西安交通大學管理学院
THE SCHOOL OF MANAGEMENT
XI'AN JIAOTONG UNIVERSITY

2 SQL语句基础

{课本第三章}

刘跃文 博士，副教授
西安交通大学管理学院
信息管理与电子商务系
liuyuewen@mail.xjtu.edu.cn

V1, 2019-9-16

Outline 提纲

1. 创建表：
   创建表、主键约束、外键约束
2. 查询数据：
   简单查询、连接查询、聚合查询、嵌套查询
3. 修改数据
   增、删、改

3.1 SQL的历史

- IBM San Jose Research Laboratory开发的R系统项目中的一部分：IBM顺序语言，之后重命名为 **Structured Query Language (SQL，结构化查询语言，读作sikəu)**
- ANSI 和 ISO 标准 SQL:
  - SQL-86, SQL-89, SQL-92
  - SQL:1999, SQL:2003, SQL:2008
- 商业数据库系统一般提供**SQL-92**标准中的全部或者大部分语句，加上其它一些标准中的语句，以及一些特有的语句
  - **不同的数据库管理系统，其SQL有一定的差异**
  - **书上的例子在一些数据库管理系统中可能不能运行**

- **DDL** Data Definition Language
- **DML** Data Manipulation Language

3.2 数据定义语言
Data Definition Language

- The SQL **data-definition language (DDL)** allows the specification of information about relations, including:

- The schema 模式 for each relation.
- The domain of values associated with each attribute.
- 完整性约束 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.

- DDL的核心目的：创建表、修改表（定义数据的模式）

3.2.2 Create Table Construct 创建表 [掌握]

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

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

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

- Example:

```
create table instructor (
        ID              char(5),
        name            varchar(20) not null,
        dept_name       varchar(20),
        salary          numeric(8,2))
```

- **insert into** *instructor* **values** ('10211', 'Smith', 'Biology', 66000);
- **insert into** *instructor* **values** ('10211', null, 'Biology', 66000);

- **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)*

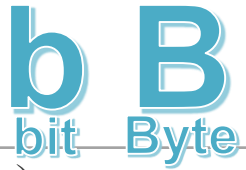
- 讨论4个问题：
- （1）创建的表是什么样子的？
- （2）varchar(10), char(5), int 都是什么？
- （3）Primary Key，Foreign Key是什么？怎么设置？
- （4）这都啥年代了，难道没有图形化界面？

（2）数据类型的设定：
3.2.1 Domain Types 数据类型 in SQL [掌握]

- **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 *n* 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.
- More are covered in Chapter 4.

数据类型表 [Yuewen Add 理解]

类型说明	取值范围
tinyint[(m)]	有符号值：–128到127（–2^7到2^7–1） 无符号值：0到255（0到2^8–1）
smallint[(m)]	有符号值：–32768到32767（–2^15到2^15–1） 无符号值：0到65535（0到2^16–1）
int[(m)]	有符号值：–2147683648到2147683647（–2^31到2^31–1） 无符号值：0到4294967295（0到2^32–1）
bigint[(m)]	有符号值：–9223372036854775808到 9223373036854775807（–2^63到2^63–1） 无符号值：0到18446744073709551615（0到2^64–1）
float	最小非零值：±1.175494351e–38
double	最小非零值：±2.2250738585072014e–308
decimal(m,d)	可变；其值的范围依赖于m和d



一些实际情况 [Yuewen Add 了解]

- 常见的主流数据库系统：
  - IBM DB2, **Oracle**, Informix, Sybase, **SQL Server**, **MySQL**, Access
- 不同的数据库系统，可用的数据类型可能会有不同。
- 例如：
  - MySQL: enum("value1", "value2", …)；65535 个成员；1 或2字节
  - SQLServer：没有枚举类型
- 使用原则：
  - **在空间够用的情况下，尽可能地宽松**
  - **尽量使用标准SQL中的数据类型**

数据库的本质 [Yuewen Add 理解]

数据库是一种：

A 科学技术
遵循理论 why

B 工程技术
遵循规范 how

（3）Primary Key和Foreign Key是什么
Integrity Constraints in Create Table [掌握]

- **not null**
- **primary key** (A_1, ..., A_n)
  - **primary key** declaration 声明 on an attribute automatically ensures **not null** **主键自动不为null**
- **foreign key** (A_m, ..., A_n) **references** r (A_m, ..., A_n)
  - **foreign key** (A_m, ..., A_n) **references** r (A'_m, ..., A'_n)

2.3 Keys 键/码 [掌握]

- Let K $\subseteq$ R [$R = (A_1, A_2, \cdots, A_n)$ is a *relation schema*]
- *K* is a **superkey 超键** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*
  - Example: {*ID*} and {ID,name} are both superkeys of *instructor.*
- Superkey *K* is a **candidate key** 候选键 if *K* is minimal
  Example: {*ID*} is a candidate key for *Instructor*
- One of the candidate keys is selected to be the **primary key 主键**.
  - which one?
- **Foreign key** 外键 constraint: Value in one relation must appear in another
  - **Referencing** relation
  - **Referenced** relation

身份证号	学号	姓名	性别	生辰	学院	GPA

学院名称	院长	院址

Example: Declare *dept_name* as the primary key for *department*

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)*

- **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*);

- **create table** *prereq* (
  course_id **varchar**(8),
  prereq_id **varchar**(8),
  primary key *(course_id, prereq_id),*
  foreign key (*course_id*) **references** *course,*
  foreign key (*prereq_id*) **references** *course*);

About Foreign Key [Yuewen Add 理解]

- Define: **FTable** (the table with foreign key); **PTable** (the table with primary key);

- Is that possible that a foreign key to be a part of a set of primary key in the FTable?
- Answer: **Yes!**
- Should a foreign key definitely be a primary key in the PTable?
- Answer: **Yes!**
- Is that possible that a foreign key to be a part of a set of primary key in the PTable?
- Answer: **No!**
- Possible reasons if you have this demand: (1) you chose an improper PTable; (2) you missed attributes in the foreign key table.

About Foreign Key Cont. [Yuewen Add 理解]

- How to associate the corresponding attributes in the PTable for a foreign key set in the FTable?
- Answer: foreign key (sID,sName) references PTable (ID,Name)

- Can an attribute to be a primary key and the same time a foreign key in the PTable?
- Answer: **Yes!**
- Possible cases: (1) the primary key is the foreign key; (2) a part of primary key to be a foreign key; (3) the primary key is combined by several sets of foreign keys.

