Principles of Database Systems

Intermediate SQL

主讲教师：
薛昕惟
办公室：信息楼320B
xuexinwei@dlut.edu.cn

Join Expressions

student(*ID*, *name*, *dept_name*, *tot_cred*)

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

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

ID	*course_id*	*sec_id*	*semester*	*year*	*grade*
00128	CS-101	1	Fall	2009	A
00128	CS-347	1	Fall	2009	A-
12345	CS-101	1	Fall	2009	C
12345	CS-190	2	Spring	2009	A
12345	CS-315	1	Spring	2010	A
12345	CS-347	1	Fall	2009	A
19991	HIS-351	1	Spring	2010	B
23121	FIN-201	1	Spring	2010	C+
44553	PHY-101	1	Fall	2009	B-
45678	CS-101	1	Fall	2009	F
45678	CS-101	1	Spring	2010	B+
45678	CS-319	1	Spring	2010	B
54321	CS-101	1	Fall	2009	A-
54321	CS-190	2	Spring	2009	B+
55739	MU-199	1	Spring	2010	A-
76543	CS-101	1	Fall	2009	A
76543	CS-319	2	Spring	2010	A
76653	EE-181	1	Spring	2009	C
98765	CS-101	1	Fall	2009	C-
98765	CS-315	1	Spring	2010	B
98988	BIO-101	1	Summer	2009	A
98988	BIO-301	1	Summer	2010	*null*

Review Join and Natural Join

- **select** *
  from student, takes
  where student.ID = takes.ID;

- **select** *
  from student **natural join** takes;

- **select** *
  from student **join** takes **using** (ID);

Join Conditions

- SQL supports another form of join, in which an arbitrary join condition can be specified.
- **select** *
  from student **join** takes **on** student.ID=takes.ID;
- The difference between **join…on** and **natural join** is that the result of join…on has the ID attribute listed twice.

Join Conditions

- **select** *
  from student **join** takes **on** student.ID=takes.ID;

- **select** *
  from student, takes
  where student.ID = takes.ID;

- Good reasons for introducing the **on** condition:
  - an SQL query is often more readable if the join condition is specified in the **on clause** and the rest of the conditions appear in the **where clause**
  - In **outer join**, on conditions do behave in a manner different from where conditions

Try...

- **For all students**, find their ID, name, dept name, and tot_cred, along with the courses that they have taken.

- Incorrect version
- **select** *
  from student, takes
  where student.ID = takes.ID;

Outer Joins

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

ID	*course_id*	*sec_id*	*semester*	*year*	*grade*
00128	CS-101	1	Fall	2009	A
00128	CS-347	1	Fall	2009	A-
12345	CS-101	1	Fall	2009	C
12345	CS-190	2	Spring	2009	A
12345	CS-315	1	Spring	2010	A
12345	CS-347	1	Fall	2009	A
19991	HIS-351	1	Spring	2010	B
23121	FIN-201	1	Spring	2010	C+
44553	PHY-101	1	Fall	2009	B-
45678	CS-101	1	Fall	2009	F
45678	CS-101	1	Spring	2010	B+
45678	CS-319	1	Spring	2010	B
54321	CS-101	1	Fall	2009	A-
54321	CS-190	2	Spring	2009	B+
55739	MU-199	1	Spring	2010	A-
76543	CS-101	1	Fall	2009	A
76543	CS-319	2	Spring	2010	A
76653	EE-181	1	Spring	2009	C
98765	CS-101	1	Fall	2009	C-
98765	CS-315	1	Spring	2010	B
98988	BIO-101	1	Summer	2009	A
98988	BIO-301	1	Summer	2010	*null*

