a.
select distinct student.name
from student natural join takes natural join section,course
where section.course_id = course.course_id and course.dept_name='Comp. Sci.'

b.
select ID,name
from student natural join takes  /*评：此处不需要natural join takes，只需要选所有学
生，毕竟后面才是限制*/
except
select ID,name
from student natural join takes
where year<2009
c.
select dept_name,max(salary)
from instructor
group by department
d.
with max_salary_dept as
    (select dept_name,max(salary) as max_sal
    from instructor
    group by department)
select  min(max_sal)
from max_salary_dept natural join department
```

a.

name
Zhang
Shankar
Levy
Williams
Brown
Bourikas

b.