- Can two foreign keys in FTable comes from the same attribute in the PTable?
- Answer: **Yes!**
- Example: PTable (couse_id); FTable (preCourse_id, course_id);

How to Interpret Foreign Keys? [Yuewen Add 理解]

- 外键的单位是“组”，每一组外键在其来源表中都是完整的一组主键。

- 例1：teaches 外键为 Teacher_ID, Couse_ID, Sec_ID, Semester, Year
- 不能光看表面，认为这个表有5个外键
- 而要溯源，看到这个外键是两组：
  - 第一组：Teacher_ID 来自 Instructor表。
  - 第二组：Couse_ID, Sec_ID, Semester, Year 来自 Section表。这样理解，Section表的主键是(Couse_ID, Sec_ID, Semester, Year)如果Section表有一个Section_ID，能唯一代表一个Section，从而成为Section表的主键，那么只需要一个外键，Section_ID即可。
- 通常情况下，来自同一个表的外键是一组。
- 例2：prereq表的外键为Course_ID, preCourse_ID
  - 这个表的外键是2组，因为这两个外键是两组完整的主键

How to Write Foreign Keys? [Yuewen Add 掌握]

- 第一步：识别有几组外键，每一组完整的主键都是一组外键
- 第二步：每一组外键都写作
  - **foreign key** (*列名，列名*) **references** *表名*
- 第三步：当来源表中的主键顺序（如 ID, Name）和要引用的表中的外键顺序（如sName, sID）不同时，使用括号标注对应关系
  - **foreign key** (s*ID, sName*) **references** *表名(ID, Name)*

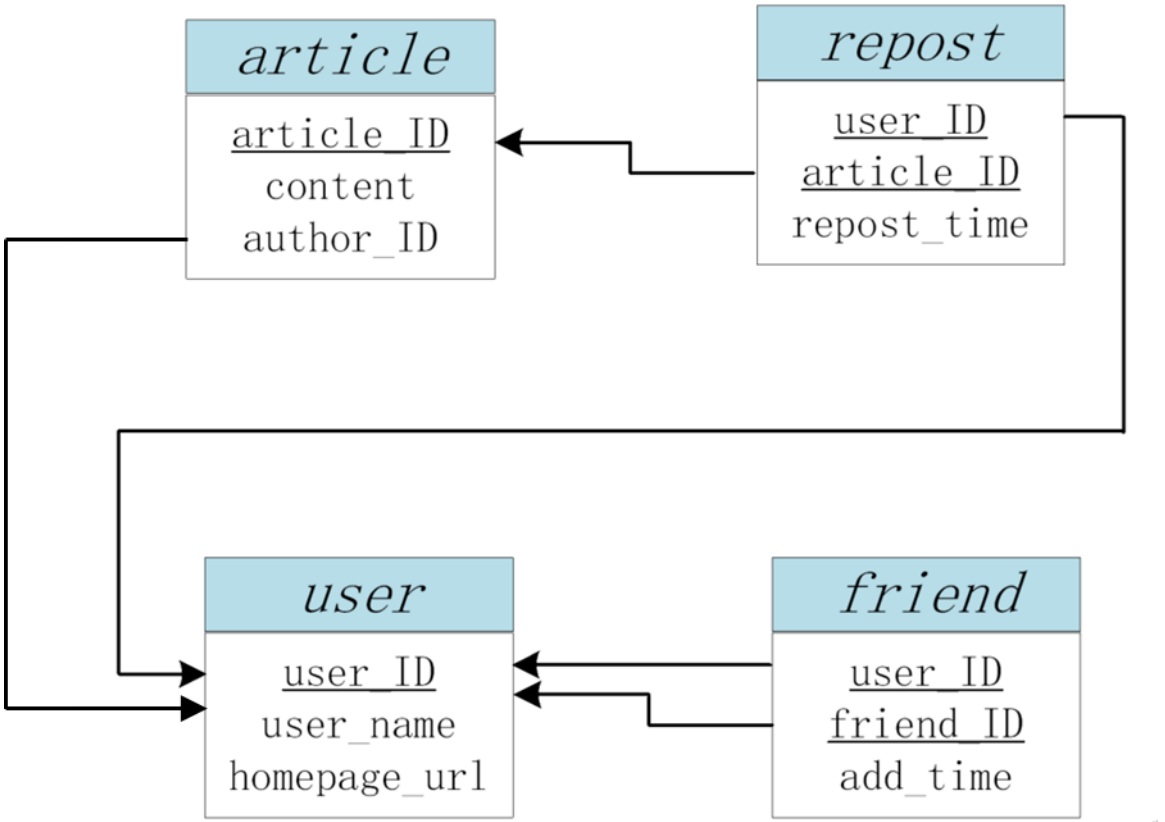
更多写法……

- **create table** *course* (
  course_id **varchar**(8) **primary key**,
  title **varchar(**50),
  dept_name **varchar**(20),
  credits **numeric**(2,0),
  foreign key *(dept_name*) **references** *department*);
  - Primary key declaration can be combined with attribute declaration as shown above

- 问题：一个公司会选择哪种写法？
  - *course_id* **varchar**(8) **primary key**
  - *course_id* **varchar**(8),
    primary key (course_id)
  - 编程语言是人和机器的对话，很多时候也是人和人的对话。
  - 正确的做法：简单规则，创建模板，规范写作。

课堂小测验 [掌握]