Observe that student Snow, with ID 70557, has not taken any courses

Outer Joins

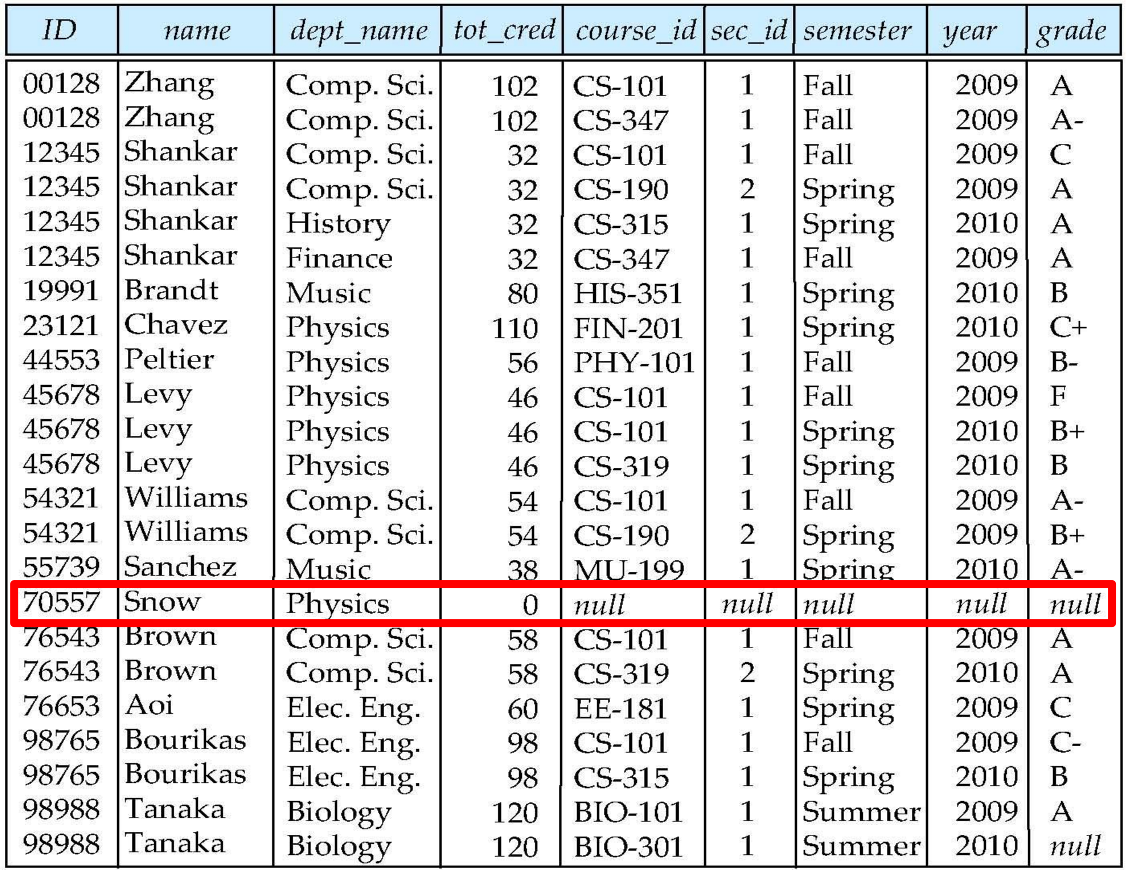
- An extension of the join operation that **avoids loss of information**. (避免信息丢失)
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join. (首先进行连接，之后加入一个关系中与另一关系任何元组都不匹配的元组)
- Uses null values.

Left Outer Join

select *
from student **natural left outer join** takes;

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

- The **left outer join** preserves tuples only in the relation named before (to the left of) the **left outer join** operation.

