0. 目录

1. Chapter 1

1.1 概览

- DBS 概念和组成
- 数据抽象与数据独立性
- DBS 设计阶段与数据模型
- Database Languages（DDL, DML）
- 数据库背景知识

1.2 DBS 概念和组成

1.2.1 数据库系统

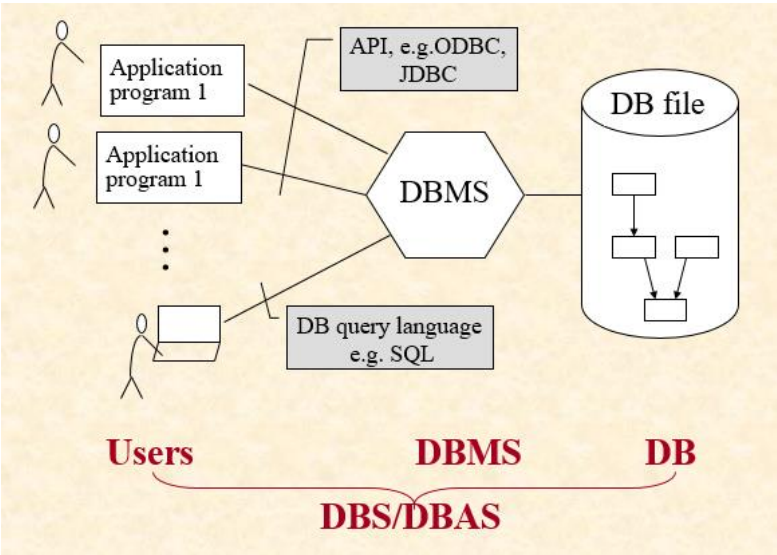
数据库系统（DBS）是由一个相互关联的数据的集合和一组用以访问这些数据的程序组成

这个数据集合成为数据库（database）

- DBMS provides an environment that is both *convenient* and *efficient* for store and retrieve information
  - definition of structures for storage of information
  - data manipulation mechanisms
  - data safety mechanisms

1.2.2 目标

提供一种可以**方便、高效**地存取数据库信息的途径



1.3 数据视图

- 数据库系统是一些互相关联的数据以及一组使得用户可以访问和修改这些数据的程序的集合。
- 数据库系统的一个主要目标是给用户提供数据的**抽象视图**，也就是说，系统隐藏关于数据存储和欸和的某些细节。

1.3.1 数据抽象

1.3.1.1 物理层（physical level）：描述数据实际上**怎样存储**的

1.3.1.2 逻辑层（logical level）：描述数据库中存储什么数据以及这些数据间存在什么关系。虽然逻辑层的实现可能涉及复杂的物理层结构，但逻辑层用户不必知道这样的复杂性，即**数据独立性**

1.3.1.3 视图层（view level）：只描述整个数据库的某个部分

1.3.2 实例和模式

- 特定时刻存储在数据库中的信息的集合称作数据库的一个**实例（instance）**
- 数据库的总体设计称作**数据库模式（schema）**;数据库模式即使发生变化，也不频繁
- 应用程序若不依赖于物理模式，即具有**物理数据独立性（Physical data independence）**，即使物理模式改变了它们也无需重写

1.3.2.1 几种模式

- 物理模式（Physical schema）:在物理层描述数据库的设计
- 逻辑模式（logical schema）：在逻辑层描述数据库设计
- 子模式（subschema）:在视图层，描述了不同的视图

1.3.3 数据模型

- 数据库结构的基础是数据模型
- 数据模型是一个描述数据、数据联系、数据语义、以及一致性约束的概念工具的集合。**a collection of conceptual tools for describing data, data relationship, data semantics, consistency**
- Definition of data model (P8) ： a collection of *conceptual tools* for describing
  - data
  - data relationships
  - data semantics
  - consistency constraints

1.3.3.1 分类

- 关系模型（relational model）：用表的集合来表示数据和数据间的关系。每个表有多个列，每列有唯一的列名。
- 实体-联系模型（entity-relationship model）:现实世界由一组称为实体的基本对象及这些对象件的联系构成
- 基于对象的数据模型（object-based data model）

- 半结构化数据模型

1.4 数据库语言

1.4.1 数据操纵语言

1.4.1.1 Data-Manipulation Language,DML 是这样一种语言：它使得用户可以访问或操纵那些按照某种适当的数据模型组织起来的数据

- 信息检索
- 向数据库插入新的信息
- 删除信息
- 修改信息

1.4.1.2 过程化 DML(procedural DML)：要求用户指定需要什么数据以及如何获得这些数据

1.4.1.3 声明化 DML(declarative DML)：只要求用户指定需要什么数据，而不声明如何获得这些数据

1.4.2 数据定义语言

1.4.2.1

- 数据库模式是通过一系列定义来说明的，这些定义由**数据定义语言（Data Defenition Language，DDL）**来表达
- 存储在数据库中的数据必须满足某些**一致性约束（consistency constraint）**
- DDL 以一些指令（语句）作为输入，生成一些输出。DDL 的输出放在**数据字典**中，数据字典包含了**元数据**

1.4.2.2

- 域约束（domin constraint）:每个属性都必须对应于一个所有可能的取值构成的域
- 参照完整性（referential integrity）:一个关系中给定属性集上的取值也在另一关系的某一属性集中的取值中出现；数据库的修改会导致参照完整性的破坏
- 断言（assertion）：一个断言就是数据库需要时刻满足的某一条件。域约束和参照完整性约束是断言的特殊形式。

1.4.2.3 授权（authorization）：对不同用户在数据库中的而不同数据值上允许不同的访问类型

1.5 数据库设计

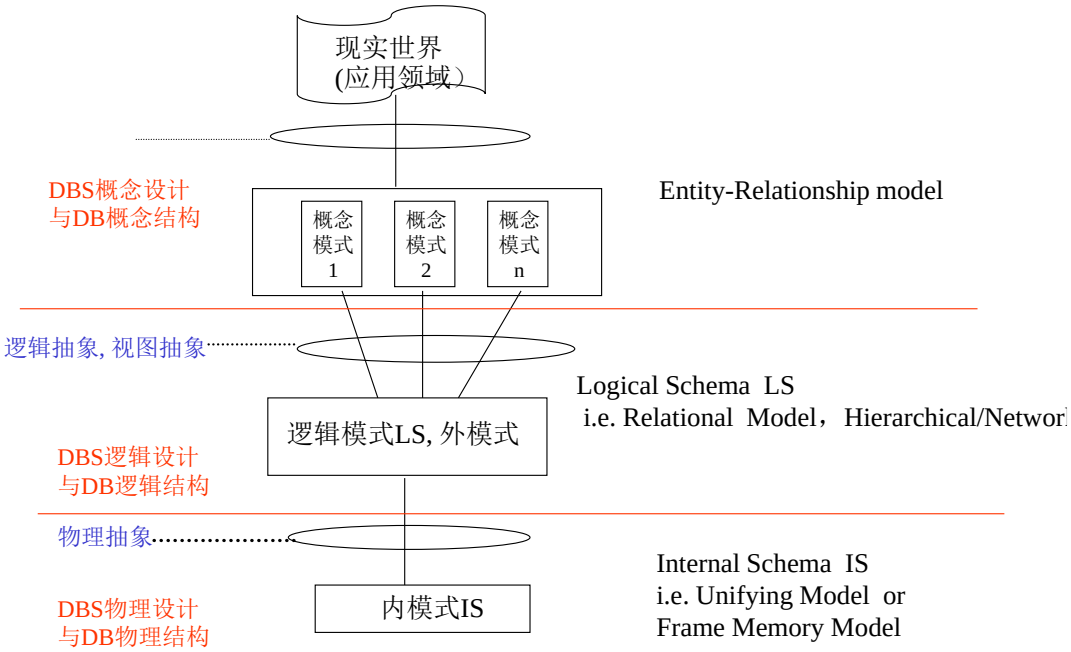
1.5.1 设计过程

从保持 data independence（数据无关性/独立性）角度出发，根据 data models 所定义的数据规范形式，在 view、logical 、 physical 三个层次，采用三种 data abstraction 方法，通过 DB 概念设计、DBS 逻辑设计、DBS 物理设计三个阶段，构造 面向具体应用领域的 DBS 的 external schema、logical schema 、internal schema

