中级SQL
Intermediate SQL

李文根/Wengen Li

Email: lwengen@tongji.edu.cn

先进数据与机器智能系统实验室（ADMIS）

https://admis.tongji.edu.cn/main.htm

同濟大學 TONGJI UNIVERSITY 电子与信息工程学院 College of Electronics and Information Engineering 计算机科学与技术系 Department of Computer Science and Technology

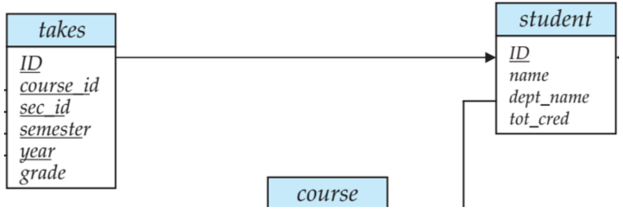
课程概要

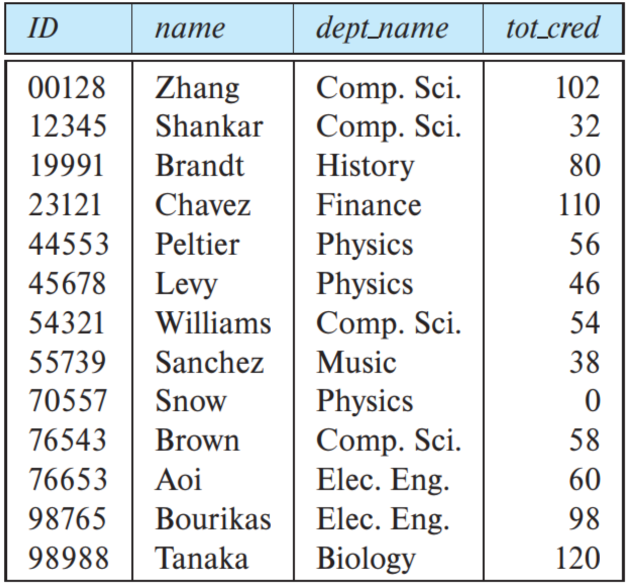
- **Part 0: Overview**
  - Ch1: Introduction
- **Part 1 Relational Databases**
  - Ch2: Relational model
  - Ch3: Introduction to SQL
  - **Ch4: Intermediate SQL**
  - Ch5: Advanced SQL
- **Part 2 Database Design**
  - Ch6: Database design based on E-R model
  - Ch7: Relational database design
- **Part 3 Application Design & Development**
  - Ch8: Complex data types
  - Ch9: Application development
- **Part 4 Big data analytics**
  - Ch10: Big data
  - Ch11: Data analytics
- **Part 5 Data Storage & Indexing**
  - Ch12: Physical storage system
  - Ch13: Data storage structure
  - Ch14: Indexing
- **Part 6 Query Processing & Optimization**
  - Ch15: Query processing
  - Ch16: Query optimization
- **Part 7 Transaction Management**
  - Ch17: Transactions
  - Ch18: Concurrency control
  - Ch19: Recovery system
- **Part 8 Parallel & Distributed Database**
  - Ch20: Database system architecture
  - Ch21-23: Parallel & distributed storage, query processing & transaction processing
- **Part 9**
  - DB Platform: **OceanBase**, MongoDB, Neo4J

目录

- Join表达式
- 视图
- 事务
- 完整性约束
- SQL中的数据类型
- SQL中的索引定义
- 授权

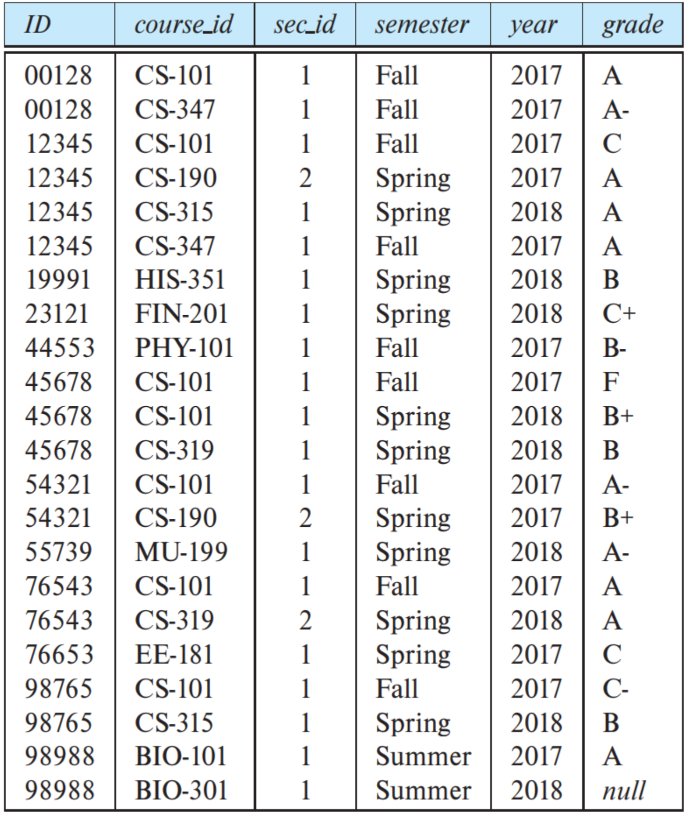
关系示例





ID	*name*	*dept_name*	*tot_cred*
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

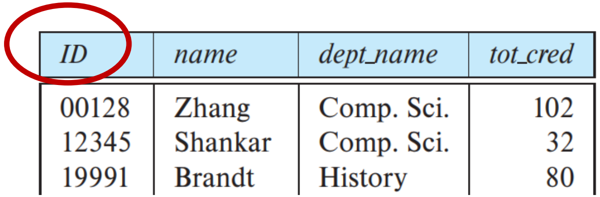
student



ID	*course_id*	*sec_id*	*semester*	*year*	*grade*
00128	CS-101	1	Fall	2017	A
00128	CS-347	1	Fall	2017	A-
12345	CS-101	1	Fall	2017	C
12345	CS-190	2	Spring	2017	A
12345	CS-315	1	Spring	2018	A
12345	CS-347	1	Fall	2017	A
19991	HIS-351	1	Spring	2018	B
23121	FIN-201	1	Spring	2018	C+
44553	PHY-101	1	Fall	2017	B-
45678	CS-101	1	Fall	2017	F
45678	CS-101	1	Spring	2018	B+
45678	CS-319	1	Spring	2018	B
54321	CS-101	1	Fall	2017	A-
54321	CS-190	2	Spring	2017	B+
55739	MU-199	1	Spring	2018	A-
76543	CS-101	1	Fall	2017	A
76543	CS-319	2	Spring	2018	A
76653	EE-181	1	Spring	2017	C
98765	CS-101	1	Fall	2017	C-
98765	CS-315	1	Spring	2018	B
98988	BIO-101	1	Summer	2017	A
98988	BIO-301	1	Summer	2018	*null*

