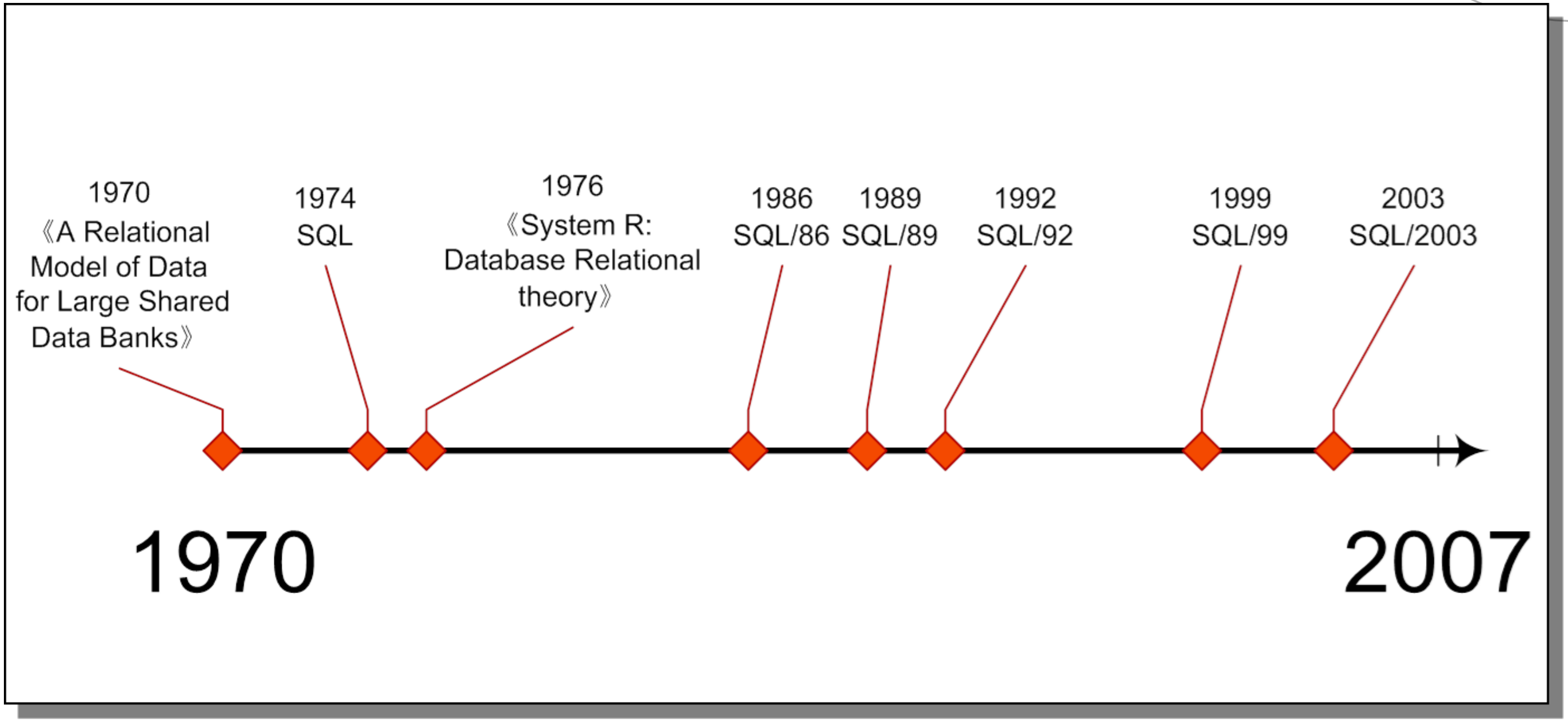
Principles of Database Systems

Introduction to SQL

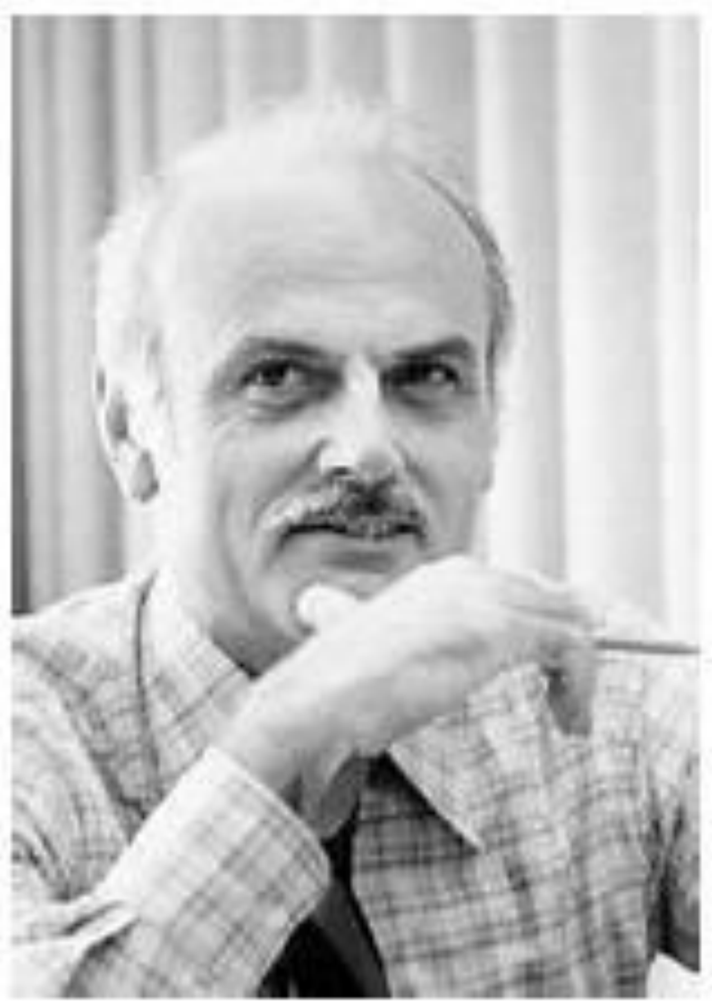
Overview of the SQL Query Language

History of SQL


1970
《A Relational Model of Data for Large Shared Data Banks》
1974
SQL
1976
《System R: Database Relational theory》
1986
SQL/86
1989
SQL/89
1992
SQL/92
1999
SQL/99
2003
SQL/2003
1970
2007

The newest SQL standard is SQL/2008

Edgar Frank Codd



埃德加 ·弗兰克 科德（Edgar Frank Codd，1923—2003）是密执安大学哲学博士，IBM公司研究员，被誉为“**关系数据库之父**”，并因为在数据库管理系统的理论和实践方面的杰出贡献于1981年获**图灵奖**。1970年，科德发表题为"**A Relational Model of Data for Large Shared Data Banks**"（大型共享数据库的关系数据模型）的论文，文中首次提出了数据库的关系模型。由于关系模型简单明了、具有坚实的数学理论基础，所以一经推出就受到了学术界和产业界的高度重视和广泛响应，并很快成为数据库市场的主流。20世纪80年代以来，计算机厂商推出的数据库管理系统几乎都支持关系模型，数据库领域当前的研究工作大都以关系模型为基础。

History of SQL

- IBM Sequel language developed as part of **System R** project at the IBM San Jose Research Laboratory
- Renamed Structured Query Language (SQL,结构化查询语言)
- ANSI and ISO standard SQL:
  - SQL-86, SQL-89, SQL-92
  - SQL:1999, SQL:2003, SQL:2008

  - ANSI: the American National Standards Institute
  - ISO: the International Organization for Standardization

Database Languages

- Database Languages as human-machine interfaces

  - **Data-Manipulation Language, DML** (数据操纵语言)

  - **Data-Definition Language, DDL** (数据定义语言)

姓名	生日	身高	项目	时间	国家
博尔特	1986.821	196	100米跑	9’79	牙买加
苏炳添	1989.8.29	172	100米跑	~~9’99~~ 9’98	中国
宁泽涛	1993.3.6	191	100米自	47’65	中国
菲尔普斯	1985.6.30	193	100米蝶	50’58	美国

Database Languages

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

- Two classes of languages
  - **Procedural (过程化DML)**– user specifies what data is required and how to get those data
  - **Declarative (nonprocedural) (声明式DML)** – user specifies what data is required without specifying how to get those data

- Query (查询):a statement requesting the retrieval of information

- SQL is the most widely used query language

Constituent Parts of SQL (SQL组成部分)

- The SQL language has several parts:
  - Data-definition language (DDL)
  - Data-manipulation language (DML)
  - Integrity (完整性) (included in DDL)
  - View definition (视图定义) (included in DDL)
  - Transaction control (事务控制)
  - Embedded SQL and dynamic SQL (嵌入式SQL及动态SQL)
  - Authorization (授权)

Implementation of SQL (SQL实现)

- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.

- Not all examples here may work on your particular system.
  - E.g. the “natural join”(自然连接) is not implemented in Microsoft SQL Server.

SQL Data Definition

Functions of DDL

- The SQL DDL allows specification of not only a set of relations, but also information about each relation, including:
  - **The schema for each relation.**
  - **The types of values associated with each attribute.**
  - The integrity constraints.
  - The set of indices to be maintained for each relation.
  - The security and authorization information for each relation.
  - The physical storage structure of each relation on disk.

Basic Types

- **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.
- **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.
- **date, time, timestamp, interval**
- Each type may include a special value called the **null** value

Basic Schema Definition-Create

- We define an SQL relation by using the **create table** command.