的集合，从而得到 conceptual DBS 、logical DBS 、physical DBS 的设计结果

1.5.1.1

- 初始阶段：全面刻画预期的数据库用户的数据需求
- 概念设计阶段：选取一个数据模型，运用该模型的概念将需求转化为数据库的概念模式。涉及到决定数据库中该包含哪些属性，以及如何将这些属性组织到多个表中
- 后者主要有两种方法：使用实体-联系模型；引入一套算法（规范化），这套算法将所有属性集作为输入，生成一组关系表
- 逻辑设计阶段：设计者将高层的概念模式映射到将要使用的数据库的实现数据模型上
- 物理设计阶段



DBS设计与设计阶段： 数据模型—数据抽象—数据模式

1.6 数据存储和查询

1.6.1 存储管理器

- 存储管理器是数据库系统中负责在数据库中存储的低层数据与应用程序以及向系统提交的查询间提供接口的部件。
- 负责数据库中的存储、检索和更新

1.6.2 查询处理器

子系统编译执行 DDL 和 DML 语句

1.7 数据库体系结构

- DBS = User/DBA + DBMS + DB
- DBMS:

query processing

storage manager

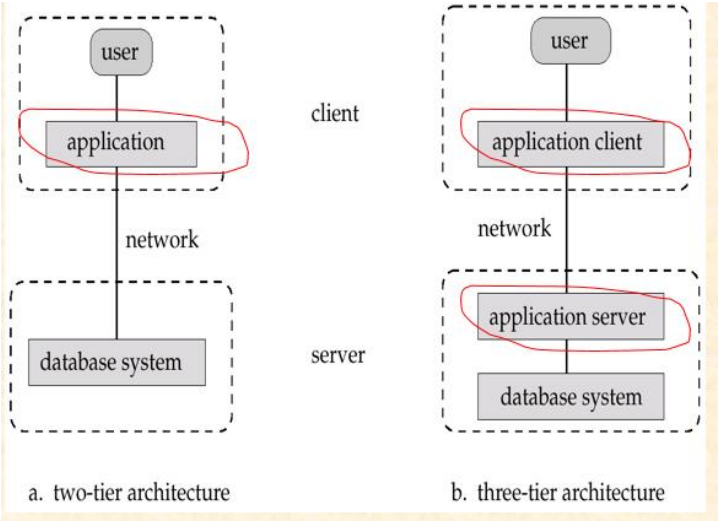
- DB files:

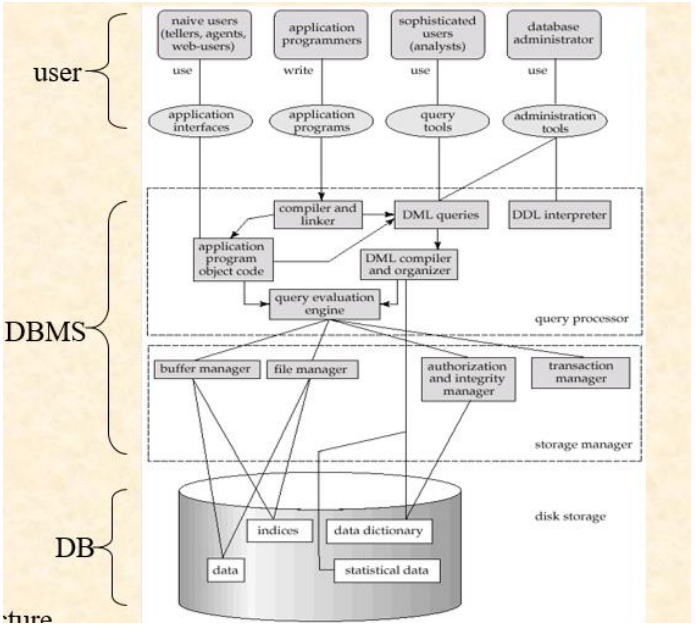
application data

data dictionary/directory

indices

statistic data





2. Chapter 2

2.1 关系数据库的结构

- 关系数据库由表的集合构成，每张表有唯一的名字
- 关系 relation 用来指代表，元组 tuple 用来指代行，属性 attribute 指代表中的列
- 关系实例 relation instance 表示一个关系的特定实例
- 元组在关系中出现的顺序是无关紧要的
- 关系中属性的允许取值的集合：域 domain
- Null 表示未知或不存在

2.2 数据库模式

- 数据库模式 （database schema）：数据库的逻辑设计
- 数据库实例（database instance）：给定时刻数据库的快照
- 关系模式 （relation schema ）：由属性序列及各属性对应域组成

2.3 键（码）

- 码是整个关系的一种性质，而不是单个元组的性质

2.3.1.1 超键

- A superkey is a set of one or more attributes that, taken collectively, can be used to identify uniquely a tuple in the relation
- 可能包含无关紧要的属性

2.3.1.2 候选键

- K is a candidate key if K is minimal super key
- A relation may have several candidate keys

2.3.1.3 主键

- 被设计者选中的、主要用来在一个关系中区别不同元组的候选码
- 不能为空

2.3.1.4 外键

- 一个关系模式 r1 可能在它的属性中包括另一个关系模式 r2 的主码，这个属性在 r1 上称作参照 r2 的外码
- 关系 r1 称作外码依赖的参照/关联关系
- 关系 r2 称作外码的被参照/关联关系

2.3.1.5 参照完整性约束（Foreign key/Referential integrity constraint）

- 要求在参照关系中任意元组在特定属性上的取值必然等于被参照关系中某个元组在特定属性上的取值

2.4 模式图（schema diagram）

2.5 关系查询语言

2.5.1 查询语言

2.5.1.1 过程化语言

2.5.1.2 非过程化语言

2.6 关系运算

2.6.1 relational algebra

2.6.1.1 a set of operations, take one or two limited relations as input and produce a new limited relation as output

2.6.1.2 Three types of operations/operators on relations

- fundamental operations,
- additive operations
- extended operations

Symbol (Name)	Example of Use
σ (Selection) 选择	$\sigma_{salary >= 85000}(instructor)$
	Return rows of the input relation that satisfy the predicate.
Π (Projection) 投影	$\Pi_{ID,\ salary}(instructor)$
	Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
× (Cartesian Product)笛卡尔积	*instructor* × *department*
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
∪ (Union) 并	$\Pi_{name}(instructor) \cup \Pi_{name}(student)$
	Output the union of tuples from the *two* input relations.
– (Set Difference)集合差	$\Pi_{name}(instructor) - \Pi_{name}(student)$
	Output the set difference of tuples from the two input relations.
⋈ (Natural Join)自然连接	*instructor* ⋈ *department*
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.

2.6.2 tuple relational calculus（演算）

2.6.3 domain relational calculus

3. SQL

3.1 SQL 数据定义

3.1.1 基本类型

3.1.1.1 数据类型

- char(n). Fixed length（固定长度） character string, with user-specified length n.
- varchar(n). Variable（可变） length character strings, with user-specified maximum length n.
- int. Integer (a finite subset of the integers that is machine-dependent).
- smallint. Small integer (a machine-dependent subset of the integer domain type).
- numeric(p,d). Fixed point number（定点数）, with user-specified precision of p digits（总长度 p 位）, with d digits to the right of decimal point（小数点）

- real, double precision. Floating point and double-precision floating point numbers, with machine-dependent precision.（精度与机器有关）
- float(n). Floating point number, with user-specified precision of at least n digits.

3.1.1.2 其他

- date. Dates, containing a (4 digit) year, month and date

  E.g. date '2001-7-27'
- time. Time of day, in hours, minutes and seconds.

