Chp 3
Introduction to SQL

王朝坤
清华大学软件学院

2025年/春

本节课的内容

内容	核心知识点	对应章节
DBMS理论	关系数据模型（以及关系演算)、**SQL语言**	CHP. 1、2、**3**
DBMS设计	存贮、索引、查询、优化、事务、并发、恢复	CHP. 12、13、14、15、16、17、18、19
DBMS实现	大作业	小班辅导
DBMS应用	实体-联系图、关系范式、DDL、JDBC	CHP. 4、5、6、7

数据库应用的核心问题

1.存放了什么数据?

2.它们之间有什么**联系?**

3.如何组织数据?

4.如何查询数据?

用户如何**高效、自然地**表达查询请求

SQL的发展简史

- **In 1974, Structured Query Language Proposed by Don Chamberlin and Ray Boyce from IBM**
  - SEQUEL, the Structured English Query Language
  - Up to 1979, First implemented by San Jose Research Laboratory of IBM

Don Chamberlin

1. **In 1986, ANSI and ISO published SQL-86**
2. **In 1989, ISO/IEC published SQL-89**
3. **In 1992, ISO/IEC published SQL-92**
4. **In 1999, ISO/IEC published SQL:1999**
5. **In 2003, ISO/IEC published SQL:2003**
6. **In 2008, ISO/IEC published SQL:2008**
7. **In 2011, ISO/IEC published SQL:2011**
8. **In 2016, ISO/IEC published SQL:2016**

Raymond F. Boyce

1946 –
June 16,1974

SQL 2023

- ISO/IEC 9075-1:2023 – Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework)
- ISO/IEC 9075-2:2023 – Information technology – Database languages – SQL – Part 2: Foundation (SQL/Foundation)

° ° °

- ISO/IEC 9075-11:2023 – Information technology – Database languages – SQL – Part 11: Information and definition schemas (SQL/Schemata)
- ISO/IEC 9075-13:2023 – Information technology – Database languages – SQL – Part 13: SQL Routines and types using the Java TM programming language (SQL/JRT)
- ISO/IEC 9075-14:2023 – Information technology – Database languages – SQL – Part 14: XML-Related Specifications (SQL/XML)
- ISO/IEC 9075-15:2023 – Information technology – Database languages – SQL – Part 15: Multidimensional Arrays (SQL/MDA)
- ISO/IEC 9075-16:2023 – Information technology – Database languages – SQL – Part 16: Property Graph Queries (SQL/PGQ)

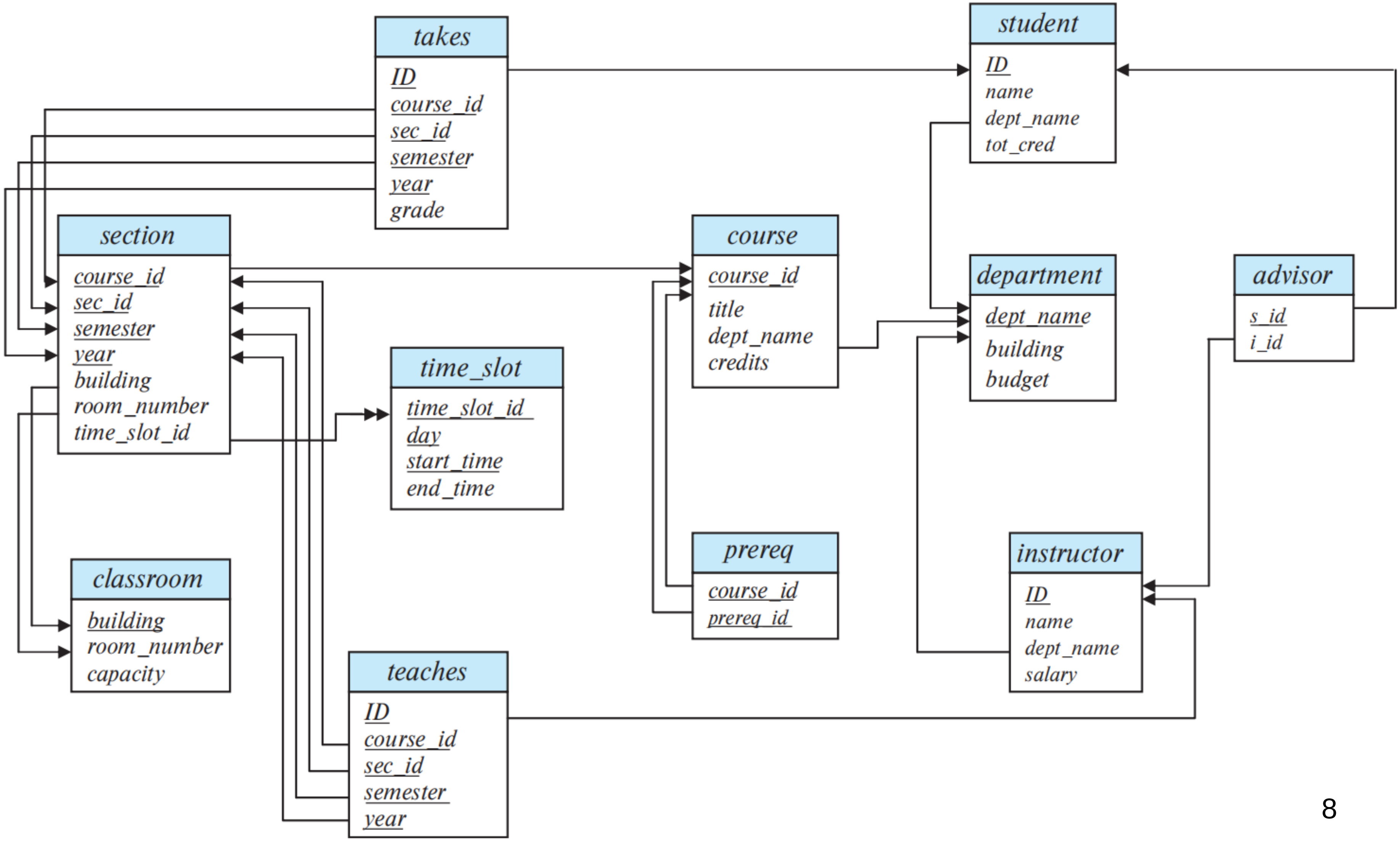
SQL的范畴

1. **Data Definition Language**
   - Commands for relation schema operations
2. **Data Manipulation Language**
   - A query language based on both RA and RC
3. Integrity
4. View definition
5. **Transaction Control Language**
6. **Embedded SQL and Dynamic SQL**
   - define how SQL statements can be embedded within high-level programming language
7. Authorization

Main Contents

- **SQL Data Definition**
- Basic Structure of SQL Queries
- Additional Operations & Null Values
- Complex SQL
  - Set Operations
  - Aggregation
  - Nested Sub-queries
- Modification of the Database

Schema Diagram for University Database



数据定义语言 - DDL

The SQL allows the specification of information about relations, including:

- The schema name and its attributes
- The domain of values associated with each attribute
- Integrity constraints

And as we will see later, also other information such as

- *The set of indices to be maintained for each relations(索引).*
- *Security and authorization information for each relation(授权).*
- *The physical storage structure of each relation on disk(存储).*

e.g. SQL Server 2019 DATA_COMPRESSION Specifies the data compression option for the specified table

表定义，也就是关系定义

- Definition: **R (A, D, dom, F)**
  - Table header
- To define a relation schema in the DBMS, DDL need to specify:
  - R, Relation Name
  - A, Attributes
  - **D**, Domain types
  - **dom**, Mapping rules from A to D
  - F, Constraints on R

SQL 典型数据类型

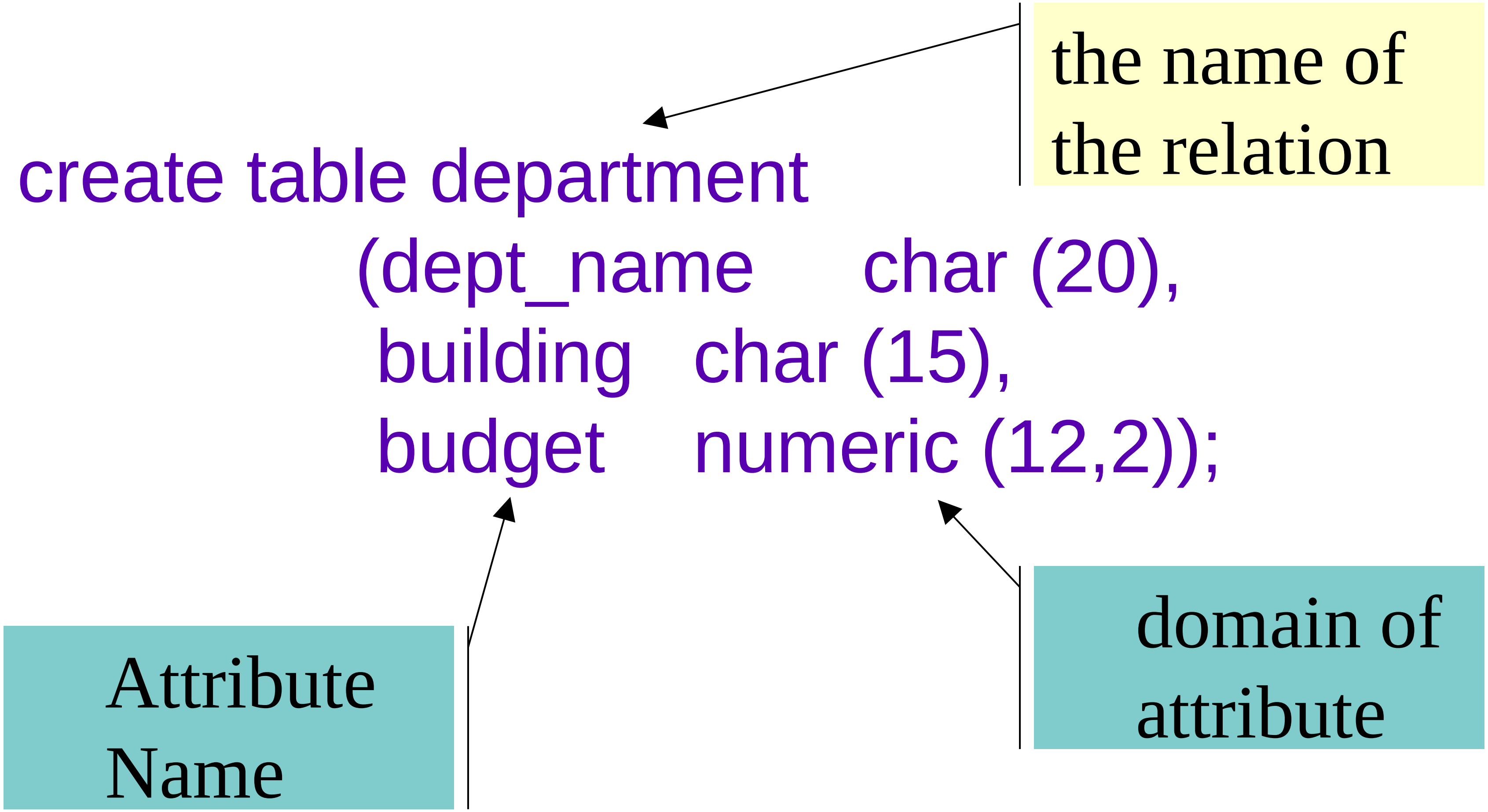
- **char(n).** Fixed length character string, with user-specified length *n.*
- **varchar(n).** Variable length character strings, with user-specified maximum length *n.*
- **int.** Integer (a finite subset of the integers that is machine-dependent).
- **smallint.** Small integer (a machine-dependent subset of the integer domain type).
- **numeric(p,d).** Fixed point number, with user-specified precision of *p* digits, with *d* digits to the right of decimal point. (ex., **numeric**(3,1), allows 44.5 to be stores exactly, ***but not 444.5 or 0.32***)
- **real, double precision.** Floating point and double-precision floating point numbers, with machine-dependent precision.
- **float(n).** Floating point number, with user-specified precision of at least *n* digits.
- More are covered in Chapter 3.

Create Table 语句

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

没有约束的简单表



the name of the relation

```
create table department
        (dept_name     char (20),
         building   char (15),
         budget     numeric (12,2));
```

Attribute Name

domain of attribute

Create Table 完整性约束

- **not null**
- **Unique or Unique***(A1, …, An)*
- **primary key** (A_1, ..., A_n)
- **foreign key** (A_m, ..., A_n) **references** *r*

```
create table instructor (
                ID              char(5),
                name            varchar(20) not null,
                dept_name  varchar(20),
                salary          numeric(8,2),
                primary key (ID),
                foreign key (dept_name) references department);
```

primary key declaration on an attribute automatically ensures **not null**

表定义举例

-
  ```
  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);
  ```

-
  ```
  create table takes (
          ID              varchar(5),
          course_id       varchar(8),
          sec_id          varchar(8),
          semester        varchar(6),
          year            numeric(4,0),
          grade           varchar(2),
          primary key (ID, course_id, sec_id, semester, year) ,
          foreign key (ID) references  student,
          foreign key (course_id, sec_id, semester, year) references section);
  ```

表定义举例(c1)

```
• create table course (
        course_id        varchar(8),
        title            varchar(50),
        dept_name        varchar(20),
        credits          numeric(2,0),
          primary key (course_id),
          foreign key (dept_name)
                references department);
```