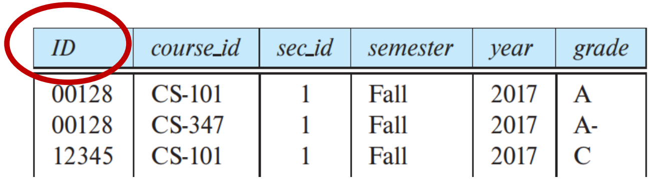
takes

自然连接

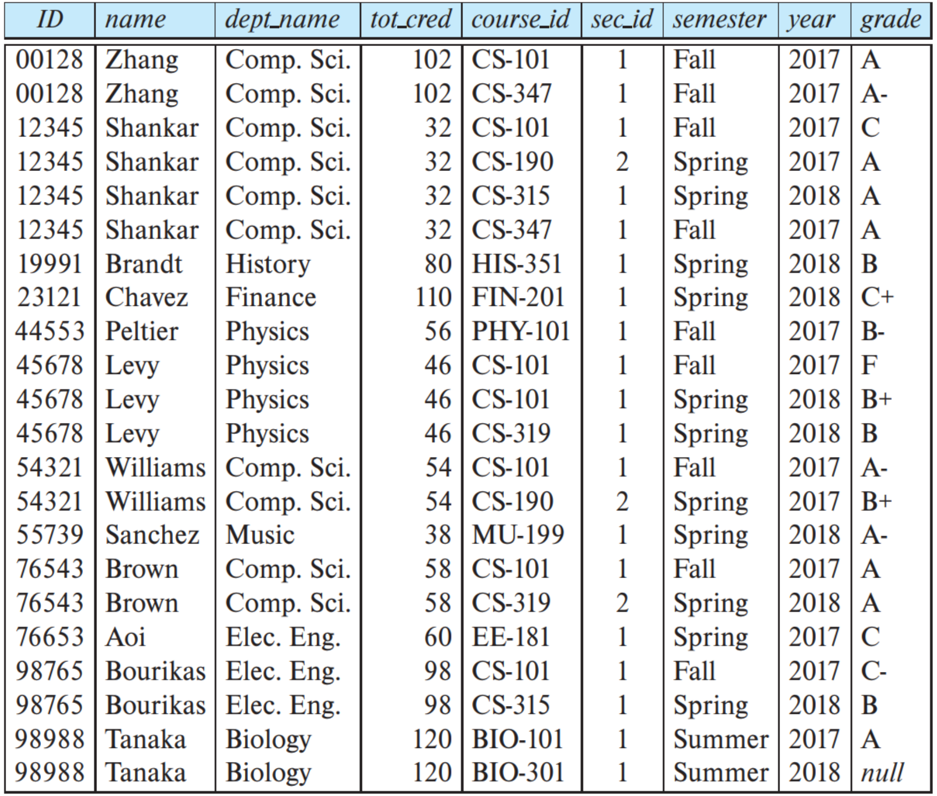


ID	*name*	*dept_name*	*tot_cred*
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80

student ***natural join*** *takes*



ID	*course_id*	*sec_id*	*semester*	*year*	*grade*
00128	CS-101	1	Fall	2017	A
00128	CS-347	1	Fall	2017	A-
12345	CS-101	1	Fall	2017	C



ID	*name*	*dept_name*	*tot_cred*	*course_id*	*sec_id*	*semester*	*year*	*grade*
00128	Zhang	Comp. Sci.	102	CS-101	1	Fall	2017	A
00128	Zhang	Comp. Sci.	102	CS-347	1	Fall	2017	A-
12345	Shankar	Comp. Sci.	32	CS-101	1	Fall	2017	C
12345	Shankar	Comp. Sci.	32	CS-190	2	Spring	2017	A
12345	Shankar	Comp. Sci.	32	CS-315	1	Spring	2018	A
12345	Shankar	Comp. Sci.	32	CS-347	1	Fall	2017	A
19991	Brandt	History	80	HIS-351	1	Spring	2018	B
23121	Chavez	Finance	110	FIN-201	1	Spring	2018	C+
44553	Peltier	Physics	56	PHY-101	1	Fall	2017	B-
45678	Levy	Physics	46	CS-101	1	Fall	2017	F
45678	Levy	Physics	46	CS-101	1	Spring	2018	B+
45678	Levy	Physics	46	CS-319	1	Spring	2018	B
54321	Williams	Comp. Sci.	54	CS-101	1	Fall	2017	A-
54321	Williams	Comp. Sci.	54	CS-190	2	Spring	2017	B+
55739	Sanchez	Music	38	MU-199	1	Spring	2018	A-
76543	Brown	Comp. Sci.	58	CS-101	1	Fall	2017	A
76543	Brown	Comp. Sci.	58	CS-319	2	Spring	2018	A
76653	Aoi	Elec. Eng.	60	EE-181	1	Spring	2017	C
98765	Bourikas	Elec. Eng.	98	CS-101	1	Fall	2017	C-
98765	Bourikas	Elec. Eng.	98	CS-315	1	Spring	2018	B
98988	Tanaka	Biology	120	BIO-101	1	Summer	2017	A
98988	Tanaka	Biology	120	BIO-301	1	Summer	2018	*null*

自然连接(续)

select $A_1, A_2, \ldots, A_n$
from r_1 ***natural join*** r_2 ***natural join …natural join*** r_m
where P;

select *name, course_id*
from *student* ***natural join*** *takes;*

查询每位同学所选课程的ID

select *name, title*
from *student* ***natural join*** *takes, course*
where *takes.course_id=course.course_id;*

查询每位同学所选课程的名称

select *name, title*
from *(student* ***natural join*** *takes) join course using(course_id);*

思考：与关系course的连接是否可以用natural join?

Join表达式

- **Join operation (连接操作)**
  - Take two relations and return another relation as the results
- **Join type (连接类型)**
  - Define how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated
- **Join condition (连接条件)**
  - Define which tuples in the two relations match, and what attributes are present in the result of the join

Join Types
inner join left outer join right outer join full outer join

Join Conditions
natural using $(A_1, A_2, \ldots, A_n)$ on <predicate>

natural和using去重复属性，on不会

关系示例

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

Relation ***loan***

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

Relation ***borrower***

Note: borrower information is missing for L-260 and loan information is missing for L-155

例1：关系连接

loan ***inner join*** *borrower* ***on***
loan.loan_number = borrower.loan_number

loan_number	*branch_name*	*amount*	*customer_name*	*loan_number*
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230

loan ***natural inner join*** *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

例2：关系连接

loan ***left outer join*** *borrower* ***on***
loan.loan_number = borrower.loan_number

loan_number	*branch_name*	*amount*	*customer_name*	*loan_number*
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230
L-260	Perryridge	1700	*null*	*null*

loan ***natural left outer join*** *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*

例3：关系连接

loan ***right outer join*** *borrower* ***on***
loan.loan_number = borrower.loan_number

loan_number	*branch_name*	*amount*	*customer_name*	*loan_number*
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230
L-155	null	null	Hayes	*null*

loan ***natural right outer join*** *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	null	null	Hayes