  E.g. time '09:00:30' time '09:00:30.75'
- timestamp: date plus time of day

  E.g. timestamp '2001-7-27 09:00:30.75'
- Interval: period of time

  E.g. Interval(区间) '1' day

  Subtracting a date/time/timestamp value from another gives an interval value

  Interval values can be added to date/time/timestamp values

3.1.2 基本模式定义

3.1.2.1 Create

- An SQL relation is defined using the **create table** command
- **create table** *r* (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integrity-constraint$_1$),完整性约束
  ...,
  (integrity-constraint$_k$)) ； 分号结束
- *r* is the name of the relation
- each A_i is an attribute name in the schema of relation *r*
- D_i is the data type of values in the domain of attribute A_i
- 完整性约束
  - not null
  - primary key (A1, ..., An), 非空且唯一
  - foreign key (Am, ..., An) references r
  - check (P), where P is a predicate

3.1.2.2 Insert

- insert into branch values ('Perryridge', 'Blooklyn', 8000)

3.1.2.3 delete

- The delete command removes tuples from the table

delete from branch

3.1.2.4 Drop

- The drop table command deletes all information (tuples and schema) about the dropped relation from the database
- drop table r

3.1.2.5 Alter

- The alter table command is used to add or delete/drop attributes to an existing relation
- **alter table** *r* **add** *A D*
  - where A is the name of the attribute to be added to relation r and D is the domain of A
  - all tuples in the relation are assigned null as the value for the new attribute
  - e.g. **alter table** *instructor* **add** *age* char(30)
- **alter table** *r* **drop** *A*
  - where A is the name of an attribute of relation r
  - e.g. a**lter table** *instructor* **drop** *age*

3.1.3 查询的基本结构

3.1.3.1 单关系查询

- distinct
  - SQL allows duplicates in relations as well as in query results.
  - To force the elimination of duplicates, insert the keyword distinct after select.
  - E.g : select distinct dept_name
    from instructor
- All
  - The keyword all specifies that duplicates should not be removed.
    select all dept_name from instructor
- Select 的其他
  - An asterisk in the select clause denotes “all attributes”

    select * from instructor
  - An attribute can be a literal(照字面的) with no from clause

    select ‘437’
  - Results is a table with one column and a single row with value “437”
  - Can give the column a name using:

    select ‘437’ as FOO
- Select 附带算术表达式
  - select ID, name, salary/12

from instructor

- would return a temporal relation that is the same as the instructor relation, except that the value of the attribute salary is divided by 12.

- Where
  - The where clause specifies conditions that the result must satisfy

3.1.3.2 多关系查询

- Examples
  - Find the names of all instructors in the Art department who have taught some course and the course_id
  - select name, course_id
    from instructor , teaches
    where instructor.ID = teaches.ID and instructor. dept_name = 'Art'
- 注意事项
  - 查询顺序：from→where→select
  - from 子句中包括多个关系表时，一定要在 where 子句中加入连接条件!
  - 防止出现耗时费力的多表笛卡尔积操作
  - 实际应用中，对频繁执行的 SQL 查询，其 from 子句中的表的个数不要过多，如不要超过 4 个!
  - 避免耗时费力的多表连接操作。
  - 如果频繁执行的 SQL 查询涉及的查询数据存放在 N≥4 张表中，可以考虑将这 N 张表中的数据进行合并

3.1.3.3 自然连接

- 自然连接运算作用于两个关系，并产生一个关系作为结果
- 仅考虑两个关系中 都出现的属性上 取值相同的元组对

3.1.4 附加的基本运算

3.1.4.1 更名运算

- As
- Tuple variables

Tuple variables are defined in the from clause via the use of the as clause ——关系的别名/简称

- 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.salary > S.salary and S.dept_name = 'Comp. Sci.'

利用 T 和 S 区分不同的 instructor，实现了对同一属性的不同值的比较

3.1.4.2 字符串运算

- Like
  - SQL includes a string-matching operator for comparisons on character strings. The operator like uses patterns that are described using two special characters:
  - percent (%). The % character matches any substring.
  - underscore (_). The _ character matches any character.
    1. 'Intro%' matches any string beginning with "Intro".
    2. '%Comp%' matches any string containing "Comp" as a substring.
    3. '_ _ _' matches any string of exactly three characters.
    4. '_ _ _ %' matches any string of at least three characters
  - Match the string "100%"

    like '100 \%' escape '\'

    in that above we use backslash (\) as the escape character.
  - SQL supports a variety of string operations such as
    1. concatenation （关联）(using "||")
    2. converting from upper to lower case (and vice versa)
    3. finding string length, extracting substrings, etc.

3.1.4.3 排列元组的显示次序

- Order by
  - The order by clause causes the tuples in the result of query to appear in sorted order, i.e. ascending order or descending order, denoted by keywords asc or desc respectively
  - ascending order is the default(默认升序)

3.1.4.4 where 子句谓词

- SQL includes a between comparison operator
- Tuple comparison
  - select name, course_id
    from instructor, teaches
    where (instructor.ID, dept_name) = (teaches.ID, 'Biology');
- SQL（select）不主动删除重复，除非用 distinct

3.1.4.5 集合运算

- Union
  - 自动去除重复，与 select 不同
  - 要想保留重复：union all
  - Find courses that run in Fall 2009 or in Spring 2010
    - (select course_id from section where sem = 'Fall' and year = 2009)
      union

(select course_id from section where sem = 'Spring' and year = 2010)

- Intersect
  - 自动去除重复
  - 要想保留重复：intersect all
- Except
  - 从第一个输入中输出所有不出现在第二个输入中的元组
  - 自动去除重复
  - 要想保留重复：except all
  - To find the largest salary of all instructors
  - Step1. Find the salaries of all instructors that are less than the largest salary.
    - select distinct T.salary
      from instructor as T, instructor as S
      where T.salary < S.salary
  - Step2. Find all the salaries of all instructors
    - select distinct salary
      from instructor
  - Step3.
  - (select "second query")
    except
    (select "first query")

3.1.4.6 空值

- It is possible for tuples to have a null value, denoted by null, for some of their attributes
  - null signifies an unknown value or that a value does not exist.
- The predicate is null can be used in Select clause to check for null values
  - E.g. Find all instructors who appear in the instructor relation with null value for salary
  - select name
    from instructor
    where salary is null
- The result of any arithmetic expression involving null is null
  - e.g. 5 + null returns null
- Any comparison with null returns unknown
  - e.g. 5 < null, null <> null , null = null
- In where clause, three-valued logic using the truth value unknown
  - OR

    (unknown or true) = true, (unknown or false) = unknown
    (unknown or unknown) = unknown
  - AND

    (true and unknown) = unknown,

    (false and unknown) = false,

(unknown and unknown) = unknown

- NOT

  (not unknown) = unknown

- “P is unknown” evaluates to true if predicate P evaluates to unknown

- Result of where clause predicate P is treated as false if it evaluates to unknown
- Null values and aggregates（聚集）
  - all aggregate operations except count(*) ignore tuples with null values on the aggregated attributes（聚集函数一般忽略空值，count(*)除外：它计算元组个数，含空值的元组也计算）
  - E.g. Total all instructor salaries

    select sum (salary)
    from instructor
  - if there are null values in salary attributes, above statement ignores null amounts

3.1.4.7 聚集函数

- 基本聚集
  - avg: average value,
  - min: minimum value,
  - max: maximum value,
  - sum: sum of values,
  - count: number of values
    - 有些时候计算聚集函数时需要先删除重复元组：可以使用关键词 distinct
- 分组聚集
  - Group by 子句中所有属性相同的元组将被分在一个组中
  - Find the names and average salaries of all departments whose average salary is greater than 42000
  - select dept_name, avg (salary)

    from instructor

    group by dept_name

    having avg (salary) > 42000;
  - 出现在 select 子句中的属性：要么在聚集函数中，要么必须出现在 group by 子句中
- Having 子句
  - Having 子句中的谓词在 group by 分组形成之后才起作用
  - 任何出现在 having 子句中，但没有被聚集的属性必须出现在 group by 子句中

- 空值和布尔值的聚集
  - All aggregate operations except count(*) ignore tuples with null values on the aggregated attributes
  - 聚集函数一般忽略空值，count(*)除外：它计算元组个数，含空值的元组也计算

3.1.5 嵌套子查询

3.1.5.1 集合成员资格

- In
    - 连接词 In 测试元组是否是集合中的成员，集合是由 select 子句产生的一组值构成的
    - Not in 测试元组是否不是集合中的成员