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

Basic Schema Definition-Create

- The general form of the **create table** command is:

  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

Integrity Constraints in DDL

- SQL supports a number of different integrity constraints:
  - **primary key** (A_{j1}, A_{j2},…, A_{jm}): The primary-key specification says that attributes A_{j1}, A_{j2},…, A_{jm} form the primary key for the relation. The primary key attributes are required to be *nonnull* and *unique*;
  - **foreign key (**$\boldsymbol{A_{k1}}$**,** $\boldsymbol{A_{k2}}$**,…,** $\boldsymbol{A_{kn}}$ **) references s**:The foreign key specification says that the values of attributes (A_{k1}, A_{k2},…, A_{kn}) for any tuple in the relation must correspond to values of the primary key attributes of some tuple in relation *s*.

Integrity Constraints in DDL (Cont.)

- SQL supports a number of different integrity constraints:
  - **not null**: The not null constraint on an attribute specifies that the null value is not allowed for that attribute
  - SQL prevents any update to the database that violates an integrity constraint.
    - For example, a tuple has null value for any primary key attribute.

Basic Schema Definition-Drop

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

drop table *r*

Basic Schema Definition-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 type of *A*

- all 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

SQL Data Manipulation

Data Manipulation

- A newly created table is empty initially, we can use **insert** command to load data into the table

```
insert into instructor
        values (10211, ’Smith’, ’Biology’, 66000);
```

- The **delete** command removes tuples from the table

```
delete from account
```

- The **update** command changes a value in a tuple without changing all values in the tuple.

```
update instructor
        set salary= salary * 1.05;
```

SQL Data Query

SQL Data Query

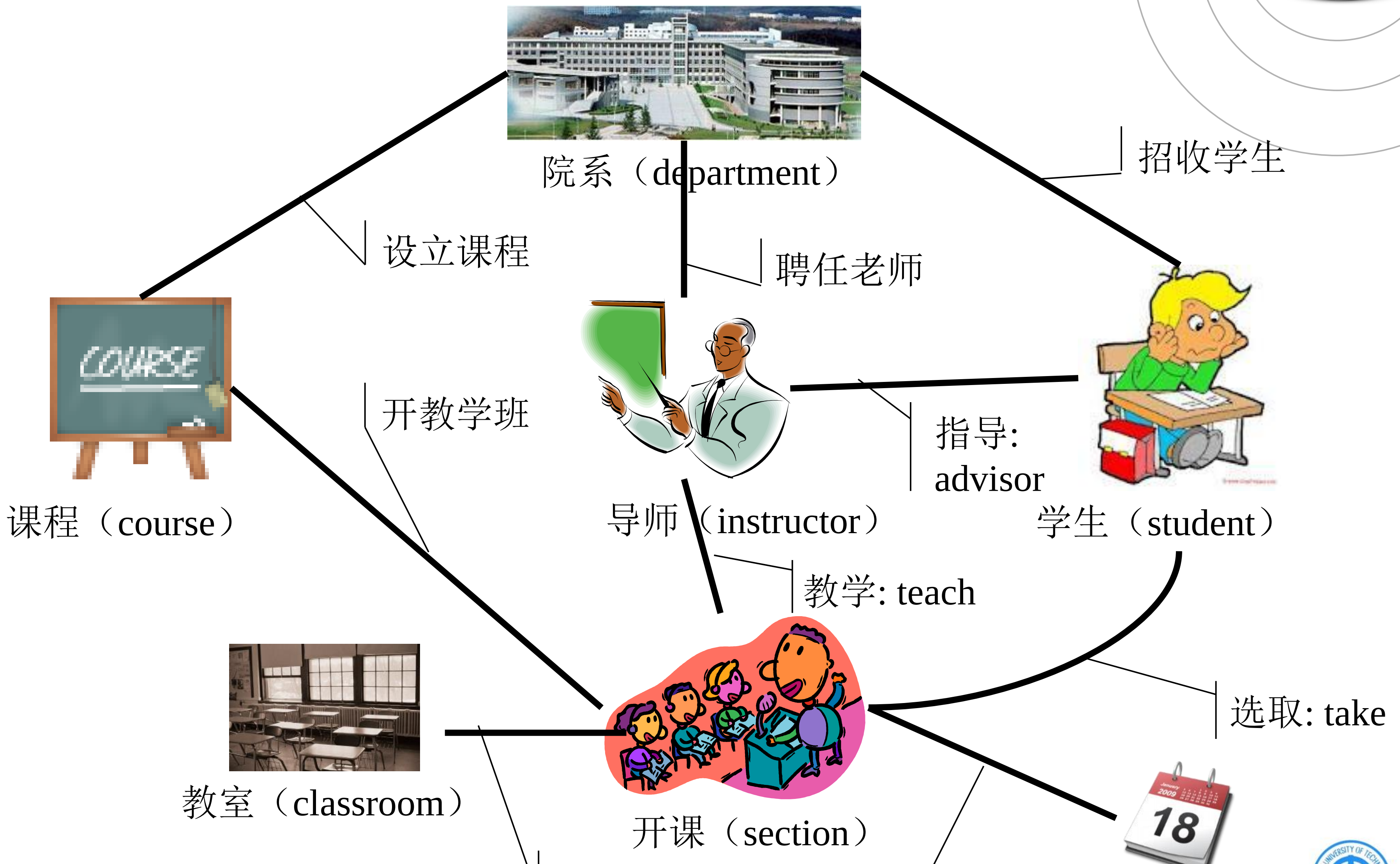
- The SQL data-manipulation language (DML) provides the ability to query information, and insert, delete and update tuples

- “Query” (查询) could be generalized definition
  - Define (定义) , **retrieve (检索)**, modify (修改), control (控制) etc. on DB
  - Define→DDL(create, alter, drop)
  - Modify→DML(insert, update, delete)
  - **Retrieve→DML(select)**
  - Control→DCL(grant,revoke)

Key points

- Always keep in mind:
  - SQL statements are case insensitive (大小写不敏感)
    - E.g.   Name ≡ NAME ≡ name
  - SQL statements must follow the fixed syntax (固定语法)
  - **Both the input and output of SQL statements are table(loosely speaking, relation)**, because SQL statements are the implementations of relational operations (关系运算). This point is especially important for understanding the underlying principle of query, that is SELECT statement.

