数据库系统原理

Database System Principle

邵蓥侠

Email：shaoyx@bupt.edu.cn

北京邮电大学计算机学院

计算机应用技术中心

PART 2

RELATIONAL DATABASES

Chapter 3

Introduction to SQL

北京郵電大學

Three Parts in Chapter 3

- ***S***tructured ***Q***uery ***Language***

  most widely used query language in use, commercially, as the standard of query languages

- Usage of SQL
  - data definition: § 3.2, DDL
  - data query/retrieve: § 3.3 —— § 3.8, ***Select*** clause
  - data manipulation: § 3.9 Modification of the Database

§ 3.1 Overview

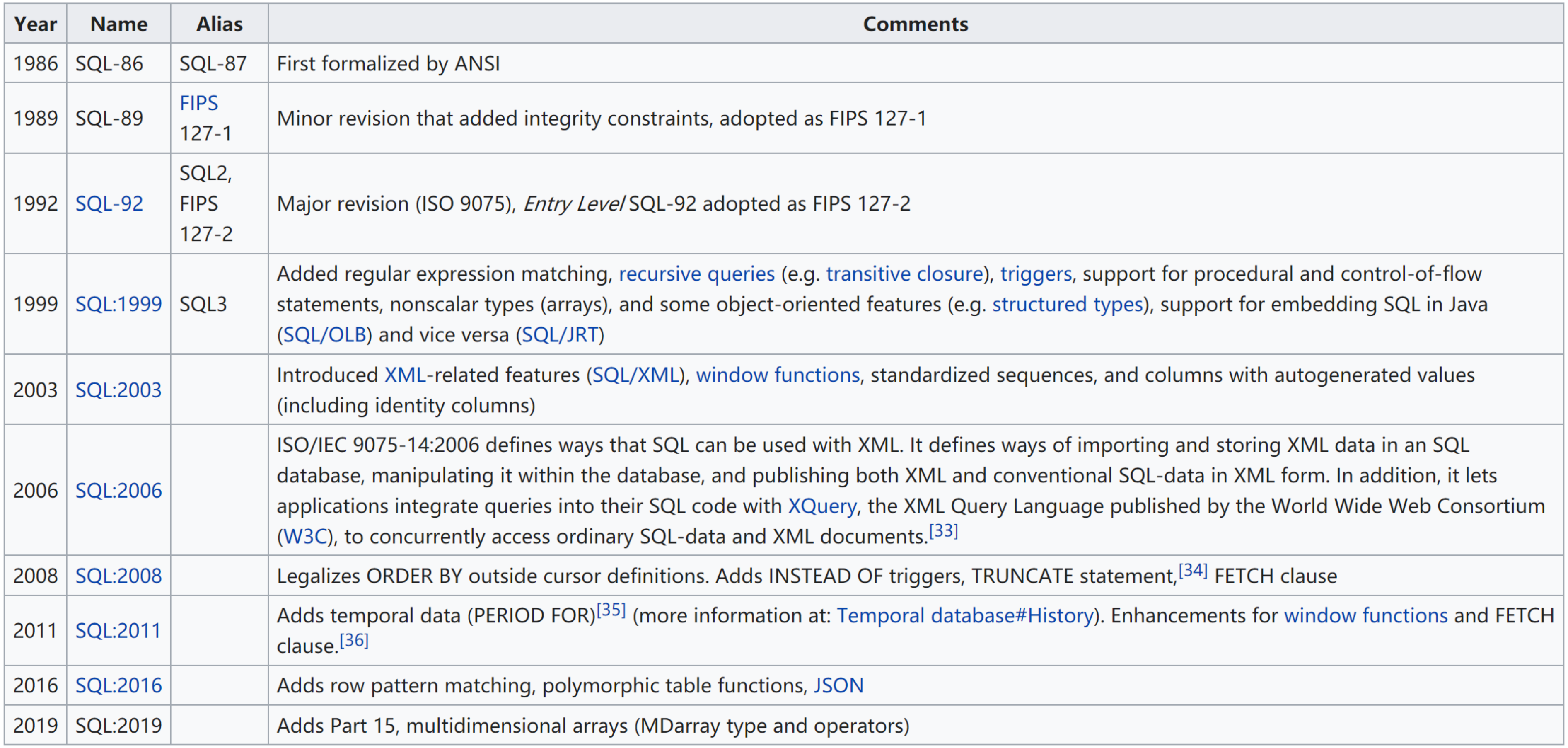
- Database ***query*** languages
  - commercial implementation of *pure* relational ***operations*** in DBS
  - interactive tools for users to *develop* , *use/access*, and *maintain*（维护） DBS
    - DBS access: *create*, *retrieval*, *insert*, *delete*, *update*, etc

- *"Query"*
  - generalized （广义的）definition
  - define, *retrieve*, modify, control etc. on DB

History

- **SQL, Structured Query Language**
  - most widely used query language in use, commercially
  - on the basis of the *relational algebra* （关系代数）and the *tuple relational calculus*（元组关系演算）
- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory in 1970s; Renamed Structured Query Language (SQL)
- As the standard of query languages, ANSI and ISO standard SQL:
  - SQL-86, SQL-89, SQL-92, SQL:1999
  - SQL:2003/2006/2008, with the ability to process **XML** data
  - the recent versions: SQL: 2011/2016/2019

History



Year	Name	Alias	Comments
1986	SQL-86	SQL-87	First formalized by ANSI
1989	SQL-89	FIPS 127-1	Minor revision that added integrity constraints, adopted as FIPS 127-1
1992	SQL-92	SQL2, FIPS 127-2	Major revision (ISO 9075), *Entry Level* SQL-92 adopted as FIPS 127-2
1999	SQL:1999	SQL3	Added regular expression matching, recursive queries (e.g. transitive closure), triggers, support for procedural and control-of-flow statements, nonscalar types (arrays), and some object-oriented features (e.g. structured types), support for embedding SQL in Java (SQL/OLB) and vice versa (SQL/JRT)
2003	SQL:2003		Introduced XML-related features (SQL/XML), window functions, standardized sequences, and columns with autogenerated values (including identity columns)
2006	SQL:2006		ISO/IEC 9075-14:2006 defines ways that SQL can be used with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database, and publishing both XML and conventional SQL-data in XML form. In addition, it lets applications integrate queries into their SQL code with XQuery, the XML Query Language published by the World Wide Web Consortium (W3C), to concurrently access ordinary SQL-data and XML documents.[33]
2008	SQL:2008		Legalizes ORDER BY outside cursor definitions. Adds INSTEAD OF triggers, TRUNCATE statement,[34] FETCH clause
2011	SQL:2011		Adds temporal data (PERIOD FOR)[35] (more information at: Temporal database#History). Enhancements for window functions and FETCH clause.[36]
2016	SQL:2016		Adds row pattern matching, polymorphic table functions, JSON
2019	SQL:2019		Adds Part 15, multidimensional arrays (MDarray type and operators)

https://en.wikipedia.org/wiki/SQL

History

- 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.

Parts in SQL

- Data definition language (DDL)
  - *define*, *delete*, *modify* on *schemas, views* and *index*
  - *view definition*
- Data-manipulation language (DML)
  - *query/retrieve*, *insert* , *delete*, *update* on tuples in DB
- Integrity（完整性）
  - specify integrity constraints in DB
- Transaction management
- authorization: security control, specifying access rights
- embedded and dynamic SQL
  - for database access, by ***application programs***

Parts in SQL

- Basic operators in SQL

data query	Select
data manipulation	Insert, Delete, Update
data definition	Create, Drop, Alter (*on schema*)
data control	Grant, Revoke (*in Chapter 4*)
transaction processing	begin transaction, commit, rollback
指针/游标控制语言（CCL）	DECLARE CURSOR， FETCH INTO和UPDATE WHERE CURRENT