例4：关系连接

loan ***natural full outer join*** *borrower*

loan ***full outer join*** *borrower using (loan_number)*

两个查询等价

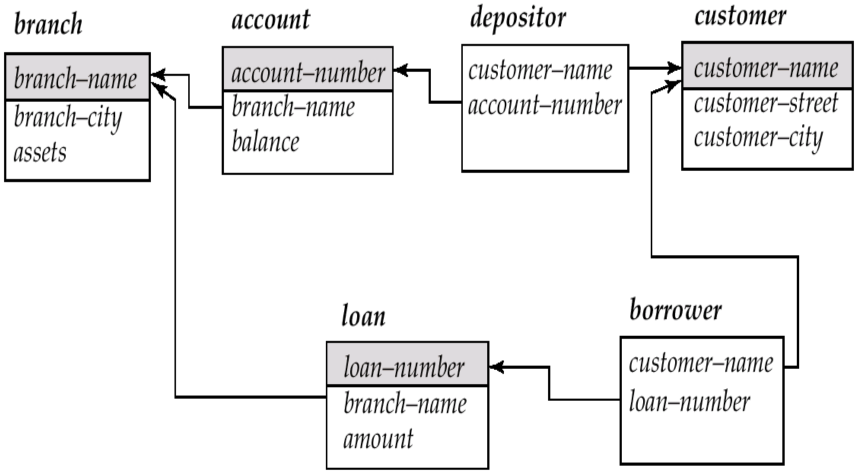
loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	null	null	Hayes

例5：关系连接

- Find all the customers who have either an account or a loan (but not both) at the bank:

select *customer_name*
from *(depositor* ***natural full outer join*** *borrower)*
where *account_number* ***is null*** *or loan_number* ***is null***

思考：如何用集合操作实现该查询?



连接条件的区别

- join on/join using
  - **on** is a predicate
  - **using** specifies the attributes for natural join

select *name, title*
from *(instructor* ***natural join*** *teaches)* ***join*** *course* ***using*** *(course_id);*

- join on/where

select ***
from *student* ***left outer join*** *takes* ***on*** *student.ID=takes.ID*

select ***
from *student* ***left outer join*** *takes* ***on true***
where *student.ID=takes.ID*

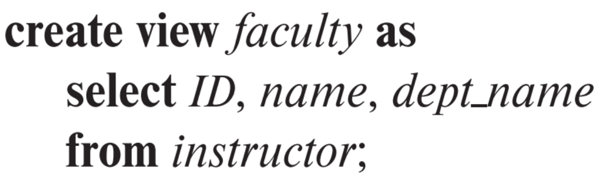
思考：两个SQL查询结果相同吗?

目录

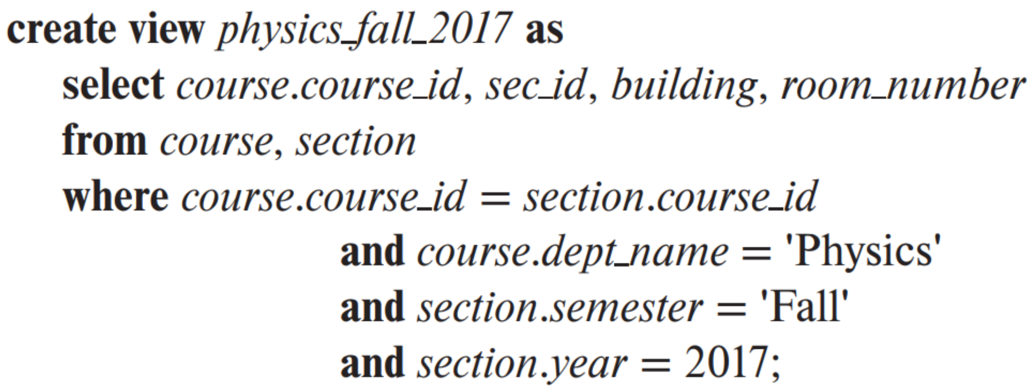
- Join表达式
- 视图
- 事务
- 完整性约束
- SQL中的数据类型
- SQL中的索引定义
- 授权

视图(View)

定义视图： **create view** *v* **as** <query expression>

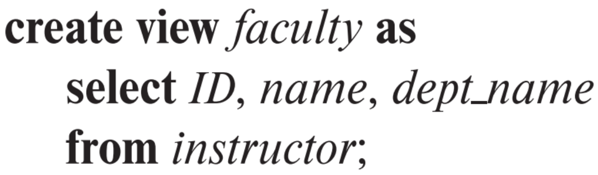


```
create view faculty as
    select ID, name, dept_name
    from instructor;
```



```
create view physics_fall_2017 as
    select course.course_id, sec_id, building, room_number
    from course, section
    where course.course_id = section.course_id
                   and course.dept_name = 'Physics'
                   and section.semester = 'Fall'
                   and section.year = 2017;
```

视图的更新



```
create view faculty as
    select ID, name, dept_name
    from instructor;
```

- Add a new tuple to relation **faculty**

```
insert into faculty
   values (‘30765', 'Green', 'Music’);
```

This insertion should be represented by the insertion of the tuple

(*30765*', *'Green*', *'Music*’, *null*)

into the relation **instructor**, and attribute salary is set to null

视图的更新(续)

- Updates on complex views are difficult or impossible to translate, and hence are disallowed
- In general, an SQL view is updatable if:
  - The **from** clause has only one relation
  - The select clause contains only attribute names of the relation, and does not have any **expressions**, **aggregates**, or **distinct specification**
  - Any attributes not listed in the select clause **can be set to null**
  - The query does not have a **group by** or **having** clause

物化视图(Materialized View)

- The relation of a view is stored, and will change if the actual relations used in the view definition change.
- Materialized view maintenance
  - **Real-time updates** vs. **periodic updates**

目录

- Join表达式
- 视图
- 事务
- 完整性约束
- SQL中的数据类型
- SQL中的索引定义
- 授权

事务(Transactions)

- A transaction is a sequence of query and update statements executed as **a single unit**
- Transactions are started implicitly and terminated by one of
  - **commit** [work]: make all updates of the transaction permanent in the database
  - **rollback** [work]: undo all updates performed by the transaction

事务(续)

- 例：transferring money from one account to another involves two steps: **deduct** from one account and **credit** to another
  - If one step succeeds and the other fails, database is in an inconsistent state
  - Either both steps should succeed or neither should
- If any step of a transaction fails, all work done by the transaction should be **undone** by **rollback**
- Rollback of incomplete transactions is done automatically, in case of system failures

事务(续)

- In most database systems, each SQL statement that executes successfully is automatically committed
  - Each transaction consists of only a single statement
  - Automatic commit can be turned off, allowing multi-statement transactions, but depends on the database system
  - Another option: enclose statements within

begin atomic

…

end

目录

- Join表达式
- 视图
- 事务
- 完整性约束
- SQL中的数据类型
- SQL中的索引定义
- 授权

完整性约束(Integrity Constraints)

- Integrity constraints guard against accidental damage to the database
  - by ensuring that authorized changes to the database do not result in a loss of **data consistency**

- **Types**
  - Domain constraints
  - Not null constraint
  - Unique constraint
  - Referential integrity
  - …

域约束(Domain Constraints)

- Domain constraints are the most elementary form of integrity constraint
- New domains can be created from existing data types, e.g.,

```
create domain Dollars numeric(12, 2)
create domain Pounds numeric(12, 2)
```

  - cannot assign or compare a value of type Dollars with a value of type Pounds

- The check clause permits domains to be restricted

```
create domain hourly_wage numeric(5, 2)
    constraint value_test check(value >= 6.00)
```