Database Explained



院系（department）

设立课程

聘任老师

招收学生

课程（course）

开教学班

导师（instructor）

指导: advisor

学生（student）

教学: teach

选取: take

教室（classroom）

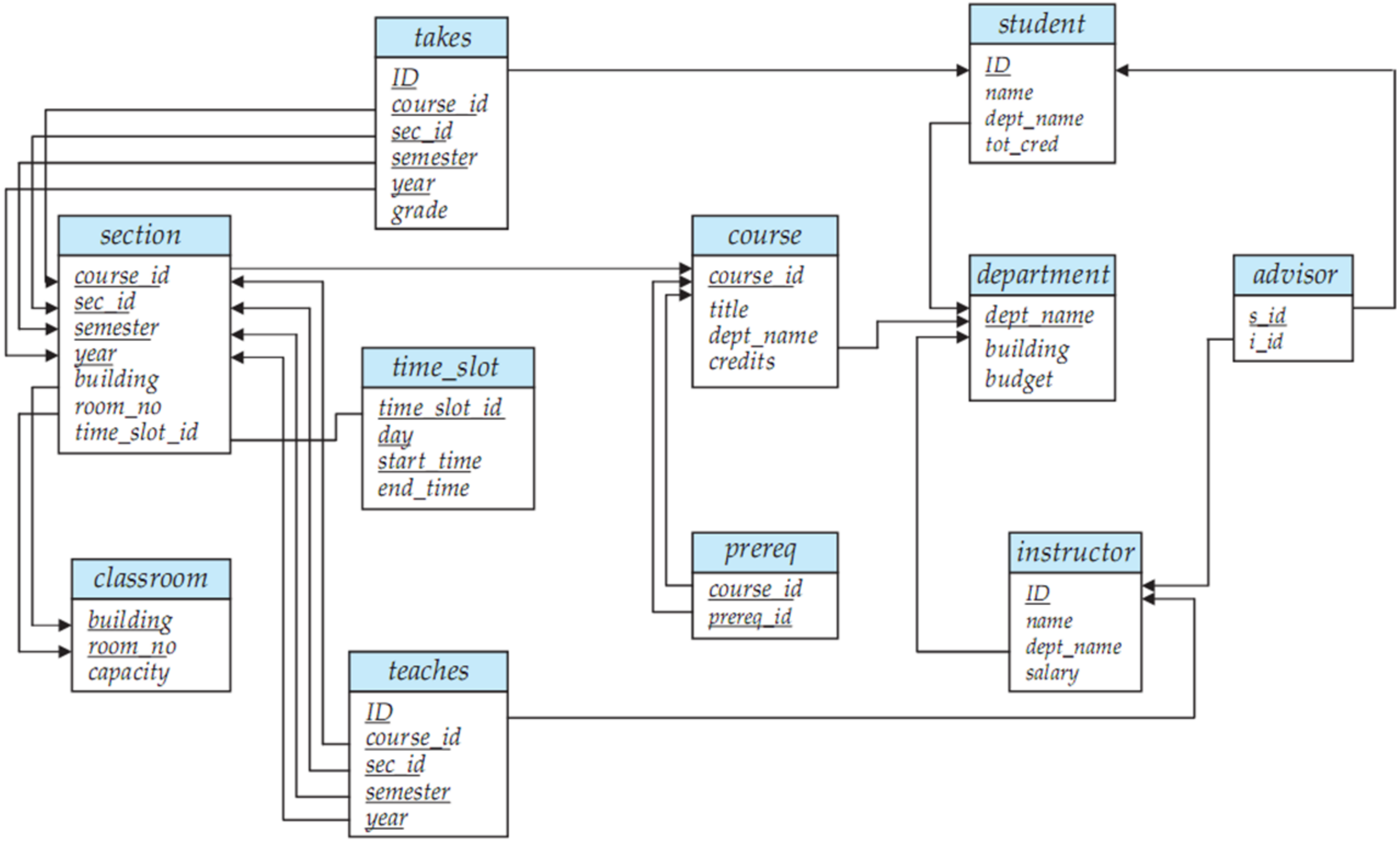
开课（section）

使用教室

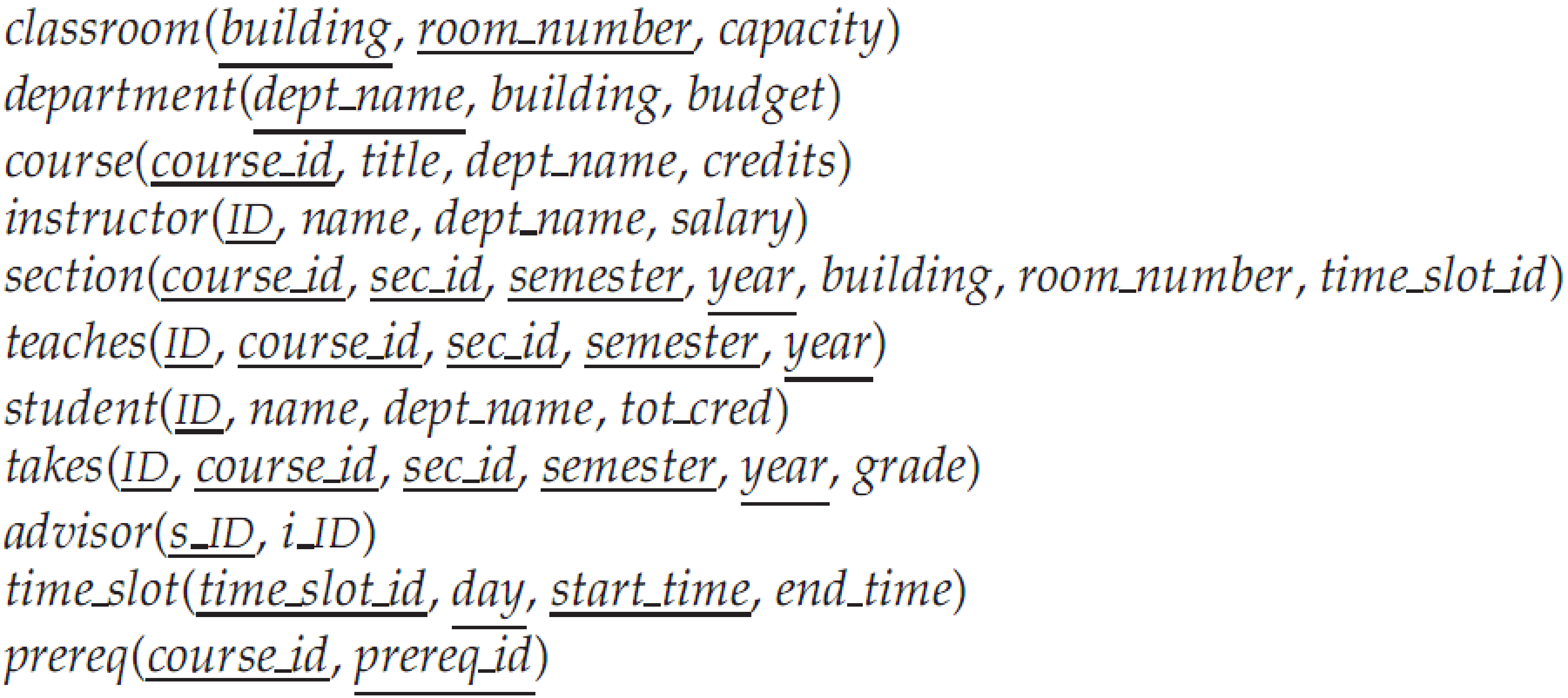
上课时间

时间段（time_slot）

Database Used



Database Used



classroom(building, room_number, capacity)

department(dept_name, building, budget)

course(course_id, title, dept_name, credits)

instructor(ID, name, dept_name, salary)

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

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

student(ID, name, dept_name, tot_cred)

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

advisor(s_ID, i_ID)

time_slot(time_slot_id, day, start_time, end_time)

prereq(course_id, prereq_id)

```
create table instructor
        (ID                       varchar(5),
         name                     varchar(20) not null,
         dept_name                varchar(20),
         salary                   numeric(8,2) check (salary > 29000),
         primary key (ID),
         foreign key (dept_name) references department
          );
```

练习一

请写出下面两个表的建表语句。

姓名	生日	身高	项目	时间	国家号
博尔特	1986.8.21	196	100米跑	9’58	1
苏炳添	1989.8.29	172	100米跑	9’99	2
宁泽涛	1993.3.6	191	100米自	47’65	2

编号	国家名
1	牙买加
2	中国
3	美国

Basic Structure of SQL Queries

First Sight on SELECT

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

- The basic structure of an SQL query consists of three clauses (子句): **select**, **from**, and **where**.

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 (属性)
  - R_i represents a relation (关系)
  - P is a predicate (谓词).

- The query takes as its input the relations listed in the **from clause**, operates on them as specified in the **where** and **select clauses**, and then produces a relation as the result

Informal Vocabulary useful in the very beginning

- select *XX* → I want get *XX* attribute.
- from *YY* → I need data from *YY.*
- where *ZZ* → The data needed should satisfy the condition of *ZZ*.

Try…

- Explain the meaning of following statement

select *name* **from** *instructor*;

- "Find the names of all instructors."

Try…

- Explain the meaning of following statement

```
select name
from instructor
where salary > 70000;
```

- "Find the names of all instructors who have salary greater than $70,000."

Try…

- Construct statement for the following query
- "Find the department names of all instructors."

select *dept_name*
from *instructor*;

Queries on a Single Relation

- The **select clause** list the attributes desired in the result of a query

  Example:

  find the names of all instructors:

  select *name*
  from *instructor*

- An asterisk(星号) in the select clause denotes "all attributes"

  select * **from** instructor

Queries on a Single Relation

- In the formal, mathematical definition of the relational model, a **relation is a set**. Thus, duplicate tuples would never appear in relations. In practice, duplicate elimination is time-consuming. Therefore, **SQL allows duplicates in relations as well as in the results of SQL expressions.**

- Run the following statement and try to understand the result.

select *dept_name* **from** *instructor*;

Queries on a Single Relation

- Insert the keyword **distinct** after **select** to force the elimination of duplicates.

  select distinct *dept_name*
  from *instructor*;

- Try it …

Queries on a Single Relation

- The **select clause** may also contain arithmetic expressions (算数表达式) involving the operators +, –, *, and / operating on constants or attributes of tuples. The literal (文字量) can also be contained in **select clause**.

```
select ID, name, dept_name, salary *1.1
from instructor;
```

- Note: it does not result in any change to the instructor relation.

Queries on a Single Relation

- The **where clause** allows us to select only those rows in the result relation of the from clause that satisfy a specified predicate.

- To find all instructors in Comp. Sci. dept with salary > 80000

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

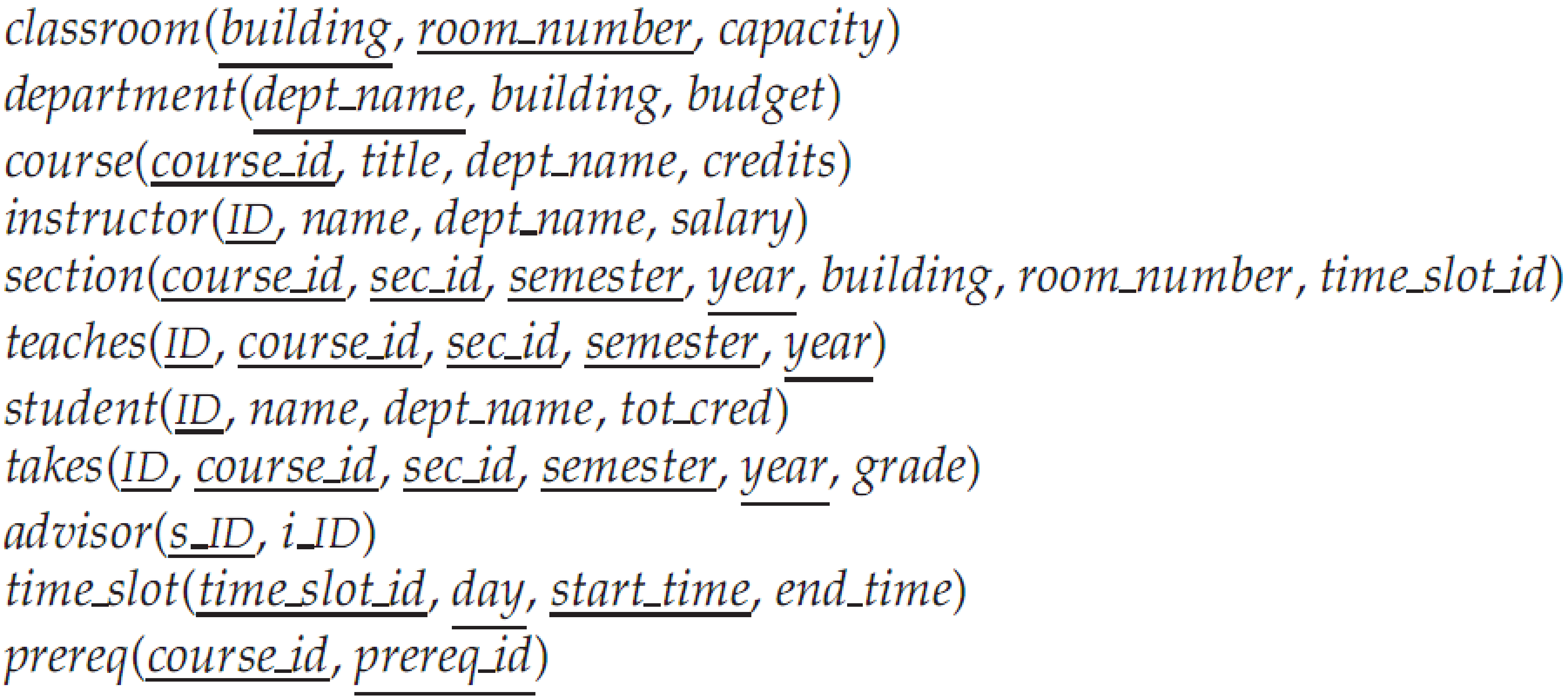
- Comparison results (比较结果) can be combined using the logical connectives **and**, **or**, and **not**.