- 写出Article，Repost，User，Friend四个表的创建表语句

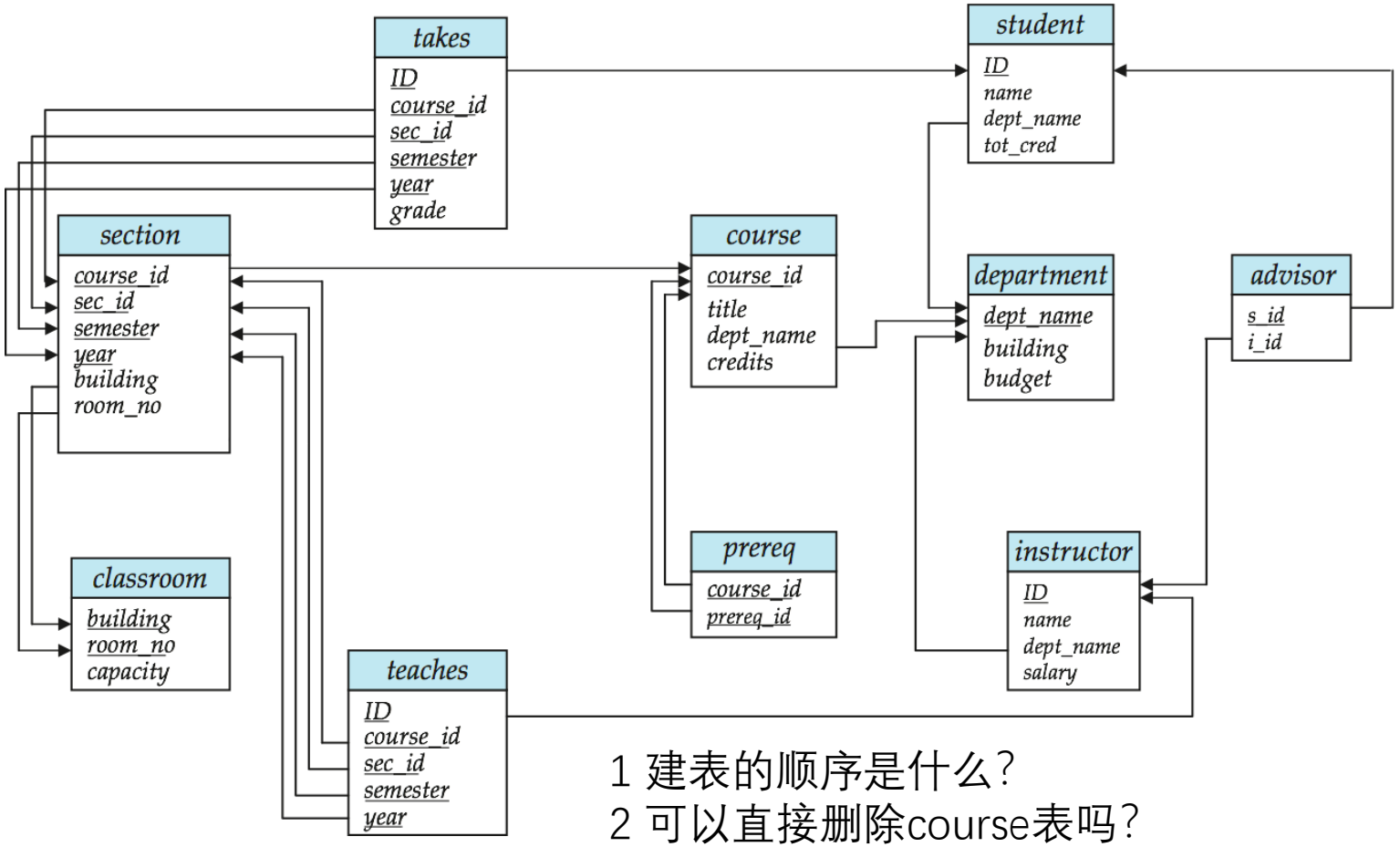


课堂测验答案

- create table user (
  user_ID varchar(50),
  user_name varchar(50),
  homepage_url varchar(500),
  primary key (user_ID))
- create table friend (
  user_ID varchar(50),
  friend_ID varchar(50),
  primary key (user_ID,friend_ID),
  foreign key (user_ID) references user,
  foreign key (friend_ID) references user)

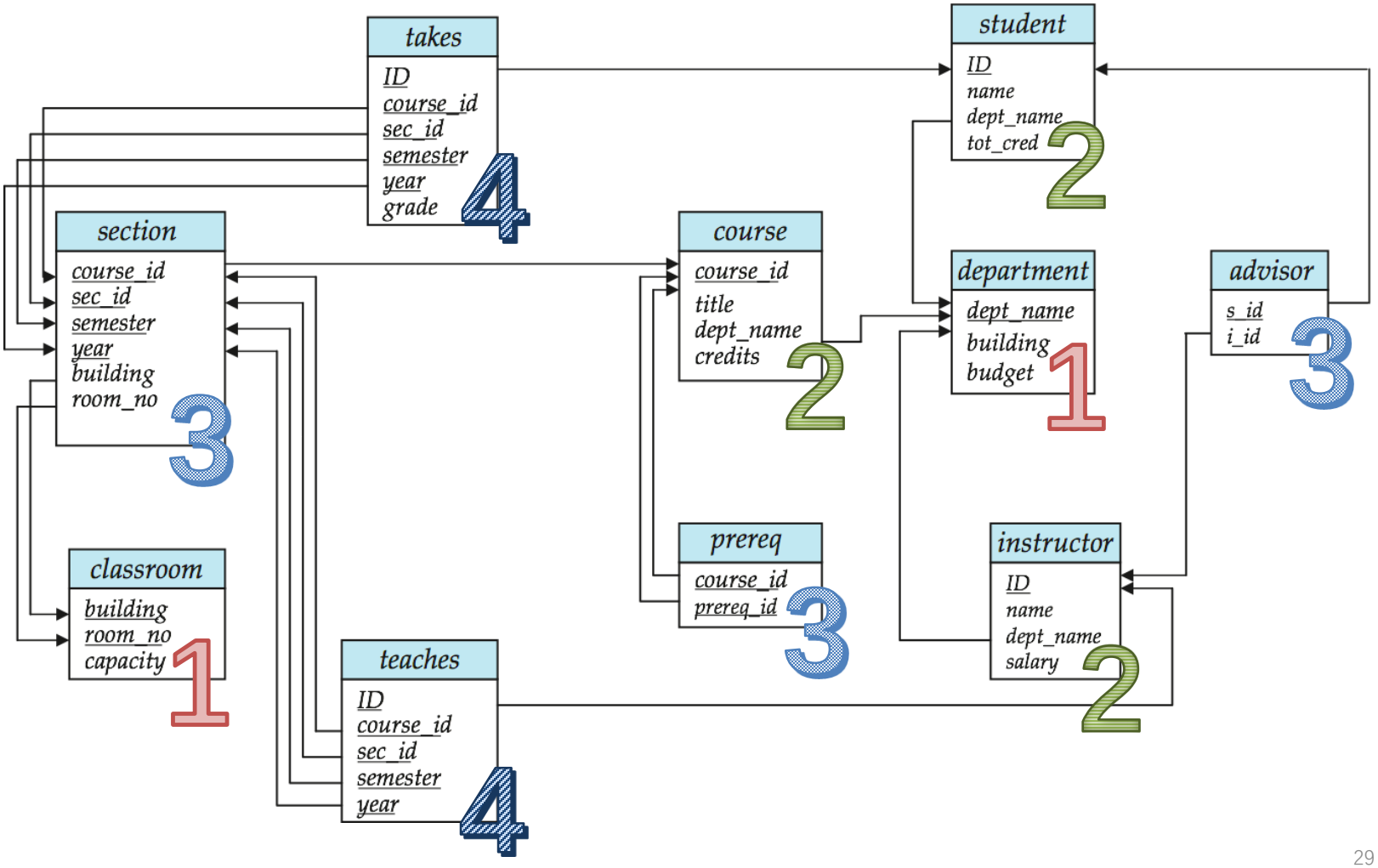
- create table article (
  article_ID varchar(50),
  content varchar(max),
  author_ID varchar(50),
  primary key (article_ID),
  foreign key (author_ID) references user)
- create table repost (
  user_ID varchar(50),
  article_ID varchar(50),
  repost_time datetime,
  primary key (user_ID,article_ID),
  foreign key (user_ID) references user,
  foreign key (article_ID) references article)