  - The domain has a constraint to ensure that the hourly_wage is greater than 6.00
  - The clause constraint ***value_test*** is optional but useful to indicate which constraint an update violates

非空约束

- Declare attribute name in relation student to be not null

  name varchar(15) not null

- Declare the domain Dollars to be not null

  create domain Dollars numeric(12,2) not null

Unique约束

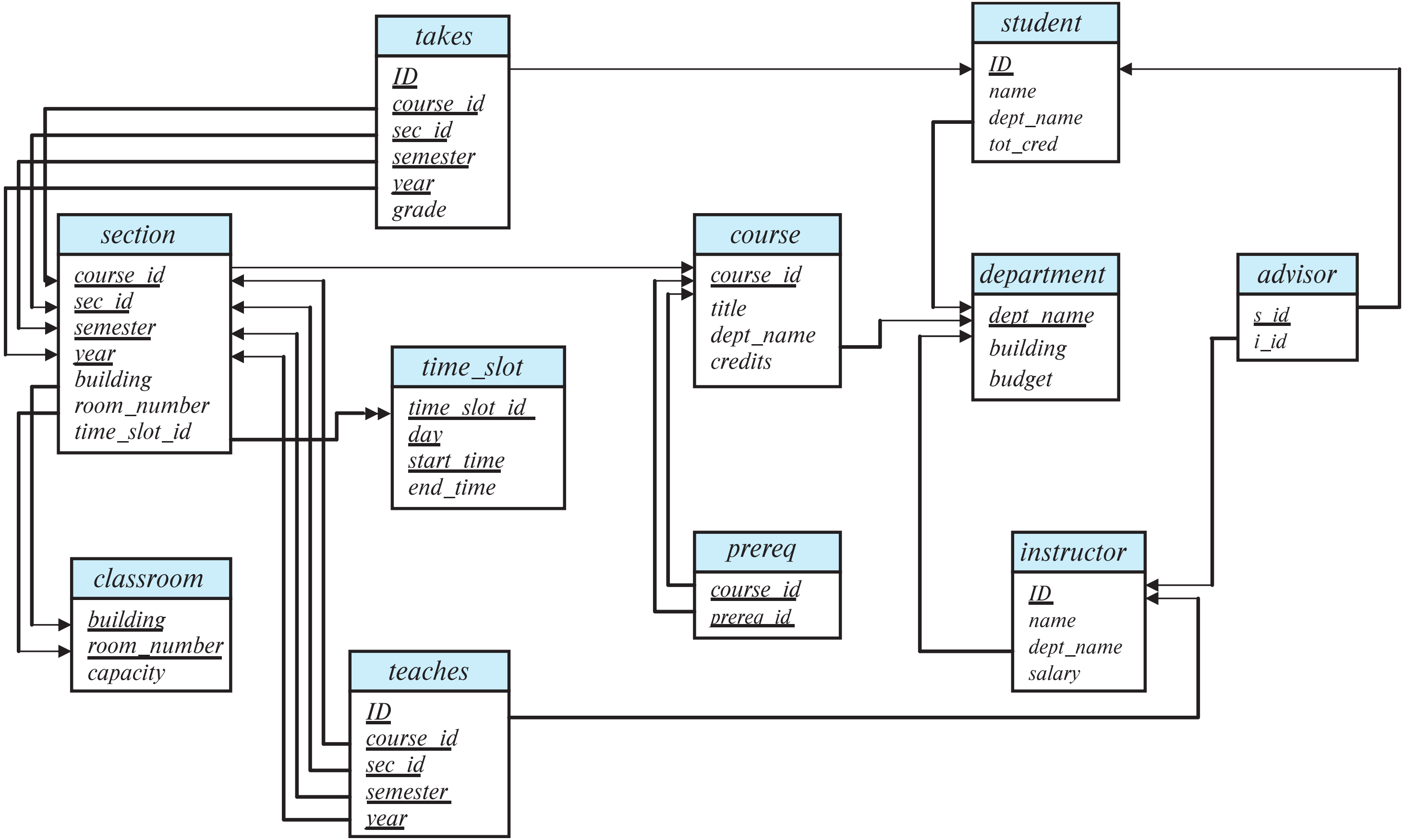
- **unique** $(A_1, A_2, \ldots, A_m)$
  - attributes $A_1, A_2, \ldots, A_m$ together form a superkey
  - Allow attributes to be null (in contrast to primary keys)

check子句

- **check** (P), where P is a predicate
  - E.g., declare branch_name as the primary key for relation **branch** and ensure that the values of assets are non-negative

```
create table branch
      (branch_name      char(15),
       branch_city      char(30),
       assets           integer,
       primary key (branch_name),
       check (assets >= 0));
```

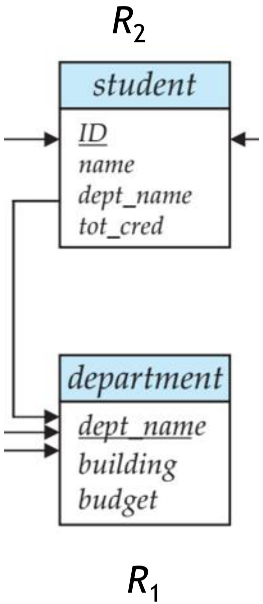
参照完整性(Referential Integrity)



参照完整性(续)

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation
- **Formal definition**
  - Let $r_1(R_1)$ and $r_2(R_2)$ be relations with primary keys K_1 and K_2, respectively
  - If the subset α of R_2 is a foreign key referencing K_1 in relation r_1, for every t_2 in r_2, there must be a tuple t_1 in r_1 such that $t_1[K_1] = t_2[\alpha]$
  - Referential integrity constraint also called subset dependency since its can be written as

$$\Pi_{\alpha}(r_2) \subseteq \Pi_{K_1}(r_1)$$



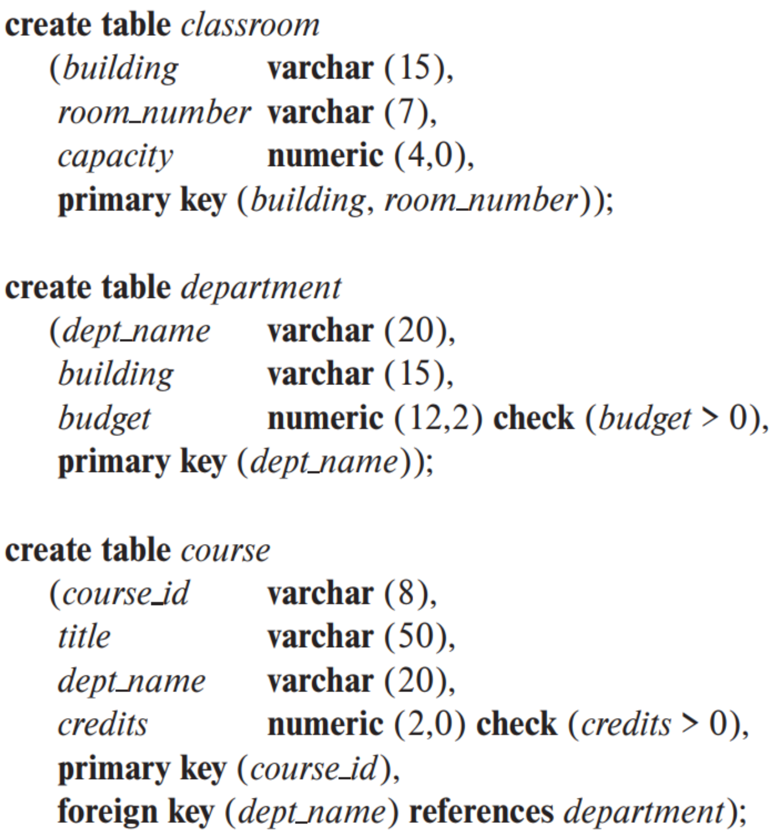
参照完整性(续)