- Comparisons can be applied to results of arithmetic expressions.

Queries on Multiple Relations

- Try to answer the query “Retrieve the names of all instructors, along with their department names and department building name.”

Database Used



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

Queries on Multiple Relations

- The **from clause** lists the relations involved in the query

- Listing multiple relations in **from clause** **without** any restriction specified in **where clause** will lead the result of Cartesian product over the relations

Queries on Multiple Relations

- E.g. Find the Cartesian product (笛卡尔积) instructor X teaches

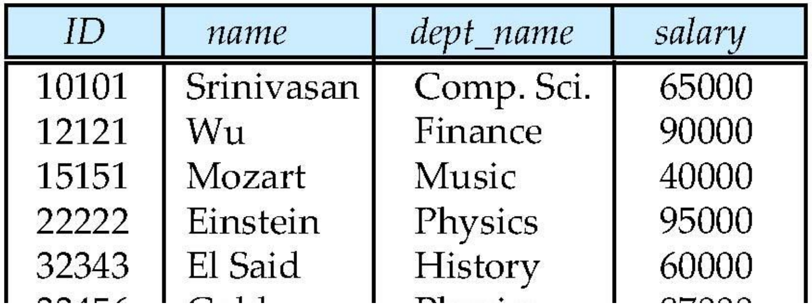
```
select *
from instructor, teaches
```

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

- Cartesian product is not very useful directly, but useful combined with where-clause condition

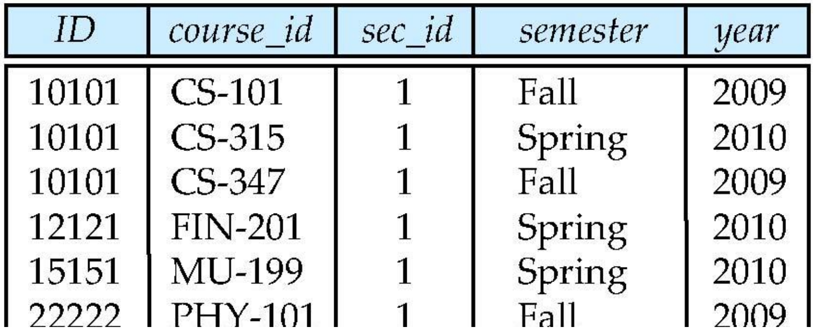
Cartesian Product: *instructor X teaches*

instructor

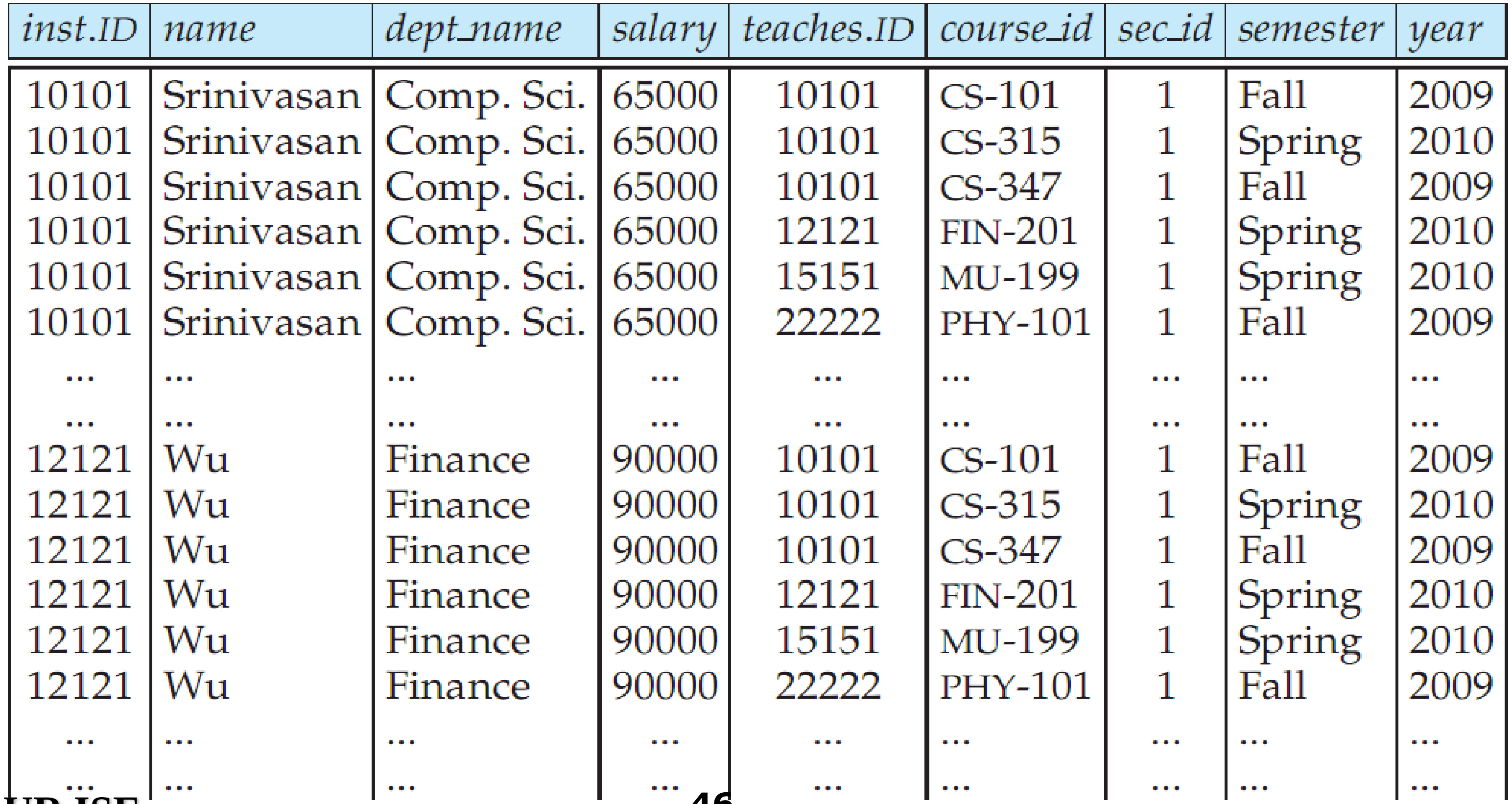


ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000

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	Finance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Join(连接)

- “Retrieve the names of all instructors, along with their department names and department building name.”

- To answer the query, each tuple in the instructor relation must be matched with the tuple in the department relation whose dept_name value **matches** the dept_name value of the instructor tuple.

```
select name, instructor.dept name, building
from instructor, department
where instructor.dept_name= department.dept_name;
```