3.1 Background (cont.)

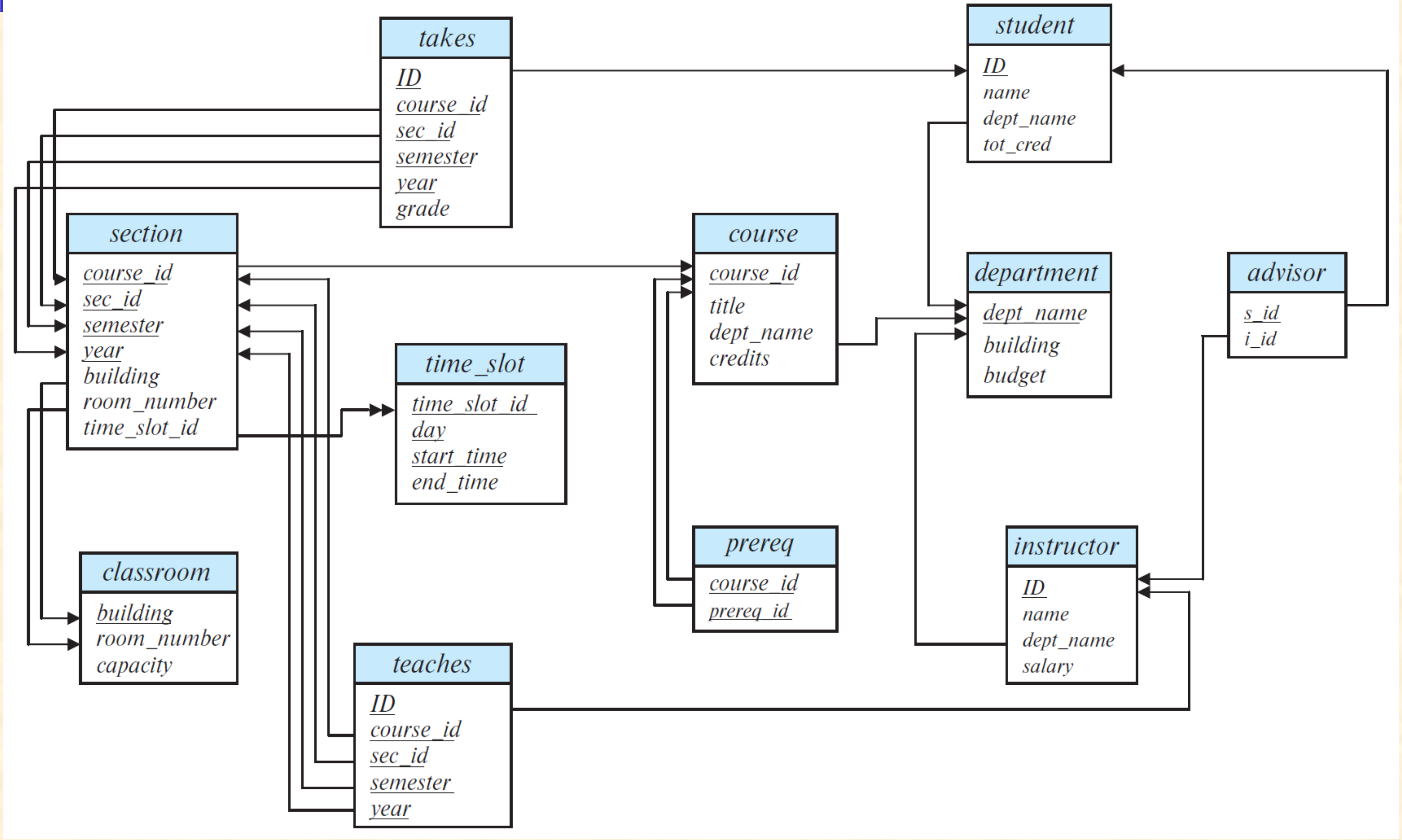


Fig. 3.1 Schema of the ***University*** database

return

§ 3.2 Data Definition

- DDL in SQL allows the specification（规范） of information about each relation, including
  - the *schema* for each relation or view -----***create table/view***
  - 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（索引） 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（定点数）, with user-specified precision of *p* digits（位）, with *d* digits to the right of decimal point（小数点）. (ex., **numeric**(3,1), allows 44.5 to be stores exactly, but not 444.5 or 0.32)

Domain Types in SQL

- **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 4

Date/Time Types in SQL (Cont.) Examples

- date. Dates, containing a (4 digit) year, month and date
  - E.g. **date** ‘2001-7-27’
- **time.** Time of day, in hours, minutes and seconds.
  - E.g. **time** ’09:00:30’ **time** ’09:00:30.75’
- **timestamp**: date plus time of day
  - E.g. **timestamp** ‘2001-7-27 09:00:30.75’
- **Interval**: period of time
  - E.g. Interval ‘1’ day
  - Subtracting a date/time/timestamp value from another gives an interval value
  - Interval values can be added to date/time/timestamp values

- Null values are allowed in all the domain types. Declaring an attribute to be **not null** prohibits null values for that attribute.

- **create domain** construct in SQL-92 creates user-defined domain types 自定义类型
  - **create domain** *person-name* **char**(20) **not null**

注意

- 在Oracle, DB2等大型商用数据库系统中，关系表的属性的名字只能取英文名，不支持中文属性名
- SQL Server支持**中**英文属性名
- 开发大型数据库应用时，程序所访问的表中的属性名字最好取为英文名，便于应用程序的可移植性！！

3.2.2 Schema Definition (cont.)

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

  create table ***r*** $(A_1\ D_1, A_2\ D_2, \ldots, A_n\ D_n,$
  (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 Create Table

- not null
- primary key (A_1, ..., A_n)
- foreign key (A_m, ..., A_n) references *r*
- check (P), where *P* is a predicate