- By default, a foreign key references the primary key attributes of the referenced table
  - ***foreign key*** *(account_number)* ***references*** *account*
- Short form for specifying a single column as foreign key
  - *account_number char (10)* ***references*** *account*

```
create table customer
      (customer_name          char(20),
      customer_street         char(30),
      customer_city           char(30),
      primary key (customer_name))；
create table branch
      (branch_name            char(15),
      branch_city             char(30),
      assets                  integer,
      primary key (branch_name))；
```

```
create table account
      (account_number      char(10),
      branch_name          char(15),
      balance              integer,
      primary key (account_number),
      foreign key (branch_name) references branch)；
create table depositor
      (customer_name       char(20),
      account_number       char(10),
      primary key (customer_name, account_umber),
      foreign key (account_number) references account,
      foreign key (customer_name) references customer)
      ；
```

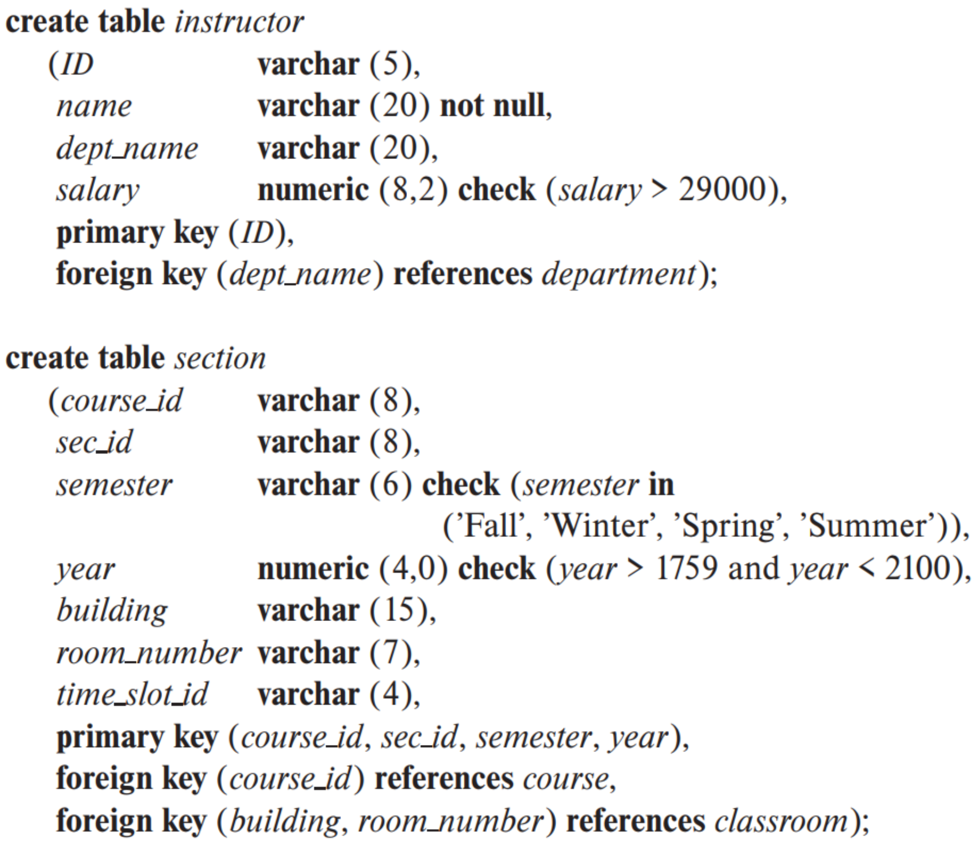
参照完整性(续)



```
create table classroom
    (building      varchar (15),
     room_number   varchar (7),
     capacity      numeric (4,0),
     primary key (building, room_number));

create table department
    (dept_name     varchar (20),
     building      varchar (15),
     budget        numeric (12,2) check (budget > 0),
     primary key (dept_name));

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



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

create table section
    (course_id     varchar (8),
     sec_id        varchar (8),
     semester      varchar (6) check (semester in
                        ('Fall', 'Winter', 'Spring', 'Summer')),
     year          numeric (4,0) check (year > 1759 and year < 2100),
     building      varchar (15),
     room_number   varchar (7),
     time_slot_id  varchar (4),
     primary key (course_id, sec_id, semester, year),
     foreign key (course_id) references course,
     foreign key (building, room_number) references classroom);
```

参照完整性(续)

- 例:

```
create table person (
        ID  char(10),
        name char(40),
        spouse char(10),
        primary key ID,
        foreign key spouse references person)
```

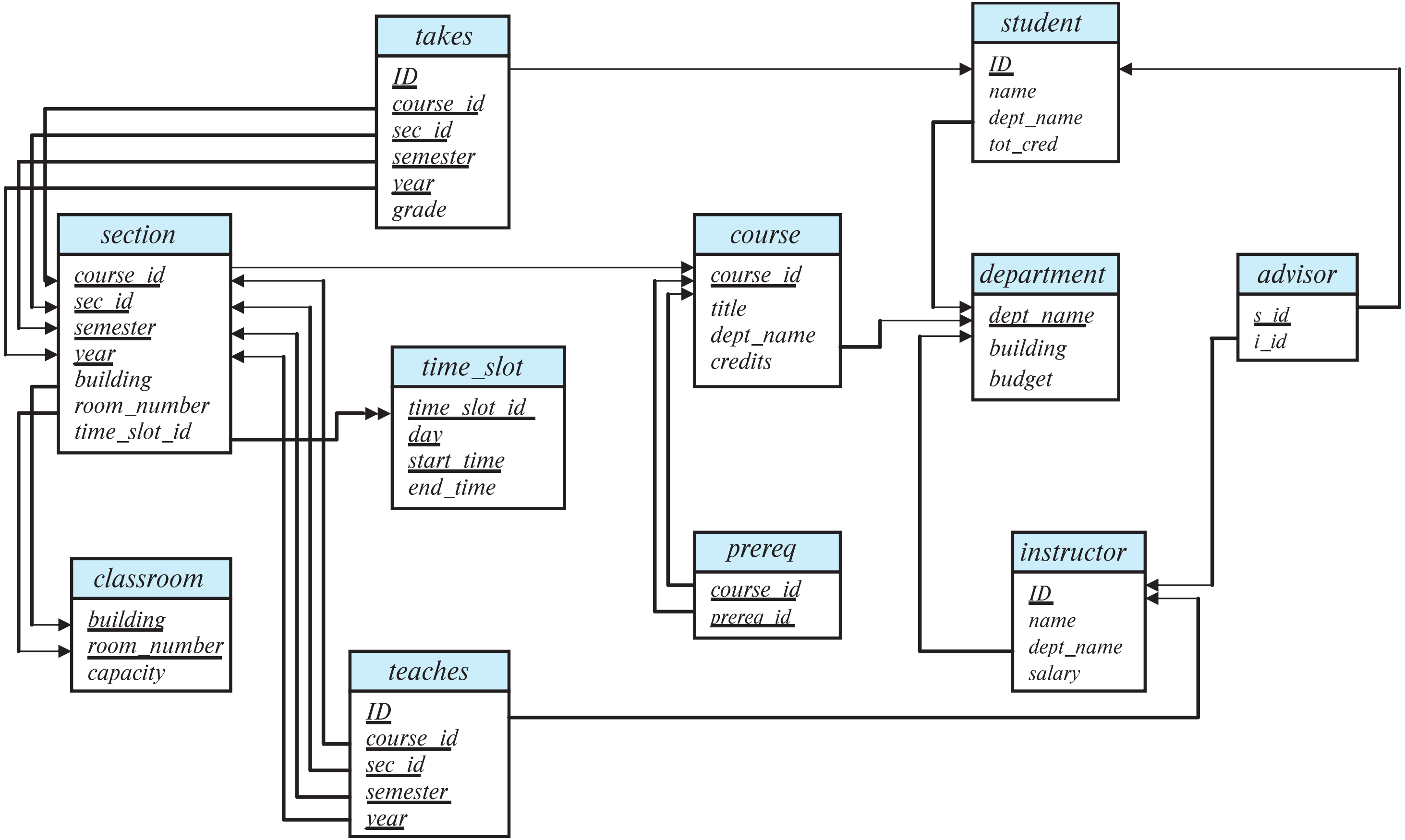
思考：如何插入一对夫妻的信息而不违反约束条件?

