Principles of Database Systems

Introduction to SQL(3)

Queries on Multiple Relations

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

Database Used

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

Queries on Multiple Relations

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

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

Queries on Multiple Relations

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

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

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

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

Cartesian Product: *instructor X teaches*

instructor

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

teaches

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009

inst.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Join(连接)

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

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

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

Join

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

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

Join

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

Database Used

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

Join

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

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

Join

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

Database Used

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

Join

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

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

Join

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

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

```
select instructor.ID,course_id
from instructor join teaches on
        instructor.ID=teaches.ID
```

Natural Join(自然连接)

- Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column 自然连接仅考虑那些在两个关系模式中都出现的属性上取值相同的元素对，每一个相同属性列仅留一个拷贝。
- **select** *
  from *instructor* **natural join** *teaches*;

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

Natural Join Example

- List the names of instructors along with the course ID of the courses that they taught.
  - **select** name, course_id
    from instructor, teaches
    where instructor.ID = teaches.ID;

  - **select** name, course_id
    from instructor **natural join** teaches;

- Danger in natural join: beware of unrelated attributes with same name which get equated incorrectly.

- Note: MS SQL Server doesn't support keyword "natural join".

Database Used

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

Self-join(自连接)

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

Self-join

- Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.
  - **select** distinct T. name
    from **instructor as T, instructor as S**
    where T.salary > S.salary and S.dept_name = 'Comp. Sci.'
  - Keyword as is optional and may be omitted
    instructor as T ≡ instructor T
  - Keyword as must be omitted in Oracle
  - The key of self-join is the relation renaming operation

Understanding the Operational Order

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

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

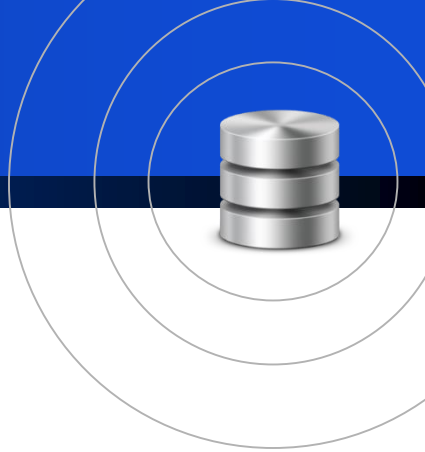
Meaning of an SQL Query

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

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

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

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



Set Operations(集合运算)

Set Operations

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

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

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

Set Operations

- Find courses that ran in Fall 2009 or in Spring 2010

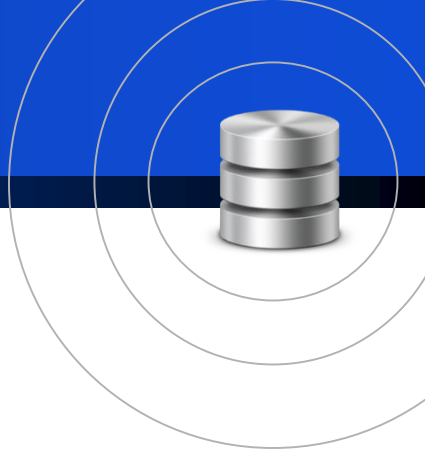
(**select** *course_id* **from** *section* **where** *sem* = 'Fall' **and** *year* = 2009)
union
(**select** *course_id* **from** *section* **where** *sem* = 'Spring' **and** *year* = 2010)

- 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)



Null Values

Null Values

- It is possible for tuples to have a null value, denoted by null, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving null is null
  - Example: 5 + null returns null

Null Values

- The predicate **is null** can be used to check for null values.
  - Example: Find all instructors whose salary is null.
  -
```
select name
from instructor
where salary is null
```

Null Values and Three Valued Logic

- Any comparison with *null* returns *unknown*
  - Example*: 5 < null or null <> null or null = null*
- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - AND: *(true* **and** *unknown)* = *unknown*,
    (false **and** *unknown)* = *false*,
    (unknown **and** *unknown)* = *unknown*
  - NOT*: (***not** *unknown)* = *unknown*
  - "*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*
- **Result of where clause predicate is treated as *false* if it evaluates to *unknown***
  - Try: **select** * **from** instructor **where** ((null+1) is not null);