Chapter 3: Introduction to SQL

Database System Concepts, 6th Ed.

Outline

- Overview of The SQL Query Language
- Data Definition
- Basic Query Structure
- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions
- Nested Subqueries
- Modification of the Database

3.1 SQL查询语言概览

- History
  - IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
  - Renamed Structured Query Language (SQL)
  - ANSI and ISO standard SQL:
    - SQL-86
    - SQL-89
    - SQL-92
    - SQL:1999 (language name became Y2K compliant!)
    - SQL:2003
  - Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.

SQL语言有以下几个部分：

- 数据定义语言(Data- Definition Language， DDL)：SQL DDL提供定义关系模式、删除关系以及修改关系模式的命令。
- 数据操纵语言(Data- Manipulation Language， DML)：SQL DML提供从数据库中查询信息，以及在数据库中插入元组、删除元组、修改元组的能力。

- 完整性(integrity)：SQL DDL包括定义完整性约束的命令，保存在数据库中的数据必须满足所定义的完整性约束。破坏完整性约束的更新是不允许的。
- 视图定义(view definition)：SQL DDL包括定义视图的命令。
- 事务控制(transaction control)：SQL包括定义事务的开始和结束的命令。

- 嵌入式SQL和动态SQL(embedded SQL and dynamic SQL)：嵌入式和动态SQL定义SQL语句如何嵌入到通用编程语言，如C、C++和Java中。
- 授权(authomation)：SQL DDL包括定义对关系和视图的访问权限的命令。

3.2 SQL Data Definition Language

- 数据库中的关系集合必须由数据定义语言(DDL)指定给系统。
- The SQL data-definition language (DDL) allows the specification of information about relations, including:
  - The schema for each relation.
  - The domain of values associated with each attribute.
  - Integrity constraints

- also other information such as
  - The set of indices to be maintained for each relations.
  - Security and authorization information for each relation.
  - The physical storage structure of each relation on disk.

3.2.1 Domain Types in SQL

- **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.

- 空值Null表示一个缺失的值，该值可能存在但并不为人所知，或者可能根本不存在。
- char数据类型存放固定长度的字符串。
  - 例如，属性A的类型是char(10)。
    - 如果我们为此属性存入字符串“Avi”，那么该字符串后会追加7个空格来使其达到10个字符的串长度。
    - varchar(10)，则不会增加空格。
- 比较一个char类型和一个varchar类型的时候，如果长度不同，比较结果将是假。

- 议始终使用varchar类型而不是char类型来避免这样的问题。

3.2.2 Create Table Construct

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

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

  - 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

- Example:

```
create table instructor (
        ID          char(5),
        name        varchar(20),
        dept_name   varchar(20),
        salary      numeric(8,2))
```

Integrity Constraints in Create Table

- **not null**
- **primary key** (A_1, ..., A_n)
- **foreign key** (A_m, ..., A_n) **references** *r*
- **primary key** declaration on an attribute automatically ensures **not null**

- *Example:*
- **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);
```

- 书P35页

And a Few More Relation Definitions

- 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);
```

- Note: *sec_id* can be dropped from primary key above, to ensure a student cannot be registered for two sections of the same course in the same semester

Updates to tables

- **Insert**
  - **insert into** *instructor* **values** (‘10211’, ’Smith’, ’Biology’, 66000);
- **Delete**
  - Remove all tuples from the *student* relation
    - **delete from** *student*
- **Drop Table**
  - **drop table** *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.

3.3 Basic Query Structure

- 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_i represents an attribute
  - R_i represents a relation
  - P is a predicate.
- The result of an SQL query is a relation.

The select Clause

- 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.

The select Clause (Cont.)

- 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
```

The select Clause (Cont.)

- 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*

- 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”

The select Clause (Cont.)

- 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
    ```

The where Clause

- 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.
- 逻辑连词的运算对象可以是包含比较运算符<、<=、>、>=、=和<>的表达式。

3.3.2 The from Clause

- 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
```

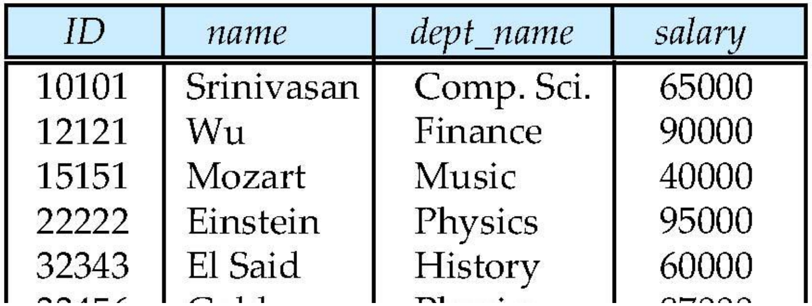
  - generates every possible instructor – teaches pair, with all attributes from both relations.