p.28 大学模式图（修改版）

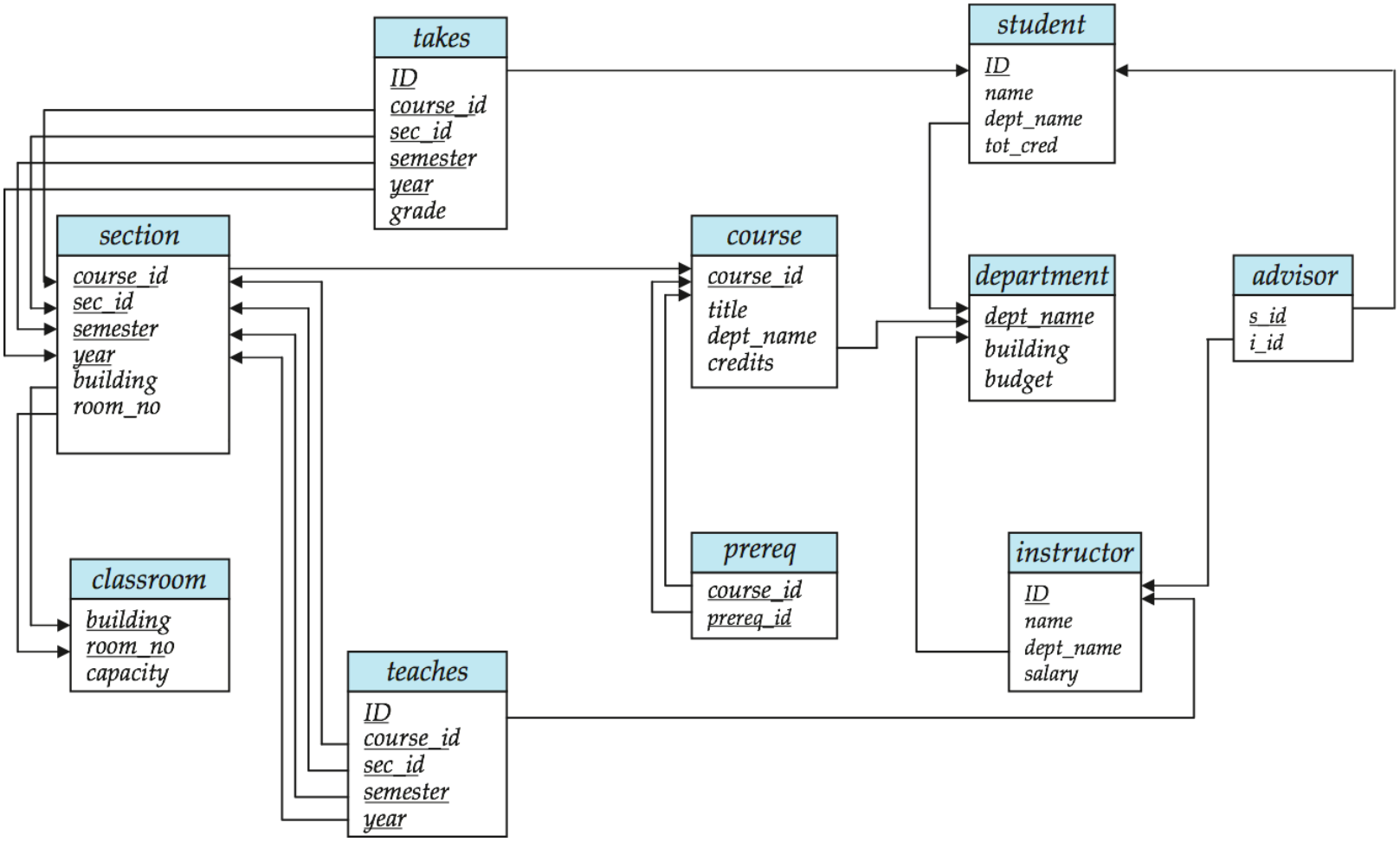


1 建表的顺序是什么？
2 可以直接删除course表吗？
3 插入数据的顺序是什么？
4 可以直接删除student表中的一行数据吗？

建表的顺序（理解什么是约束）



删表的顺序、增删数据的顺序



（4）可视化界面

- 不能用可视化界面吗?
- 答案：大部分平台可以。

- SQL语句的好处：
- （1）各平台通用;
- （2）可配置性高;
- （3）在自动化创建/创建大量表的时候，效率高;

- 查看 create 到查询编辑器。

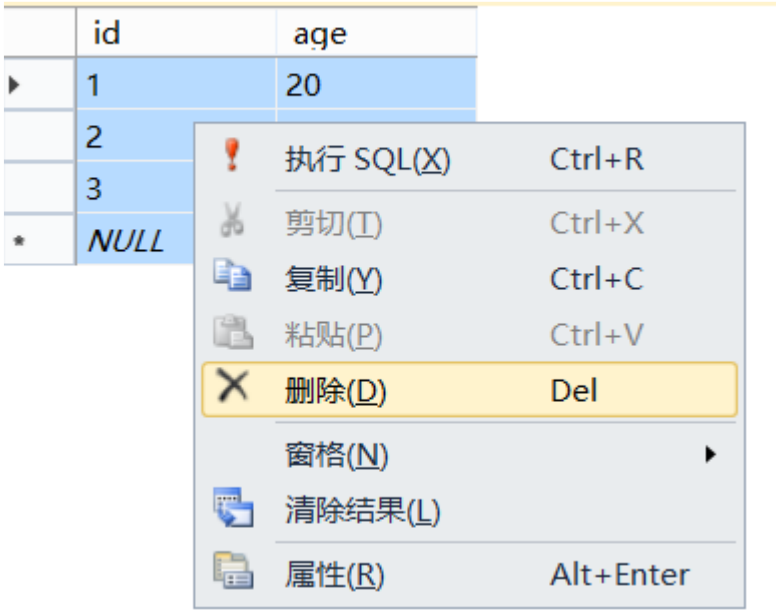
Drop Table Constructs [掌握]