3.1.5.2 集合的比较-some

- Some
    - 至少比某一个大:some
    - SQL allows < some/all, <= some/all, >= some/all, =some/all, and <>some/all comparisons
- All
    - 比所有的都大;
    - select name

      from instructor

      where salary > all (select salary

      from instructor

      where dept name = 'Biology');

3.1.5.3 空关系测试

- Exists
    - SQL uses exists, not exists and except constructs to test whether or not a subquery (which is a set of tuples) is nonempty or not respectively,
    - exists $r \leftrightarrow r \neq \emptyset$
    - not exists $r \leftrightarrow r = \emptyset$
    - except: $X - Y$
    - $X - Y = \emptyset \leftrightarrow X \subseteq Y$
    - 集合包含，$X \subseteq Y \leftrightarrow (X - Y) = \emptyset \leftrightarrow$ not exists (X except Y)
    - $X \subseteq Y \leftrightarrow$ not exists (X except Y)
        - 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*));

First nested query lists all courses offered in Biology

Second nested query lists all courses a particular student took

- Note that $X - Y = \emptyset \quad \Leftrightarrow \quad X \subseteq Y$

3.1.5.4 重复元组存在性

- Unique
  - The unique construct evaluates to “true” if a given subquery contains no duplicates
  - 如果作为参数的子查询结果中无重复元组，则 unique 返回 true
- 1>=
  - 不使用 unique 的情况下，可使用 1>=(select count(..)) 的方式

3.1.5.5 From 子查询

- 任何 select-from-where 返回的结果都是关系，因而可以被插入到另一个 select-from-where 中任何关系可以出现的位置
  - 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;

3.1.5.6 with 子句

- 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
- 将 1 个复杂查询分解为若干步，每个视图定义 1 个各步的中间计算结果，
- 这些定义只对包含 with 子句的查询有效
- Find all departments with the maximum *budget*
  - **with** *max_budget* (*value*) **as**
    (**select max**(*budget*)
    from *department*)
    select *department.name*
    from *department*, *max_budget*
    where *department.budget* = *max_budget.value*;

3.1.5.7 标量子查询

- Scalar subquery is one which is used where a single value is expected
- List all *departments* along with the number of instructors in each department

■ **select** *dept_name*,
(**select count**(*)
from *instructor*

where *department.dept_name* = *instructor.dept_name*)
as *num_instructors*
from *department*;

3.1.6 数据库的修改

3.1.6.1 Delete

- Delete all instructors whose salary is less than the average salary of instructors
  - **delete from** *instructor*

  where *salary* < (**select avg** (*salary*)

  from *instructor*);
- Problem: as we delete tuples from deposit, the average salary changes
- Solution used in SQL:
  1. First, compute **avg** (salary) and find all tuples to delete
  2. Next, delete all tuples found above (without recomputing **avg** or retesting the tuples)

3.1.6.2 Insert

- 插入时只给出了模式中部分属性的值，则其余属性将被赋空值，用 Null 表示

3.1.6.3 Updata

- 不改变整个元组的情况下改变其部分属性的值
- **update** *instructor*
  set *salary* = **case**
  when *salary* <= 100000 **then** *salary* * 1.05
  else *salary* * 1.03
  end

4. 中级 SQL

4.1 连接表达式

- Join operations take two relations and return as a result another relation.
- A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition). It also specifies the attributes that are present in the result of the join
- The join operations are typically used as subquery expressions in the from clause

4.1.1 连接条件

- On

Join types
inner join
left outer join
right outer join
full outer join

Join Conditions
natural
on <predicate>
using (A_1, A_1, …, A_n)

4.1.2 外连接

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join.
- Uses null values.

4.1.2.1 左外连接（left outer join）

- 只保留出现在左外连接运算之前（左边）的关系中的元组

course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*

4.1.2.2 右外连接

- 只保留出现在右外连接运算之后（右边）的关系中的元组

course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

4.1.2.3 全外连接

- 保留出现在两个关系中的元组

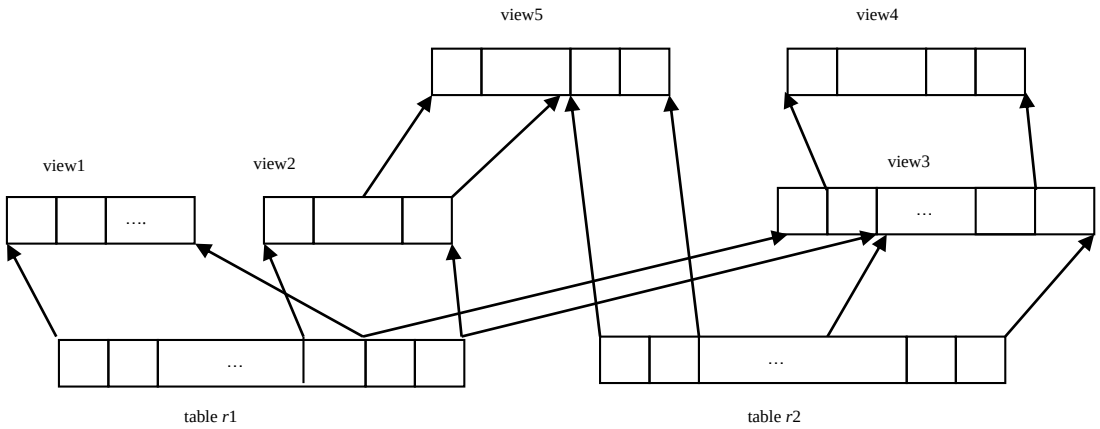
course_id	*title*	*dept_name*	*credits*	*prere_id*
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

4.2 视图

- In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database.)
- 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.

4.2.1 视图的定义



- A view is defined using the create view statement which has the form

  create view v as <query expression >

- where <query expression> is any legal SQL expression. The view name is represented by v.
  - E.M

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

    Find all instructors in the *Biology* department
    select *name*
    from *faculty*
    where *dept_name* = ‘Biology’

4.2.2 SQL 查询中使用视图

- 如果一个视图关系被计算并存储，一旦用于定义该视图的关系被修改，视图就会过期

  定义视图时，数据库系统存储视图定义本身，而不存储定义该视图的查询表达式的执行结果

  一旦视图关系出现在查询中，它就被已经存储的查询表达式代替

  无论何时执行这个查询，视图关系都被重新计算

- 一个视图可能被用到定义另一个视图的表达式中
  - **create view** *physics_fall_2009* **as**
    select *course.course_id*, *sec_id*, *building*, *room_number*

from *course*, *section*
where *course.course_id* = *section.course_id*
and *course.dept_name* = 'Physics'
and *section.semester* = 'Fall'
and *section.year* = '2009';

- **create view** *physics_fall_2009_watson* **as**
  select *course_id*, *room_number*
  from *physics_fall_2009*
  where *building*= 'Watson';

4.2.3 物化视图

4.2.4 视图更新

- 如果用视图来表达更新，插入、或删除，可能带来严重问题：困难在于，用视图表达的数据库修改必须被翻译为对数据库模型中实际关系的修改

4.2.4.1 可更新的

- From 子句只有一个数据库关系
- Select 子句中包含关系的属性名，不包含任何表达式、聚集、或 distinct 声明
- 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 groupby 或 having 子句

在这些限制下，这些试图允许执行 update insert delete 操作

4.3 完整性约束

- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency（一致性）

4.3.1 单个关系上的约束

- An SQL relation is defined using the **create table** command

  create table ***r*** (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$))

  - *r* is the name of the relation
  - each A_i is an attribute name in the schema of relation *r*
  - D_i is the data type of values in the domain of attribute A_i

- The allowed integrity constraints include

  primary key

  not null

  unique

  check (P), where P is a predicate

4.3.2 Not null

- Declare *name* and *budget* to be not null

name **varchar**(20) **not null**
budget **numeric**(12,2) **not null**

- 禁止在声明属性上插入空值
- 主码必不为空值，不必显式声明

4.3.3 Unique

- The Unique Constraint is as follows

  unique (A_1, A_2, ···, A_m)

  it states that the attributes

  *A*1, *A*2, ··· *A*m
  form a **candidate key**

  candidate keys are permitted to be null (in contrast to primary keys)

  e.g. **create table** *customer*
  (*customer-id* char(15)
  customer-name char(15)
  customer-city char(30),
  primary key (*customer-id*)
  unique (*customer-name*)