Join

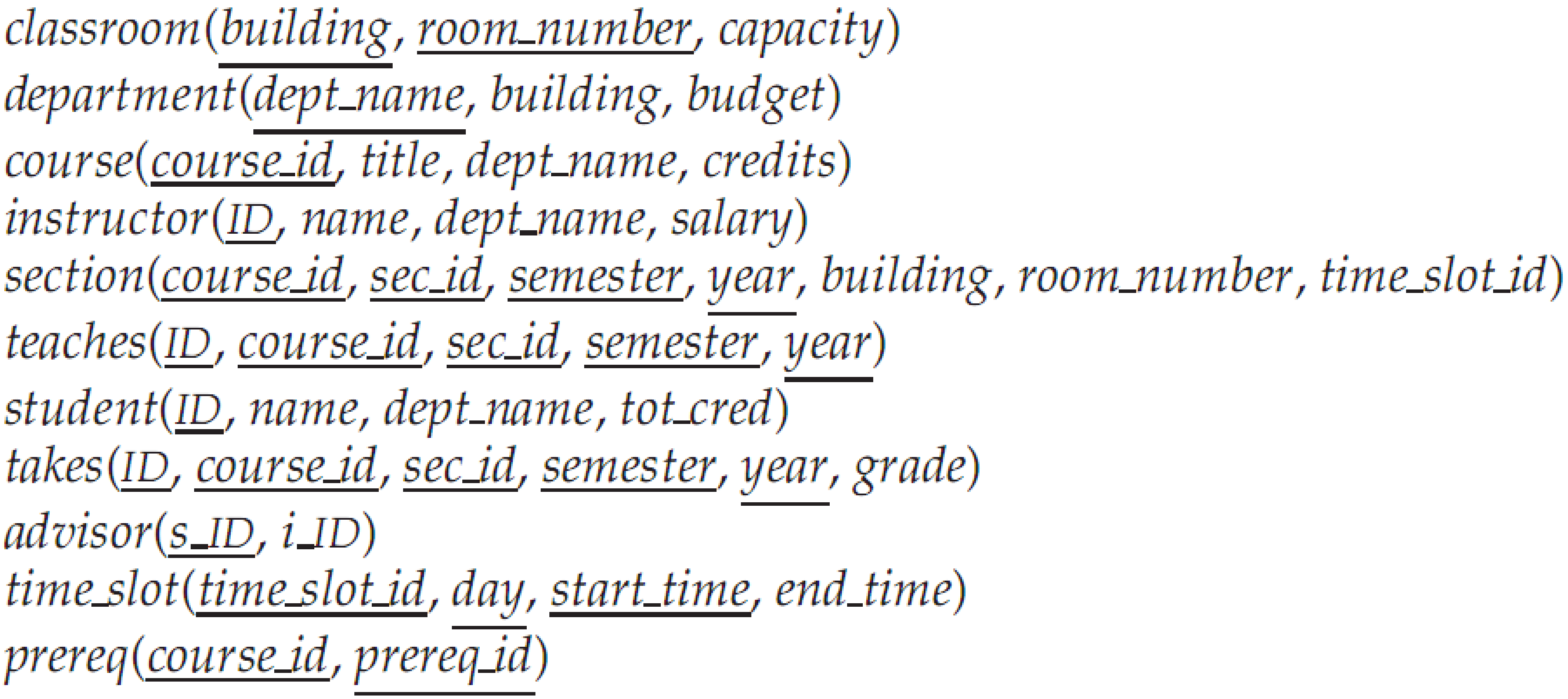
- Loosely speaking, join is the operation which combines records(rows or tuples) from two or more tables(relations) in a database.

- The records combined should follow some given conditions corresponding to specified business logic(e.g. having same value in the attributes that have the same name).

Join

- Some other examples
  - For all instructors who have taught some course, find their names and the course ID of the courses they taught.

Database Used



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

Join

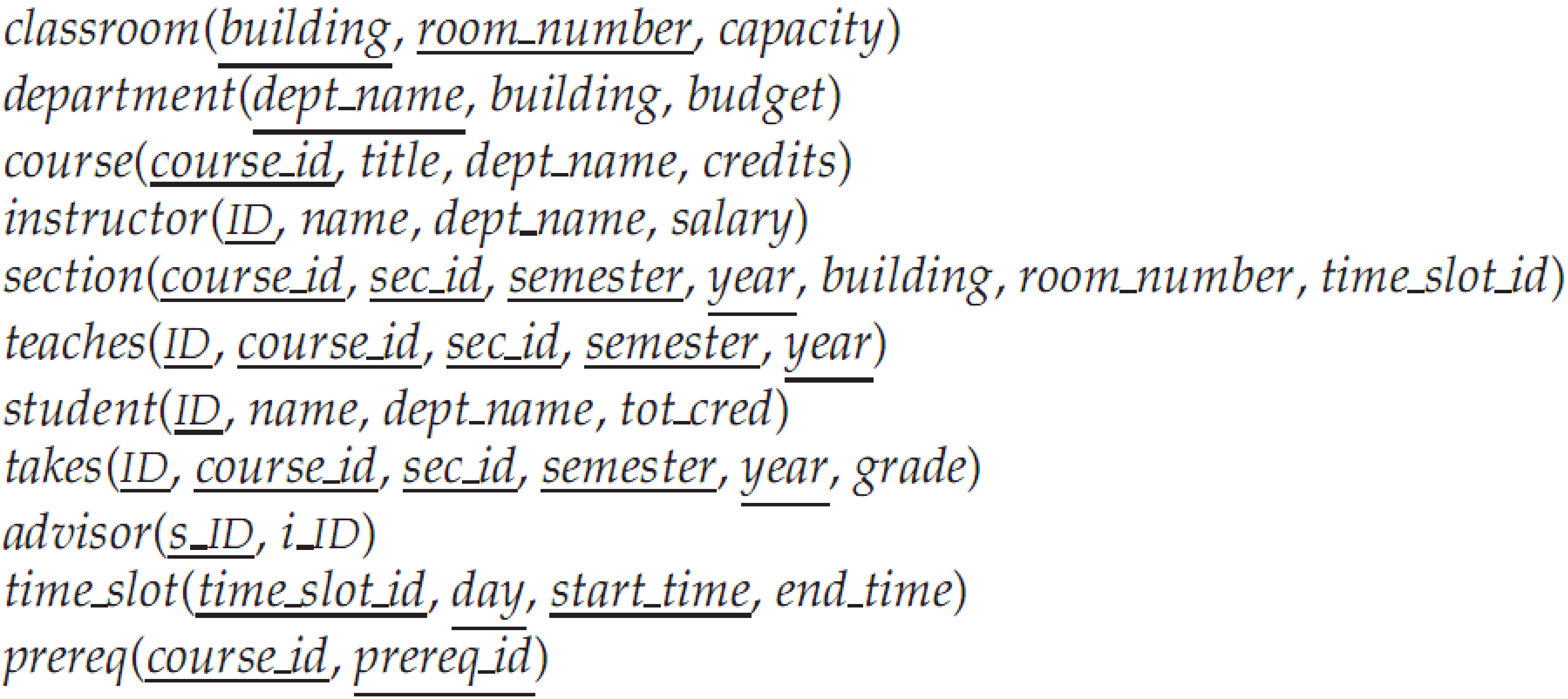
- Some other examples
  - For all instructors who have taught some course, find their names and the course ID of the courses they taught.

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

Join

- Some other examples
  - Find the course ID, semester, year and title of each course offered by the Comp. Sci. department

Database Used



classroom(building, room_number, capacity)

department(dept_name, building, budget)

course(course_id, title, dept_name, credits)

instructor(ID, name, dept_name, salary)

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

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

student(ID, name, dept_name, tot_cred)

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

advisor(s_ID, i_ID)

time_slot(time_slot_id, day, start_time, end_time)

prereq(course_id, prereq_id)

Join

- Some other examples
  - Find the course ID, semester, year and title of each course offered by the Comp. Sci. department

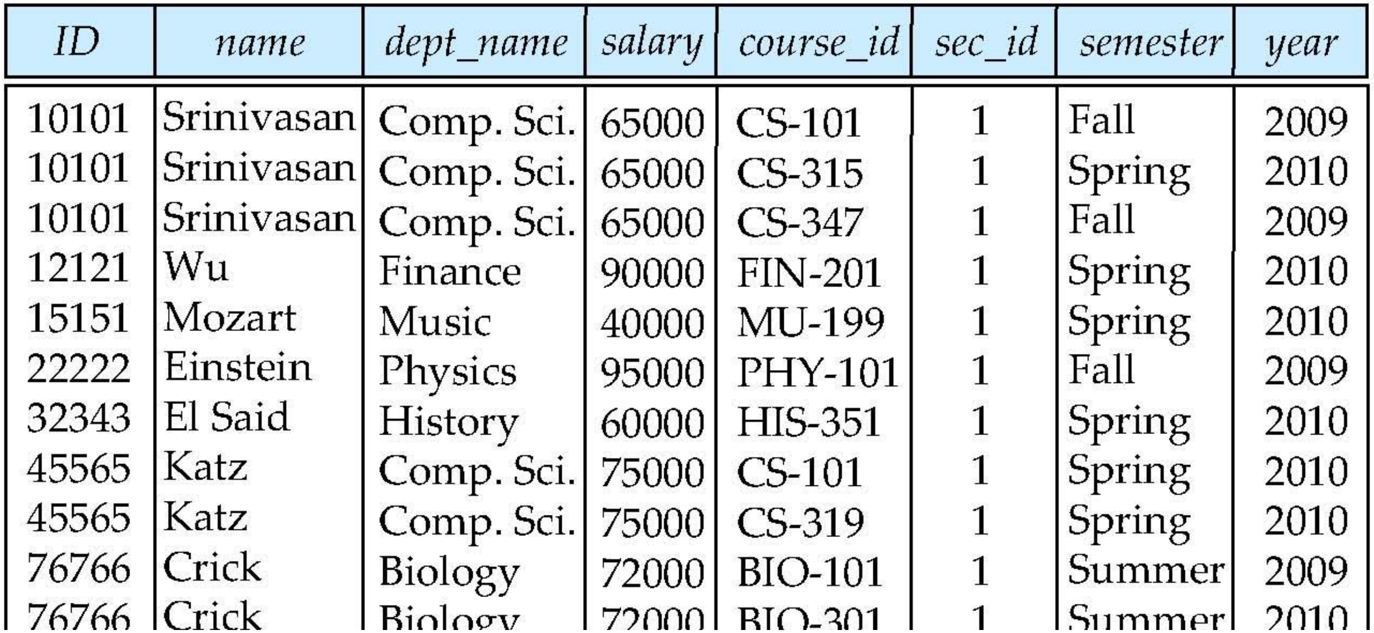
```
select section.course_id, semester, year, title
from section, course
where   section.course_id = course.course_id
   and   dept_name = 'Comp. Sci.'
```

Join

- Besides “From+Where” method, join can also performed in SQL with “JOIN” keywords(see Chapter 4).

Natural Join(自然连接)

- Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column 自然连接仅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素对，每一个相同属性列仅留一个拷贝。
- **select** *
  from *instructor* **natural join** *teaches*;



ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010

Natural Join Example

- List the names of instructors along with the course ID of the courses that they taught.
  - **select** name, course_id
    from instructor, teaches
    where instructor.ID = teaches.ID;
  - **select** name, course_id
    from instructor **natural join** teaches;
- Danger in natural join: beware of unrelated attributes with same name which get equated incorrectly.
- Note: MS SQL Server doesn't support keyword "natural join".