- **drop table** *student*
  - Deletes the table and its contents
  - Conditions: if OBJECT_ID('targetTB') is not null drop table targetTB

- **delete from** *student*
  - Deletes all contents of table, but retains table
- **delete from** *student* where GPA<2.0
- 图形化界面删除数据
- **truncate** table *student*

Advanced Tech. to Delete Data [Yuewen Add 了解]

- 删除全部数据的方法：
  - 图形化界面删除
  - delete from tbName
  - truncate table tbName
- 三种方法的比较
- 学会使用baidu和google解决问题



Alter Table Constructs [掌握]

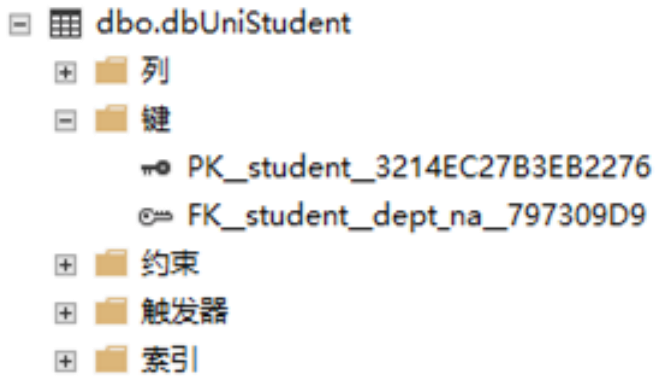
- **alter table**
  - **alter table** *r* **add** *A D*
    - where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A.*
    - All tuples in the relation are assigned *null* as the value for the new attribute.
    - e.g. alter table *student* add GPA numeric(3,2)
    - e.g. alter table test add GPA numeric(3,2), GPB numeric(3,2)
  - **alter table** *r* **drop** *A*
    - where *A* is the name of an attribute of relation *r*
    - Dropping of attributes is not supported by many databases
    - e.g. alter table *student* drop column GPA
    - e.g. alter table *student* drop column GPA, GPB

Advanced Tech. to Alter Constraints [Yuewen Add 了解]

- 增加not null：
  - e.g., alter table student alter column id char(5) not null

- 增加主键、外键：
  - e.g., alter table student add primary key (id), foreign key (dept_name) references department
  - e.g., alter table school add constraint pk_school primary key (school_name), constraint fk_school_dean foreign key (dean) references instructor (ID);
  - 不同之处在于，第二条语句提供了约束名（constraint name）

- 删除主键、外键：
  - 首先需要知道主键、外键名；
  - alter table school
    drop constraint pk_schook, fk_school_dean

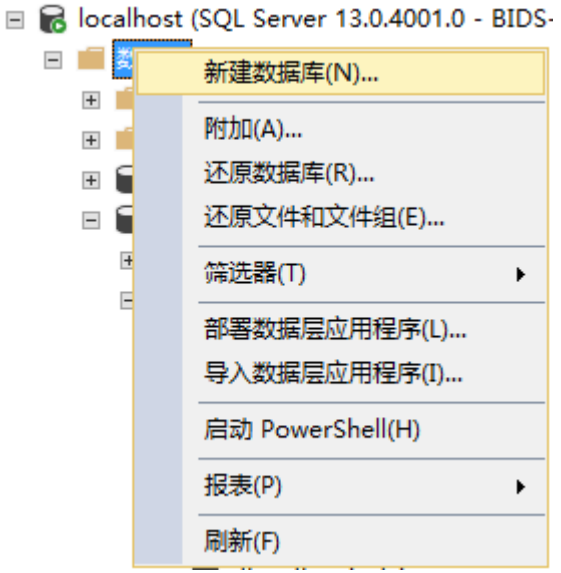


Create Database [了解]

- 图形化界面操作方法

- 创建数据库语句：

Create database dbname on
(name='dbname_data',filename='D:\dbname.mdf',size=50MB,**filegrowth=10%**)
log on (name='dbname_log',
filename='D:\dbname.log', size=50MB,
maxsize=100MB, **filegrowth=1MB**)



3.3 Basic Query Structure 基本查询结构[掌握]

- The SQL **data-manipulation language (DML)** 数据操纵语言 provides the ability to query information, and insert, delete and update tuples 元组：一行数据
- A typical SQL query has the form:

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

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

3.3.1 The select Clause 子句 [掌握]

- The **select** clause list the attributes desired in the result of a query
  - corresponds to the projection 投影 operation of the relational algebra 关系代数
  - 为什么说 select 是一个投影？select可以在原有属性基础上，筛选、新增、变换、重命名各个属性，从而形成一个自己想要的属性列表。
- Example: find the names of all instructors:

  select *name* **from** *instructor*
- NOTE: SQL names are case insensitive 大小写不敏感 (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. [数据库软件会将保留字自动变色]

```
--This is an explaination
select *,len(dept_name),building from dbUniStudent where dept_name='physics'
```

The select Clause (Cont.) [掌握]

- SQL allows duplicates 重复 in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after select.
- Find the names of all departments with instructor, and remove duplicates

```
select distinct dept_name
from instructor
```

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

```
select all dept_name
from instructor
```

- Select **top n** dept_name from instructor (试探性查询/动态加载)
  - e.g., select **top 100** from goods

The select Clause (Cont.) [掌握]

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

  select *
  from *instructor*

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

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

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

- 如果没有from会发生什么：select ID; select 3+4
- 重复的字段：select ID, ID, ID, ID, ID from instructor

The where Clause [掌握]

- The **where** clause specifies conditions that the result must satisfy
  - Corresponds to the selection predicate 谓词 of the relational algebra.
- To find all instructors in Comp. Sci. dept with salary > 80000

  select *name*
  from *instructor*
  where *dept_name* = 'Comp. Sci.' **and** *salary* > 80000
- **比较的运算符： = != <> > < >= <=**
- Comparison results can be combined using the logical connectives **and, or,** and **not.**
- Yuewen's Note: and or not 存在优先级，not 高于 and 高于 or
- Yuewen's Note: **存在多个条件时，一定要加括号，避免逻辑混乱**
- Comparisons can be applied to results of arithmetic expressions 算术表达式.