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,
    check(salary >0) );
```

Key 一定不能用空值

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

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

And more still

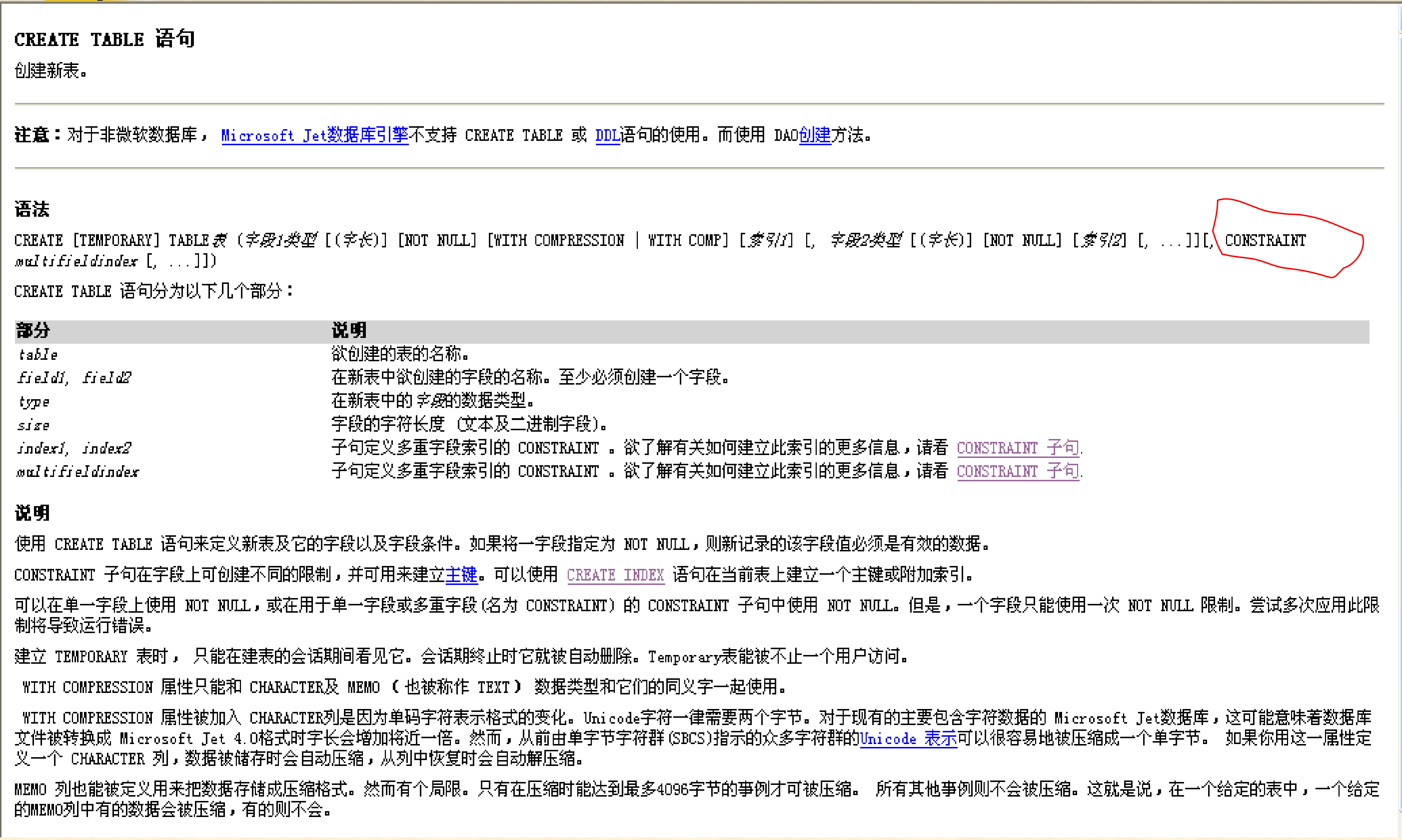
- **create table** *course* (
  ```
        course_id       varchar(8),
        title           varchar(50),
        dept_name       varchar(20),
        credits         numeric(2,0),
         primary key (course_id),
         foreign key (dept_name) references department);
  ```

Schema Definition (cont.)

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

    values (‘Perryridge’, ‘Blooklyn’, 8000)
- The ***delete*** command removes tuples from the table
  - **delete from** *branch*

create table in SQL Server



CREATE TABLE 语句

创建新表。

注意：对于非微软数据库， Microsoft Jet数据库引擎不支持 CREATE TABLE 或 DDL语句的使用。而使用 DAO创建方法。

语法

CREATE [TEMPORARY] TABLE *表* (*字段1类型* [(*字长*)] [NOT NULL] [WITH COMPRESSION | WITH COMP] [*索引1*] [, *字段2类型* [(*字长*)] [NOT NULL] [*索引2*] [, ...]][, CONSTRAINT *multifieldindex* [, ...]])

CREATE TABLE 语句分为以下几个部分：

部分	说明
table	欲创建的表的名称。
field1, field2	在新表中欲创建的字段的名称。至少必须创建一个字段。
type	在新表中的 *字段*的数据类型。
size	字段的字符长度 (文本及二进制字段)。
index1, index2	子句定义多重字段索引的 CONSTRAINT 。欲了解有关如何建立此索引的更多信息，请看 CONSTRAINT 子句.
multifieldindex	子句定义多重字段索引的 CONSTRAINT 。欲了解有关如何建立此索引的更多信息，请看 CONSTRAINT 子句.

说明

使用 CREATE TABLE 语句来定义新表及它的字段以及字段条件。如果将一字段指定为 NOT NULL，则新记录的该字段值必须是有效的数据。

CONSTRAINT 子句在字段上可创建不同的限制，并可用来建立主键。可以使用 CREATE INDEX 语句在当前表上建立一个主键或附加索引。

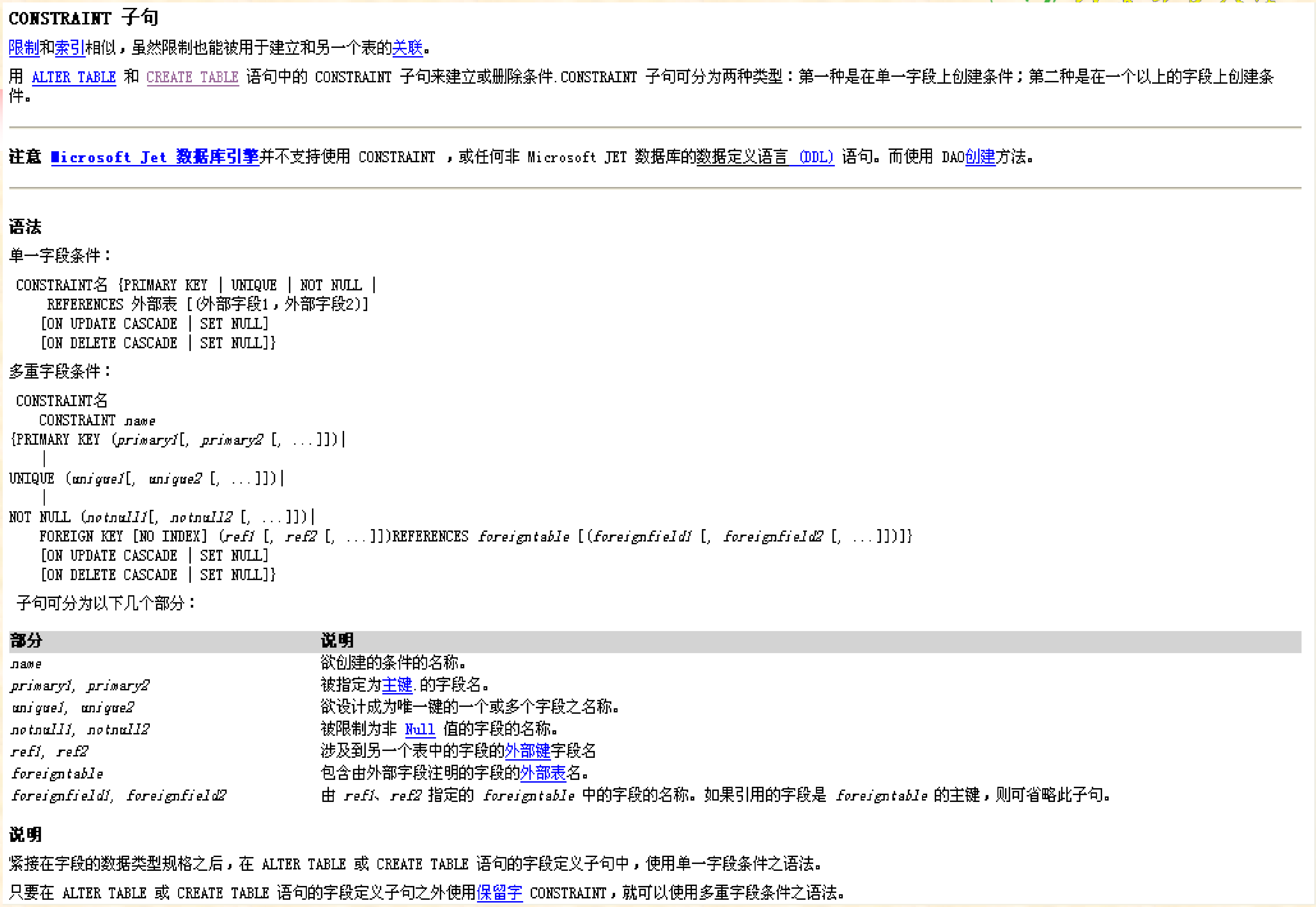
可以在单一字段上使用 NOT NULL，或在用于单一字段或多重字段(名为 CONSTRAINT) 的 CONSTRAINT 子句中使用 NOT NULL。但是，一个字段只能使用一次 NOT NULL 限制。尝试多次应用此限制将导致运行错误。

建立 TEMPORARY 表时， 只能在建表的会话期间看见它。会话期终止时它就被自动删除。Temporary表能被不止一个用户访问。

WITH COMPRESSION 属性只能和 CHARACTER及 MEMO （也被称作 TEXT） 数据类型和它们的同义字一起使用。

WITH COMPRESSION 属性被加入 CHARACTER列是因为单码字符表示格式的变化。Unicode字符一律需要两个字节。对于现有的主要包含字符数据的 Microsoft Jet数据库，这可能意味着数据库文件被转换成 Microsoft Jet 4.0格式时字长会增加将近一倍。然而，从前由单字节字符群(SBCS)指示的众多字符群的Unicode 表示可以很容易地被压缩成一个单字节。 如果你用这一属性定义一个 CHARACTER 列，数据被储存时会自动压缩，从列中恢复时会自动解压缩。

MEMO 列也能被定义用来把数据存储成压缩格式。然而有个局限。只有在压缩时能达到最多4096字节的事例才可被压缩。 所有其他事例则不会被压缩。这就是说，在一个给定的表中，一个给定的MEMO列中有的数据会被压缩，有的则不会。



CONSTRAINT 子句

限制和索引相似，虽然限制也能被用于建立和另一个表的关联。

用 ALTER TABLE 和 CREATE TABLE 语句中的 CONSTRAINT 子句来建立或删除条件.CONSTRAINT 子句可分为两种类型：第一种是在单一字段上创建条件；第二种是在一个以上的字段上创建条件。

注意 **Microsoft Jet 数据库引擎**并不支持使用 CONSTRAINT ，或任何非 Microsoft JET 数据库的数据定义语言 (DDL) 语句。而使用 DAO创建方法。

语法

单一字段条件：