删除、修改表定义

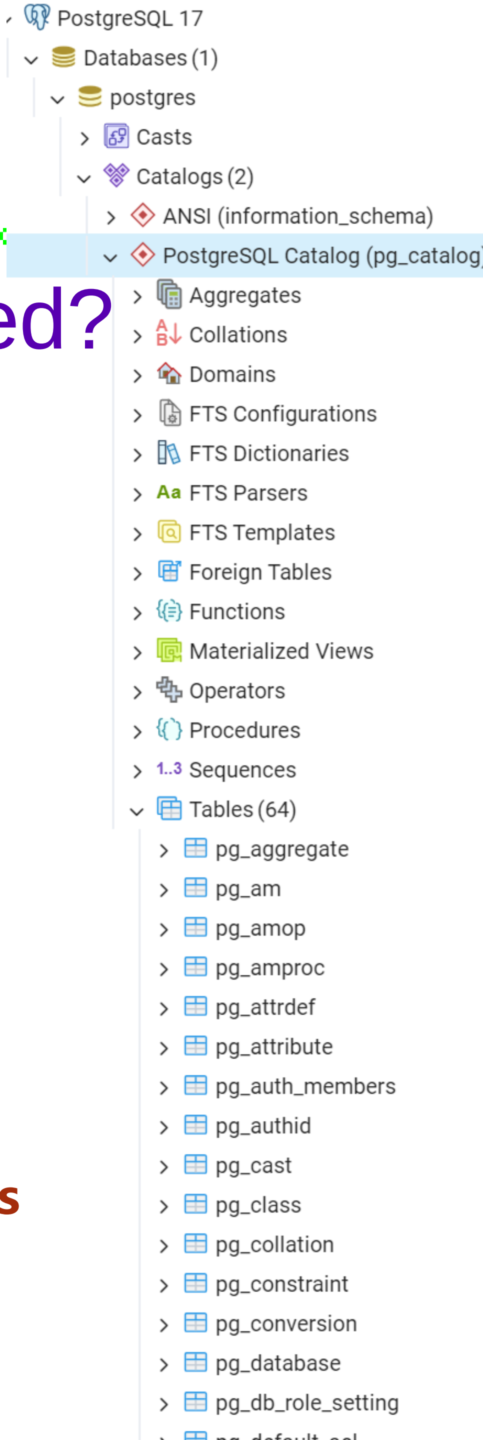
- **Drop Table**
  - **drop table** *r*
  - a more drastic action than **delete from** *r*
- **Alter**
  - **alter table** *r* **add** *A D*
    - where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A.*
    - All exiting tuples in the relation are assigned *null* as the value for the new attribute.
  - **alter table** *r* **drop** *A*
    - where *A* is the name of an attribute of relation *r*
- **Dropping of attributes not supported by many databases, Why?**

数据字典

- Where the relation schema is stored? With what format?
  - – **In Data Dictionary**
    - also called system catalogs
  - – Also with relation structure
    - also called system tables

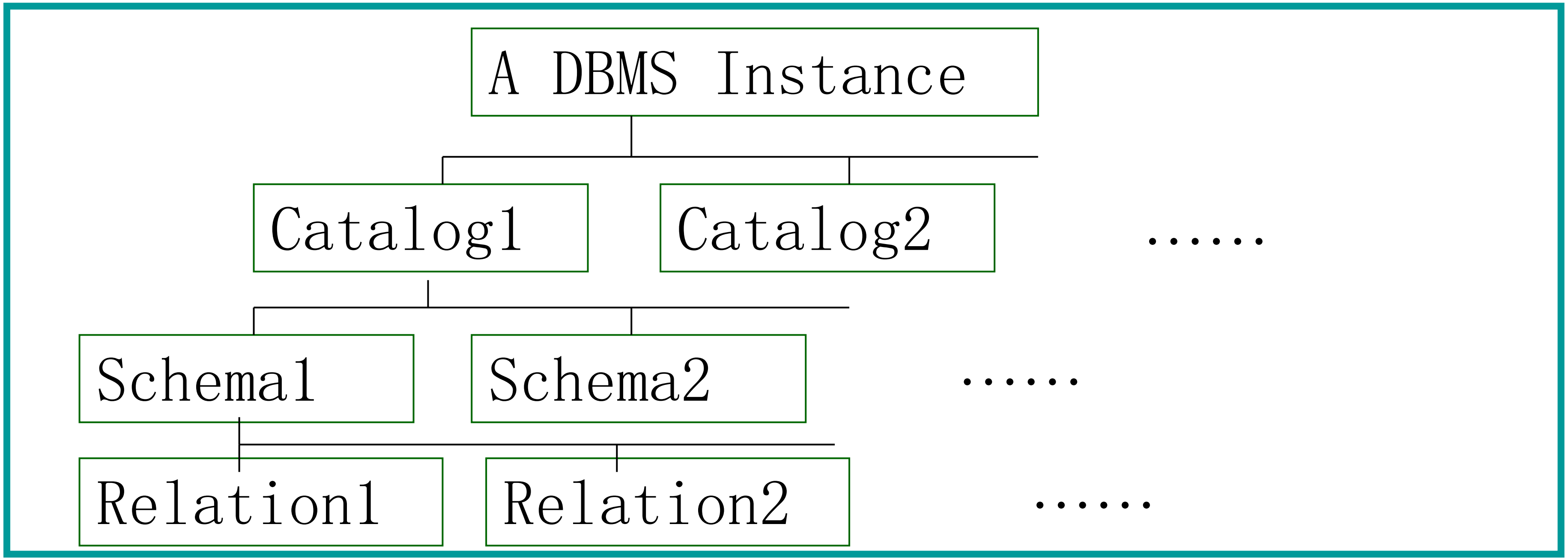
PostgreSQL 中数据库postgres的表定义放到哪里?

提示：pgAdmin 4 -> Catalogs -> pg_catalog -> Tables



- PostgreSQL 17
  - Databases (1)
    - postgres
      - Casts
      - Catalogs (2)
        - ANSI (information_schema)
        - PostgreSQL Catalog (pg_catalog)
          - Aggregates
          - Collations
          - Domains
          - FTS Configurations
          - FTS Dictionaries
          - FTS Parsers
          - FTS Templates
          - Foreign Tables
          - Functions
          - Materialized Views
          - Operators
          - Procedures
          - Sequences
          - Tables (64)
            - pg_aggregate
            - pg_am
            - pg_amop
            - pg_amproc
            - pg_attrdef
            - pg_attribute
            - pg_auth_members
            - pg_authid
            - pg_cast
            - pg_class
            - pg_collation
            - pg_constraint
            - pg_conversion
            - pg_database
            - pg_db_role_setting
            - pg_default_acl

数据库表的层次结构



- A DBMS Instance contains multiple catalogs/databases
- each catalog can contain multiple schemas（表目录）
- SQL objects such as relations and views are contained within a schema

数据库与表目录

- **Database（数据库）**
  - **Create Database**
  - **Drop Database**

PG 是怎样启动数据库的?