  - For common attributes (e.g., *ID*), the attributes in the resulting table are renamed using the relation name (e.g., *instructor.ID*)

- Cartesian product not very useful directly, but useful combined with ***where-clause*** condition (selection operation in relational algebra).

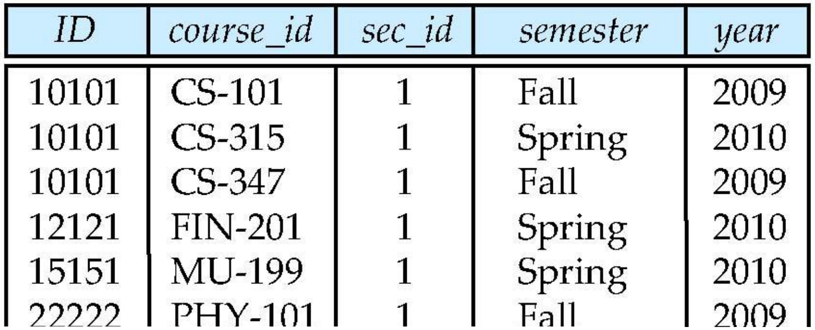
Cartesian Product

instructor

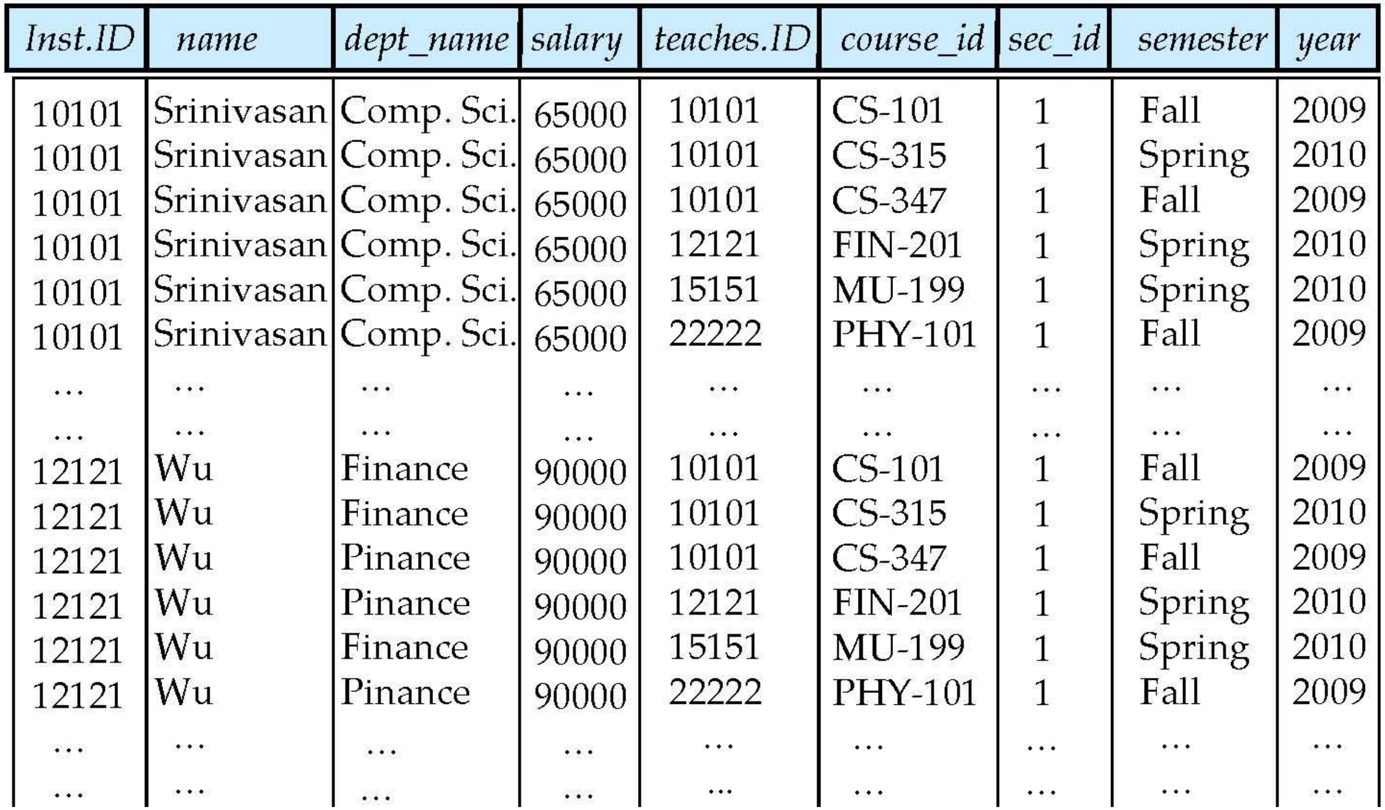


ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000

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
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Examples

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

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

3.3.3 自然连接

- 自然连接(natural join）运算作用于两个关系，并产生一个关系作为结果。
- 不同于两个关系上的笛卡儿积，
  - 笛卡儿积将第一个关系的每个元组与第二个关系的所有元组都进行连接；
  - 自然连接只考虑那些在两个关系模式中都出现的属性上取值相同的元组对。

- 自然连接没有重复列出那些在两个关系模式中都出现的属性，这样的属性只出现一次。
- 自然连接列出属性的顺序：
  - 先是两个关系模式中的共同属性，
  - 然后是那些只出现在第一个关系模式中的属性，
  - 最后是那些只出现在第二个关系模式中的属性。

●例如：instructor关系和teaches关系的自然连接

n **select** name, course_id

n **from** instructor， teaches

n **where** instruclor.ID=teaches.ID:

●该查询可以用SQL的自然连接运算更简洁地写作：

n **select** name, course_id

n **from** instructor **natrual join** teaches;

●以上两个查询产生相同的结果。

- 在一个SQL查询的from子句中，可以用自然连按将多个关系结合在一起，如下所示：