```
 CONSTRAINT名 {PRIMARY KEY | UNIQUE | NOT NULL |
     REFERENCES 外部表 [(外部字段1，外部字段2)]
    [ON UPDATE CASCADE | SET NULL]
    [ON DELETE CASCADE | SET NULL]}
```

多重字段条件：

```
 CONSTRAINT名
    CONSTRAINT name
{PRIMARY KEY (primary1[, primary2 [, ...]])|
     |
UNIQUE (unique1[, unique2 [, ...]])|
     |
NOT NULL (notnull1[, notnull2 [, ...]])|
    FOREIGN KEY [NO INDEX] (ref1 [, ref2 [, ...]])REFERENCES foreigntable [(foreignfield1 [, foreignfield2 [, ...]])]}
    [ON UPDATE CASCADE | SET NULL]
    [ON DELETE CASCADE | SET NULL]}
```

子句可分为以下几个部分：

部分	说明
name	欲创建的条件的名称。
primary1, *primary2*	被指定为主键.的字段名。
unique1, *unique2*	欲设计成为唯一键的一个或多个字段之名称。
notnull1, *notnull2*	被限制为非 Null 值的字段的名称。
ref1, *ref2*	涉及到另一个表中的字段的外部键字段名
foreigntable	包含由外部字段注明的字段的外部表名。
foreignfield1, *foreignfield2*	由 *ref1*、*ref2* 指定的 *foreigntable* 中的字段的名称。如果引用的字段是 *foreigntable* 的主键，则可省略此子句。

说明

紧接在字段的数据类型规格之后，在 ALTER TABLE 或 CREATE TABLE 语句的字段定义子句中，使用单一字段条件之语法。

只要在 ALTER TABLE 或 CREATE TABLE 语句的字段定义子句之外使用保留字 CONSTRAINT，就可以使用多重字段条件之语法。

Schema Definition (cont.)

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

§ 3.2.2 Schema Definition (cont.)

- **alter table** *r* **drop** *A*
  - where *A* is the name of an attribute of relation *r*
  - e.g. **alter table** *instructor* **drop** *age*

  - **dropping of attributes not supported by some databases**

§ 3.3 Basic structure — *Select* query

- 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 attributes
  - r_i represents relations
  - P is a predicate（谓词）

query result :

	A_1	A_2	…	…	A_n
t_1	*	*	*	*	*
t_2	*	*	*	*	*
	*	*	*	*	*
	*	*	*	*	*
t_k	*	*	*	*	*

- This query is equivalent to the relational algebra expression

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

  - the result of an SQL query is a relation

- For more details about complete syntax definition for ***Select*** clause, *refer to Appendix A. Syntax Definition of Select Clause*

31. Query on a Single Relation ---The *select*, *where* 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*

  - in the “pure” relational algebra syntax, the query would be

$$\Pi_{\text{name}}(\textit{instructor})$$

- 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（星号） 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（算数） 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 ***temporal*** 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
  - corresponding to the *selection predicate p in the relational algebra operation* σ_p

- E.g. To find all instructors in Comp. Sci. dept

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

The where Clause

- The **where** clause specifies conditions that the result must satisfy
  - corresponding to the *selection predicate p in the relational algebra operation* σ_p
- 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, e.g. *age* mod10=0.

3.3.2 Query on multiple relations ——The *from* Clause

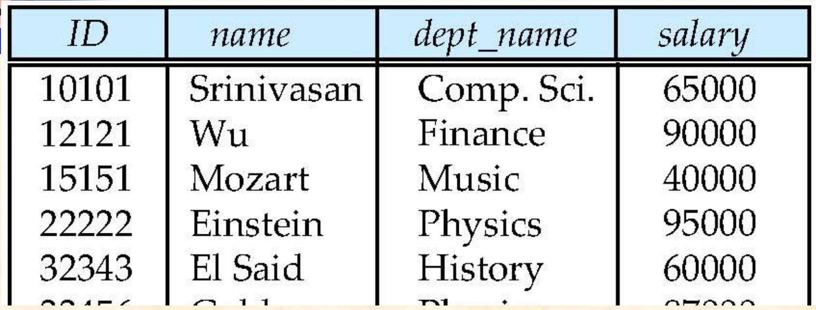
- The **from** clause lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra.
- Find the Cartesian product *instructor X teaches*

  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).

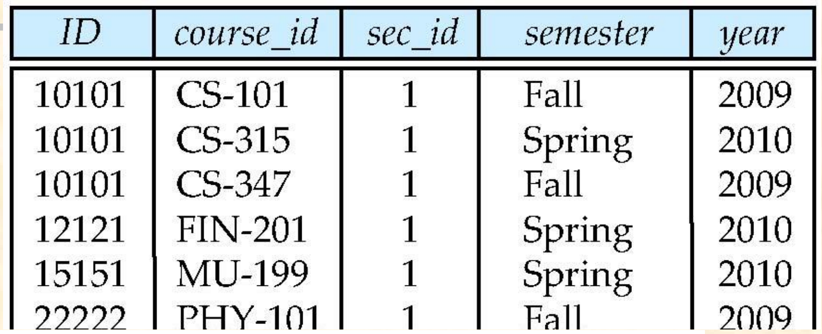
Cartesian Product

instructor

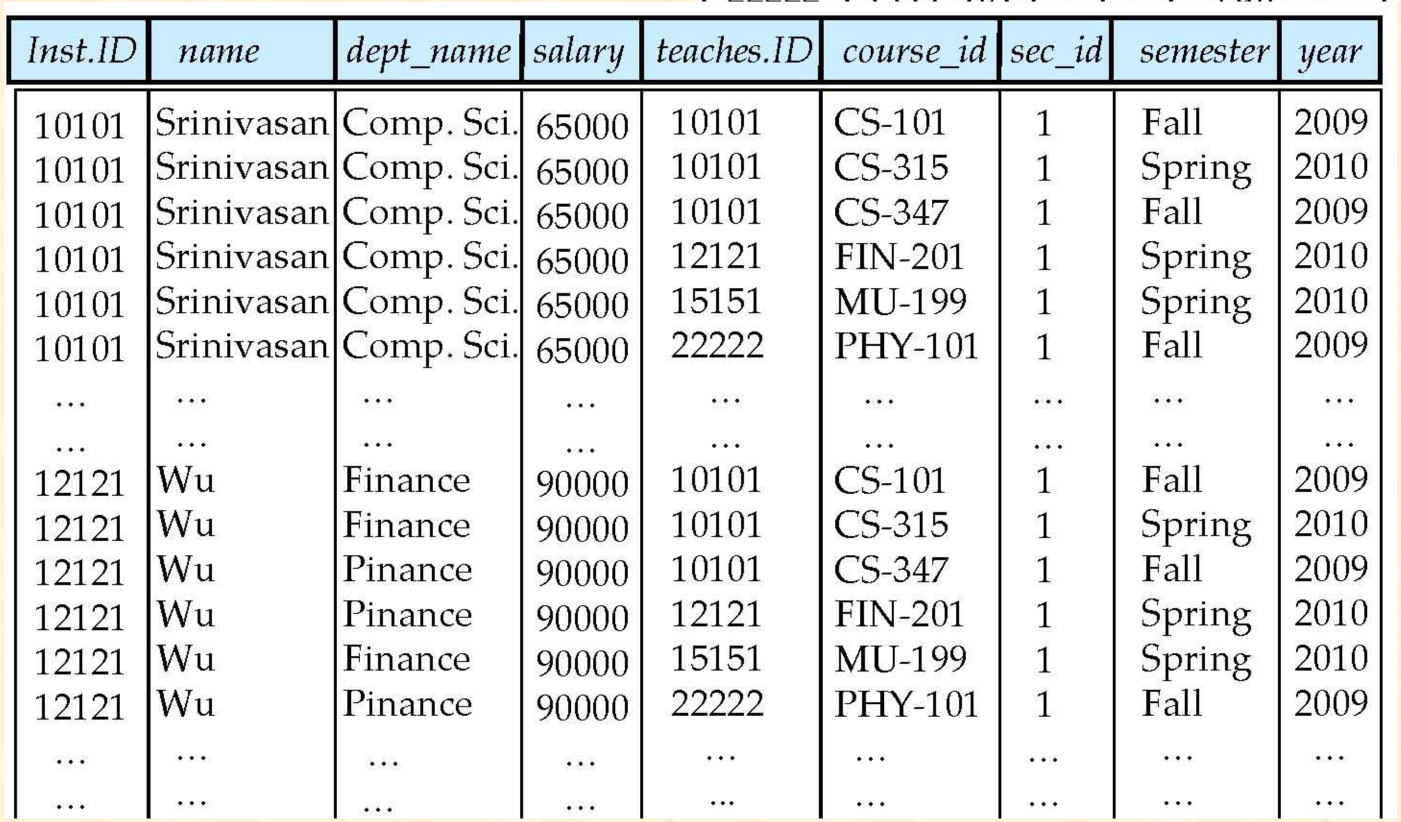


ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000

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'

Query on multiple relations (cont.)

- 注意事项
  - ***from*** 子句中包括多个关系表时，一定要在*where*子句中加入连接条件！

    防止出现耗时费力的多表笛卡尔积操作
  - 实际应用中，对频繁执行的SQL查询，其*from*子句中的表的个数不要过多，如不要超过4个！

    避免耗时费力的多表连接操作。

    如果频繁执行的SQL查询涉及的查询数据存放在$N \geq 4$张表中，可以考虑将这N张表中的数据进行合并

3.3.3 natural join

- SQL also supports "*natural join*" in the *from* subclause
  - E.g. 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* ***natural join*** *teaches*
where *instructor. dept_name* = 'Art'

3.4 Additional Basic Operations

3.4.1 The *Rename* Operation

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

  old-name **as** *new-name*

- E.g. **select** *name* ***as*** *instructor_name*
  from *instructor*

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

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

Tuple Variables

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

- Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.
  - **select distinct** *T.name*
    from *instructor* **as** *T*, *instructor* **as** *S*
    where *T.salary* > *S.salary* **and** *S.dept_name* = *'Comp. Sci.'*

Tuple Variables (cont.)

- E.g.2 Find the ***names*** of all ***instructors*** whose salary is greater than at least one instructor in the Biology department
  - /*涉及到对*instructor*关系中属性*salary*的不同值的比较
  - **select distinct** *T.name*
    from *instructor* **as** *T*, *instructor* **as** *S*
    where *T. salary* > *S. salary* **and** *S.dept_name* = 'Biology'

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

3.4.2 String Operations

- 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（转义) character.

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（串接） (using “||”)
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, etc.