提示：可以通过 Service 来控制

```
path: …\PostgreSQL\17\bin
pg_ctl start -D  ..\data
pg_ctl stop -D  ..\data
```

- **Schema（表目录）**
  - **Create Schema [*myschema_name*] [Authorization *user*] …相当于Dos命令md**
  - **Drop Schema**

Main Contents

- SQL Data Definition
- **Basic Structure of SQL Queries**
- Additional Operations & Null Values
- Complex SQL
  - Set Operations
  - Aggregation
  - Nested Sub-queries
- Modification of the Database

数据操作语言-DML

- Language for **accessing and manipulating** the data organized by the appropriate data model
  - DML also known as **query language**

- Two Types of languages
  - Procedural – user specifies what data is required and how to get those data
  - Nonprocedural – user specifies what data is required without specifying how to get those data

SQL语句基本结构

Select $A_1, A_2, \ldots, A_n$
from $R_1, R_2, \ldots, R_m$
where P

A_i represent attributes
R_i represent relations
P is a predicate formula

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

SQL Reserved Words are case insensitive

SQL语句的多值集合基础

见书中80页 Note 3.1

relations r_1 and r_2 are multiset

- If there are c_1 copies of tuple t_1 in r_1, and t_1 satisfies selections $\boldsymbol{\sigma}_{\theta,}$, then there are c_1 copies of t_1 in $\sigma_\theta(r_1)$
- For each copy of tuple t_1 in r_1, there is a copy of tuple $\Pi_A(t_1)$ in $\boldsymbol{\Pi_A(r_1)}$ where $\Pi_A(t_1)$ denotes the projection of the single tuple t_1
- If there are c_1 copies of tuple t_1 in r_1 and c_2 copies of tuple t_2 in r_2, there are c_1 x c_2 copies of the tuple t_1. t_2 in r_1 x r_2

关系代数的多值集合语义

multiset relations r_1 (A, B) and r_2 (C)

r_1 = {(1, a) (2,a)} r_2 = {(2), (3), (3)}

- $\Pi_B(r_1)$ would be

  {(a), (a)}

- $\Pi_B(r_1)$ x r_2 would be

  {(a,2), (a,2), (a,3), (a,3), (a,3), (a,3)}

- $\sigma_{C=3}$ ($\Pi_B(r_1)$ x r_2)

  { (a,3), (a,3), (a,3), (a,3)}

Select b, c
from r1,r2
where c=3

Select 子句

- The **select** clause lists the attributes desired in the result of a query
  - corresponds to the projection operation of the relational algebra
- Example: find the names of all instructors:

  select *name*

  from *instructor*
- NOTE: SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
  - E.g., *Name* ≡ *NAME* ≡ *name*
  - Some people use upper case wherever we use bold font.

Select 子句(C1)

- SQL allows duplicates in relations as well as in query results
- To force the elimination of duplicates, insert the keyword **distinct** after select
- Find the department names of all instructors, and remove duplicates

  select distinct *dept_name*
  from *instructor*

- The keyword **all** specifies that duplicates should not be removed

  select all *dept_name*
  from *instructor*

Select 子句(C2)

- An asterisk in the select clause denotes “all attributes”

```
select *
from instructor
```

- An attribute can be a literal 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
```

  - **HSQL 2.5.0 does Not support this format**
- An attribute can be a literal with **from** clause

```
select 'A'
from instructor
```

  - Result is a table with one column and *N* rows (number of tuples in the *instructors* table), each row with value “A”

Select 子句(C3)

- The **select** clause can contain arithmetic expressions involving the operation, +, –, ∗, and /, and operating on constants or attributes of tuples.
  - The query:

    select *ID, name, salary/12*
    from *instructor*

    would return a relation that is the same as the *instructor* relation, except that the value of the attribute *salary* is divided by 12.
  - Can rename "*salary/12*" using the **as** clause:

    select *ID, name, salary/12* **as** *monthly_salary*
    from *instructor*

Generalized Projection in Select

From 子句

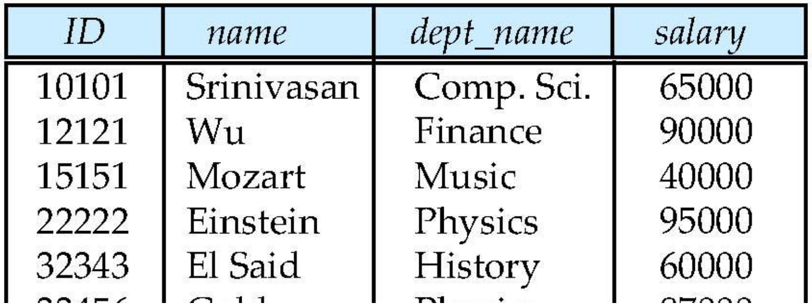
- The **from** clause lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product *instructor X teaches*

  select *
  from *instructor, teaches*

  - generates every possible instructor – teaches pair
  - Common attributes (e.g., *ID*) are renamed using the relation name (e.g., *instructor.ID*)
- Cartesian product not very useful directly, but useful combined with **where-clause**

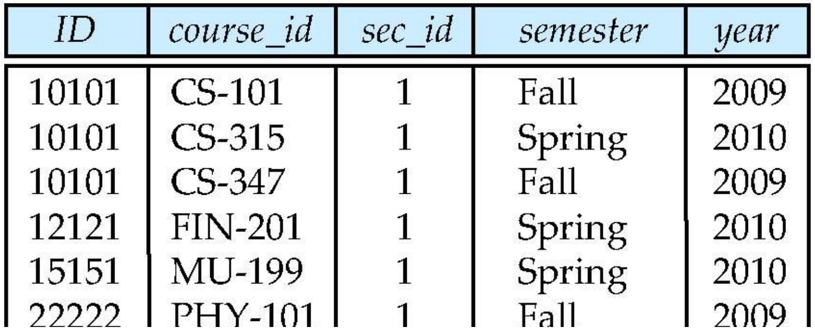
笛卡儿积运算

instructor

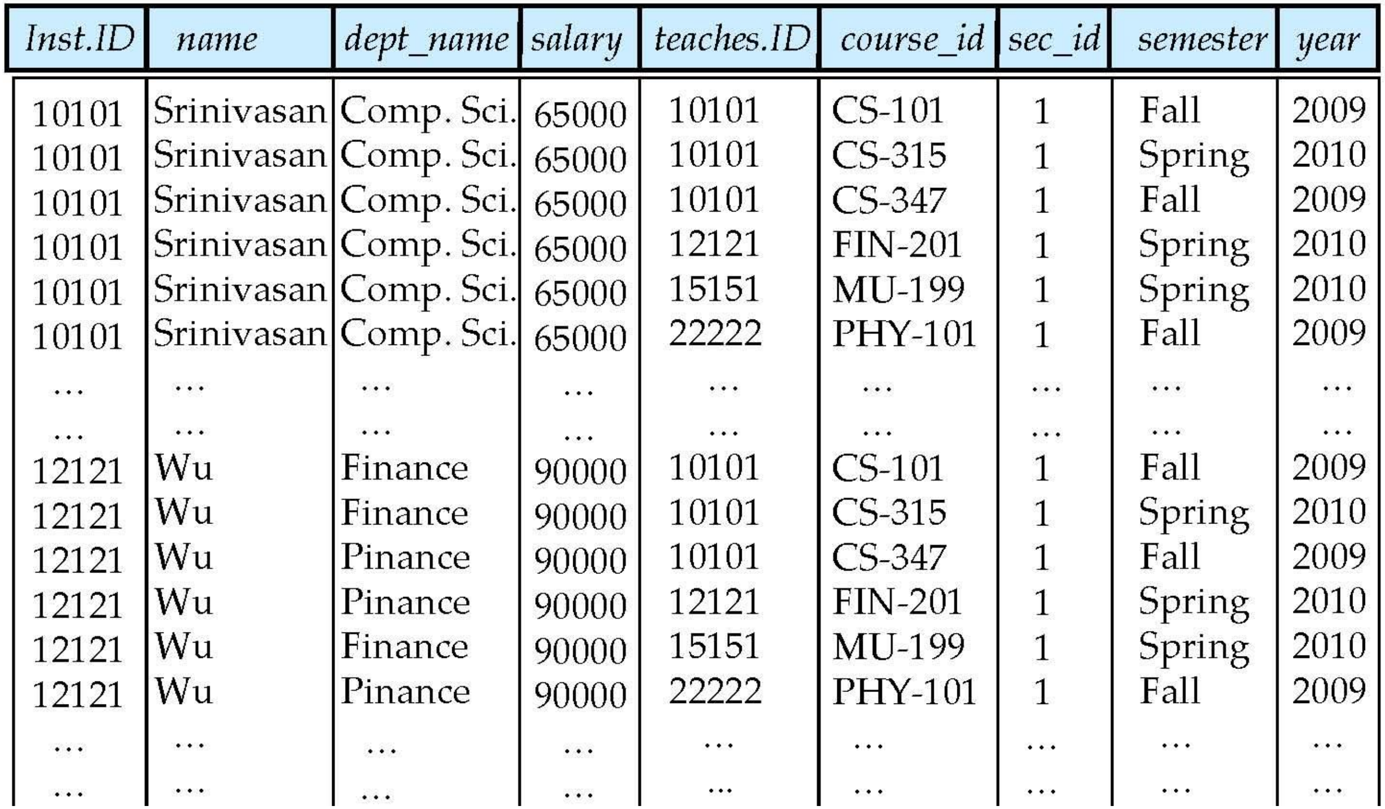


ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000

teaches



ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009



Inst.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Pinance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Pinance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Pinance	90000	22222	PHY-101	1	Fall	2009
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…

Where 子句

- The **where** clause specifies conditions that the result must satisfy
  - Corresponds to the selection predicate of the relational algebra.
- To find all instructors in Comp. Sci. dept