3.3.2 The from Clause [掌握]

- The **from** clause lists the relations involved in the query
  - Corresponds to the Cartesian product 笛卡尔积 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
- Cartesian product is not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra)

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

X

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

Joins 连接 [掌握]

- 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
```

- 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.'
```

连接查询的理解（1） [理解]

- select * **from A,B**
- “from A,B”的作用是，同时从两张表取数据，合并成一张表。合并方法是：对于A中的每一行，匹配B中的每一行。所以，假设A是m行，B是n行，就形成了一个mn行的表，也就是笛卡尔积。
- select * from A,B **where A.id=B.id**
- “where A.id=B.id”的作用是，从mXn行的表中，选择满足条件“A.id=B.id”的数据，这样结果表的数据量一下就降下来。
- 但是，如果A表中id=1的数据有3条，B表中id=1的数据有2条，那么在执行结果中，id=1的表仍然会有6条。

A

id	S_id
1	a
1	b
1	c

B

id	Course
1	Calculus
1	CS
2	DB

连接查询的理解（2） [理解]

- select * from A,B **where A.Score<B.Score**
- 既然where后面的条件是将两个表的笛卡尔积的结果中部分结果筛选出来，那么这个条件未必就必须是“=”的条件。
- 假设A,B两表是学生成绩表，上述条件筛选的就是：A表中每个记录，匹配所有比其Score高的B的记录。

A

id	Score
1	100
2	60
3	80

B

id	Score
1	75
2	90
3	40

id	Score	id	Score
2	60	1	75
2	60	2	90
3	80	2	90

连接查询的理解（3） [理解]

- select **A.*,B.name** from A,B where A.id=B.id
- select后面的列名，如果是*，是指按照from后表的顺序，依次罗列A,B表的所有字段。但是这些字段名是可以自己指定的。
- A.*是指罗列A表中的全部字段
- B.name是指列出B表中的name字段。如果一个字段名是唯一的，那么可以直接写字段名；如果字段名不唯一，必须写A.name或者B.name，否则会报错。

3.3.3 Natural Join [不许掌握]

- Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column [看起来省事的办法]
- **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

3.4.1 The Rename Operation
重命名操作 [掌握]

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

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

- E.g.
  - **select** *ID, name, salary/12* **as** *monthly_salary*
    from *instructor*

- 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

Yuewen Add: 重命名的要点[掌握]

- 1 可以重命名字段名、表名。
- 2 重命名的字段名，只有在order by子句中能使用，其它地方不能使用。一些错误的用法如下:
- e.g.，select ID, exam*0.7 as exam_Score, attendance*0.3 as atten_Score, exam_Score+atten_Score as final_score from stuPerformance
- e.g., select ID, exam*0.7 as exam_Score from stuPerformance where exam_Score>70
- 3 表被重命名之后，旧的表名失效，不能再被使用
- 4 在一个表被使用多次时，需要重命名表名，否则会有语法错误

表与自身的连接操作 [理解]

- 表与自身的连接操作，可以视作两张一样的表的连接操作。
- 表与自身的连接是非常有用的。避免写程序循环，提高计算效率。举两个例子：
- （1）好友推荐
- 好友推荐就是推荐好友的好友（潜在好友）
- select A.*,B.fid as ffid from tb as A, tb as B where A.fid=B.id

tb as A

id	fid
1	2
2	4
3	1

tb as B

id	fid
1	2
2	4
3	1

id	fid	ffid
1	2	4
3	1	2

- （2）识别排队顺序
- 假设有一张销售记录表，各列分别为PosID,CustomerID,Seq(排队顺序),Ptime(交易时间)
- 查看排队的前一个人：
- Select A.*,B.CusID as BID from tb as A, tb as B where A.PosID=B.PosID and A.Seq=B.Seq+1 and D[illegible]ime)<=[illegible]

A

PosID	CusID	Seq	PTime
1	a	1	3:02
1	b	2	3:03
1	c	3	3:05

B

PosID	CusID	Seq	PTime
1	a	1	3:02
1	b	2	3:03
1	c	3	3:05

PosID	CusID	Seq	PTime	BID
1	b	2	3:03	a
1	c	3	3:05	b

3.4.2 String 字符串 Operations [掌握]

- 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.

- e.g., 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** 转义字符 '\'

String Operations (Cont.) [理解]

- 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 连接 (using "||") Yuewen's Note: 这个与DBMS的版本有关。连接字符在SQL Server中可以用+，用||会报错。
  - converting from upper to lower case (and vice versa) *upper(), lower()*
  - finding string length, extracting substrings, etc. *length(), left(), right(), ltrim(), rtrim(), substring(), charindex()*

字符串比较的要点[掌握]

- %可以代表0~无穷多字符，而_代表1个字符（不能是0个）
- 1个汉字字符和英文字符都是1个字符
- 当需要使用通配符时，一定要使用 like，而不是=
- 使用=时，通配符被当做普通字符来处理了。

- 例如：假设表tb如右图：
- select * from tb where name='%'
  只会返回第一条数据
  因为这里'%'被当做普通字符串
- select * from tb where name like '%'
  只会返回全部5条数据
  因为这里'%'被当做通配符

	id	name
	1	%
	2	hello % world
	3	%%%
	4	%_%
	5	A

3.4.4 Ordering the Display of Tuples 结果排序 [掌握]

- 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 多标准排序
  - Example: **order by** *dept_name, name*

Where Clause Predicates 谓词 [不许掌握]

- 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');