4.3.4 Check

- The check clause is applied to relation declaration as well as to domain declaration

  check (*P*), where *P* is a predicate

4.3.5 参照完整性

- **Referential integrity**:ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation
- 保证在一个关系中给定属性集上的取值也在另一关系的特定属性集的取值中出现
  - **create table** *course (*
    course_id **char**(5) **primary key**,
    title **varchar**(20),
    dept_name **varchar**(20) **references** *department*
)
  - **create table** *course* (
    ...
    dept_name **varchar**(20),
    foreign key (*dept_name*) **references** *department*
    on delete cascade

    //级联删除：如果删除 department 中的元组导致违反了完整性约束，系统不拒绝删除，而是对 course 关系做级联删除
    on update cascade, //
    . . .
)
- alternative actions to cascade: **set null**, **set default**

4.3.6 复杂 check 条件与断言

4.3.6.1 Check

- E.g. For the relation *section*, the time_slot_id in each tuple should be is actually the identifier of a time_slot in the *time_slot* relation. This constraint can be defined as

  check (*time_slot_id* **in** (**select** *time_slot_id* **from** *time_slot*))

- *check* in *create table*

- **create table** *student* (
  ID **varchar**(5),
  name **varchar**(20) not null,
  dept_name **varchar**(20),
  tot_cred **numeric**(3,0),
  primary key *(ID),*
  foreign key *(dept_name*) **references** *department*)
  check(tot_cred>0)

4.3.6.2 assertion

- An assertion is a predicate expressing a condition that we wish the database always to satisfy
- 一个断言就是一个谓词，它表达了我们希望数据库总能满足的一个条件

  e.g. domain constraints, referential-integrity constraint
- An assertion in SQL takes the form

  create assertion <assertion-name> **check** <predicate>
- When an assertion is made, the DBMS tests it for validity. Any modification to DB is allowed only if it does not cause that assertion to be violated
  - this testing may introduce a significant amount of overhead, hence assertions should be used with great care
- For each tuple in the *student* relation, the value of the attribute *tot_cred* must equal the sum of credits of courses that the student has completed successfully

  create assertion *credits_earned_constraints* **check**
  (**not exists** (**select** *ID*
  from *student*
  where *tot_cred*
  <> **select sum**(*credits*)
- **from** *takes* **natural join** *course*
  where *student.ID= takes.ID*
  and *grade* **is not null** and grade<>'F')

4.4 SQL 的数据类型与模式

4.4.1 日期和时间

- Date
- Time
- Timestamp

4.4.2 默认值

Create table student(
ID varchar(5) **default** 0)

- 如果没有给出 ID 的值，元组在该属性上的取值就被置为 0

4.4.3 创建索引

Create index student_id on student(ID)

- 多个属性上建立索引：（ID,NAME…）逗号分隔

4.5 授权

- Forms of authorization on parts of the database include

  Read - allows reading, but not modification of data

  Insert - allows insertion of new data, but not modification of existing data

  Update - allows modification, but not deletion of data Delete - allows deletion of data

4.5.1 授权的授予

- The **grant** statement is used to confer（授予） authorization

  grant <privilege list>

  on <relation name or view name> **to** <user list>

  <user list> is:

- a user-id
- **public**, which allows all valid users the privilege granted

- **select:** allows read access to relation,or the ability to query using the view
- Example: grant users U_1, U_2, and U_3 **select** authorization on the *instructor* relation:

  grant select on *instructor* **to** U_1, U_2, U_3
- **insert**: the ability to insert tuples
- **update**: the ability to update using the SQL update statement
- **delete**: the ability to delete tuples.
- **all privileges**: used as a short form for all the allowable privileges

- The **revoke** statement is used to revoke authorization.

  revoke <privilege list>

  on <relation name or view name> **from** <user list>
- Example:

  revoke select on *branch* **from** U_1, U_2, U_3
- <privilege-list> may be **all** to revoke all privileges the revoker may hold.
- If <revokee-list> includes **public,** all users lose the privilege except those granted it explicitly.

4.5.2 角色

- **create role** instructor;
- **grant** *instructor* **to Amit;**
- Privileges can be granted to roles:

  grant select on *takes* **to** *instructor*;
- Roles can be granted to users, as well as to other roles

  create role *teaching_assistant*

  grant *teaching_assistant* **to** *instructor*;

Instructor inherits all privileges of *teaching_assistant*

4.5.3 视图的授权

- create view geo_instructor as
  (select *

from instructor
where dept_name = 'Geology');

- grant select on geo_instructor to geo_staff

4.5.4 其他

- **references** privilege to create foreign key

  grant reference (*dept_name*) **on** *department* **to** Mariano;

- transfer of privileges

  grant select on *department* **to** Amit **with grant option**;

  revoke select on *department* **from** Amit, Satoshi **cascade**;

  revoke select on *department* **from** Amit, Satoshi **restrict（restrict，防止级联收回）**;

5. 高级 SQL

5.1 使用程序设计语言访问数据库

- API (application-program interface) for a program to interact with a database server
- Application conducts complex data processing, and makes calls to
  - Connect with the database server
  - Send SQL commands to the database server
  - Fetch tuples of result one-by-one into program variables
- Various tools:
  - JDBC (Java Database Connectivity) works with Java
  - ODBC (Open Database Connectivity) works with C, C++, C#, and Visual Basic
  - Other API's such as ADO.NET sit on top of ODBC
  - Embedded SQL

5.1.1 JDBC

- JDBC is a Java API for communicating with database systems supporting SQL.
- JDBC supports a variety of features for querying and updating data, and for retrieving query results.
- JDBC also supports metadata retrieval, such as querying about relations present in the database and the names and types of relation attributes.
- Model for communicating with the database:
  - Open a connection
  - Create a "SQL statement" object
  - Execute queries using the Statement object to send queries and fetch results
  - Exception mechanism to handle errors

- Open a connection, named as conn
  - using the getConnection method of the DriverManager class
  - parameters

    URL/machine to be connected;

user identifier, password

- Create a SQL “statement” object on the connection conn
  - using the createStatement method
  - its statement handle is named as stmt
  - allows the Java program to invoke methods that ship an SQL statement given as an argument for execution by the DBMS
- Execute queries using the Statement object stmt to send queries and fetch results
  - using execute.query or execute.update such as insert/delete/update/createtable,
  - parameters
    - the SQL statement to be executed, represented as a string
  - using the try{…}/catch{…}construct to catch any exceptions or error conditions

- Fetch the query result
  - retrieve the set of tuples in the result into a ResultSet object rset, and fetch them one tuple at a time
  - the next() method tests whether or not the result set has at least one tuple and if so, fetches it
- At the end of the query, the statement stmt and the connection conn are closed

5.1.2 嵌入式 SQL

- Approaches to take SQL as DB query tools
  - interactive SQL
    - SQL is used directly as DML and DDL through DBS human-machine interfaces
  - dynamic SQL, e.g JDBC, ODBC
  - embedded SQL
    - SQL is embedded in general-purpose programming languages, e.g. *C* language
    - executing of general-purpose programming language programs with SQL statement embedded results in DB access

- The SQL standard defines embeddings of SQL in a variety of programming languages such as C, C++, Pascal, Fortran, and Cobol
- A language to which SQL queries are embedded is referred to as a *host language*(宿主语言), and the SQL structures permitted in the host language comprise *embedded* SQL
- Merits of embedded SQL
  - 交互式 SQL 只能进行 DB 的访问操作，不能对 DB 访问结果进行进一步的数据处理
  - Embedded SQL 将 SQL 的数据库访问功能与 C 语言等宿主语言的数据处理能力相结合，提高了数据应用系统的能力

5.1.2.1 Exec SQL

- **EXEC SQL** statement is used to identify embedded SQL request to the preprocessor

  - EXEC SQL <embedded SQL statement >;

- Note: this varies by language:

- In some languages, like COBOL, the semicolon(分号) is replaced with END-EXEC
- In Java embedding uses # SQL { …. };

- Before executing any SQL statements, the program must first connect to the database. This is done using:
  - EXEC-SQL **connect to** *server* **user** *user-name* **using** *password*;
  - Here, *server* identifies the server to which a connection is to be established.