- How to insert a tuple without causing constraint violation?
  - Set spouse to null initially, update after inserting all persons (not possible if spouse attributes declared to be not null)
  - OR defer constraint checking
    - ***set constraints*** *constraint_list* ***deferred***

复杂的check子句

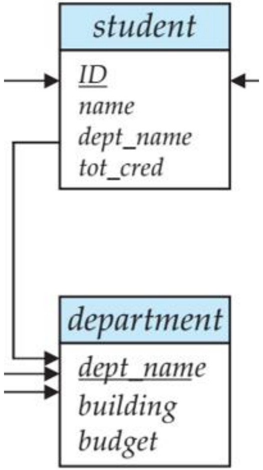
- For relation section: ***check*** *(time_slot_id* ***in*** *(**select** time_slot_id* ***from*** *time_slot))*
  - 思考：Can we use a foreign key here?
- Every section is taught by at least one instructor
  - ***check*** *((course_id, sec_id, semester, year)* ***in*** *(**select** course_id, sec_id, semester, year* ***from*** *teaches))*
- Unfortunately, subquery in check clause is not supported by many database systems
  - Alternative: triggers (later)

大学数据库模式


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

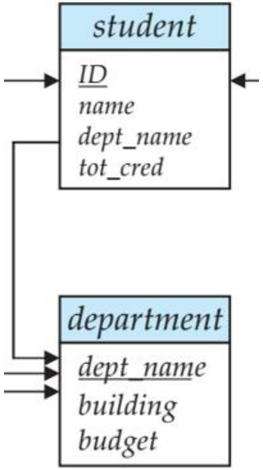
数据库修改

- r_2's attribute set α reference r_1 on attributes K
- **Insert**
  - If a tuple t_2 is inserted into r_2, the system must ensure that there is a tuple t_1 in r_1 such that $t_1[K] = t_2[\alpha]$, i.e., $t_2[\alpha] \in \Pi_K(r_1)$
- **Delete**
  - If a tuple t_1 is deleted from r_1, the database system must compute the set of tuples in r_2 that reference t_1: $\sigma_{\alpha=t_1[K]}(r_2)$
  - If this set is not empty
    - either the delete command is rejected as an error, or
    - the tuples that reference t_1 must be deleted (cascading deletions are possible)



数据库修改(续)

- **Update**
  - If a tuple t_2 is updated in relation r_2 and the update modifies values for foreign key α, then a test similar to the insert case is made
  - If a tuple t_1 is updated in r_1, and the update modifies values for the primary key(*K*), then a test similar to the delete case is made:
    - The system must compute $\sigma_{\alpha=t_1[K]}(r_2)$ using the old value of t_1
    - If this set is not empty
      - the update may be rejected as an error, or
      - the update may be cascaded to the tuples in the set, or
      - the tuples in the set may be deleted.



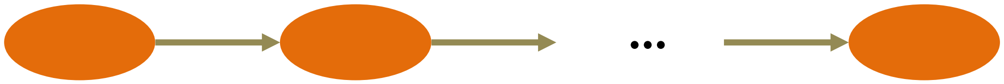
SQL中的级联操作

```
create table course (
    course_id       char(5) primary key,
    title           varchar(20),
    dept_name       varchar(20),
    foreign key (dept_name) references department
           on delete cascade
           on update cascade)
```

- Due to the on delete cascade clauses, if the delete of a tuple in department results in referential-integrity constraint violation, the delete “cascades” to the course relation, and the tuples that refer to the department should be deleted
- Cascading updates are similar

SQL中的级联操作(续)

- If there is a chain of foreign-key dependencies across multiple relations, with on delete cascade specified for each dependency, a deletion or update at the end of the chain can propagate across the entire chain



- Referential integrity is only checked at the end of a transaction
  - Intermediate steps are allowed to violate referential integrity provided later steps remove the violation
  - Otherwise, it would be impossible to create some database states, e.g. inserting two tuples whose foreign keys point to each other
    - E.g., the spouse attribute of relation married_person(name, address, spouse)

SQL中的参照完整性

- Alternative to cascading
  - **on delete set null**
  - **on delete set default**

- Null values in foreign key attributes complicate SQL referential integrity semantics
  - if any attribute of a foreign key is null, the tuple is defined to satisfy the foreign key constraint

断言(Assertions)

- An assertion is a predicate expressing a condition that we wish the database always to satisfy
- An assertion in SQL takes the form

  create assertion <assertion-name> check <predicate>

- When an assertion is made, the system tests it for validity, and tests it again on every update that may violate the assertion
  - This testing may introduce a significant amount of overhead
  - Assertions should be used with great care

例1

- The sum of all loan amounts for each branch must be less than the sum of all account balances at the branch

```
create assertion sum_constraint check
  (not exists
    (select * from branch
    where (select sum(amount) from loan
             where loan.branch_name = branch.branch_name)
                  >=
             (select sum(balance) from account
             where account.branch_name = branch.branch_name)))
```

例2

- Every loan has at least one borrower who has an account with a minimum balance at least $1000

```
create assertion balance_constraint check
(not exists (
   select * from loan
   where not exists (
        select *
        from borrower, depositor, account
        where loan.loan_number = borrower.loan_number
           and borrower.customer_name = depositor.customer_name
           and depositor.account_number = account.account_number
           and account.balance >= 1000)))
```

Note: SQL has no (for all) predicate, so $(\forall x)P \equiv \neg(\exists x(\neg P))$