```
select name
from instructor
where dept_name = 'Comp. Sci.'
```

- Comparison results can be combined using the logical connectives **and, or,** and **not**
  - To find all instructors in Comp. Sci. dept with salary > 80000

```
select name
from instructor
where dept_name = 'Comp. Sci.'  and salary > 80000
```

- Comparisons can be applied to results of arithmetic expressions.

课堂练习

- Find the names of all instructors who have taught some course and the course_id

- Find the names of all instructors in the Music department who have taught some course and the course_id

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

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

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

```
select name, course_id
from instructor , teaches
where instructor.ID = teaches.ID  and  instructor. dept_name = 'Music';
```

Main Contents

- SQL Data Definition
- Basic Structure of SQL Queries
- **Additional Operations & Null Values**
- Complex SQL
  - Set Operations
  - Aggregation
  - Nested Sub-queries
- Modification of the Database

换名-The Rename Operation

- The SQL allows renaming relations and attributes using the **as** clause:

  old-name as new-name

  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.';
    ```

- Keyword **as** is optional and may be omitted

  instructor **as** *T* ≡ *instructor T*

元组变量-Tuple Variables

- Tuple variables (**元组变量**)
  - are defined in the **from** clause with the **as clause**
- Example 1

```
select customer-name, L.loan-number, L.amount
from borrower, loan as L
where  borrower.loan-number = L.loan-number
```

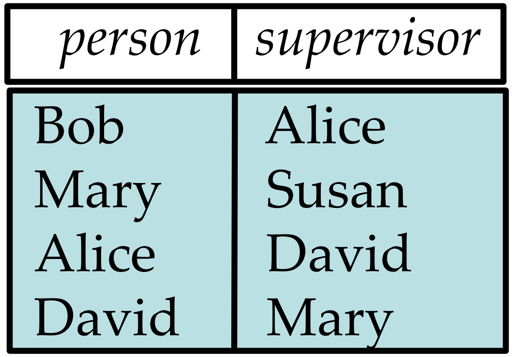
- Example 2
  - **Find the names of all branches that have greater assets than some branch located in Brooklyn**

```
select distinct T.branch-name
from branch as T, branch as S
where T.assets > S.assets and S.branch-city = 'Brooklyn';
```

Loan (loan-number, branch-name, amount)
borrower (customer-name, loan-number)
Branch (branch-name, branch-city, assets)

换名的例子

Relation *emp_super*



person	*supervisor*
Bob	Alice
Mary	Susan
Alice	David
David	Mary

Find the supervisor of “Bob”

Find the supervisor of the supervisor of “Bob”

通过PG完成练习~

Find ALL the supervisors (direct and indirect) of “Bob”

字符串运算（函数）

- 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.
- Find the names of all instructors whose name includes the substring “dar”.

```
select name
from instructor
where name like '%zar%'
```

- Match the string “100%”

```
like '100\%'  escape  '\'
```

in that above we use backslash (\) as the escape character.

字符串运算(Cont.)

- Patterns are case sensitive.
- Pattern matching examples:
  - ‘Intro%’ matches any string beginning with “Intro”.
  - ‘%Comp%’ matches any string containing “Comp” as a substring.
  - ‘_ _ _’ matches any string of exactly three characters.
  - ‘_ _ _ %’ matches any string of at least three characters.

- SQL supports a variety of string operations such as
  - concatenation (using “||”)
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, etc.

结果集中元组的顺序

- List in alphabetic order the names of all instructors

  select distinct *name*
  from *instructor*
  order by *name*

- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default.
  - Example: **order by** *name* **desc**
- Can sort on multiple attributes
  - Example: **select distinct** *dept_name*, *name*
    from *instructor*
    order by *dept_name, name*

Where 子句的条件

- SQL includes a **between** comparison operator
- Example: Find the names of all instructors with salary between $90,000 and $100,000 (that is, ≥ $90,000 and ≤ $100,000)

  -
    ```
    select name
    from instructor
    where salary between 90000 and 100000;
    ```
- Tuple comparison

  -
    ```
    select name, course_id
    from instructor, teaches
    where (instructor.ID, dept_name) =
                                       (teaches.ID, 'Biology');
    ```

Null 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 result of any arithmetic expression involving *null* is *null*
  - Example: 5 + *null* returns null
- The predicate **is null** can be used to check for null values.
  - Example: Find all instructors whose salary is null.