- Variables of the host language can be used within embedded SQL statements. They are preceded by a colon (:) to distinguish from SQL variables
  - e.g., :*credit_amount*
- Variables used as above must be declared within DECLARE section, as illustrated below. The syntax for declaring the variables, however, follows the usual host language syntax.
  - EXEC-SQL BEGIN DECLARE SECTION}
    - int *credit-amount* ;
  - EXEC-SQL END DECLARE SECTION;

- E.g.1 From within a host language, find the ID and name of students who have completed more than the number of credits stored in variable *credit_amount* in the host langue
- *credit_amount* is the shared variable defined in the declaration part

EXEC SQL BEGIN DECLARE SECTION

int *credit_amount*;

EXEC SQL END DECLARE SECTION

credit_amount := e.g. input-from-screen-by-users

EXEC SQL

select *ID, name*

from *student*

where tot_cred *> :credit_amount*

END-EXEC

5.1.2.2 cursor

- To write an embedded SQL query, we use the
  - **declare** *c* **cursor for <SQL query>**

  tatement.
  - The variable *c* is used to identify the query
- EXEC SQL
  - **declare** *c* **cursor for**
    select *ID, name*
    from *student*
    where tot_cred *> :credit_amount*

  END_EXEC
- 利用 Embedded SQL 进行查询时，查询结果有可能包括多个元组，此时无法直接将多个元组通过共享变量赋值传递给宿主程序

- *系统开辟专门 working 区域存放 SQL 查询的结果关系，并利用查询游标 **c** 指向此区域。宿主程序根据 **c** 指向的查询结果关系集合，使用 **open, fetch, close** 依次获取结果关系中的各元组

- Usage of cursor in embedded SQL

  declare cursor – open – fetch – close

  - EXAMPLE2

    EXEC SQL BEGIN DECLARE SECTION

    int *credit_amount*;

    char *si*, *sn*;

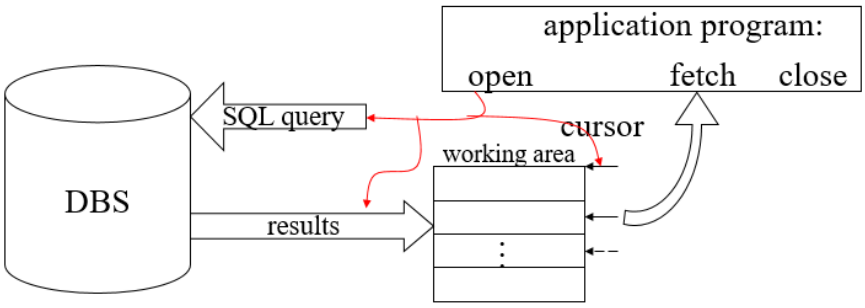
    EXEC SQL END DECLARE SECTION

    credit_amount:= input-from-screen-by-users

    EXEC SQL

    declare *c* **cursor for**

    select *ID, name*
    from *student*
    where tot_cred > *:credit_amount*



END-EXEC

*EXEC SQL **open c** END-EXEC*

*EXEC SQL **fetch c into** **:si**, **:sn** END-EXEC*

*EXEC SQL **close c** END-EXEC*

- The **open** statement for our example is as follows:
  - EXEC SQL **open** *c* ;
  - This statement causes the database system to execute the query and to save the results within a temporary relation. The query uses the value of the host-language variable *credit-amount* at the time the **open** statement is executed.
- The **fetch** statement causes the values of one tuple in the query result to be placed on host language variables.
  - EXEC SQL **fetch** *c* **into** *:si, :sn* END_EXEC
  - Repeated calls to fetch get successive tuples in the query result
- The **close** statement causes the database system to delete the temporary relation that holds the result of the query.

  EXEC SQL **close** *c* ;
- Note
  - above details vary with language

- for example, the Java embedding defines Java iterators（迭代器） to step through result tuples.

5.2 触发器

- Trigger
  - a statement that is executed automatically by DBMS as a side effect of a modification to the database
  - 对数据库作修改时，它被系统自动执行
- Trigger is an *event-condition-action* model based integrity definition, checking, remedy mechanism
  - specify what **event**s cause the trigger to be executed (e.g. insert, delete, update), and under which **conditions** the trigger execution will proceed
    - integrity constraints checking
  - specify the **actions** to be taken when the trigger executes
    - if constraints is violated, remedy actions are taken

6. 形式化关系查询语言

6.1 关系代数

Relational algebra

6.1.1 基本运算

- Six basic operators
  - select: σ
  - project: ∏
  - union: ∪
  - set difference: –
  - Cartesian product: x
  - rename: ρ
- The operators take one or two relations as inputs and produce a new relation as a result.

6.1.1.1 选择运算

- Notation: $\sigma_p(r)$
- p is called the selection predicate
- Defined as:

$$\sigma_p(r) = \{t \mid t \in r \text{ and } p(t)\}$$

  - Where p is a formula in propositional calculus consisting of terms connected by : ∧ (and), ∨ (or), ¬ (not)
    Each term is one of:
    - <attribute> *op* <attribute> or <constant>
    - where *op* is one of: =, ≠, >, ≥. <. ≤

- Example of selection: $\sigma_{dept\_name="Physics"}(instructor)$

■ **Relation r**

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

■ $\sigma_{A=B \wedge D > 5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

6.1.1.2 投影运算

- Notation:

$$\Pi_{A_1, A_2, \ldots, A_K}(r)$$

  - where A_1, A_2 are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing（去掉） the columns that are not listed
- Duplicate rows removed from result, since relations are sets（由于关系是集合，重复行均被去除）
- Example: To eliminate the *dept_name* attribute of *instructor*

$$\Pi_{ID, name, salary}(instructor)$$

■ **Relation r:**

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

$\Pi_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

6.1.1.3 关系运算的组合

- Several relational algebra operations can be composed together into a **relational algebra expression**

- E.g.2. $\Pi_{id,\ name,\ salary}(\sigma_{dept\_name="Physic"}(instructor))$

- **Can build expressions using multiple operations**
- **Example:** $\sigma_{A=C}(r \times s)$
- $r \times s$

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

- $\sigma_{A=C}(r \times s)$

A	B	C	D	E
α	1	α	10	a
β	2	β	10	a
β	2	β	20	b

6.1.1.4 并运算

- Notation: $r \cup s$
- Defined as:

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

- For $r \cup s$ to be valid.
- r, s must have the *same* **arity（数目）** (same number of attributes)
  - The attribute domains must be **compatible** (example: 2nd column of r deals with the same type of values as does the 2nd column of s)

6.1.1.5 集合差

- Notation $r - s$
- Defined as:

$$r - s = \{t \mid t \in r \text{ and } t \notin s\}$$

- Set differences must be taken between **compatible** relations.
  - r and s must have the same arity
  - attribute domains of r and s must be compatible
- Note: $r \cap s = r - (r - s)$
-

- **Relations r, s:**

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- **$r - s$:**

A	B
α	1
β	1

6.1.1.6 笛卡儿积

- Notation $r \times s$

- Defined as:
  - $r \times s = \{t \mid t = t_1\, t_2$ **and** $t_1 \in$ ***r*** **and** $t_2 \in$ ***s*** $\}$
- Assume that attributes of r(R) and s(S) are disjoint. (That is, $R \cap S = \varnothing$).
- If attributes of *r(R)* and *s(S)* are not disjoint, then renaming must be used.