Understanding the Operational Order

- Although the clauses must be written in the order **select**, **from**, **where**, the easiest way to understand the operations specified by the query is to consider the clauses in operational order:

  first **from**,
  then **where**,
  and then **select**.

Meaning of an SQL Query

- In general, the meaning of an SQL query can be understood as follows:

1. Generate a **Cartesian product**(笛卡尔积) of the relations listed in the **from clause**

2. Apply the **predicates**(谓词) specified in the **where clause** on the result of Step 1.

3. For each tuple in the result of Step 2, output the attributes (or results of expressions) specified in the **select clause**.

Additional Basic Operations

The Rename Operation(更名运算)

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

  old-name **as** new-name

- E.g.

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

Self-join(自连接)

- A self-join is joining a table to itself.
- **Try:** “Find the names of all instructors who have a higher salary than some instructor in ‘Comp. Sci’” .

Self-join

- 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
  - Keyword as must be omitted in Oracle
  - The key of self-join is the relation renaming operation

String Operations(字符串运算)

- SQL specifies strings by enclosing them in **single quotes**(单引号), for example, ’Computer’.

- A single quote character that is part of a string can be specified by using two single quote characters; for example, the string “**It’s right**” can be specified by “**It”s right**”

String Operations

- **The SQL standard** specifies that the equality operation on strings is **case sensitive**

- SQL also permits a variety of functions on character strings, such as concatenating (using “| |”), converting strings to uppercase (using **upper(s)**) and lowercase (using the function **lower(s)**) and so on.

String Operations

- SQL includes a string-matching operator for comparisons on character strings. The operator “like” uses patterns(模式) that are described using two special characters:
  - percent (%). The % character matches any **substring**(子字符串).
  - underscore (_). The _ character matches any **character**(字符).

String Operations

- Try to explain the following patterns:
  - ’Intro%’
  - ’%Comp%’
  - ’_ _ _ ’
  - ’_ _ _ %’

String Operations

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

String Operations

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

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

String Operations

- Backslash (\) is used as the escape character.

- E.g.
  - **like** ’ab\%cd%’ **escape** ’\’ matches all strings beginning with “ab%cd”.

  - **like** ’ab\\cd%’ **escape** ’\’ matches all strings beginning with “ab\cd”.

Ordering the Display of Tuples

- The order by clause causes the tuples in the result of a query to appear in sorted order

```
select name
from instructor
where dept_name = ’Physics’
order by name;
```

Ordering the Display of Tuples

- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default. 默认升序
  - Example: **order by** name **desc**

- Can sort on multiple attributes
  - Example: **order by** dept_name, name

Where Clause Predicates

- 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');
    ```
  - SQL server doesn't support this feature

Set Operations(集合运算)

Set Operations

- 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)

n 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)

n 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)

Set Operations

- Set operations **union**(并), **intersect** (交), and **except** (差)

- Each of the above operations automatically eliminates duplicates(重复)

- To retain all duplicates use the corresponding multiset(多重集) versions **union all**, **intersect all** and **except all**.
  - MS SQL Server doesn't support **intersect all** and **except all**

Null Values

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

Null Values

- 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
    ```

Null Values and Three Valued Logic

- Any comparison with *null* returns *unknown*
  - Example*: 5 < null or null <> null or null = null*

- 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 is treated as *false* if it evaluates to *unknown***
  - Try: **select** * **from** instructor **where** ((null+1) is not null);

Aggregate Functions (聚集函数)

Aggregate Functions

- Aggregate functions are functions that take a collection (a set or multiset) of values as input and return a single value.
  - Average: **avg**
  - Minimum: **min**
  - Maximum: **max**
  - Total : **sum**
  - Count: **count**

Basic Aggregation

- Example: “Find the average salary of instructors in the Computer Science department.”
  - **select avg** (salary)
    from instructor
    where dept_name= ’Comp. Sci.’;
  - a meaningful name can be given to the result attribute by using the **as clause**
  - Retaining duplicates is important in computing an average.

Basic Aggregation

- If we do want to eliminate duplicates, we use the keyword **distinct** in the aggregate expression.

  – E.g. “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;

Basic Aggregation

- We use the aggregate function **count** frequently to count the number of tuples in a relation.
- The notation for this function in SQL is **count (*)**

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

- SQL does not allow the use of **distinct** with **count (*)**. It is legal to use **distinct** with **max** and **min**, even though the result does not change.

Aggregation with Null Values

- All aggregate operations except count(*) ignore tuples with null values on the aggregated attributes (除了count(*)外所有的聚集函数都忽略输入集合中的空值)

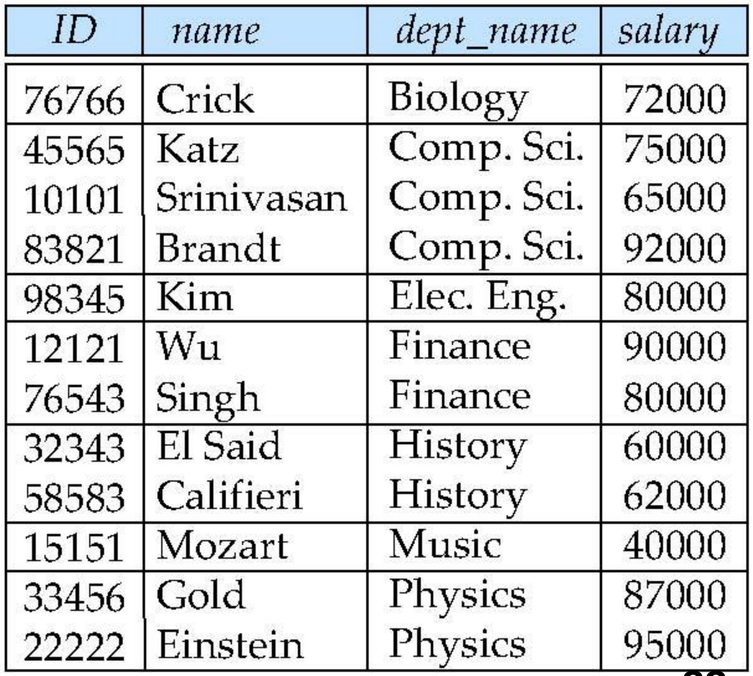
- What if collection has only null values?
  - count returns 0
  - all other aggregates return null

Aggregation with Grouping(分组)

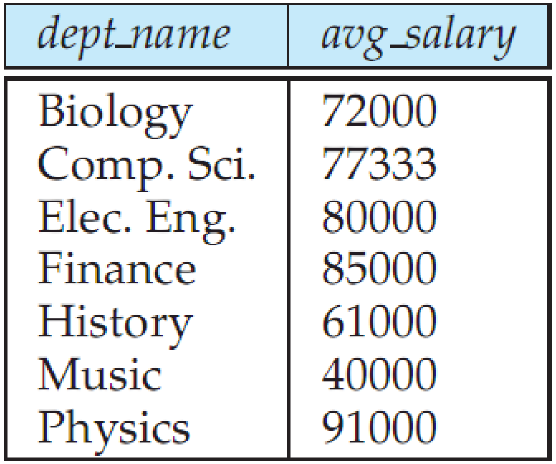
- apply the aggregate function not only to **a single set of tuples**, but also to **a group of sets of tuples**
  - E.g. “Find the average salary in each department.”

Aggregation with Grouping(分组)

- apply the aggregate function not only to **a single set of tuples**, but also to **a group of sets of tuples**
  - E.g. “Find the average salary in each department.”



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 with Grouping

- apply the aggregate function not only to **a single set of tuples**, but also to **a group of sets of tuples**
  - E.g. “Find the average salary in each department.”

    ```
    select dept_name, avg (salary) as avg_salary
    from instructor
    group by dept_name;
    ```

  - Note: departments with no instructor will not appear in result

Aggregation with Grouping

- Attributes in **select clause** outside of aggregate functions must appear in **group by** list (出现在select子句中但没有被聚集的属性必须出现在group by子句中)

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

The Having Clause

- Try: Find the names and average salaries of all departments whose **average salary** is greater than 42000

- Can we use the where clause to satisfy the query?

The Having Clause

- Try: Find the names and average salaries of all departments whose **average salary** is greater than 42000

- This query requires restriction on tuple groups rather than tuples, so the where clause is useless in the case. Instead, **having clause** should be used to filter(过滤) tuple groups.