```
select name
from instructor
where salary is null;
```

三值逻辑

- Three values – *true*, *false*, *unknown*
- Any comparison with *null* returns *unknown*
  - Example*: 5 < null   or   null <> null   or   null = null*
- Three-valued logic using the 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 is treated as *false* if it evaluates to *unknown*

课堂练习

判断正误

- *5 + null returns false* (X)
- *5 < null returns unknown* (√)
- *5 < null returns true* (X)
- The % character matches any substring (√)
- ‘Intro\%’ matches any string beginning with ‘Intro’ (X)

Main Contents

- SQL Data Definition
- Basic Structure of SQL Queries
- Additional Operations & Null Values
- **Complex SQL**
  - **Set Operations**
  - Aggregation
  - Nested Sub-queries
- Modification of the Database

集合操作

- **union, intersect,** and **except**
  - correspond to the relational algebra operations $\cup$, $\cap$, –
  - **Each of the above operations automatically eliminates duplicates**

- **union all, intersect all** and **except all**
  - Corresponding to multiset versions
  - Suppose a tuple occurs *m* times in *r* and *n* times in *s*
    - *m* + *n* times in *r* **union all** *s*
    - min(*m*,*n)* times in *r* **intersect all** *s*
    - max(0, *m* – *n)* times in *r* **except all** *s*

见教材97页 Note3.2

集合操作举例

- **Find courses that ran in Fall 2017 or in Spring 2018**

(**select** *course_id* **from** *section* **where** *semester* = 'Fall' **and** *year* = 2017)
union
(**select** *course_id* **from** *section* **where** *semester* = 'Spring' **and** *year* = 2018)

Find courses that ran in Fall 2017 and in Spring 2018

(**select** *course_id* **from** *section* **where** *semester* = 'Fall' **and** *year* = 2017)
intersect
(**select** *course_id* **from** *section* **where** *semester* = 'Spring' **and** *year* = 2018)

Find courses that ran in Fall 2017 but not in Spring 2018

(**select** *course_id* **from** *section* **where** *semester* = 'Fall' **and** *year* = 2017)
except
(**select** *course_id* **from** *section* **where** *semester* = 'Spring' **and** *year* = 2018)

找到最高薪水

- 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*

- Find all the salaries of all instructors
  - **select distinct** *salary*
    from *instructor*

- Find the largest salary of all instructors.
  - second query
    except
    (first query)

Main Contents

- SQL Data Definition
- Basic Structure of SQL Queries
- Additional Operations & Null Values
- **Complex SQL**
  - Set Operations
  - **Aggregation**
  - Nested Sub-queries
- Modification of the Database

聚合函数-Aggregate Functions

- These functions operate on the multiset of values of a column of a relation, and return a value

avg: average value
min: minimum value
max: maximum value
sum: sum of values
count: number of values

聚合函数(Cont.)

- Find the average salary of instructors in the Computer Science department

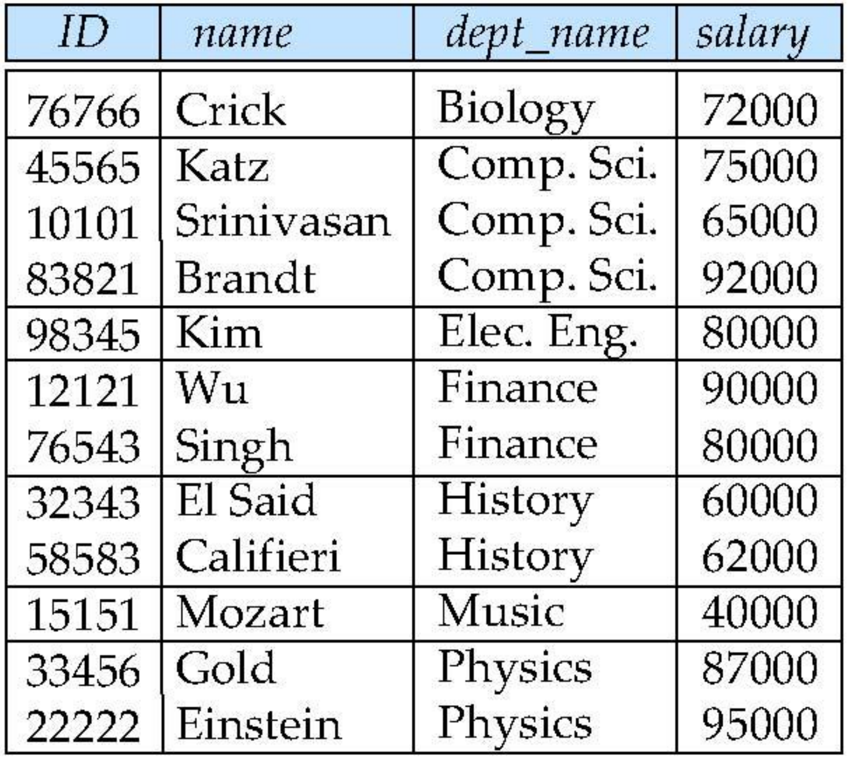
  -
    ```
    select avg (salary)
    from instructor
    where dept_name= 'Comp. Sci.';
    ```
- Find the total number of instructors who teach a course in the Spring 2018 semester

  -
    ```
    select count (distinct ID)
    from teaches
    where semester = 'Spring' and year = 2018;
    ```
- Find the number of tuples in the *course* relation

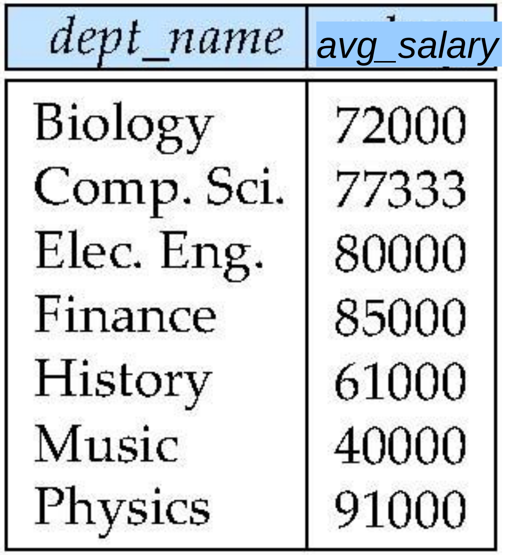
  -
    ```
    select count (*)
    from course;
    ```

Group By 子句

- Find the average salary of instructors in each department
  - **select** *dept_name*, **avg** (*salary*) **as** *avg_salary*
    from *instructor*
    group by *dept_name*;



ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000



dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

带有Group by子句的结果集模式

- Attributes in **select** clause outside of aggregate functions must appear in **group by** list

  -
    ```
    /* erroneous query */
    select dept_name, ID, avg (salary)
    from instructor
    group by dept_name;
    ```

Having 子句

- Find the names and average salaries of all departments whose average salary is greater than 82000