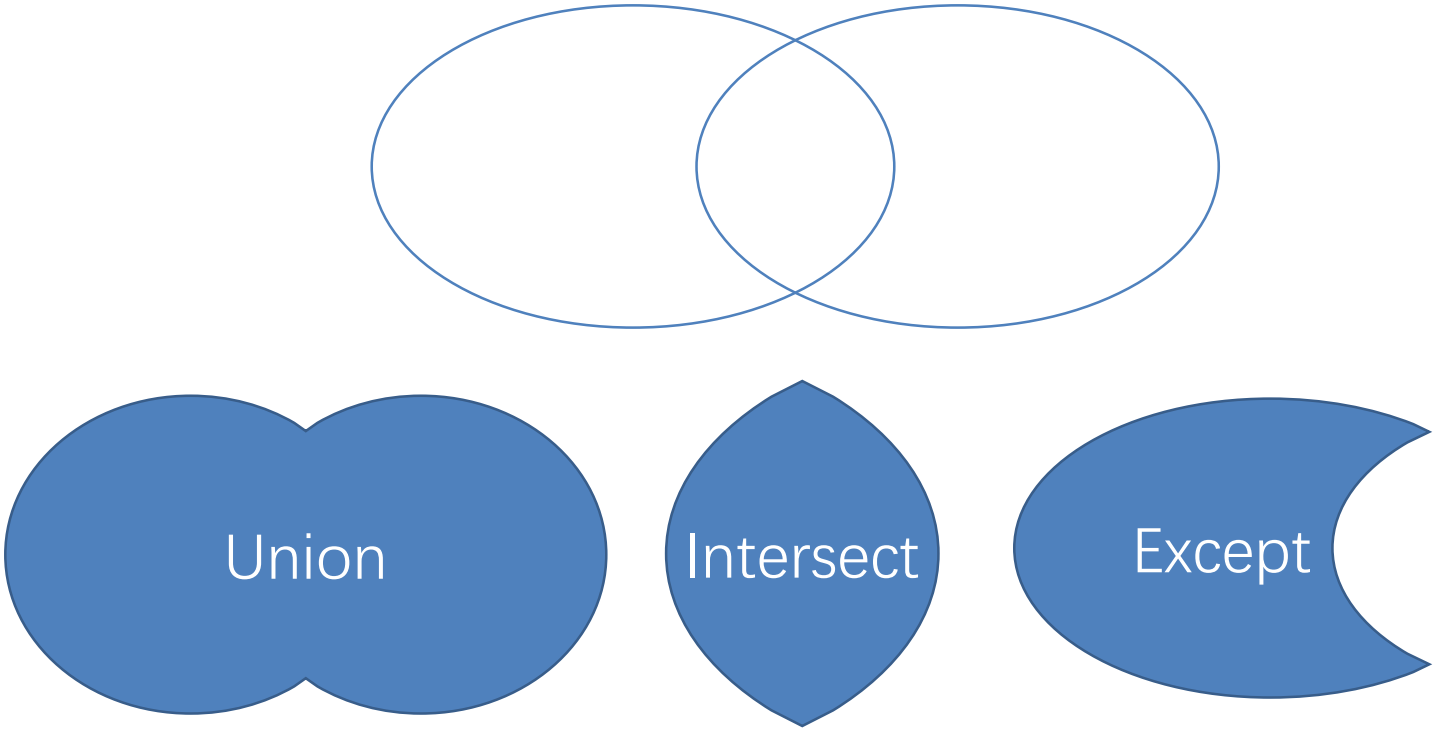
Set Operations 集合操作 [掌握]

- Find courses that ran in Fall 2009 or in Spring 2010
- **select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009
  union
  select *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010

- Find courses that ran in Fall 2009 and in Spring 2010
- **select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009
  intersect
  select *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010

- Find courses that ran in Fall 2009 but not in Spring 2010
- **select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009
  except
  select *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010

集合运算图示



Set Operations [不许掌握]

- Set operations **union**, **intersect**, and **except**
  - Each of the above operations automatically eliminates duplicates
- To retain all duplicates use the corresponding multiset versions **union all, intersect all** and **except all.**

  Suppose a tuple occurs *m* times in *r* and *n* times in *s,* then, it occurs:
  - *m + n* times in *r* **union all** *s*
  - min(*m,n)* times in *r* **intersect all** *s*
  - max(0, *m – n)* times in *r* **except all** *s*

关于集合运算的说明[掌握]

- 1 集合运算要求掌握4种：Union, Union all, Intersect, Except
- 2 Union，Intersect，Except都会将结果去重。操作顺序是：现将表中的数据进行去重，再进行集合运算。
- 3 Union all不对表中的数据做去重操作。
- 4 SQL Server中，不支持Intersect all和Except all

集合运算举例[掌握]

- 例：有A、B两个表如下：

- select name from A
  union/union all/intersect/except
  select name from B

A

	id	name
	1	1
	2	1
	3	1
	4	2
	5	2

B

	id	name
	1	1
	2	1
	3	2
	4	2
	5	3

Union

	name
1	1
2	2
3	3

Union all

	name
1	1
2	1
3	1
4	2
5	2
6	1
7	1
8	2
9	2
10	3

Intercept

	name
1	1
2	2

Except

	name

3.6 Null Values 空值 [掌握]

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null*
  - Example: 5 + *null* returns null
- The predicate **is null** can be used to check for null values.
  - Example: Find all instructors whose salary is null.

select *name*
from *instructor*
where *salary* **is null**

Null Values and Three Valued Logic 逻辑值 [了解]

- 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*

Null的几个要点[掌握]

- 1 Null是指不存在数据，它不是空字符串''，也不是0
- 2 判断是否为Null要用is，不能用等于号=
- 3 Null参与逻辑运算，不一定能得到True或者False，还可能得到Unknown

3.7 Aggregate Functions 聚集函数 [掌握]

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

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

Aggregate Functions (Cont.) [掌握]

- Find the average salary of instructors in the Computer Science department
  - **select avg** (*salary*)
    from *instructor*
    where *dept_name*= 'Comp. Sci.';
- Find the total number of instructors who teach a course in the Spring 2010 semester
  - **select count** (**distinct** *ID*)
    from *teaches*
    where *semester* = 'Spring' **and** *year* = 2010
- Find the number of tuples in the *course* relation
  - **select count** (*)
    from *course*;
- Yuewen备注：要注意 avg(整数) 得到整数。如何得到小数？
  - 将整数*1.0，将其转换为小数

Aggregate Functions – Group By [掌握]

- Find the average salary of instructors in each department
  - **select** *dept_name*, **avg** (*salary*)
    from *instructor*
    group by *dept_name*;
  - Note: departments with no instructor will not appear in result

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

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

Aggregation (Cont.) [掌握]

- 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*;

- Yuewen：Group by语句只有两种列
  - 一种是被Group by的列名
  - 一种是被聚集函数统计的统计值
  - 为什么?

选择列表中的列 'B.id' 无效，因为该列没有包含在聚合函数或 GROUP BY 子句中。

常见错误[掌握]

- E.g., select good_id, good_price-min(good_price) from goods
- 这句话报错。错误出现在哪里?

Good_ID	Good_Name	Good_Price
1	iPhone	5000.00
2	SamSung	2000.00
3	Leshi	1000.00
4	Huawei	3000.00

- 判断方法
- min(good_price) 能得到几条数据? 1条
- good_price 有几条数据? 4条
- 条数不匹配: 错误!!!