3.4.3 Attribute Specification

- The asterisk symbol "*" can be used in the *select* clause to denote "all attributes"
  - **select** *
    from *instructor* , *teaches*
    where *instructor.ID* = *teaches.ID*
  - **select** *instructor.**
    from *instructor* , *teaches*
    where *instructor.ID* = *teaches.ID*

3.4.4 Ordering the Display of Tuples

- The ***order by*** clause causes the tuples in the result of query to appear in sorted order, i.e. *ascending* order or *descending* order, denoted by ***keywords*** ***asc*** or ***desc*** respectively
  - *ascending* order is the default
- 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** 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’);

Duplicates

- Viewing relations with duplicates as multisets, SQL can define not only what tuples will appear in the result of a query, but also how many copies of these tuples appear in the result
- The duplicate semantics of *select* clause is based on multiset versions of the relational algebra operators
- SQL不主动删除重复，除非用distinct
- For more details, refer to *Appendix C Duplicates in Select Clause*

§ 3.5 Set Operations

- In SQL, the set operations **union, intersect,** and **except** operate on relations and correspond to the relational algebra operators $\cup$, $\cap$, and –
  - each of the above operations ***automatically eliminates duplicates***
  - to retain（保留） all duplicates use the corresponding multiset versions ***union all***, ***intersect all*** and ***except all***

- 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)

Set Operations (cont.)

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)

Set Operations (Cont.)

To find the largest salary of all instructors,

- Step1. Find the salaries of all instructors that are less than the largest salary.
  - **select distinct** *T.salary*
    from *instructor* **as** *T*, *instructor* **as** *S*
    where *T.salary* < *S.salary*
- Step2. Find all the salaries of all instructors
  - **select distinct** *salary*
    from *instructor*
- Step3.
  - (**select** "second query")
    except
    (**select** "first query")

§ 3.6 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 predicate ***is null*** can be used in ***Select*** clause to check for null values
- E.g. Find all ***instructors*** who appear in the *instructor* relation with *null* value for *salary*
  - **select** *name*
    from *instructor*
    where *salary* **is null**
- The result of any *arithmetic expression* involving ***null*** is ***null***
  - e.g. 5 + null returns null

北京邮电大学

Null Values (cont.)

- Any comparison with *null* returns *unknown*
  - e.g. *5 < null*, *null <> null* , *null = null*
- In *where* clause, three-valued logic using the truth value *unknown* (i.e. *null, introduced in SQL 1999*)
  - 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

Null Values (cont.)

- NOT
  - (**not** *unknown*) = *unknown*
- "*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*

- Result of **where** clause predicate P is treated as *false* if it evaluates to *unknown*

- Null values and aggregates
  - all aggregate operations **except count(*)** ignore tuples with *null* values on the aggregated attributes

Null Values (cont.)

- E.g. Total all ***instructor salaries***

  select sum (*salary*)
  from *instructor*

  - if there are ***null*** values in ***salary*** attributes, above statement ignores ***null*** amounts
  - the result is ***null*** if there is no non-null *salary*

聚集一般忽略空值

惟一例外： **count(*)** 计算元组个数，含空值的元组也计数

§ 3.7 Aggregate Functions

- The following aggregate functions operate on the set (*or 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
  - these functions corresponds to

$$_{G1,\ G2,\ \ldots,\ Gn}\ \mathcal{G}\ _{F1(A1),\ F2(A2),\ldots,\ Fn(An)}\ (E)$$

Aggregate Functions (cont.)

- How to conduct *group by* subclause and *aggregate* function

Select {$A_1, A_2, ., A_i$}, ***ag_fun***(A_{i+1}),…, ***ag_fun***(A_{i+k})
From $r_1, r_2, \ldots, r_m$
Where *P1*
{ Group by $A_1, A_2, ., A_i$
{ having *P2* } }

- notes:
  - *P1* is defined on all attributes of $r_1, r_2, \ldots, r_m$
  - *P2* is defined on $A_1, A_2, ., A_i, .., A_j, \ldots, A_k, \ldots, A_n$, as the constraints on the group formed by **Group by** A_1, A_2, A_j
  - attributes in the *select* clause outside of aggregate functions must appear in group by list

北京邮电大学

Aggregate Functions (cont.)

- step1. temporal relation ***T1*** is derived from

  Select $A_1, A_2, \ldots, A_i, \ldots, A_j, \ldots, A_{i+k}$
  From $r_1, r_2, \ldots, r_m$
  Where $P1$

- step2. *T1* is grouped on $A_1, A_2, \ldots, A_i$, resulting in temporal relation *T2*
- step3.

  if *having* subclause exists,

  then **the group**, which is not in accordance with *P2*, is filtered out of *T2*

- step4. for each group, conduct ***ag_fun*** on $A_{i+1}, \ldots, A_{i+k}$

  Select $A_1, A_2, \ldots, A_i$, ***ag_fun***$(A_{i+1}), \ldots,$ ***ag_fun***(A_{i+k})
  From *T2*

Aggregate Functions (cont.)

E.g.1 Find the average salary of instructors in the Computer Science department