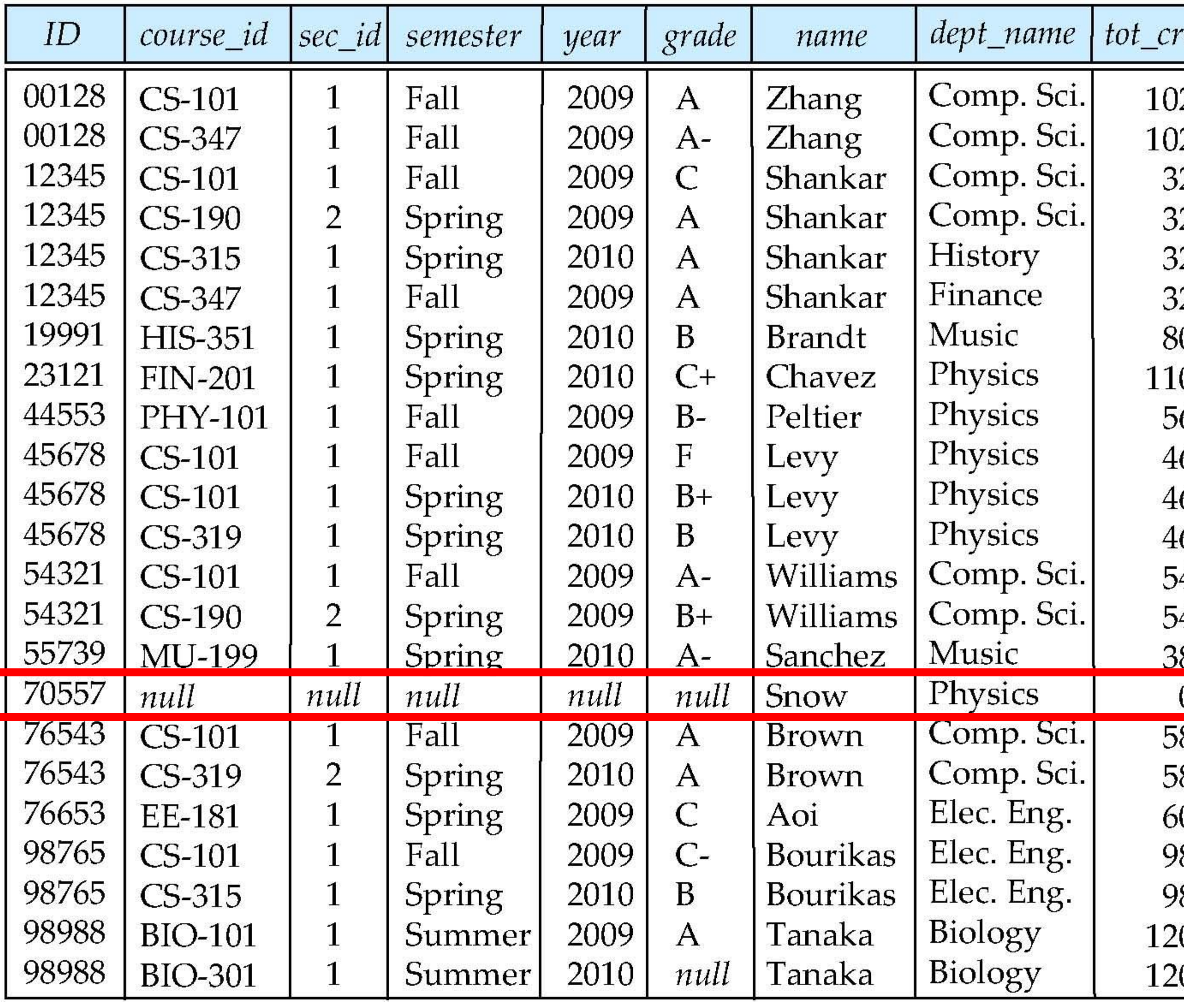


ID	*name*	*dept_name*	*tot_cred*	*course_id*	*sec_id*	*semester*	*year*	*grade*
00128	Zhang	Comp. Sci.	102	CS-101	1	Fall	2009	A
00128	Zhang	Comp. Sci.	102	CS-347	1	Fall	2009	A-
12345	Shankar	Comp. Sci.	32	CS-101	1	Fall	2009	C
12345	Shankar	Comp. Sci.	32	CS-190	2	Spring	2009	A
12345	Shankar	History	32	CS-315	1	Spring	2010	A
12345	Shankar	Finance	32	CS-347	1	Fall	2009	A
19991	Brandt	Music	80	HIS-351	1	Spring	2010	B
23121	Chavez	Physics	110	FIN-201	1	Spring	2010	C+
44553	Peltier	Physics	56	PHY-101	1	Fall	2009	B-
45678	Levy	Physics	46	CS-101	1	Fall	2009	F
45678	Levy	Physics	46	CS-101	1	Spring	2010	B+
45678	Levy	Physics	46	CS-319	1	Spring	2010	B
54321	Williams	Comp. Sci.	54	CS-101	1	Fall	2009	A-
54321	Williams	Comp. Sci.	54	CS-190	2	Spring	2009	B+
55739	Sanchez	Music	38	MU-199	1	Spring	2010	A-
70557	Snow	Physics	0	*null*	*null*	*null*	*null*	*null*
76543	Brown	Comp. Sci.	58	CS-101	1	Fall	2009	A
76543	Brown	Comp. Sci.	58	CS-319	2	Spring	2010	A
76653	Aoi	Elec. Eng.	60	EE-181	1	Spring	2009	C
98765	Bourikas	Elec. Eng.	98	CS-101	1	Fall	2009	C-
98765	Bourikas	Elec. Eng.	98	CS-315	1	Spring	2010	B
98988	Tanaka	Biology	120	BIO-101	1	Summer	2009	A
98988	Tanaka	Biology	120	BIO-301	1	Summer	2010	*null*