- **select** A,, Az,…. A.
- **from** rl **natural join** r2 **natural join**…**natural join** rm
- **where** P;
- 更为一般地，from子句可以为如下形式：
  - **from** El, E2,…, E。
  - 其中每个E可以是单个关系或一个包含自然连接的表达式。

- 例如：查询“列出教师的名字以及他们所讲授课程的名称”
  - **select** name, title
  - **from** instructor **natural join** teaches, course
  - **where** teaches.course id= course.course id;
- 注意：下面的SQL查询不会计算出相同的结果
  :
  - **select** name, title
  - **from** instructor **natural join** teaches **natural join** course;

- 原因是：
  - instructor和teaches的自然连接包括属性（ID，name，*dept_name*，salary．*course_id*，sec_id），
  - 而course关系包含的属性是（*course_id*，title，*dept_name*，credits）。
  - 二者自然连接的结果，是需要来自这两个输入的元组既要在属性dept_name上取值相同，还要在course_id上取值相同。

- SQL提供了一种自然连接的构造形式，允许用户来指定需要哪些列相等。
  - **select** name, title
  - **from** (instructor **natural join** teaches) **join** course **using** (course_id);
- **join... using**运算中需要给定一个属性名列表，其中，两个输入中都必须具有指定名称的属性。

- 运算r1 **join** r2 **using**(A1，A2)，
- 它与r1和r2的自然连接类似，只不过
  - r1.Al= t2.A1, 并且t1.A2= t2.A2
- 即使r和r2都具有名为A3的属性，也不需要r1．A3=r2．A3成立。

3.4 附加的基本运算
3.4.1 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*

- SQL提供了一个重命名*结果关系*中属性的方法
  - from子句的两个关系中可能存在同名属性，在这种情况下，结果中就会出现重复的属性名；
  - 其次，如果我们在select子句中使用算术表达式，那么结果属性就没有名字；
  - 再次，属性名可以从基关系导出，但我们也许想要改变结果中的属性名字。