```
select avg (salary)
from instructor
where dept_name= ’ Comp. Sci.’;
  /*定义在多重集合上的计算
```

ID	*name*	*dept_name*	*salary*
45565	Jian	Comp. Sci	$100
74281	Ye	Comp. Sci	$250
98753	Du	Comp. Sci	$340
54123	Lin	Comp. Sci	$200

$\}$ **avg**

3.5 Aggregate Functions (cont.)

- E.g.2 Find the number of tuples in the *course* relation
  - **select count** (*)
    from *course*

- E.g.3 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/**删除重复元组*

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

注意！

- 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*;

Aggregate Functions – 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

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 (嵌套) Subqueries

嵌套查询： 中间结果，便于构思化大为小， 化繁为简。

DBMS实现时，要优化，不一定要保留中间值

- SQL provides a mechanism for the nesting of subqueries, to implement more complex query
- A subquery is a *select-from-group* 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.

Nested (嵌套) Subqueries

- The subquery is often nested in the *where clause/having clause* , *from clause*

- Subquery in the *where clause/having ,* perform tests for
  - *set membership*
  - *set comparisons*
  - *set cardinality*
  - *duplicate tuples*

3.8.1 Set Membership

- 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);
```

语句执行过程：
1)从头至尾，依次扫描*section*中每一行；
2)对*section*中每一行，
  a. 判断其semester、year是否满足查询条件
  b. 执行嵌入在where子句中的*select*查询，得到10年春季的*course_id*集合；
3) 判断本行的*course_id*是否在此集合中，决定本行的*course_id*是否所需查询结果

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

- 说明：上述语句的执行效率较低，why?!

  被嵌入的select子句可能多次重复执行
- 实际应用中应尽量避免membership操作，参见“数据库物理设计及查询优化”

Set Membership

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.

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’);
```

3.8.2 Set Comparison – "some" Clause

- SQL allows < **some/all**, <= **some/all**, >= **some/all**, =**some/all**, and <>**some/all** comparisons
  - for more details, refer to *Appendix D Some/All Set Comparisons* ▷

Set Comparison-all (cont.)

- 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’);
```

3.8.3 Test for Empty Relations

- E.g. Find all ***students*** who take all the courses that takes by ZhangLi
  /查询选修了“ZhangLi”所选修的全部课程的同学
  - 方法**:**

  1. 计算 ZhangLi选修课程集合*X*
  2. 对每个学生，计算他选修的课程集合***Y***
  3. 判断***Y***是否包含X

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

Test for Empty Relations(cont.)

- 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);
```

- **Correlation name（相关名称）** – variable S in the outer query
- **Correlated subquery（相关子查询）** – 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 $X - Y = \emptyset \;\Leftrightarrow\; X \subseteq Y$

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.
- The **unique** construct evaluates to “true” if a given subquery contains no duplicates
  - /* 如果作为参数的子查询结果中无重复元组，则unique 返回true
- 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);
```

3.8.5 Subqueries in the *From* Clause

- SQL allows a subquery expression to be used 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;
```

- 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;
```

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
    - /* 将1个复杂查询分解为若干步，每个视图定义1个各步的中间计算结果

- Find all departments with the maximum *budget*

找出所有最高budget的department（可能不只一个

```
with max_budget (value) as
        (select max(budget)
         from department)
select department.name
from department, max_budget
where department.budget = max_budget.value;
```

视图max-budget(value) = { 200000}, e.g.

定义局部视图

//取子集作为中间逻辑结果

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;
```

北京邮电大学

3.8.7 Scalar（标量） Subquery

——Subqueries in the *Select* Clause

- 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

Deletion

- 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 instructor, 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)

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*);

Insertion (Cont.)

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

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

- 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

Updates

- 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 (next slide)

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
```

Updates with Scalar（标量） Subqueries

- Recompute and update tot_creds value for all students

```
update student S
 set tot_cred = (select sum(credits)
                 from takes, course
                 where takes.course_id = course.course_id and
                        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
```

Update on Multiple Tables！！

务必掌握，实际中应用很多!

- 不同DBMS平台下的多表update语句语法定义不相同

- *Student*(S#, Sname, age …, C#, Grade, …),

  Sgrade(S#, C#, Grade)