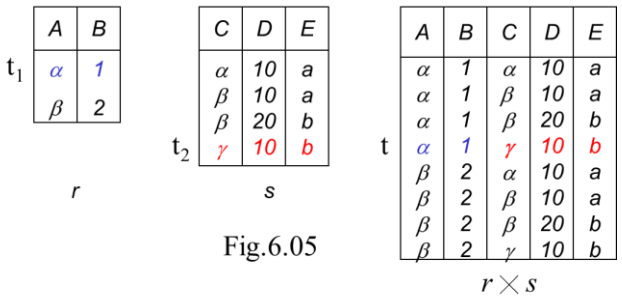
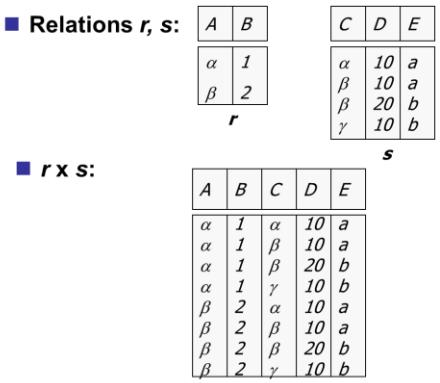


Fig.6.05

- For relation ***r(R)*** and ***s(S)***, $r \times s$ is a relation whose schema is ***R'***, which is the concatenation （级联） of *R* and *S*
  - ***R*** $(A_1, A_2, \cdots, A_m)$, ***S*** $(B_1, B_2, \cdots, B_n)$
  - ***R'*** $(A_1, A_2, \cdots, A_m, B_1, B_2, \cdots, B_n)$
- for $t \in r \times s$, there are $t_1 \in r$ and $t_2 \in s$, such that
  - $t[R] = t_1[R]$
  - $t[S] = t_2[S]$
- assume that attributes of r(R) and s(S) are disjoint. i.e. $\{A_1, A_2, \cdots, A_m\} \cap \{B_1, B_2, \cdots, B_n\} = \varnothing$
- if attributes of *r(R)* and *s(S)* are not disjoint, then renaming must be used.



6.1.1.7 Fundamental Operations vs *Select* statement in SQL

- A typical SQL query has the form
  - **select** $A_1, A_2, \ldots, A_n$
    from $r_1, r_2, \ldots, r_m$
    where P
  - A_is represent attributes
  - r_is represent relations
  - P is a predicate
- Equivalent to the relational algebra expression

$$\Pi_{A1, A2, \ldots, An}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

6.1.1.8 更名运算

- The rename operation on relation *r* or relational algebra expression ***E***

  $\rho_{x}(r)$, $\rho_{x}(E)$

- renaming ***r*** or ***E*** as ***X***
- If a relational-algebra expression ***E*** has arity *n*, then
  - $\rho_{x(A1, A2, \cdots, An)}(E)$

  ρ returns the result of expression *E* under the name *X*, and with the attributes renamed to *A1, A2, ···, An*

- E.g. $\rho_{s(A1, A2, \cdots, An)}(r(B_1, B_2, \cdots, B_n)) = s(A_1, A_2, \cdots, A_n)$

- Find the high*est salary* in the *univsersity*
- Principle
  - /*将对一个关系 *instructor* 中同一个属性 *salary* 的不同取值的比较转换为利用换名操作、迪卡尔乘积和选择操作对关系 *instructor* 和它的副本$\rho_{d}(instructor)$在同一属性 *salary* 上取值的比较, *refer to Fig.6.0.6*
- Step1.
  - $instructor \times \rho_{d}(instructor)$
- Step2. all tuples that have *non-largest salary* values in *instructor* relation:

  $\sigma_{instructor.salary < d.salary}(instructor \times \rho_{d}(instructor))$

ins.ID	*ins.name*	*ins.salary*
120005	Zhang	1000
130007	Wang	2000
198032	Li	3000

(a) *instructor*

d.ID	*d.name*	*d.salary*
120005	Zhang	1000
130007	Wang	2000
198032	Li	3000

(b) $d = \rho(instructor)$

- Step3. all *non-largest salary* values in relation *instructor*
  - $\prod_{instructor.salary}($

    $\sigma_{instructor.salary < d.salary}(instructor \times \rho_{d}(instructor))$
- Step4. All largest *salary* in relation *univsersity*
- $\prod_{salary}(instructor)$ —

  $\prod_{instructor.salary}($

  $\sigma_{instructor.salary < d.salary}(instructor \times \rho_{d}(instructor))$

- *branch (branch_name, branch_city, assets)*

  customer (customer_name, customer_street, customer_city)

  account (account_number, branch_name, balance)

  loan (loan_number, branch_name, amount)

  depositor (customer_name, account_number)

borrower (customer_name, loan_number)

- **Find all loans of over $1200**

$$\sigma_{amount > 1200}(loan)$$

- **Find the loan number for each loan of an amount greater than $1200**

$$\prod_{loan\_number}(\sigma_{amount > 1200}(loan))$$

- **Find the names of all customers who have a loan, an account, or both, from the bank**
  - $\prod_{customer\_name}(borrower) \cup \prod_{customer\_name}(depositor)$
- **Find the names of all customers who have a loan at the Perryridge branch.**
  - **Query 1**

$$\prod_{customer\_name}(\sigma_{branch\_name = \text{“Perryridge”}}(\sigma_{borrower.loan\_number = loan.loan\_number}(borrower \times loan)))$$

  - **Query 2**

$$\prod_{customer\_name}(\sigma_{loan.loan\_number = borrower.loan\_number}((\sigma_{branch\_name = \text{“Perryridge”}}(loan)) \times borrower))$$

6.1.2 关系代数的形式化定义

- Relational algebra expression E is defined as follows
  - a basic expression is
  - a relation r in the database, or
  - a constant relation $C = \{c_1, c_2, .., c_n\}$, which is written by listing its tuples in the relation
- a general expression is defined as:

  if E_1 and E_2 are relational algebra expressions, then

- $E_1 \cup E_2$
- $E_1 - E_2$
- $E_1 \times E_2$
- $\sigma_p(E_1)$
- $\prod s(E_1)$,
- $\rho_x(E_1)$

are all relational algebra expressions

6.1.3 附加的关系代数运算

- Additional operations
  - $\cap$, set-intersection（集合交）
  - ∞, natural-join （自然、外连接）
  - $\leftarrow$, assignment（赋值）
- With respect to querying powers, additional operations are equivalent with fundamental operations, i.e. can be replaced (or re-represented) by the six basic operations operations
  - a additional operation can be replaced/re-represented by basic operations
  - additional operations simplify representations of queries

6.1.3.1 集合交运算

- For relation ***r*** and ***s***, set-intersection operation on **r** and **s** is defined as

  $r \cap s = \{ t \mid t \in r \text{ and } t \in s \}$

  - ***r, s*** have the *same arity*
  - attributes of **r** and **s** are compatible
- $r \cap s = r - (r - s)$

- **Relation *r, s*:**

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cap s$

A	B
α	2

6.1.3.2 自然连接运算

- For relations $r(R)$ and $s(S)$, assuming $R \cap S = \{A_1, A_2, \cdots, A_n\}$, the natural join of r and s, denoted by $r \bowtie s$, is a relation on *schema* $R \cup S$,

  r $\bowtie$ **s** =

  $\prod_{R \cup S}(\sigma_{r.A1=s.A1 \wedge r.A2=s.A2 \wedge \cdots \wedge r.An=s.An}(r \times s))$
- E.g.1 (cont.)

  $R = (A, B, C, D)$, $S = (B, D, E)$, $r(R)$ and $s(S)$,

  - $r \bowtie s =$ $\prod_{r.A, r.B, r.C, r.D, s.E}(\sigma_{r.B = s.B \wedge r.D = s.D}(r \times s))$
- defined on the schema (A, B, C, D, E)
- Natural-join operation $\bowtie$ combine σ and $\times$ on some relations into one operation
  - to improve execution efficiency of query

- E.g.2 Find the *names* of all instructors in the Comp.Sci department together with the course titles of all courses the instructors teach

name	*title*
Brandt	Game Design
Brandt	Image Processing
Katz	Image Processing
Katz	Intro. to Computer Science
Srinivasan	Intro. to Computer Science
Srinivasan	Robotics
Srinivasan	Database System Concepts

Figure 6.16 Result of $\Pi_{name, title}(\sigma_{dept\_name = \text{“Comp. Sci.”}}(instructor \bowtie teaches \bowtie course))$.