```
select dept_name, avg (salary)
from instructor
where salary > 80000
group by dept_name
having avg (salary) > 82000;
```

Note: predicates in the having clause are applied after the formation of groups whereas predicates in the where clause are applied before forming groups

聚合操作中Null值的处理

- Total all salaries

  select sum (*salary*)
  from *instructor*

  - Above statement ignores null amounts
  - Result is *null* if there is no non-null amount
- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes
- What if collection has only null values?
  - count returns 0
  - all other aggregates return null

Main Contents

- SQL Data Definition
- Basic Structure of SQL Queries
- Additional Operations & Null Values
- **Complex SQL**
  - Set Operations
  - Aggregation
  - **Nested Sub-queries**
- Modification of the Database

子查询-Nested Subqueries

- SQL provides a mechanism for the nesting of subqueries. A **subquery** is a **select-from-where** expression that is nested within another query.
- The nesting can be done in the following SQL query

  select $A_1, A_2, ..., A_n$
  from $r_1, r_2, ..., r_m$
  where P

-

  as follows:
  - A_i can be replaced be a subquery that generates a single value.
  - r_i can be replaced by any valid subquery
  - P can be replaced with an expression of the form:

    B <operation> (subquery)

  Where *B* is an attribute and <operation> to be defined later.

Subqueries in the Where Clause

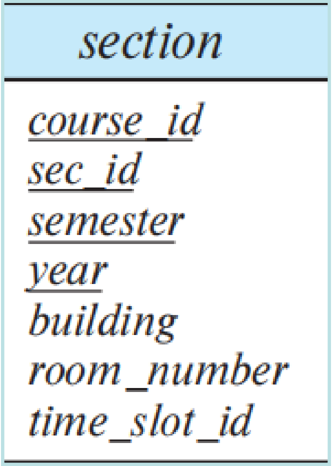
Where子句中的子查询

- A common use of subqueries is to perform tests:
  - For set membership
  - For set comparisons
  - For set cardinality.

集合元素判断-Set Membership

- Find courses offered in Fall 2017 and in Spring 2018

```
select distinct course_id
from section
where semester = 'Fall' and year= 2017 and
          course_id in (select course_id
                            from section
                            where semester = 'Spring'
and year= 2018);
```

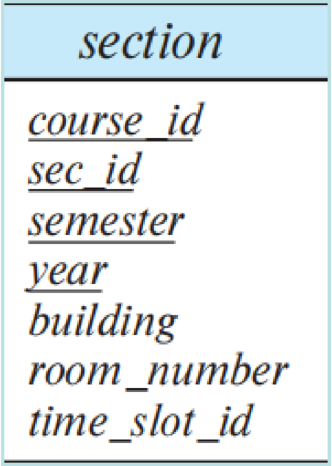


section

course_id
sec_id
semester
year
building
room_number
time_slot_id

集合元素判断(c1)

Find courses offered in Fall 2017 but not in Spring 2018



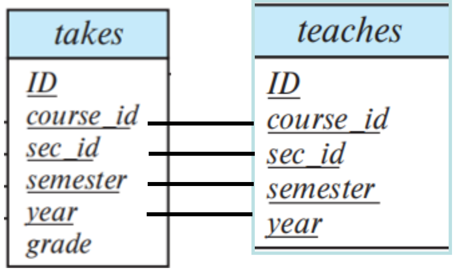
section

course_id
sec_id
semester
year
building
room_number
time_slot_id

```
select distinct course_id
from section
where semester = 'Fall' and year= 2017 and
        course_id  not in (select course_id
                            from section
                          where semester = 'Spring' and year= 2018);
```

集合元素判断(c2)

- Find the total number of (distinct) students who have taken course sections taught by the instructor with *ID* 10101



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

Note: Above query can be written in a much simpler manner.
The formulation above is simply to illustrate SQL features.

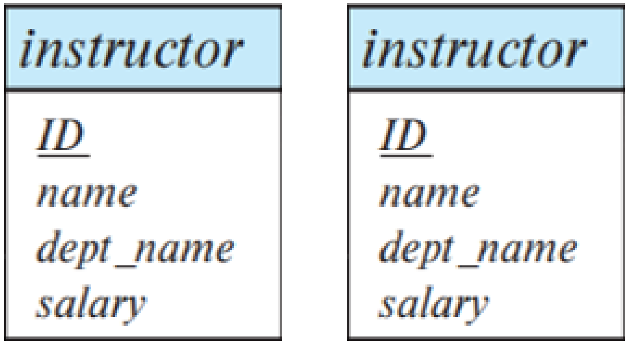
“Some” 子句 -集合比较

- 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';
```

Same query using > **some** clause

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



instructor
ID
name
dept_name
salary

instructor
ID
name
dept_name
salary

“Some” 子句

F <comp> some *r* ⇔ ∃ *t* ∈ *r* s.t. (F <comp> *t*)

Where <comp> can be: <, ≤, >, =, ≠

$$\left(5 < \text{some} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}\right) = \text{true}$$

$$\left(5 < \text{some} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline \end{array}\right) = \text{false}$$

$$\left(5 = \text{some} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline \end{array}\right) = \text{true}$$

$$\left(5 \neq \text{some} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline \end{array}\right) = \text{true (since } 0 \neq 5\text{)}$$

(= some)	≡	**in**
(≠ some)	$\not\equiv$	**not in**

“all” 子句

F <comp> all r $\Leftrightarrow \forall$ t $\in$ r (F <comp> t)

$(5 < \text{all} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{false}$

$(5 < \text{all} \begin{array}{|c|}\hline 6 \\ \hline 10 \\ \hline \end{array}) = \text{true}$

$(5 = \text{all} \begin{array}{|c|}\hline 4 \\ \hline 5 \\ \hline \end{array}) = \text{false}$

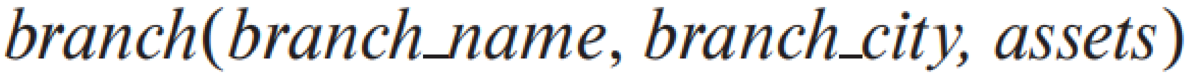
$(5 \neq \text{all} \begin{array}{|c|}\hline 4 \\ \hline 6 \\ \hline \end{array}) = \text{true (since } 5 \neq 4 \text{ and } 5 \neq 6)$

($\neq$ all)	$\equiv$	**not in**
(= all)	$\not\equiv$	**in**

查询举例

- Find the names of all branches that have greater assets than all branches located in Brooklyn

```
select branch-name
from branch
where assets > all
    (select assets from branch
    where branch-city = 'Brooklyn’);
```



branch(*branch_name*, *branch_city*, *assets*)

Exits子句：空关系测试

- The **exists** construct returns the value **true** if the argument subquery is nonempty.
- **exists** $r \Leftrightarrow r \neq \varnothing$
- **not exists** $r \Leftrightarrow r = \varnothing$

“exists” 子句

- Yet another way of specifying the query “Find all courses taught in both the Fall 2017 semester and in the Spring 2018 semester”

```
select course_id
from section as S
where semester = 'Fall' and year = 2017 and
        exists (select *
                from section as T
                where semester = 'Spring' and year= 2018
                        and S.course_id = T.course_id);
```

- **Correlation name** – variable S in the outer query
- **Correlated subquery** – the inner query

“not exists” 子句

- 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 $X – Y = Ø \quad \Leftrightarrow \quad X \subseteq Y$

Note: Cannot write this query using = **all** and its variants

Another Example of Division Operator

Table PilotSkills is about the pilots and the planes they can fly (dividend), table Hangar is about planes in the hangar (divisor), we want the names of the pilots who can fly every plane (quotient) in the hangar.