- 在from子句中，重命名关系的原因
  - 是把一个长的关系名替换成短的，这样，在查询的其他地方使用起来就更为方便
  - 另一个原因是为了适用于需要比较*同一个关系*中的元组的情况。
- 像T和S那样，被用来重命名关系的**标识符**在SQL标准中被称作**相关名称(correlation name)**，**表别名(table alias)**，或者**相关变量（correlation variable）**，或者**元组变量(tuple variable)**。

3.4.2 String Operations

- SQL使用一对单引号来标示字符串
- 如果单引号是字符串的组成部分，那就用两个单引号字符来表示
- 在SQL标准中，字符串上的**相等**运算是区分大小写的
- 然而，一些数据库管理系统，如MySQL和SQL Server，在匹配字符串时并不区分大小写
- 这种默认方式，可以在数据库级或特定属性级被修改的。

- 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** '%dar%'
- Match the string “100%”

  like ‘100 \%’ **escape** ‘\’

  in that above we use **backslash** (\) as the **escape** character.
- escape character，转义字符直接放在特殊字符前面，表示该特殊字符被当成普通字符。

String Operations (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)，**upper（s）**，**lower（s）**
  - 去掉字符串后面的空格(使用**trim(s)**)
  - finding string length,
  - extracting substrings, etc.

3.4.3 select子句中的属性说明

- 星号“*”可以用在select子句中，表示“所有的属性”,
- 因而，如下查询的select子句中使用instructor.*
  - select instructor.*
  - from instructor，teaches
  - where instructor. ID= teaches.ID；
- 表示instructor中的所有属性都被选中。

3.4.4 Ordering the Display of Tuples

- **order by**子句就可以让查询结果中元组按排列顺序显示
- 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:
  - order by dept_name, name

3.4.5 Where Clause Predicates

- SQL includes a **between...and...** comparison operator
- Example:
  - Find the names of all instructors with salary between $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’);
- SQL允许我们用记号(v1, v2,...,vn)来表示一个分量值分别为 v1, v2,...,vn的n维元组。
- 在元组上可以运用比较运算符，按**字典顺序**进行比较运算。

Duplicates

- In relations with duplicates, SQL can define how many copies of tuples appear in the result.
- **Multiset（多重集合）** versions of some of the relational algebra operators – given multiset relations r1 and r2:
  - 1. σ_θ (r1): If there are c1 copies of tuple t1 in r1, and t1 satisfies selections σ_θ, then there are c1 copies of t1 in σ_θ (r1).
  - 2. Π_A (r1): For each copy of tuple t1 in r1, there is a copy of tuple Π_A (t1) in Π_A (r1)， where Π_A (t1) denotes the projection of the single tuple t1.

- 3. r1 x r2: If there are c1 copies of tuple t1 in r1 and c2 copies of tuple t2 in r2, there are c1 x c2 copies of the tuple t1. t2 in r1 x r2

- Example: Suppose multiset relations r_1 (A, B) and r_2 (C) are as follows:

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

- Then $\Pi_B(r_1)$ would be {(a), (a)}, while $\Pi_B(r_1)$ x r_2 would be

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

Duplicates (Cont.)

- SQL duplicate semantics:

  select $A_{1,}$, A_2, ..., A_n
  from r_1, r_2, ..., r_m
  where P

  is equivalent to the *multiset* version of the expression:

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

3.5 Set Operations
3.5.1~3.5.3 并、交、差运算

- SQL作用在关系上的union、intersect和except运算对应于数学集合论中的∪, ∩, 和−运算。

- Find courses that ran 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)

- Find courses that ran in Fall 2009 and in Spring 2010

- (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
  intersect
  (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

- Find courses that ran in Fall 2009 but not in Spring 2010
- (**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
  except
  (**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)
- union、intersect和except运算自动去除重复
- 如果想保留所有重复，就必须用union all、intersect all和except all代替union、intersect和except

Set Operations (Cont.)