Aggregate Functions – Having Clause
Having子句[掌握]

- 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

- **比较以下语句：有何不同？**

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

Null Values and Aggregates
聚集函数对空值的处理 [了解]

- 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 [null会被忽略]

- [极端情况] What if collection has only null values?
  - count returns 0
  - all other aggregates return null

3.8 Nested Subqueries 嵌套子查询 [掌握]

- SQL provides a mechanism for the nesting of subqueries.
- A **subquery** is a **select-from-where** expression that is nested within another query.
- A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality.

Example Query
嵌套在where条件中: in [掌握]

- Find courses offered in Fall 2009 and in Spring 2010

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

- Find courses offered in Fall 2009 but not in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
        course_id  not in (select course_id
                      from section
                      where semester = 'Spring' and year= 2010);
```

Example Query [不许掌握]

- 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: SQL Server不支持这种多列的写法。

Set Comparison 集合比较 [理解]

- Find names of instructors with salary greater than that of some (at least one) instructor in the Biology department.

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

- Same query using > **some** clause

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

Definition of Some Clause
Some子句 [理解]

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

(5 < **some** | 0 | 5 | 6 |) = true

(5 < **some** | 0 | 5 |) = false

(5 = **some** | 0 | 5 |) = true

(5 ≠ **some** | 0 | 5 |) = true (since 0 ≠ 5)

Example Query
all子句[理解]

- 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
all子句 [理解]

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

(5 < **all** | 0, 5, 6 |) = false

(5 < **all** | 6, 10 |) = true

(5 = **all** | 4, 5 |) = false

(5 ≠ **all** | 4, 6 |) = true (since 5 ≠ 4 and 5 ≠ 6)

Test for Empty Relations
存在子句 [理解]

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

- e.g., 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);
```

Subqueries in the From Clause
From子句中的嵌套查询 [掌握]

- 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) as tb
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;
```

- [Yuewen' Note] SQL Server要求from中的嵌套子查询必须有别名。

With Clause
提前定义一个模块/关系 [掌握]

- 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.
- Find all departments with the maximum budget

```
with max_budget (value) as
    (select max(budget)
      from department)
select 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
- 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;
```

- Note：一个with配对一个select

Scalar Subquery 标量子查询 在Select子句中的嵌套 [理解]

- 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

嵌套标量子查询[掌握]

- E.g.，select good_id, good_price-min(good_price) from goods
- 这句话该怎么修改?

- Select good_id, good_price-(select min(good_price) from goods) from goods

3.9 Modification of the Database
数据库修改 [掌握]

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

insert delete update

Modification of the Database – Deletion
删除 [掌握]

- Delete all instructors

  delete from *instructor*

- Delete all instructors from the Finance department

  delete from *instructor* **where** *dept_name* = 'Finance';

- Delete all tuples in the *instructor* relation for those instructors associated with a department located in the Watson building.

```
delete from instructor
where dept_name in (select dept_name
                    from department
                    where building = 'Watson');
```

Deletion (Cont.)

- Delete all instructors whose salary is less than the average salary of instructors

  delete from *instructor*
  where *salary* < (**select avg** (*salary*) **from** *instructor*);

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

Delete的几种套路[掌握]

- 套路1：删全表数据。常用truncate table tb
- delete from tb
- 套路2：依据固定条件，删表中部分数据。
- delete from tb where salary<10000
- 套路3：依据其它表，删表中部分数据。
- delete from tb where id in (select distinct id from tbIDList)
- delete from tb where salary<(select min(salary) from tbFactory)

Modification of the Database – Insertion
插入 [掌握]

- 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***);

- 注意：values的值和表的字段顺序要一致！！！

Insertion (Cont.) [掌握]

- 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 problems, if *table1* did not have any primary key defined.

- 注意：select后面的内容要和被插入表的字段数量、顺序完全一致
- 或者： insert into student(ID,name) values (222,'Lilei')
  insert into student(ID,name) select ID,name from instructor

Insert的几种套路[掌握]

- 套路1：按照表的字段顺序插入数据（值可以为空）
- Insert into tb values(1,'Tony',300,'CompSci',,,)
- 套路2：按照指定的顺序插入数据
- Insert into tb(id,name,deptName) values(1,'Tony','CompSci')
- 套路3：从别的表复制数据到本表中
- Insert into tb select id,name,dept_name from sourceTB
- Insert into tb(id,name,deptName,salary,nStudents) select id,name,dept_name,salary*12,0 from sourceTB where dept_name='CompSci'

Modification of the Database – Updates
Update更新语句 [掌握]

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others receive a 5% raise
  - 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 case when语句 [掌握]

- Same query as before but with case statement

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

- **Case when 条件 then 值 else 值 end**
- **Case when 也可以用于select子句中。**

```
select  case when salary <= 100000 then salary * 1.05
else salary * 1.03 end
from instructor
```

case when语句的几点说明[理解]

- **case when 可以写多个when，如：**
- case when score>=90 then ‘A’ when score>=80 then ‘B’ when score>=70 then ‘C’ when score>=60 then ‘D’ else ‘F’ end

- **case when 可视作一个列名，可以用在多种子句中，如：**
- select ID, case when ID%2=0 then ‘男’ else ‘女’ end as gender from tb

- **case when 采取的策略是，一旦条件满足就不再比下一条，如：**
- case when ID%2=0 then ‘男’ when ID%4=0 then ‘外星人’ else ‘女’ end
- case when ID%4=0 then ‘外星人’ when ID%2=0 then ‘男’ else ‘女’ end
- 这两条语句结果不同。第一条语句永远不会有’外星人’出现。

update语句的几种套路[掌握]

- 套路1：直接更新表格数据
- update tb set name='Tony',salary=salary+12 where id=2
- 套路2：把一个表格中的内容更新到另外一个表格中
- update A set address=B.address, name=B.name from B where A.id=B.id

知识点总结

- create table：int, varchar, numeric, float, 主键，外键（特别是多组外键references同一个表），建表、删表、插入数据、删除数据的次序
- alter table：增加，删除字段
- drop table, delete from table
- select：top n, distinct, 各种运算, as
- where：and or not 加括号, = < <= > >= != <>
- from：单表，多表，笛卡尔积，连接条件，as
- 字符串： like % _ escape，函数 len(), substring(), charindex()等
- order by: asc desc
- 集合操作：union, union all, intersect, except

- NULL：is，NULL的涵义，unknown
- 聚集函数：min max count avg sum
- group by：select中只能有两种列，having，空值被忽略
- 嵌套子查询：
  - where：in，not in，some，all，exists
  - from：查询结果表，with提前定义表
  - select：标量子查询
- insert: values, select, 注意前后字段顺序一致
- delete: where
- update: where
- case when then else end

谢谢！

liuyuewen@xjtu.edu.cn