PilotSkills

pilot	plane
'Celko'	‘C919'
'Higgins'	‘H-6N Bomber'
'Higgins'	‘J-35 Fighter'
'Higgins'	‘C919'
'Jones'	‘H-6N Bomber'
'Jones'	‘J-35 Fighter'
'Smith'	‘H-6K Bomber'
'Smith'	‘H-6N Bomber'
'Smith'	‘J-35 Fighter'
'Wilson'	‘H-6K Bomber'
'Wilson'	‘H-6N Bomber'
'Wilson'	‘J-35 Fighter'
'Wilson'	‘J-20 Fighter'

Hangar

plane
‘H-6K Bomber'
‘H-6N Bomber'
‘J-35 Fighter'

Unique子句：重复元组测试

- The **unique** construct tests whether a subquery has any duplicate tuples in its result.
  - This construct is not yet widely implemented

- The **unique** construct evaluates to “true” if a given subquery contains no duplicates .

- 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);
```

Subqueries in the From Clause

From 子句中的子查询

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

```
select dept_name, avg(salary)
 from instructor
 group by dept_name
 having avg(salary) > 42000;
```

- **Note that we do not need to use the having clause**
- Another way to write above query

```
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;
```

lateral 子句

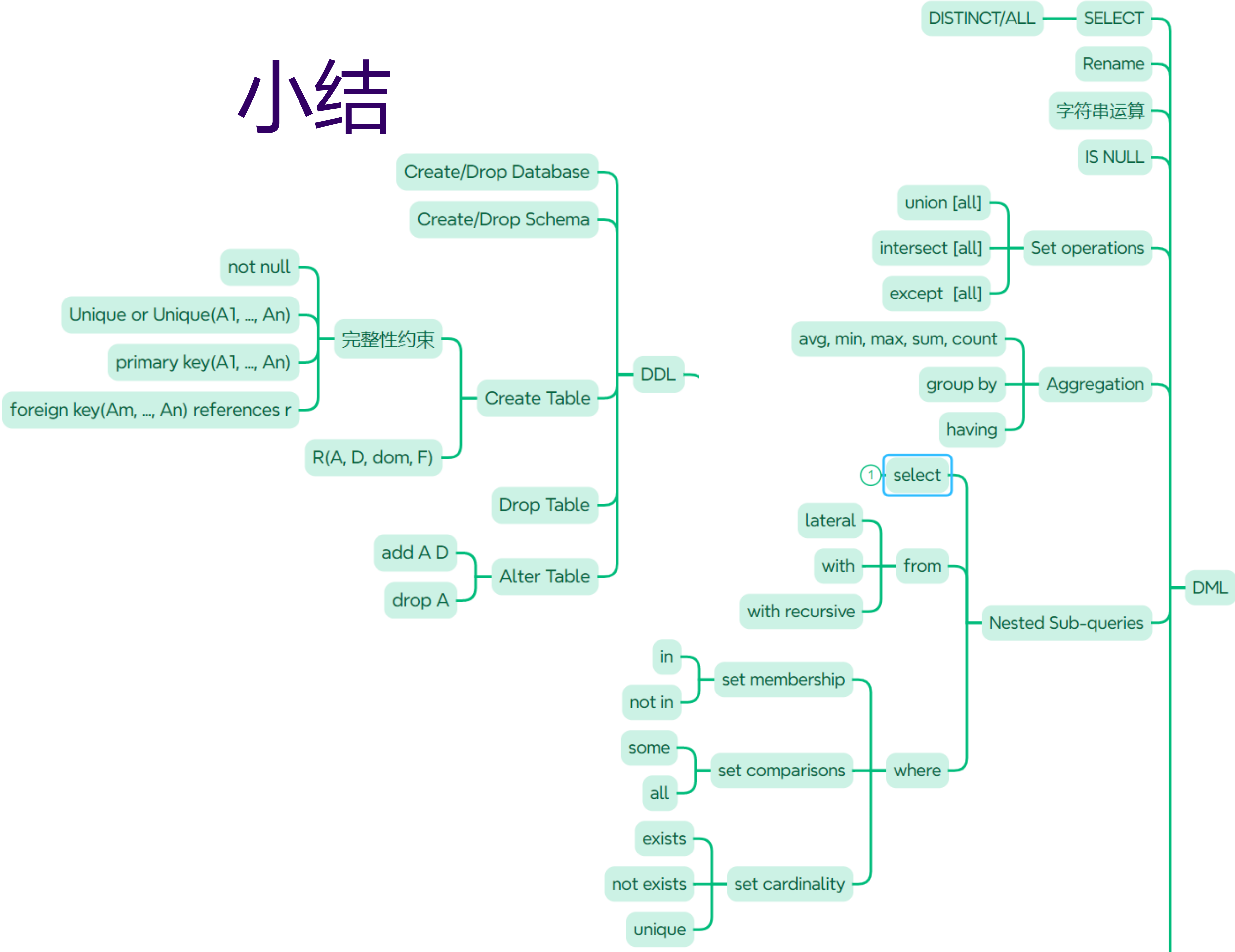
- **print the names of each instructor, along with their salary and the average salary in their department**
- E.g.

```
select name, salary, avg_salary
from instructor I1, lateral (select avg(salary) as avg_salary
                             from instructor I2
                             where I2.dept_name= I1.dept_name);
```

- Without the **lateral** clause, the subquery cannot access the correlation variable *I1* from the outer query.

- **Lateral is firstly introduced in SQL:2003.** Currently, only a few SQL implementations, such as **IBM DB2, PostgreSQL,** support the **lateral** clause.

小结



- DDL
  - Create/Drop Database
  - Create/Drop Schema
  - Create Table
    - 完整性约束
      - not null
      - Unique or Unique(A1, ..., An)
      - primary key(A1, ..., An)
      - foreign key(Am, ..., An) references r
    - R(A, D, dom, F)
  - Drop Table
  - Alter Table
    - add A D
    - drop A
- DML
  - SELECT
    - DISTINCT/ALL
  - Rename
  - 字符串运算
  - IS NULL
  - Set operations
    - union [all]
    - intersect [all]
    - except [all]
  - Aggregation
    - avg, min, max, sum, count
    - group by
    - having
  - Nested Sub-queries
    - select
    - from
      - lateral
      - with
      - with recursive
    - where
      - set membership
        - in
        - not in
      - set comparisons
        - some
        - all
      - set cardinality
        - exists
        - not exists
        - unique

谢谢!