目录

- Join表达式
- 视图
- 事务
- 完整性约束
- SQL中的数据类型
- SQL中的索引定义
- 授权

SQL内置数据类型

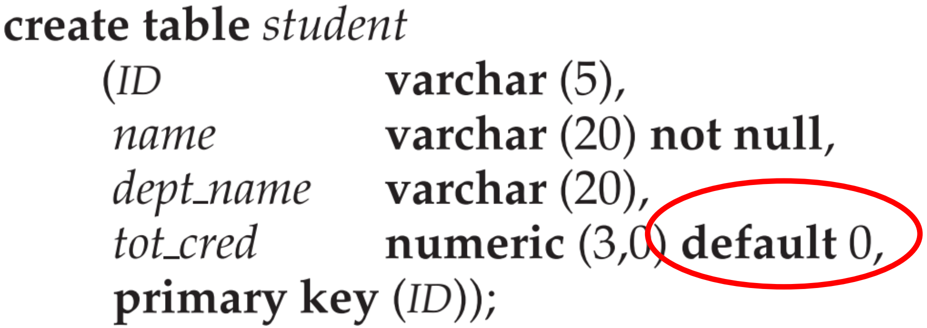
- **date**: containing year, month and day
  - E.g., date ‘2005-07-27’
- **time**: time of day, in hours, minutes and seconds
  - E.g., time ‘09:00:30’ , time ‘09:00:30.75’
- **timestamp**: date + time of day
  - E.g., timestamp ‘2005-07-27 09:00:30.75’
  - timestamp (p): specifies the number of digits after the decimal point
- **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

SQL内置数据类型(续)

- Extract values of individual fields from date/time/timestamp
  - ***extract*** *(year from current_date)*
- Cast string types to date/time/timestamp
  - ***cast*** *<string-valued-expression> as date*
  - ***cast*** *<string-valued-expression> as time*

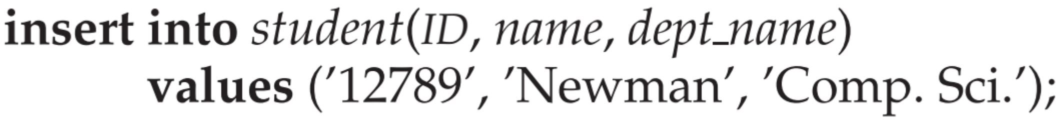
Default Values

- Specify a default value for an attribute



```
create table student
    (ID            varchar (5),
     name          varchar (20) not null,
     dept_name     varchar (20),
     tot_cred      numeric (3,0) default 0,
     primary key (ID));
```

- How an insertion can omit the value for the *tot_cred* attribute?



```
insert into student(ID, name, dept_name)
        values ('12789', 'Newman', 'Comp. Sci.');
```

大对象类型

- Large objects, e.g., photos, videos, and CAD files
  - **blob** (binary large object): object is a large collection of uninterpreted binary data. The interpretation is left to an application outside of the database system

    image **blob** (10MB)

    movie **blob** (2GB)
  - **clob** (character large object): object is a large collection of character data

    book_review **clob**(10KB)
  - When a query returns a large object, a locator (pointer) is returned rather than the large object itself

用户定义的类型

- Create type construct in SQL creates user-defined type
  - ***create type*** *Dollars* ***as numeric*** *(12,2) [****final****]*
- Create domain construct in SQL-92 creates user-defined domain types
  - ***create domain*** *person_name char(20)* ***not null***
- Types and domains are similar. Domains can have constraints, e.g., not null/default values, specified on them

create domain *degree_level varchar(10)*
constraint *degree_level_test* ***check*** *(value in (’Bachelors’, ’Masters’, ’Doctorate’));*

目录

- Join表达式
- 视图
- 事务
- 完整性约束
- SQL中的数据类型
- SQL中的索引定义
- 授权

创建索引

- Many queries reference only a small proportion of the records in a table. It is inefficient to read every record to find a record with particular value
- An **index** on an attribute of a relation is a data structure that allows the database system to find those tuples in the relation that have a specified value for that attribute efficiently, without scanning through all the tuples of the relation.
- Create an index:

create index <name> on <relation-name> (attribute);

例：创建索引

```
create table student
(ID             varchar (5),
name            varchar (20) not null,
dept_name       varchar (20),
tot_cred        numeric (3,0) default 0,
primary key (ID))
create index studentID_index on student(ID)
```

- The following query can be executed by using the index to find the required record, without scanning all records of student

```
select *
from  student
where  ID = '12345'
```

目录

- Join表达式
- 视图
- 事务
- 完整性约束
- SQL中的数据类型
- SQL中的索引定义
- 授权

数据库的安全性

- **Security** - protection from malicious (恶意的) attempts to steal or modify data
  - **Database system level**
    - Authentication and authorization mechanisms to allow specific users access only to the required data
  - **Operating system level**
    - Operating system superusers usually can do anything they want to the database
  - **Network level:** use encryption to prevent
    - Eavesdropping (窃听): unauthorized reading of messages
    - Masquerading (冒充): pretending to be an authorized user or sending messages supposedly from authorized users)

数据库的安全(续)

- Protection from malicious attempts to steal or modify data
  - **Physical level**
    - Physical access to computers allows destruction of data by intruders, and traditional lock-and-key security is needed
    - Computers must also be protected from floods, fire, etc.
  - **Human level**
    - Users must ensure that authorization is not given to intruders
    - Users should be trained on password selection and secrecy

授权(Authorization)

- Types of authorization on **parts** of the database
  - **Read authorization** - allows reading, but not modification of data
  - **Insert authorization** - allows insertion of new data, but not modification of existing data
  - **Update authorization** - allows modification, but not deletion of data
  - **Delete authorization** - allows deletion of data

- Types of authorization to **modify** the database schema
  - **Index authorization** - allows creation and deletion of indices
  - **Resources authorization** - allows creation of new relations
  - **Alteration authorization** - allows addition or deletion of attributes
  - **Drop authorization** - allows deletion of relations

视图的授权

- **View**
  - Simplify usage of the system and enhance security by allowing users access only to data they need for their job
- Suppose a bank clerk needs to know the names of the customers of each branch, but is not authorized to see specific loan information
  - The cust_loan view is defined as follows:

```
create view cust_loan as
    select branch_name, customer_name
    from   borrower, loan
   where borrower.loan_number = loan.loan_number
```

- The clerk is authorized to see the result of the query:

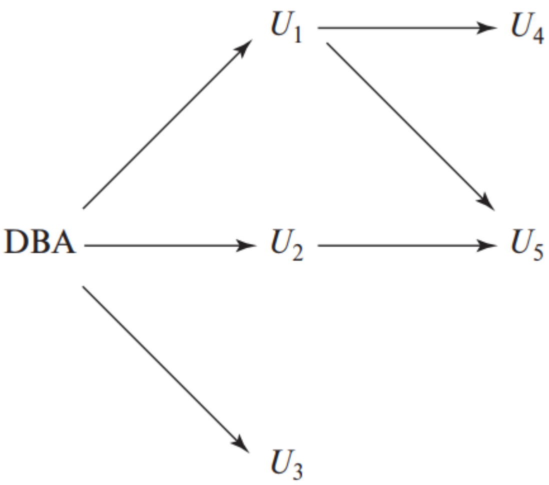
```
select * from cust_loan
```

视图的授权(续)