  利用*Sgrade*的内容，更新*Student*

  update *Student* as A

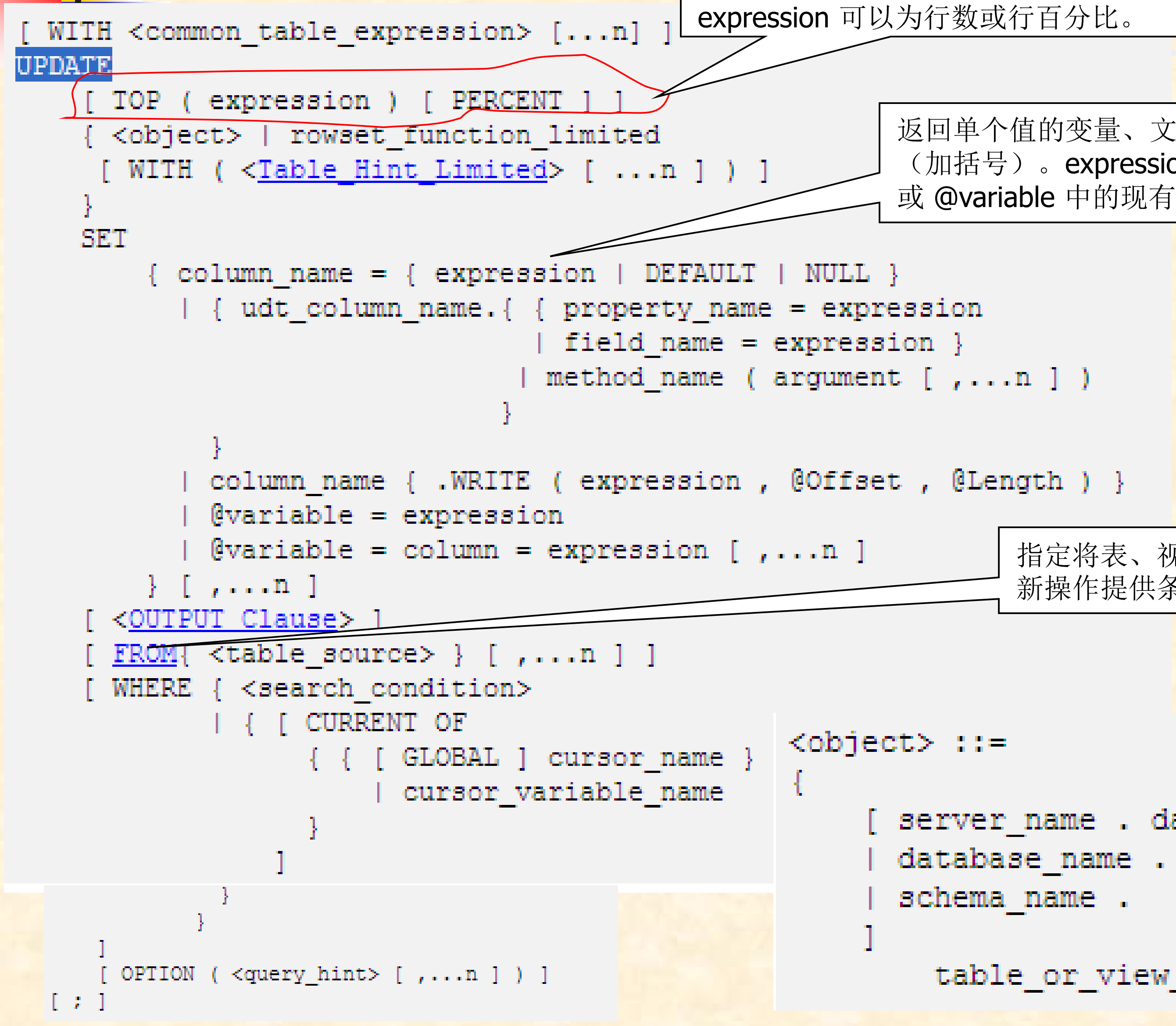
  set *Grade*=B.Grade

  from *Sgrade* as B

  where A.S#=B.S# and A.C#=B.C#

Update in SQL Server

北京郵電大學

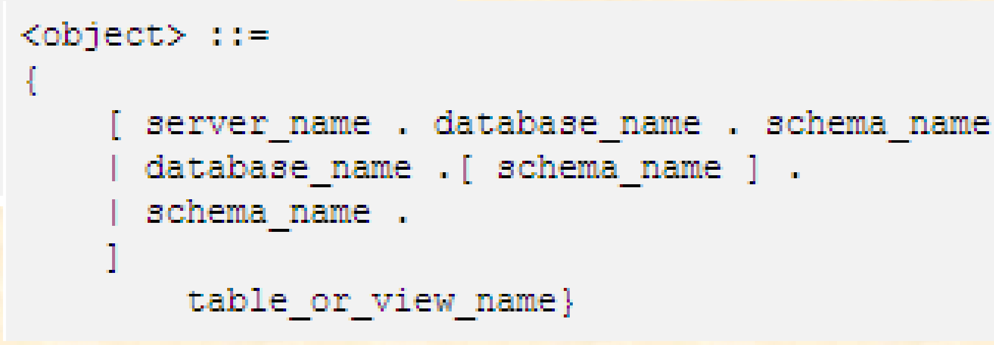


```
[ WITH <common_table_expression> [...n] ]
UPDATE
    [ TOP ( expression ) [ PERCENT ] ]
    { <object> | rowset_function_limited
     [ WITH ( <Table_Hint_Limited> [ ...n ] ) ]
    }
    SET
        { column_name = { expression | DEFAULT | NULL }
          | { udt_column_name.{ { property_name = expression
                                | field_name = expression }
                               | method_name ( argument [ ,...n ] )
                              }
            }
          | column_name { .WRITE ( expression , @Offset , @Length ) }
          | @variable = expression
          | @variable = column = expression [ ,...n ]
        } [ ,...n ]
    [ <OUTPUT Clause> ]
    [ FROM{ <table_source> } [ ,...n ] ]
    [ WHERE { <search_condition>
            | { [ CURRENT OF
                  { { [ GLOBAL ] cursor_name }
                      | cursor_variable_name
                  }
                ]
              }
            }
    ]
    [ OPTION ( <query_hint> [ ,...n ] ) ]
[ ; ]
```

指定将要更新的行数或行百分比。
expression 可以为行数或行百分比。

返回单个值的变量、文字值、表达式或嵌套 select 语句（加括号）。expression 返回的值替换 column_name 或 @variable 中的现有值。

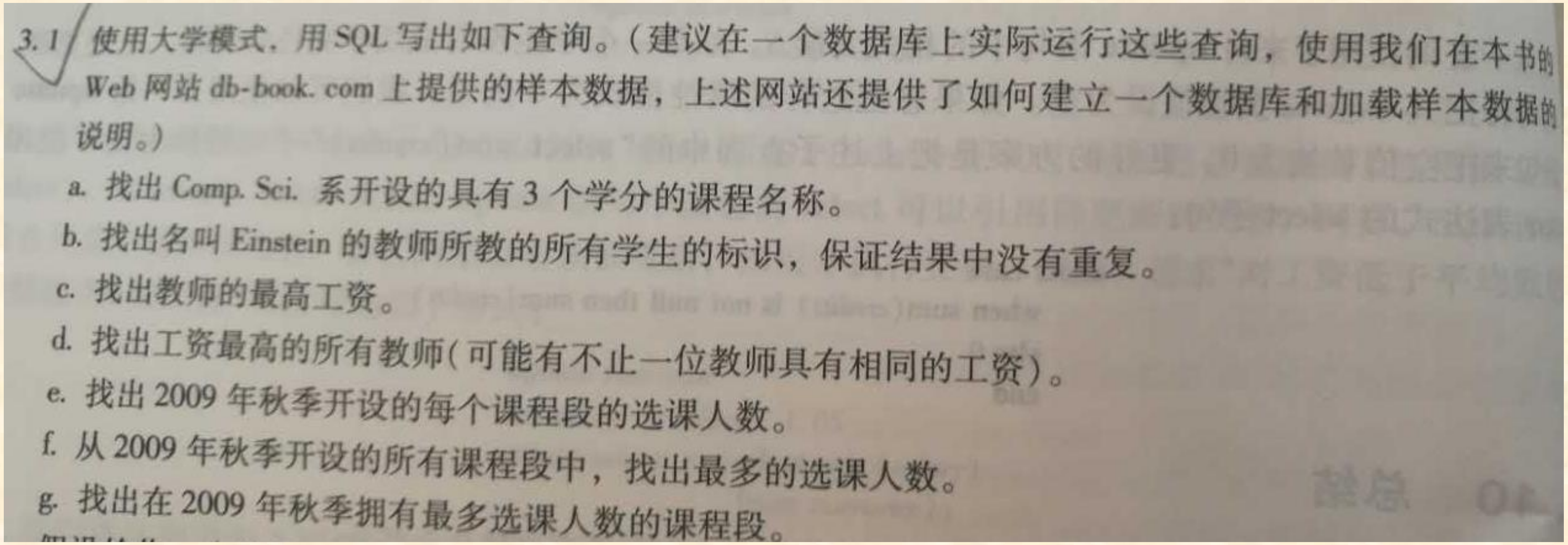
指定将表、视图或派生表源用于为更新操作提供条件。



```
<object> ::=
{
    [ server_name . database_name . schema_name
    | database_name .[ schema_name ] .
    | schema_name .
    ]
        table_or_view_name}
```

北京邮電大學

Excises



3.1 使用大学模式，用 SQL 写出如下查询。(建议在一个数据库上实际运行这些查询，使用我们在本书的 Web 网站 db-book. com 上提供的样本数据，上述网站还提供了如何建立一个数据库和加载样本数据的说明。)