Right Outer Join

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

- The **right outer join** preserves tuples only in the relation named after (to the right of) the **right outer join operation.**



ID	*course_id*	*sec_id*	*semester*	*year*	*grade*	*name*	*dept_name*	*tot_cr*
00128	CS-101	1	Fall	2009	A	Zhang	Comp. Sci.	102
00128	CS-347	1	Fall	2009	A-	Zhang	Comp. Sci.	102
12345	CS-101	1	Fall	2009	C	Shankar	Comp. Sci.	32
12345	CS-190	2	Spring	2009	A	Shankar	Comp. Sci.	32
12345	CS-315	1	Spring	2010	A	Shankar	History	32
12345	CS-347	1	Fall	2009	A	Shankar	Finance	32
19991	HIS-351	1	Spring	2010	B	Brandt	Music	80
23121	FIN-201	1	Spring	2010	C+	Chavez	Physics	110
44553	PHY-101	1	Fall	2009	B-	Peltier	Physics	56
45678	CS-101	1	Fall	2009	F	Levy	Physics	46
45678	CS-101	1	Spring	2010	B+	Levy	Physics	46
45678	CS-319	1	Spring	2010	B	Levy	Physics	46
54321	CS-101	1	Fall	2009	A-	Williams	Comp. Sci.	54
54321	CS-190	2	Spring	2009	B+	Williams	Comp. Sci.	54
55739	MU-199	1	Spring	2010	A-	Sanchez	Music	38
70557	*null*	*null*	*null*	*null*	*null*	Snow	Physics	0
76543	CS-101	1	Fall	2009	A	Brown	Comp. Sci.	58
76543	CS-319	2	Spring	2010	A	Brown	Comp. Sci.	58
76653	EE-181	1	Spring	2009	C	Aoi	Elec. Eng.	60
98765	CS-101	1	Fall	2009	C-	Bourikas	Elec. Eng.	98
98765	CS-315	1	Spring	2010	B	Bourikas	Elec. Eng.	98
98988	BIO-101	1	Summer	2009	A	Tanaka	Biology	120
98988	BIO-301	1	Summer	2010	*null*	Tanaka	Biology	120