The Having Clause

- Try: Find the names and average salaries of all departments whose **average salary** is greater than 42000

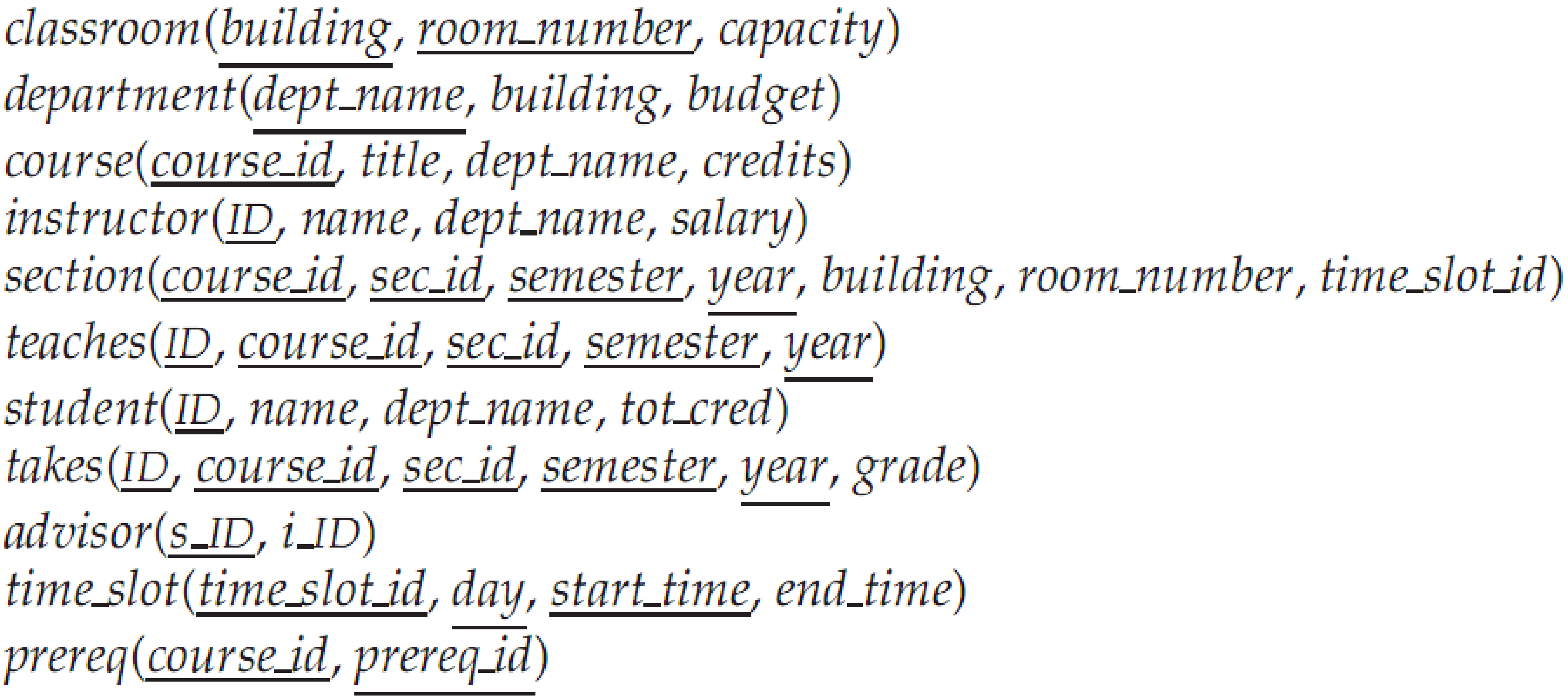
  select dept_name, **avg** (salary)
  from instructor
  group by dept_name
  having avg (salary) > 42000;

- Note: predicates in the **having clause** are applied **after** the formation of groups whereas predicates in the **where clause** are applied **before** forming groups (用在having中的谓词在形成分组后才起作用，而where子句中的谓词在分组前起作用)

The Having Clause

- Try: “For each course section offered in 2009, find the average total credits (tot_cred) of all students enrolled in the section, if the section had at least 2 students.”

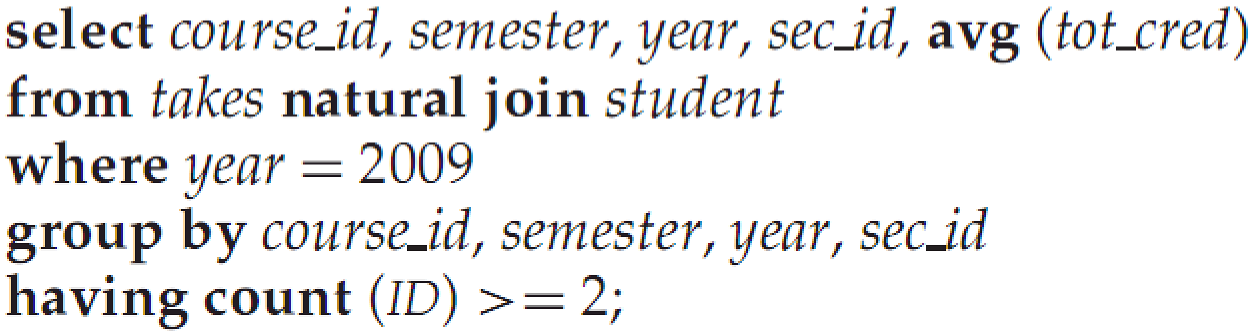
Database Used



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

The Having Clause

- Try: “For each course section offered in 2009, find the average total credits (tot_cred) of all students enrolled in the section, if the section had at least 2 students.”



```
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;
```

Nested Subqueries
(嵌套子查询)

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. (子查询是嵌套在另一个查询中的select-from-where表达式)
- A common use of subqueries is to perform tests for **set membership**, **set comparisons**, **and set cardinality**. (子查询通常被用来对集合成员资格、集合的比较以及集合的基数进行检查)

Set Membership(集合的成员资格)

- SQL allows testing tuples for membership in a relation. The **in** connective tests for set membership, where the set is a collection of values produced by a **select clause**. The **not in** connective tests for the absence of 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)

Set Membership

- The **in** and **not in** operators can also be used on enumerated(枚举) sets. (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’);
```

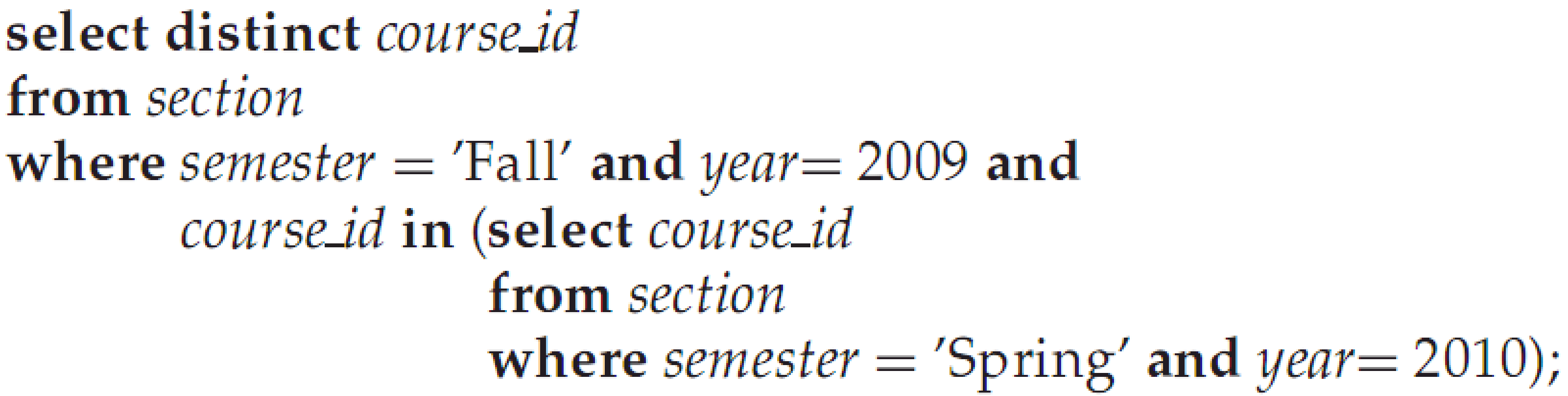
Set Membership

- “Find all the courses taught in the both the Fall 2009 and Spring 2010 semesters.”

- Step 1

```
(select course id
from section
where semester = ’Spring’ and year= 2010)
```

- Step 2



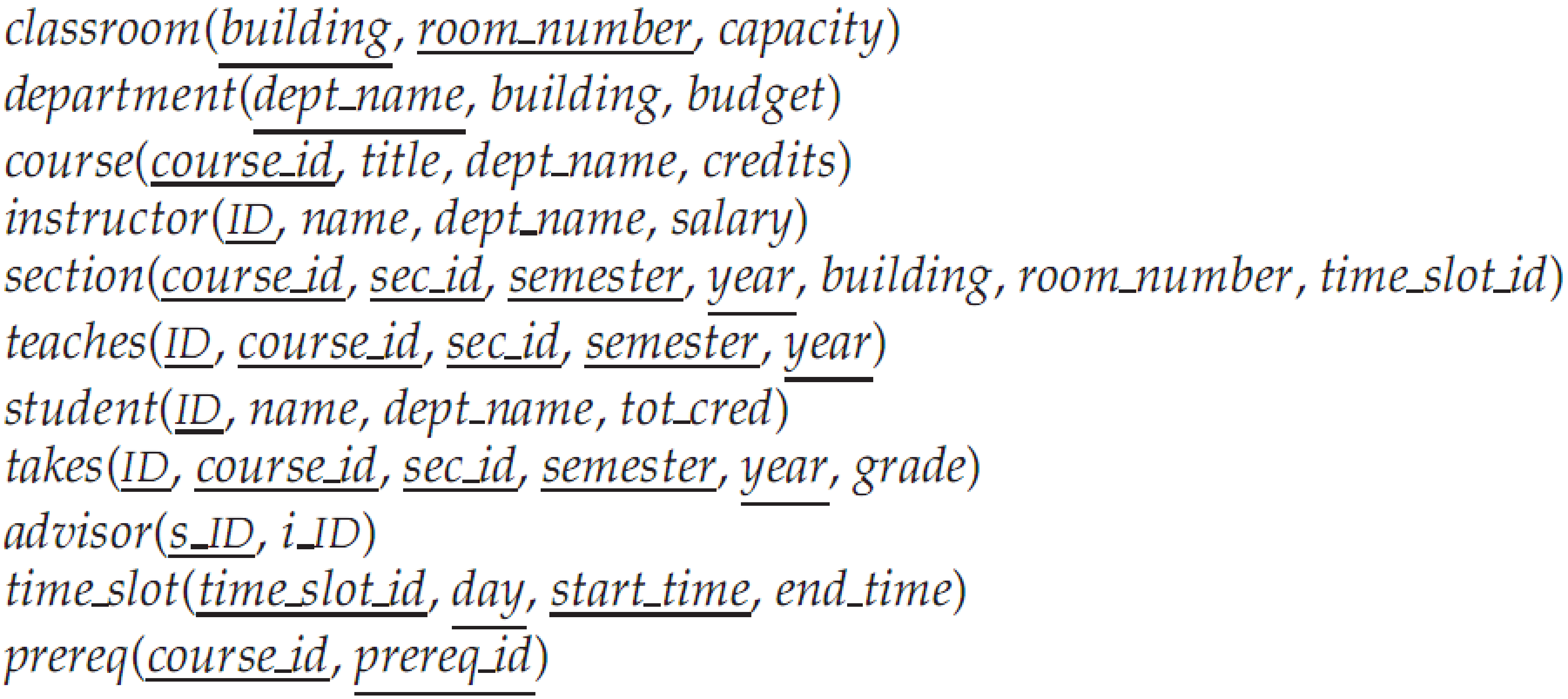
```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
        course_id in (select course_id
                      from section
                      where semester = 'Spring' and year= 2010);
```

Set Membership

- “Find all the courses taught in the both the Fall 2009 but not in Spring 2010 semesters.”