- Suppose a tuple occurs m times in r and n times in s, then, it occurs:
  - 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

3.6 Null Values

- 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**
- SQL将涉及空值的任何比较运算的结果视为**unknown**
  - 既不是谓词**is null**，也不是**is not null**

Null Values and Three Valued Logic

- **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*
- 某些SQL，还允许我们使用子句**is unknown**和**is not unknown**来测试一个表达式的结果是否为unknown，而不是true或false。

- 当一个查询使用**select distinct**子句时，重复元组将被去除。
- 为了达到这个目的，当比较两个元组对应的属性值时，
  - 如果这两个值都是非空并且值相等，或者都是空，那么它们是相同的。
- 这里，对待空值的方式与谓词中对待空值的方式是不同的，
  - 在谓词中“null= null”会返回unknown，而不是true。

- 如果元组在所有属性上的取值相等，那么它们就被当作相同元组，即使某些值为空。
- 上述方式还应用于集合的并、交和差运算

3.7 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

3.7.1 Aggregate Functions (Cont.)

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

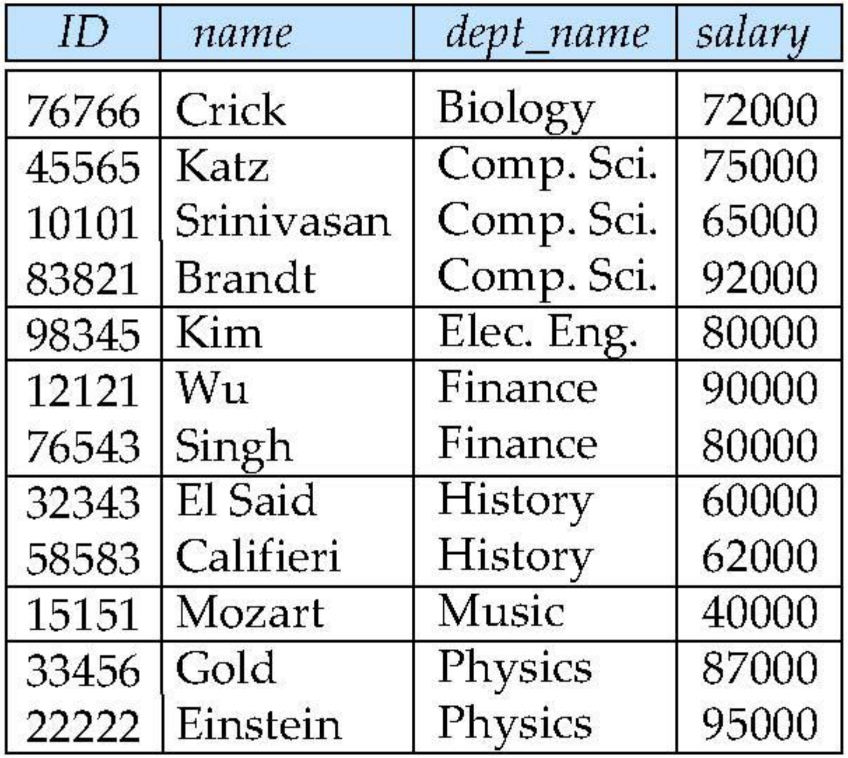
  -
    ```
    select avg (salary)
    from instructor
    where dept_name= ’Comp. Sci.’;
    ```
- Find the total number of instructors who teach a course in the Spring 2010 semester

  -
    ```
    select count (distinct ID)
    from teaches
    where semester = ’Spring’ and year =
    2010;
    ```

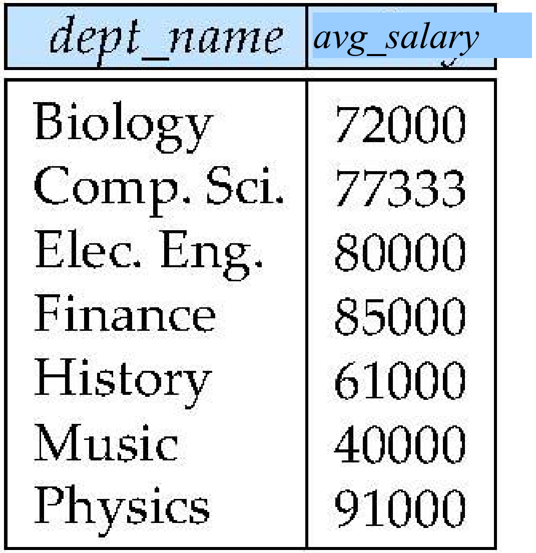
- Find the number of tuples in the course relation
  - select **count (*)**
    from course;
- SQL不允许在用count(*)时使用distinct。
- 在用max和min时使用distinct是合法的，尽管结果并无差别。

3.7.2 Aggregate Functions – Group By

- 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

Aggregation (Cont.)