- Properties of natural join

$r \bowtie (s \bowtie t) = (r \bowtie s) \bowtie t$

if $R \cap S = \Phi$（不含有任何相同属性）, then $r(R) \bowtie s(S) = r(R) \times s(S)$

- **Relations r, s:**

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	∈

s

- **r ⋈ s**

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

- 等价的 sql:
  - Select * from r natural join s

    Where r.B = s.B and r.D = s.D

6.1.3.3 赋值运算

- **Assign** (←) *relation algebra expression E* to a *relational variable* var_r, i.e. denoting *E* as var_r.

$$\text{var\_r} \leftarrow E$$

  - The assignment operation (←) provides a convenient way to express complex queries

  - E.g. Write $r \div s$ as

$$temp1 \leftarrow \prod_{R-S}(r)$$
$$temp2 \leftarrow \prod_{R-S}((temp1 \times s) - \prod_{R-S,S}(r))$$
$$result = temp1 - temp2$$

- Definition in terms of the basic algebra operation

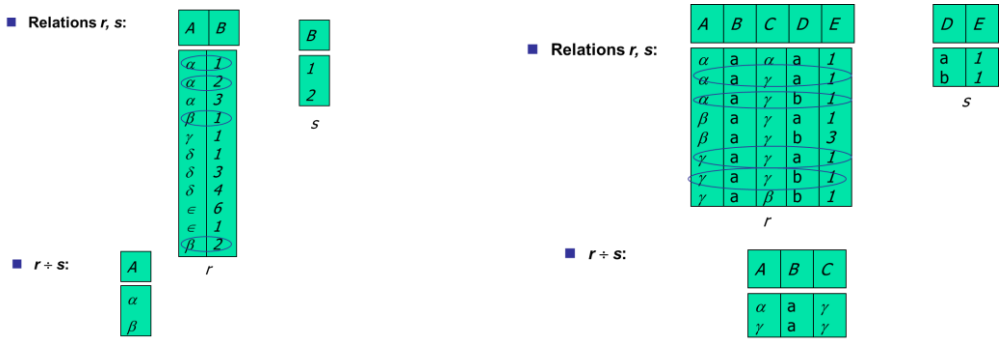
Let $r(R)$ and $s(S)$ be relations, and let $S \subseteq R$

$$r \div s = \prod_{R-S}(r) - \prod_{R-S}((\prod_{R-S}(r) \times s) - \prod_{R-S,S}(r))$$

- To see why
  - $\prod_{R-S,S}(r)$ simply reorders attributes of r
  - $\prod_{R-S}(\prod_{R-S}(r) \times s) - \prod_{R-S,S}(r))$ gives those tuples t in

    $\prod_{R-S}(r)$ such that for some tuple $u \in s$, $tu \notin r$.

  - $r \div s = \{ t \mid t \in \prod_{R-S}(r) \wedge \forall u \in s (tu \in r) \}$
  - Where tu means the concatenation（串连） of tuples t and u to produce a single tuple



6.1.3.4 外连接

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values:
  - *null* signifies that the value is unknown or does not exist
  - All comparisons involving *null* are (roughly speaking) false by definition.
  - We shall study precise meaning of comparisons with nulls later

Relation *loan*

loan_number	branch_name	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

Relation *borrower*

customer_name	loan_number
Jones	L-170
Smith	L-230
Hayes	L-155

Join

loan ⋈ *borrower*

loan_number	branch_name	amount	customer_name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

Left Outer Join（左外连接）

loan ⟕ *borrower*

loan_number	branch_name	amount	customer_name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*

Right Outer Join（右外连接）

loan ⟖ *borrower*

loan_number	branch_name	amount	customer_name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

Full Outer Join（全外连接）

loan ⟗ *borrower*

loan_number	branch_name	amount	customer_name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

6.1.3.5 Modification （修改）of the Database

- The content of the database may be modified using the following operations:
  - Deletion（删除）
  - Insertion（插入）
  - Updating（更新）
- All these operations are expressed using the assignment （赋值）operator.

- A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.
  - Can delete only whole tuples; cannot delete values on only particular attributes
  - A deletion is expressed in relational algebra by:

    $r \leftarrow r - E$

    where r is a relation and E is a relational algebra query.
  - **Delete all loan records with amount in the range of 0 to 50**

    $loan \leftarrow loan - \sigma_{amount \geq 0 \text{ and } amount \leq 50}(loan)$

  - To insert data into a relation, we either:
    - specify a tuple to be inserted
    - write a query whose result is a set of tuples to be inserted
  - in relational algebra, an insertion is expressed by:
    - $r \leftarrow r \cup E$
    - where r is a relation and E is a relational algebra expression.
  - The insertion of a single tuple is expressed by letting E be a constant relation containing one tuple.
    - Insert information in the database specifying that Smith has $1200 in account A-973 at the Perryridge branch.
      - $account \leftarrow account \cup$ {("A-973", "Perryridge", 1200)}
      - depositor ← $depositor \cup$ {("Smith", "A-973")}
  - A mechanism to change a value in a tuple without charging *all* values in the tuple
  - Use the generalized （广义投影）projection operator to do this task
    -

      $$r \leftarrow \prod_{F_1, F_2, \ldots, F_l}(r)$$

  - Each F_i is either
    - the I^{th} attribute of r, if the I^{th} attribute is not updated, or,
    - if the attribute is to be updated, F_i is an expression, involving only constants and the attributes of r, which gives the new value for the attribute
      - Make interest（利息） payments by increasing all balances by 5 percent.
        - $account \leftarrow \prod_{account\_number, branch\_name, balance * 1.05}(account)$

6.1.4 扩展的关系代数运算

6.1.4.1 广义投影

- Generalized project extends the *projection* operation by allowing arithmetic functions to be used in the projection list.

  $\prod_{F1, F2, \cdots, Fn}(E)$

- , where
- E is any relational-algebra expression
- each of $F_1, F_2, \cdots, F_n$ are are arithmetic expressions involving constants and attributes in the schema of E.

- involving arithmetic operation on values of *different attributes in one relation*,
  - E.g.

    $\prod_{ID,\ name,\ department\_name,\ salary/12}$ *(instructor)*

6.1.4.2 聚集

Aggregation functions

- Aggregation functions
  - mapping of a collection of attribute values in a relation r into a single value, i.e. attribute domains → domain

  avg: average value

  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values
- A *convenient* scheme for computing and comparing of the values in one attribute of the relation, *in contrast with*

- Aggregate operation in relational algebra

  G1, G2, ⋯, Gn $\mathcal{G}$ F1(A1), F2(A2),⋯, Fm(Am) (E)
  - E is any relational-algebra expression
  - G1, G2 ⋯, Gn is a list of attributes on which to group (may be empty)
    - a list of attributes in E on which to group (分组) , i.e. tuples in E are classified into several groups E1, E2, ⋯, En, each group Ei is a subset of E, and tuples in each group Ei have the same values for G1,G2,⋯,Gn
  - if G1, G2,⋯,Gn do not appears, aggregations are conducted on all tuples in E, that is, E are not classified into groups

- each Ai is an attribute on which to conduct aggregation functions Fi
  - also the attribute names for aggregation results
- aggregation functions Fi , e.g. avg, count, min, max, sum
- computing procedure
  - separating E into groups on the basis of G1, G2 ⋯, Gn
  - computing F1, F2,⋯, Fn on each group, and return computing results denoted as A1, A2, ⋯, An

- Relation account grouped by branch-name:

branch_name	account_number	balance
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$branch\_name\ \mathcal{G}_{\mathbf{sum}(balance)}\ (account)$

branch_name	**sum**(balance)
Perryridge	1300
Brighton	1500
Redwood	700

- Result of aggregation does not have a name
  - Can use rename operation to give it a name
  - For convenience, we permit renaming as part of aggregate operation
  - Result of aggregation does not have a name
  - Can use rename operation to give it a name
  - For convenience, we permit renaming as part of aggregate operation
    - $branch\_name\ \mathcal{G}\ \mathbf{sum}(balance)\ \mathbf{as}\ sum\_balance\ (account)$