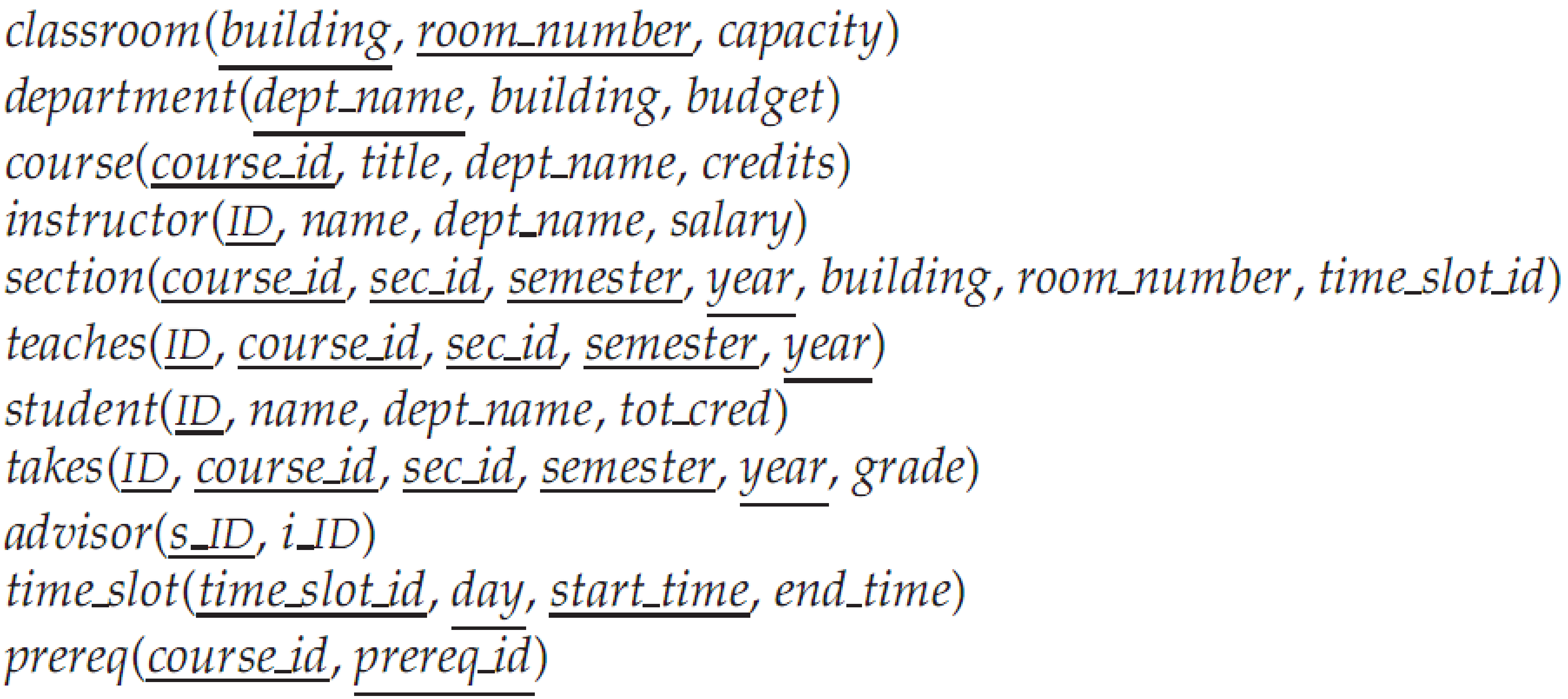
Full Outer Join

- The **full outer join** preserves tuples in both relations.
- Display a list of all students in the Comp. Sci. department, along with the course sections, if any, that they have taken in Spring 2009; all course sections from Spring 2009 must be displayed, even if no student from the Comp. Sci. department has taken the course section.

```
select *
from (select *
        from student
        where dept name= ’Comp. Sci’)
        natural full outer join
       (select *
        from takes
        where semester = ’Spring’ and year = 2009);
```

Try…

- Find all students who have not taken a course



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

Try…

- Find all students who have not taken a course
  - **select** ID
    from student **left outer join** takes **on** student.ID=takes.ID
    where course_id **is** null;

Comparison

- **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**;**

Joined Relations

- The default **join** type, when the join clause is used without the outer prefix is the **inner join.**

- **select ***
  from student ***join*** takes ***on*** student.ID=takes.ID

- **select ***
  from student ***inner join*** takes ***on*** student.ID=takes.ID**;**