- 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;
```

3.7.3 Having Clause

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

- 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，否则查询就被当成是错误的。
- 包含聚集、group by或having子句的查询的含义可通过下述操作序列来定义：
  1. 最先根据from子句来计算出一个关系。
  2. 如果出现了where子句，where子句中的谓词将应用到from子句的结果关系上。
  3. 如果出现了group by子句，满足where谓词的元组通过group by子句形成分组。如果没有group by子句，满足where谓词的整个元组集被当作一个分组。

4. 如果出现了having子句，它将应用到每个分组上；不满足having子句谓词的分组将被抛弃。
5. select子句利用剩下的分组产生出查询结果中的元组，即在每个分组上应用聚集函数来得到单个结果元组。

- select *course_id*, *semester*, *year*, *sec_id*, avg (*tot_cred*)
- from *takes* natural join *student*
- where *year* = 2009
- group by *course_id*, *semester*, *year*, *sec_id*
- having count (*ID*) >= 2;

3.7.4 Null Values and Aggregates

- 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**

3.8 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

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

3.8.1 Set Membership

- SQL允许测试元组在关系中的成员资格。
  - 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。
  - 连接词**not in**则测试元组是否不是集合中的成员。
- Find courses offered in Fall 2009 and in Spring 2010
  - **select distinct** *course_id*
  - **from** *section*
  - **where** *semester* = ’Fall’ **and** *year*= 2009 **and**
    course_id **in** (**select** *course_id*
  - **from** *section*
  - **where** *semester* = ’Spring’ **and**
    year= 2010);

- Find courses offered in Fall 2009 but not in Spring 2010
  - **select distinct** *course_id*
  - **from** *section*
  - **where** *semester* = ’Fall’ **and** *year*= 2009 **and** *course_id* **not in** (**select** *course_id*
  -   **from** *section*
  -   **where** *semester* = ’Spring’ **and** *year*= 2010);

Set Membership (Cont.)

- 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.

- in和not in操作符也能用于枚举集合。
  - **select** distinct name
  - **from** instructor
  - **where** name **not in** ('Mozart', 'Einstein');

3.8.2 Set Comparison – “some” Clause

- 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’);

Definition of "some" Clause

- F <comp> **some** $r \Leftrightarrow \exists\, t \in r$ such that (F <comp> t)
  Where <comp> can be: $<$, $\leq$, $>$, $=$, $\neq$

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

(read: 5 < some tuple in the relation)

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

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

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

(= **some**) ≡ **in**

However, (≠ **some**) $\not\equiv$ **not in**

Set Comparison – “all” Clause

- Find the names of all instructors whose salary is greater than the salary of all instructors in the Biology department.

```
select name
from instructor
where salary > all (select salary
                    from instructor
                    where dept name =
’Biology’);
```

Definition of “all” Clause

- F <comp> **all** $r \Leftrightarrow \forall\ t \in r\ \ (\text{F <comp> } t)$

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

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

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

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

(≠ **all**) ≡ **not in**

However, (**= all**) ≡ **i̸n**

3.8.3 Test for Empty Relations

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

Use of "exists" Clause

- Yet another way of specifying the query "Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester"

```
select course_id
from section as S
where semester = 'Fall' and year = 2009 and
          exists (select *
                        from section as T
                        where semester = 'Spring' and
year= 2010
                                    and S.course_id =
T.course_id);
```

- SQL的一个特性，来自外层查询的一个相关名称（上述查询中的S）可以用在where子句的子查询中。
- 使用了来自外层查询相关名称的子查询被称作相关子查询(correlated subquery)。
- **Correlation name** – variable S in the outer query
- **Correlated subquery** – the inner query

Use of “not exists” Clause

- 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 $X - Y = \emptyset \iff X \subseteq Y$
- Note: *Cannot* write this query using **= all** and its variants

3.8.4 Test for Absence of Duplicate Tuples

- The **unique** construct tests whether a subquery has any duplicate tuples in its result.
  - "true" , if a given subquery contains no duplicates .
- For example:
- Find all courses that were offered at most once in 2009