- A combination of **relational-level security** and **view-level security** can precisely limit a user's access to the data that he needs
- Creation of a view does not require **resources authorization** since no real relation is being created
- The creator of a view gets only those privileges that provide no additional authorization beyond that he already had
  - E.g., if creator of view cust_loan had only read authorization on borrower and loan, he gets only read authorization on cust_loan

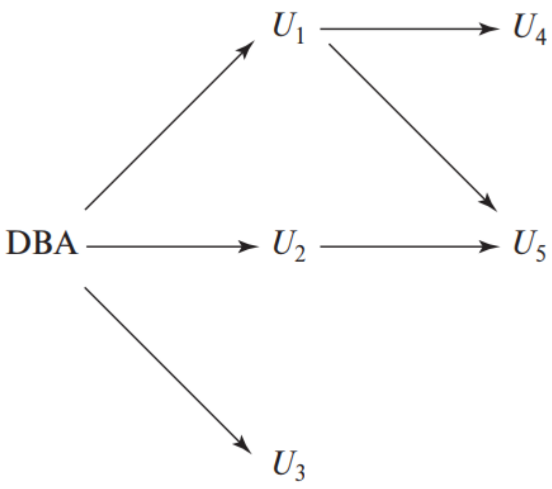
权限传递与授予

- The passage of authorization from one user to another may be denoted by an authorization graph(授权图)
  - nodes are the users
  - the root is the database administrator
  - An edge $U_i \rightarrow U_j$ indicates that user U_i has granted authorization to U_j



授权图

- All edges in an authorization graph must be part of some path originating with the root
- If DBA revokes grant from U_1:
  - Grant must be revoked from U_4 since U_1 no longer has authorization
  - Grant must not be revoked from U_5 since U_5 has another authorization path from DBA through U_2
- Must prevent cycles of grants with no path from the root:
  - DBA grants authorization to U_7
  - U_7 grants authorization to U_8
  - U_8 grants authorization to U_7
  - DBA revokes authorization from U_7
  - Must revoke grant from U_7 to U_8 and from U_8 to U_7 since there is no path from DBA to U_7 or to U_8 anymore



授权

- The **grant** statement is used to confer authorization

    grant *<privilege list>*

    on *<relation name or view name>*

    to *<user list>*

- <user list>
    - a user-id
    - public, which allows all valid users the privilege granted
    - a role (more on this later)
- The grantor of the privilege must already hold the privilege on the specified item

SQL中的权限

- **select**: allows read access to relation, or the ability to query using the view
  - E.g., ***grant select on*** *branch* ***to*** *U_1, U_2, U_3*
- **insert**: the ability to insert tuples
- **update**: the ability to update using the SQL update statement
- **delete**: the ability to delete tuples
- **references**: the ability to declare foreign keys when creating relations
- **usage**: In SQL-92, authorize a user to use a specified domain
- **all privileges**: used as a short form for all the allowable privileges

授权权限

- **with grant option**
  - Allow a user who is granted a privilege to pass on the privilege to other users
  - 例如：give U_1 the select privilege on branch and allows U_1 to grant this privilege to others

    grant select on *branch* ***to*** *U_1* ***with grant option***

角色(Roles)

- Permit common privileges for **a class of users** to be specified by creating a “role”
- Privileges can be granted to or revoked from roles, just like users
- Roles can be assigned to users, and even to other roles

```
create role teller
create role manager

grant select on branch to teller
grant update (balance) on account to teller
grant all privileges on account to manager

grant teller to manager
grant teller to alice, bob
grant manager to avi
```

收回权限

- The **revoke** statement is used to revoke authorization

  revoke *<privilege list>*

  on *<relation name or view name>* ***from*** *<user list> [**restrict**|**cascade**]*

- Example:

  revoke *select* ***on*** *branch* ***from*** U_1, U_2, U_3 *cascade*

- Revocation of a privilege from a user may cause other users to lose that privilege, i.e., cascading of the revoke

- Prevent cascading by specifying restrict:

  revoke *select* ***on*** *branch* ***from*** U_1, U_2, U_3 ***restrict***

  – the revoke command fails if cascading revokes are required

收回权限(续)

- <privilege-list> may be all to revoke all privileges
- If <revoke-list> includes public, all users lose the privilege except those granted explicitly
- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation

SQL授权的限制

- SQL does not support authorization at a tuple level
  - E.g., we cannot restrict students to see only their own grades
- With the growth in Web access to databases, database accesses come primarily from application servers
  - End users don't have database user ids, they are all mapped to the same database user id
- The authorization in such cases falls on the application program, without support from SQL
  - **Benefit**
    - fine-grained authorizations, such as to individual tuples, can be implemented by the application
  - **Drawbacks**
    - Authorization must be done in application code, and may be dispersed all over the application
    - Checking for the authorization loopholes (漏洞) becomes very difficult since it requires reading large amounts of application code

审计追踪(Audit Trails)

- An audit trail is a log of all changes (inserts / deletes / updates) to the database along with information such as
  - which user performed the change
  - when the change was performed
- Used to track erroneous/fraudulent(欺骗性的) updates
- Can be implemented using **triggers**, but many database systems provide direct support

加密

- Data can be encrypted when database authorization provisions do not offer sufficient protection
- Properties of good encryption techniques:
  - Relatively simple for authorized users to encrypt and decrypt data
  - Encryption scheme depends not on the secrecy of the algorithm but on the secrecy of a parameter of the algorithm called the encryption key (密钥)
  - Extremely difficult for an intruder to determine the encryption key

加密(续)

- **Data Encryption Standard (DES)**
  - Substitutes characters and rearranges their order on the basis of an encryption key which is provided to authorized users via a secure mechanism
  - Scheme is no more secure than the key transmission mechanism since the key has to be shared
- **Advanced Encryption Standard (AES)**
  - a new standard replacing DES, and is based on the Rijndael algorithm, but is also dependent on shared secret keys

加密(续)

- **Public-key encryption**
  - each user has two keys:
    - public key – used to encrypt data, but cannot be used to decrypt data
    - private key -- used to decrypt data
  - Encryption scheme is impossible or extremely hard to decrypt data given only the public key
  - The RSA public-key encryption scheme is based on the hardness of factoring a very large number (100's of digits) into its prime components.

小结

- **连接类型与条件**
  - Inner and outer join
  - Left, right and full outer join
  - Natural, using, and on
- **视图**
  - 定义
  - 物化视图
  - 视图更新
- **事务**
  - Commit work
  - Rollback work
- **完整性约束**
  - 实体完整性、域约束、唯一性约束
  - Check子句
  - 参照完整性
  - 断言

- **数据类型**
  - 日期与时间类型
  - 默认值
  - 大对象
  - 用户自定义类型
- **索引定义**
- **授权**
  - 权限的授予与收回
  - 权限：select、insert、update、all privileges
  - 角色
  - 视图的授权
  - 行级授权

作业

- **Exercises**
  - 4.7, 4.16
- **Submission**
  - Canvas上提交，上传单个PDF文件
  - Deadline: 2024年4月3日