Joined Relations

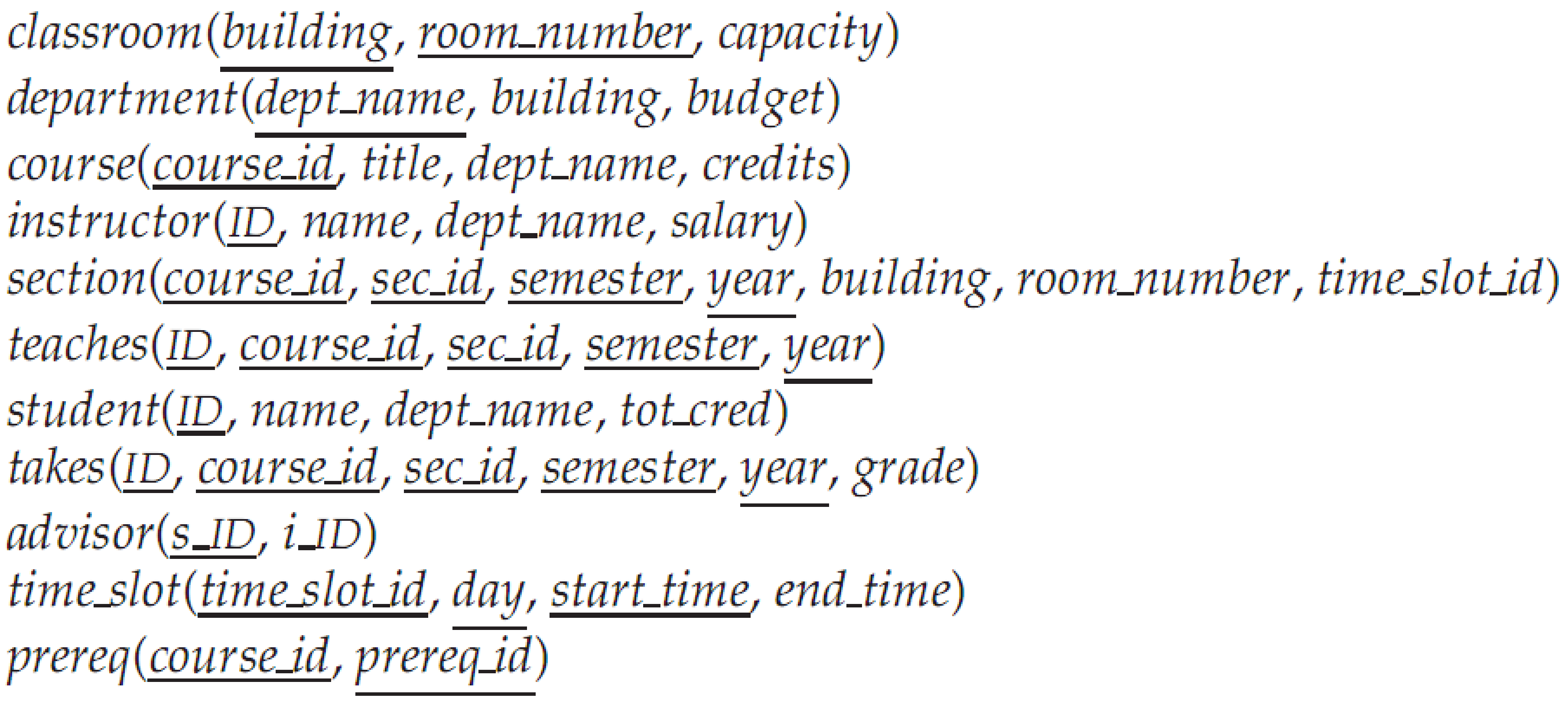
- **Join operations** take two relations and return as a result another relation.
- These additional operations are typically used as subquery expressions in the **from** clause
- **Join condition** (连接条件)– defines which tuples in the two relations match, and what attributes are present in the result of the join.
- **Join type**(连接类型) – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

Join types
inner join
left outer join
right outer join
full outer join

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

Try…

- Find the information of all courses, along with their prerequisite course ID.



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

Try…

- Find the information of all courses, along with their prerequisite course ID.

  - **select** *

    from course **left outer join** prereq **on** course.course_id=prereq.course_id

Constituent Parts of SQL (SQL组成部分)

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

Views

Views

- In some cases, it is not desirable for all users to see the entire logical model.