```
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);
```

Database Used



classroom(building, room_number, capacity)

department(dept_name, building, budget)

course(course_id, title, dept_name, credits)

instructor(ID, name, dept_name, salary)

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

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

student(ID, name, dept_name, tot_cred)

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

advisor(s_ID, i_ID)

time_slot(time_slot_id, day, start_time, end_time)

prereq(course_id, prereq_id)

Set Comparison(集合的比较)

- Recall the query “Find names of instructors with salary greater than that of some (at least one) instructor in the Biology ”

  select distinct T.name
  from instructor as T, instructor as S
  where T.salary > S.salary **and** S.dept_name=’Biology’;

Set Comparison

- Alternative style for the query

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

Note: The keyword any is synonymous(同义的) to some in SQL

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**) $\equiv$ **in**

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

Set Comparison

- 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** $r \Leftrightarrow \forall\, t \in r$ (F <comp> t)

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

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

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

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

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

Set Comparison

- In many cases, using subqueries with **some**, **any** or **all** is logical equivalent to equality comparison which uses the subqueries containing certain aggregation.(在很多情况下，使用some,any或all的子查询逻辑等价于对使用聚合的子查询进行相等比较)

Test for Empty Relations

- 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);
```

- **Correlated subquery相关子查询**

Correlated Subquery

- A subquery that uses a correlation name from an **outer query** is called a **correlated subquery**

- Observe and explain the following query:

```
select ID,name
from student
where (select COUNT(*)
           from takes
           where takes.ID=student.ID
           group by takes.ID)>2;
```

Not Exists

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

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

Not Exists

- 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) );
```

Subqueries in the From Clause

- SQL allows a subquery expression to be used in the from clause.

- **The key concept applied here is that any select-from-where expression returns a relation as a result and, therefore, can be inserted into another select-from-where anywhere that a relation can appear.**

Subqueries in the From Clause

- 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;
```

Subqueries in the From 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;
```

With Clause

- The **with** clause provides a way of defining a temporary view whose definition is available only to the query in which the **with** clause occurs. (With子句提供了定义临时视图的方法，该定义只对包含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 dept_name,budget
from department, max_budget
where department.budget = max_budget.value;
```

Complex Queries using With Clause

- With clause is very useful for writing complex queries
- Supported by most database systems, with minor syntax variations

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;
```

Scalar Subquery(标量子查询)

- Scalar subquery is one which is used where a single value is expected
- E.g.

```
select dept_name,
       (select count(*)
          from instructor
         where department.dept_name = instructor.dept_name)
       as num_instructors
 from department;
```

- E.g.

```
select name
from instructor
where  salary * 10 >
       (select budget  from department
         where department.dept_name = instructor.dept_name)
```

- **Runtime error if subquery returns more than one result tuple** (如果执行后结果不止一个元组，则产生一个运行时错误)

Modification of the Database

Modification of the Database

- **Deletion** of tuples from a given relation (删除)
- **Insertion** of new tuples into a given relation (插入)
- **Updating** values in some tuples in a given relation (修改)

- Subquery can be used in modification of the database

Deletion

- Try: “delete all tuples in the *instructor* relation for those instructors associated with a department located in the Watson building.”

Deletion

- 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’);
```

Insertion

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

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

Updates

- Recall the basic update statement

```
update instructor
set salary = salary *1.05
where salary < 70000;
```

- Observe the update with subquery

```
update instructor
set salary = salary *1.05
where salary < (select avg (salary)
                from instructor);
```

Updates

- Can you figure out the statement?
  “all instructors with salary over $100,000 receive a 3 percent raise, whereas all others receive a 5 percent raise.”

- Does the following statements works?
  Write two update statements:

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

Updates

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

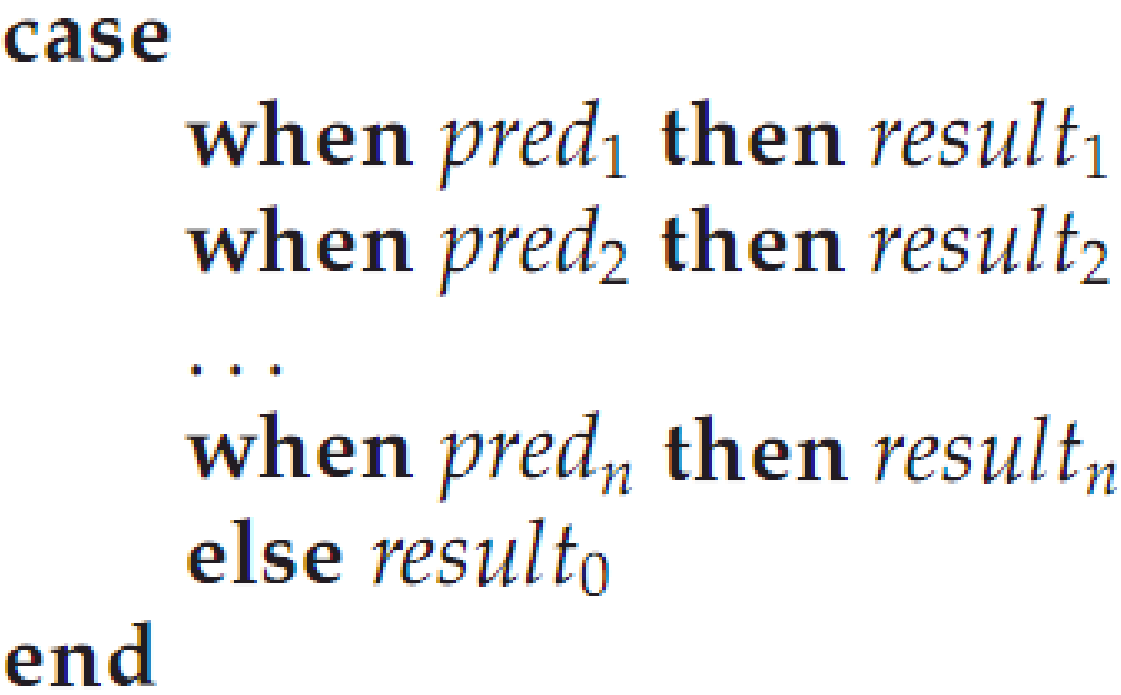
- **The order is important!!!**
- **Can be done better using the case statement**

Case Statement

- “all instructors with salary over $100,000 receive a 3 percent raise, whereas all others receive a 5 percent raise.”

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

Case Statement



```
case
    when $pred_1$ then $result_1$
    when $pred_2$ then $result_2$
    . . .
    when $pred_n$ then $result_n$
    else $result_0$
end
```

Updates

- 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:

```
        case
            when sum(credits) is not null
                then sum(credits)
            else 0
        end
```

Review

Review

- Overview of the SQL Query Language
  - History of SQL
  - Constituent Parts of SQL
  - Implementation of SQL
- SQL Data Definition
  - Functions of DDL
  - Basic Types
  - Create, drop, alter
  - Integrity Constraints in DDL
    - Primary key, foreign key, not null, unique

Review

- SQL Data Manipulation
  - Insert, update, delete
- SQL Data Query
  - Key points
  - Basic Structure of SQL Queries
    - Select-from-where
    - Queries on a Single Relation
    - Queries on Multiple Relations
    - Join
    - Natural Join
    - Understanding the Operational Order

Review

- Additional Basic Operations
  - The Rename Operation
    - as clause
    - Self-join
  - String Operations
    - Like
  - Ordering the Display of Tuples
    - Order by
  - Where Clause Predicates
    - Between … and …

Review

- Set Operations
  - Union, union all, intersect, except
- Null Values
  - is null , is not null
  - comparison with null
- Aggregate Functions
  - Basic Aggregation
    - max, min, avg, sum, count
  - Aggregation with Grouping
  - The Having Clause
  - Aggregation with Null Values

Review

- Nested Subqueries
  - set membership
    - in , not in
  - set comparisons
    - some, any, all
  - Test for Empty Relations
    - exists, not exists
    - Correlated subquery
  - Subqueries in the From Clause
  - The with Clause
  - Scalar Subqueries
- Modification of the Database (with subquery)

Thanks