```
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 = 2009);
```

- 在不使用unique结构的情况下，上述查询的一种等价表达方式是：
- select T.course id
- from course as T
- where **1 <=** (select count(R.course id)
    - from section as R
    - where T.course id= R.course id and
    - R.year = 2009);

- **not unique**结构测试在一个子查询结果中是否存在重复元组。
- 对一个关系的**unique**测试结果为**假**的定义是:
  - 当且仅当在关系中存在着两个元组t1和t2，且t1=t2。
  - 由于在t1或t2的**某个域为空**时，判断t1=t2为假
- 所以，尽管一个元组有多个副本，只要该元组有一个属性为空，unique测试就有可能为真。

3.8.5 Subqueries in the Form Clause

- SQL allows a subquery expression to be used in the **from** clause
- 任何select-from-where表达式返回的结果都是**关系**，
- 因而该关系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;
```

- 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;
```

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

- 很多（但并非全部）SQL实现都支持在from子句中嵌套子查询。
- 注意，某些SQL实现要求对每一个子查询结果关系都给一个名字，即使该名字从不被引用；
- Oracle允许对子查询结果关系命名（省略掉关键字as），但是不允许对关系中的属性重命名。

- 在from子句嵌套的子查询中，不能使用来自from子句其他关系的相关变量。
- 然而SQL:2003，允许ftom子句中的子查询用**关键词lateral**作为前缀，以便访问from子句中在它前面的表或子查询中的属性。
- 例如：
- 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);

3.8.6 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.
- 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;
```

Complex Queries using With Clause

- 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*;

3.8.7 Scalar Subquery

- 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;
```

- Runtime error if subquery returns more than one result tuple

3.9 Modification of the Database

- **Deletion** of tuples from a given relation.
- **Insertion** of new tuples into a given relation
- **Updating** of values in some tuples in a given relation

3.9.1 Deletion

- 删除整个元组，而不能只删除某些属性上的值
  - **delete from** r
  - whert P；
- 其中P代表一个谓词，r代表一个关系。
- **delete**语句，
  - 首先从r中找出所有使P(t)为真的元组，
  - 然后，把它们从r中删除。

- Delete all instructors

```
delete from instructor
```

- Delete all instructors from the Finance department

```
delete from instructor
where dept_name= ’Finance’;
```

- Delete all tuples in the *instructor* relation for those instructors associated with a department located in the Watson building.
- delete from instructor
- where dept_name **in** (**select** dept name
  from *department*
  where *building* =’Watson’);

Deletion (Cont.)

- 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)

3.9.2 Insertion

- Add a new tuple to *course*

    insert into *course*
    values (’CS-437’, ’Database Systems’, ’Comp. Sci.’, 4);

- or equivalently

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

- Add a new tuple to *student* with *tot_creds* set to *null*

```
insert into student
        values ('3003', 'Green', 'Finance', null);
```

- Add all instructors to the *student* relation with tot_creds set to 0

```
insert into student
  select ID, name, dept_name, 0
   from   instructor
```

Insertion (Cont.)

- The **select from where** statement is evaluated fully before any of its results are inserted into the relation.

  Otherwise queries like

  insert into *table*1 **select * from** *table*1

  would cause **problem**

3.9.3 Updates

- 不改变整个元组的情况下，改变其部分属性的值
  - **update** r
  - **set** A=new value
  - **where** P;
- **update**语句中可以嵌套的select可以引用待更新的关系。
- 同样，SQL首先检查关系中的所有元组，看它们是否应该被更新，然后才执行更新。

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others by a 5%
  - Write *two* **update** statements:

```
update instructor
    set salary = salary * 1.03
    where salary > 100000;
update instructor
     set salary = salary * 1.05
     where salary <= 100000;
```

  - The *order* is important
  - Can be done better using the **case** statement.

Case Statement for Conditional Updates

- Same query as before but with **case** statement

```
update instructor
 set salary = case
        when salary <= 100000 then salary * 1.05
        else salary * 1.03
        end
```

- **case**语句的一般格式如下：
  - **case**
    - **when** pred1 **then** result1
    - **when** pred2 **then** result2
    - …
    - **when** predn **then** resultn
    - **else** result0
  - **end**

Updates with Scalar Subqueries

- 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);
```

- Sets *tot_creds* to null for students who have *not taken any* course
- Instead of **sum**(*credits*), use:

```
update student S
   set tot_cred =case
    when sum(credits) is not null then
   sum(credits)
     else 0
     end
```

End of Chapter 3

Database System Concepts, 6th Ed.