- Consider a person who needs to know an instructors name and department, but not the salary. This person should see a relation described, in SQL, by

```
select ID, name, dept_name
from instructor
```

- **Any disadvantages?**

Views

- A **view** provides a mechanism to hide certain data from the view of certain users.
- Any relation that is not of the conceptual model but is made visible to a user as a "**virtual relation**" is called a **view**.(不是概念模型的一部分，对用户可见的"虚关系")

Views

- SQL provides the **create view** command to create a view, which is stored in the database **in the long run**
  - **create view** v **as** <query expression>
  - where
    - <query expression> is any legal expression
    - the **view** name is represented by v

- View definition is not the same as creating a new relation by evaluating the query expression. (创建视图与创建关系不同)
  - Rather, a view definition causes the **saving of an expression**; the expression is substituted into queries using the view. (存储的是表达式)

Try…

- A view of instructors without their salary

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

Try…

- Find all instructors in the Biology department

```
select name
  from faculty
  where dept_name = ‘Biology’
```

Try…

- Create a view of department salary totals

```
    create view
departments_total_salary(dept_name, total_salary)
as
      select dept_name, sum (salary)
      from instructor
     group by dept_name;
```

Views Defined Using Other Views

- Explain the following views:

- **create view** physics_fall_2009 **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 = '2009';
```

- **create view** physics_fall_2009_watson **as**

```
    select course_id, room_number
    from physics_fall_2009
    where building= 'Watson';
```

Views Defined Using Other Views

- Expand use of a view in a query/another view

```
create view physics_fall_2009_watson as
(select course_id, room_number
from (select course.course_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 = '2009')
where building= 'Watson';
```

Materialized Views

- **Materializing a view**(物化视图): **create a physical table** containing all the tuples in the result of the query defining the view

- If relations used in the query are updated, the materialized view result becomes out of date
  - Need to **maintain** the view(维护视图), by updating the view whenever the underlying relations are updated.

Drop View

- The **Drop View** command deletes the definition the view from the **data dictionary**.

```
drop view view_name;
```

Other views depending on this dropped view should be deleted explicitly.

Update of a View

- Add a new tuple to faculty view which we defined earlier

  insert into faculty **values** (’30765’, ’Green’, ’Music’);

- Two reasonable approaches:
  - Reject the insertion
  - Insert the tuple

    (’30765’, ’Green’, ’Music’, null)

    into the instructor relation

    (必须转化为对实际关系的修改)

Some Updates cannot be Translated Uniquely

- **create view** instructor_info **as**
  select ID, name, building
  from instructor, department
  where instructor.dept_name= department.dept_name;

- **insert into** instructor_info **values** ('69987', 'White', 'Taylor');
  - which department, if multiple departments in Taylor?
  - what if no department is in Taylor?

Some Updates cannot be Translated Uniquely

- Most SQL implementations allow updates only on simple views
  - The **from** clause has only **one** database relation.
  - The **select** clause contains only attribute names of the relation, and does **not have any expressions**, **aggregates**, or **distinct** specification.
  - Any attribute not listed in the **select** clause can be set to null.
  - The query does **not have** a **group** by or **having** clause.

And Some Not at All

- **create view** history_instructors **as**
  select *
  from instructor
  where dept_name= ’History’;

- What happens if we insert (’25566’, ’Brown’, ’Biology’, 100000) into history_instructors?

- **with check option:** if a tuple inserted into the view does not satisfy the view’s **where clause** condition, the insertion is rejected by the database system

Transactions

Transactions

- A transaction is a sequence of queries and update statements on DB, executed as a single, and are started implicitly and terminated by one of commit work(提交)or rollback/abort work
  - **Commit work** : makes the updates performed by the transaction become permanent in the database.
  - **Rollback work:** undoes all the updates performed by the SQL statements in the transaction.

Transactions

- Unit of work
- Atomic transaction
  - either fully executed or rolled back as if it never occurred
- Transactions begin implicitly
  - Ended by **commit work** or **rollback work**
- But default on most databases: each SQL statement commits automatically
  - Can turn off auto commit for a session (e.g. using API)
  - In SQL:1999, can use: **begin atomic** …. **end**
    - Not supported on most databases

Integrity Constraints

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.
  - An instructor name cannot be *null.*
  - No two instructors can have the same instructor ID.
  - The budget of a department must be greater than $0.00.

Integrity Constraints on a Single Relation

- primary key
- not null
- Unique
- foreign keys
- check (P), where P is a predicate

Not Null

- Declare name and budget to be **not null**
  - name **varchar**(20) **not null**
  - budget **numeric**(12,2) **not null**

Unique Constraints

- **unique** (A_1, A_2, …, A_m)
  - The unique specification states that the attributes A_1, A_2, … A_m form a **candidate key**.
  - Candidate keys are **permitted to be null** (in contrast to primary keys).

The check clause

- The **check** clause is applied to relation declaration
  - check (P), where P is a predicate which must be satisfied by every tuple in the relation.

- Example: ensure that the budget of a department must be greater than $0.00
  - **create table** department

    (dept name **varchar (20),**

    building **varchar (15),**

    budget **numeric (12,2),**

    primary key (dept name**)**

    check(budget>0**));**

Try…

- Ensures that semester is one of fall, winter, spring or summer:

```
  create table section (
  course_id  varchar (8),
  sec_id  varchar (8),
  semester  varchar (6),
  year  numeric (4,0),
  building  varchar (15),
  room_number  varchar (7),
 time slot id  varchar (4),
 primary key (course_id, sec_id, semester, year),
 check (semester  in (’Fall’, ’Winter’, ’Spring’, ’Summer’))
);
```

Referential Integrity

- 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.

  - Example: If "Biology" is a department name appearing in one of the tuples in the *course* relation, then there exists a tuple in the *department* relation for "Biology". -- **foreign key**

```
create table course (
    course_id   char(5) primary key,
    title        varchar(20),
    dept_name varchar(20) references department
)
```

Cascading Actions in Referential Integrity

- When the DB is modified by **Insert, Delete**, and **Update**, the tests must be made in order to preserve the referential integrity constraint.

- **create table** course (

```
    …
    dept_name varchar(20),
    foreign key (dept_name) references department
                on delete cascade
                 on update cascade,
    . . .
)
```

- alternative actions to **cascade**: **set null**, **set default**

Integrity Constraint Violation During Transactions

- E.g. **create table** person (
  ID **char**(10) **primary key** ,
  name **char**(40),
  spouse **char**(10),
  foreign key spouse **references** person)

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

Defer Constraint Checking

- The SQL standard allows a clause **initially deferred** to be added to a constraint specification.

- For constraints declared as **deferrable**, executing a statement **set constraints** constraint-list **deferred** as part of a transaction causes the checking of the specified constraints to be deferred to the end of that transaction.

Complex Check Conditions and Assertions

- **check** (time_slot_id
  in (**select** time_slot_id **from** time_slot))
- Unfortunately: subquery in check clause not supported by pretty much any database.

Assertions

- An **assertion** is a predicate expressing a condition that we wish the database always to satisfy.
  - e.g. domain constraints, referential-integrity constraint

- An assertion in SQL takes the form

  create assertion <assertion-name> **check** <predicate>

Assertions

- E.g. The value of the attribute tot_cred for each student must equal the sum of credits of courses that the student has completed successfully.

```
create assertion credits_earned_constraint check
  (not exists (select ID from student
    where tot_cred < > (

           select sum(credits)

           from takes join course

           on takes.course_id= course.course_id

           where student.ID=takes.ID and grade is not

                                    null and grade < > 'F')
```

Assertions

- When an assertion is made, the DBMS tests it for validity. Any modification to DB is allowed only if it does not cause that assertion to be violated.
  - This testing may introduce a significant amount of overhead, hence **assertions should be used with great care.**

- Not supported by every DBMS.

Review

Review

- Join Expressions
  - left outer join, right outer join, full outer join
  - inner join = join
  - Join types and join conditions
- Views
  - Create view
  - Use views in SQL queries
  - Update view: with check option
- Transactions
  - Atomic
  - Commit work, Rollback work

Review

- Integrity Constraints
  - **Not null**
  - **unique** (A_1, A_2, …, A_m), candidate key, null
  - **Check**(P)
  - Referential Integrity, foreign key, **on delete/update cascade**, on delete/update set null, on delete/update set default
  - defer constraint checking
  - **create assertion** <assertion-name> **check** <predicate>