a. 找出 Comp. Sci. 系开设的具有 3 个学分的课程名称。

b. 找出名叫 Einstein 的教师所教的所有学生的标识，保证结果中没有重复。

c. 找出教师的最高工资。

d. 找出工资最高的所有教师(可能有不止一位教师具有相同的工资)。

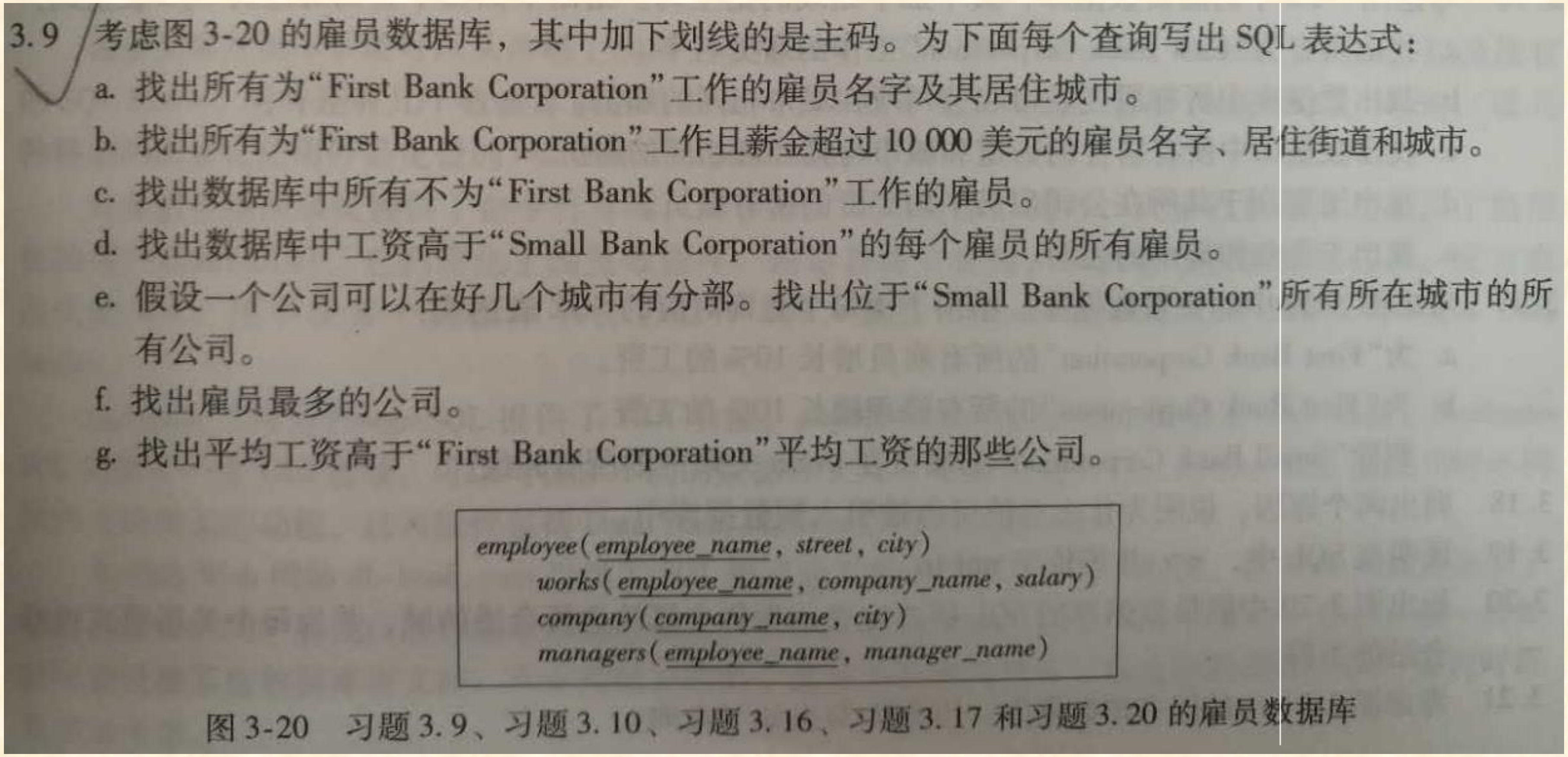
e. 找出 2009 年秋季开设的每个课程段的选课人数。

f. 从 2009 年秋季开设的所有课程段中，找出最多的选课人数。

g. 找出在 2009 年秋季拥有最多选课人数的课程段。

北京邮電大學

Excises


3.9 考虑图 3-20 的雇员数据库，其中加下划线的是主码。为下面每个查询写出 SQL 表达式：

a. 找出所有为“First Bank Corporation”工作的雇员名字及其居住城市。

b. 找出所有为“First Bank Corporation”工作且薪金超过 10 000 美元的雇员名字、居住街道和城市。

c. 找出数据库中所有不为“First Bank Corporation”工作的雇员。

d. 找出数据库中工资高于“Small Bank Corporation”的每个雇员的所有雇员。

e. 假设一个公司可以在好几个城市有分部。找出位于“Small Bank Corporation”所有所在城市的所有公司。

f. 找出雇员最多的公司。

g. 找出平均工资高于“First Bank Corporation”平均工资的那些公司。

employee(*employee_name*, *street*, *city*)
works(*employee_name*, *company_name*, *salary*)
company(*company_name*, *city*)
managers(*employee_name*, *manager_name*)

图 3-20　习题 3.9、习题 3.10、习题 3.16、习题 3.17 和习题 3.20 的雇员数据库

Appendix A
Syntax Definition of Select Clause

- The ***Select*** clause is defined as

```
Select {ALL | DISTINCT } <column expression1> {,
                                      <column expression1> }
From   <table_name or view_name> {,
                                   <table_name or view_name> }
{ Where  < conditional expression1 > }
{ Group by  < column_name1 >
                   HAVING{ < conditional expression2 > } }
{ Order by  <column_name2 >   {ASC | DESC } }
```

Appendix A
Syntax Definition of Select Clause (cont.)

- The sub-clauses in the Select clause correspond to relational algebra operations
  - *select* sub_clause --- projection
  - *from* sub_clause ---Cartesian product
  - *where* sub_clause --- select predicate
  - *group* sub_clause --- group and aggregate functions (§ 3.4)
  - *order* sub_clause --- ordering query result

Appendix A
Syntax Definition of Select Clause (cont.)

- The computation procedure of ***Select*** clause is as follows
  - selecting tuples that satisfy conditions defined by *where* sub_clause from base tables or views listed in *from* sub_clause
  - according to column expression listed in *select* sub_clause , selecting attributes from these tuples , and forming a ***result table*** as query result
  - if group sub_clause exist, grouping the result table on the basis of conditional expression2 , conducting *group/aggregate functions* on each groups, and outputting every groups as query results.

Appendix A
Syntax Definition of Select Clause (cont.)

- if **HAVING** appears, only outputting groups that satisfy *column_name1* as query results.
- if order sub_clause exist, sorting the result table or groups in descending or ascending orders.
- if distinct appears in *select* sub_clause , eliminating duplicates of tuples in the result table.

Appendix B
String Operations in SQL

- SQL specifies strings by enclosing them in single quotes, e.g. ‘Perryridge’
- SQL introduces a series of operations on strings, of which the most commonly used operation is pattern-matching using the operator ***like*** for comparisons on character strings
- Patterns are described using two special characters
  - percent (%). The % character matches any substring
  - underscore (_). The _ character matches any character
- E.g. find the ***names*** of all ***customers*** whose ***street*** includes the substring “***Main***”.

Appendix B
String Operations in SQL (cont.)

select *customer-name*
from *customer*
where *customer-street* **like** ‘%Main%’

- Match the name “Main%”

  like ‘Main\%’ **escape** ‘\’

  - ***escape***: 定义转义字符‘\’，匹配所有以 “Main%”开头的字符串

- SQL also 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.

Appendix C Duplicates in Select Clause

- In relations with duplicates, SQL can define not only what tuples will appear in the result of a query, but also how many copies of these tuples appear in the result
- The duplicate semantics of select clause is based on multiset versions of the relational algebra operators
- Given multiset relations r_1 and r_2
  - $\boldsymbol{\sigma_{\theta}(r_1)}$
    - if there are c_1 copies of tuple t_1 in r_1, and t_1 satisfies selections σ_{θ}, then there are c_1 copies of t_1 in $\sigma_{\theta}(r_1)$
  - $\boldsymbol{\Pi_A(r)}$
    - for each copy of tuple t_1 in r_1, there is a copy of tuple $\Pi_A(t_1)$ in $\Pi_A(r_1)$ where $\Pi_A(t_1)$ denotes the projection of the single tuple t_1.

Appendix C
Duplicates in Select Clause (cont.)

- $\boldsymbol{r_1}$ **x** $\boldsymbol{r_2}$
  - 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

- E.g. suppose multi-set relations r_1 (A, B) and r_2 (C) are as follows:

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

  , then $\Pi_B(r_1)$ would be {(a), (a)}, while $\Pi_B(r_1) \times r_2$ would be

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

Appendix C
Duplicates in Select Clause (cont.)

- SQL duplicate semantics

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

  - is equivalent to the expression

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

    - using the *multiset* version of relational operators Π, σ, and $\times$

Appendix D
Some/All Set Comparisons

- Definition of ***some*** set comparisons
  - F <comp> **some** *r* ⇔ ∃ *t* ∈ *r* (F <comp> *t*)
    - where <comp> can be: <, ≤, >, =, ≠
  - e.g.

$$(5 < \textbf{some} \begin{array}{c} r \\ \hline \multicolumn{1}{|c|}{0} \\ \hline \multicolumn{1}{|c|}{5} \\ \hline \multicolumn{1}{|c|}{6} \\ \hline \end{array}) = \text{true} \quad (5 < \text{some tuple in the relation})$$

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

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

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

Appendix D
Some/All Set Comparisons (cont.)

- (= **some**) ≡ **in**
- however, (≠ **some**) ≡ **not in**
- **> some** : greater than at least one

- Definition of ***all*** set comparisons
  - F <comp> **all** $r \Leftrightarrow \forall\ t \in r$ (F <comp> t)
    - where <comp> can be: $<, \leq, >, =, \neq$
- (≠ **all**) ≡ **not in**
- however, (= **all**) is not equal to **in**

Appendix D
Some/All Set Comparisons (cont.)

- E.g.

(5< **all** $\begin{array}{|c|}\multicolumn{1}{c}{r}\\\hline 0\\\hline 5\\\hline 6\\\hline\end{array}$) = false

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

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

(5 ≠ **all** $\begin{array}{|c|}\hline 4\\\hline 6\\\hline\end{array}$) = true (5 ≠ 4 and 5 ≠ 6)

Have a break