DATABASE SYSTEM PRINCIPLE - SQL QUERY LANGUAGE

李旭东
LEEXUDONG@NANKAI.EDU.CN
NANKAI UNIVERSITY

OBJECTIVES

- Overview of The SQL Query Language
- Data Definition
- Basic Query Structure
- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions
- Nested Subqueries
- Modification of the Database

©LXD

OVERVIEW OF THE SQL QUERY LANGUAGE

- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
  - Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
  - SQL-86, SQL-89, SQL-92
  - SQL:1999 (language name became Y2K compliant!)
  - SQL-2003, SQL-2006, SQL-2008
  - Not all examples here may work on your particular system.

©LXD

OVERVIEW OF THE SQL QUERY LANGUAGE

- Structured Query Language (SQL)
  - Data-Definition Language, DDL
  - Data-Manipulation Language, DML
  - Integrity完整性
  - View Definition视图定义
  - Transaction Control事务控制
  - Embedded SQL and dynamic SQL嵌入式和动态SQL
  - Authorization授权
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.

©LXD

OBJECTIVES

- Overview of The SQL Query Language
- Data Definition
- Basic Query Structure
- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions
- Nested Subqueries
- Modification of the Database

©LXD

SQL DATA DEFINITION

- SQL DDL
  - The schema for each relation
  - The types of values associated with each attribute
  - The integrity constraints
  - The set of indices to be maintained for each relation
  - The security and authorization information for each relation
  - The physical storage structure of each relation on disk

©LXD

DOMAIN TYPES IN SQL

- char(n). Fixed length character string, with user-specified length n.
- varchar(n). Variable length character strings, with user-specified maximum length n.
- int. Integer (a finite subset of the integers that is machine-dependent).
- smallint. Small integer (a machine-dependent subset of the integer domain type).
- numeric(p,d). Fixed point number, with user-specified precision of p digits, with d digits to the right of decimal point. (ex., numeric(3,1), allows 44.5, not 444.5 or 0.32)
- 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.

©LXD

BASIC SCHEMA DEFINITION

- Create table
- Insert into
- Delete from
- Albert table
- Drop table

©LXD

CREATE TABLE CONSTRUCT

- 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$_1$完整性约束),
  ...,
  (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

©LXD

CREATE TABLE CONSTRUCT

- the **create table** command
- Example:

  create table *instructor* (
  ID **char**(5),
  name **varchar**(20),
  dept_name **varchar**(20),
  salary **numeric**(8,2));

©LXD

INTEGRITY CONSTRAINTS IN CREATE TABLE

- **not null**
- **primary key** (A_1, ..., A_n)
- **foreign key** (A_m, ..., A_n) **references** *r*

  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*);

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

©LXD

CASES: CREATE TABLE

create table department
(dept name varchar (20),
building varchar (15),
budget numeric (12,2),
primary key (dept name);

create table course
(course id varchar (7),
title varchar (50),
dept name varchar (20),
credits numeric (2,0),
primary key (course id),
foreign key (dept name) references department);

©LXD

CASES: CREATE TABLE

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),
foreign key (course id) references course);

©LXD

CASES: CREATE TABLE

create table teaches
(ID varchar (5),
course id varchar (8),
sec id varchar (8),
semester varchar (6),
year numeric (4,0),
primary key (ID , course id, sec id, semester, year),
foreign key (course id, sec id, semester, year) references section,
foreign key (ID) references instructor);

©LXD

UPDATES TO TABLES

- **Insert**
  - **insert into** *instructor* **values** ('10211', 'Smith', 'Biology', 66000);
- **Delete**
  - Remove all tuples from the *student* relation
    - **delete from** *student*
- **Drop Table**
  - **drop table** *r*

©LXD

UPDATES TO TABLES

- **Alter**
  - **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 exiting tuples in the relation are assigned *null* as the value for the new attribute.
  - **alter table** *r* **drop** *A*
    - where *A* is the name of an attribute of relation *r*
    - Dropping of attributes not supported by many databases.

©LXD

OBJECTIVES

- Overview of The SQL Query Language
- Data Definition
- Basic Query Structure
- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions
- Nested Subqueries
- Modification of the Database

©LXD

BASIC STRUCTURE OF SQL QUERIES

- A typical SQL query has the form:

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

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

©LXD

QUERIES ON A SINGLE RELATION

- The **select** clause lists the attributes desired in the result of a query
  - corresponds to the projection operation of the relational algebra
- Example: find the names of all instructors:

  select *name*
  from *instructor*

- NOTE: SQL names are case insensitive
  - E.g., *Name ≡ NAME ≡ name*
  - Some people use upper case wherever we use bold font.

name
Srinivasan
Wu
Mozart
Einstein
El Said
Gold
Katz
Califieri
Singh
Crick
Brandt
Kim

©LXD

QUERIES ON A SINGLE RELATION - DEDUPLICATION去重

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword distinct after select.
- Find the department names of all instructors, and remove duplicates

  select distinct *dept_name*
  from *instructor*

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

  select all *dept_name*
  from *instructor*

©LXD

QUERIES ON A SINGLE RELATION - MORE

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

  select *
  from *instructor*

- An attribute can be a literal with no **from** clause

  select '437'

  - Results is a table with one column and a single row with value "437"
  - Can give the column a name using:

    select '437' **as** *FOO*

©LXD

QUERIES ON A SINGLE RELATION - MORE

- An attribute can be a literal with **from** clause

  select 'A'
  from *instructor*

  - Result is a table with one column and *N* rows (number of tuples in the *instructors* table), each row with value "A"

©LXD

QUERIES ON A SINGLE RELATION - MORE

- 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.
  - Can rename "*salary/12*" using the **as** clause:

    select *ID, name, salary/12* **as** *monthly_salary*

©LXD

THE WHERE CLAUSE

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

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

©LXD

THE WHERE CLAUSE

- Comparison results can be combined using the logical connectives **and**, **or**, and **not**
  - To find all instructors in Comp. Sci. dept with salary > 80000

    select *name*
    from *instructor*
    where *dept_name* = 'Comp. Sci.' **and** *salary* > 80000
- Comparisons can be applied to results of arithmetic expressions.

©LXD

QUERIES ON MULTIPLE RELATIONS

select A1 , A2 ,..., An
from r1 , r2 ,...,rm
where P;

- Each Ai represents an attribute,and each ri a relation.P is a predicate.
  - If the where clause is omitted, the predicate P is true.

©LXD

QUERIES ON MULTIPLE RELATIONS

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

  select *
  from *instructor, teaches*
  - generates every possible instructor – teaches pair, with all attributes from both relations.
  - For common attributes (e.g., *ID*), the attributes in the resulting table are renamed using the relation name (e.g., *instructor.ID*)

©LXD

CARTESIAN PRODUCT 笛卡尔积

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

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

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

©LXD

CASES: CARTESIAN PRODUCT

- Find the names of all instructors who have taught some course and the course_id
  - **select** *name, course_id*
    from *instructor , teaches*
    where *instructor.ID = teaches.ID*
- Find the names of all instructors in the Art department who have taught some course and the course_id
  - **select** *name, course_id*
    from *instructor , teaches*
    where *instructor.ID = teaches.ID* **and** *instructor. dept_name* = 'Art'

©LXD

THE NATURAL JOIN

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

More generally, a **from** clause can be of the form

from $E_1, E_2, \ldots, E_n$

- where each Ei can be a single relation or an expression involving natural joins.

©LXD

CASES: THE NATURAL JOIN

The natural join of the instructor relation with the teaches relation

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

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

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
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

©LXD

CASES: THE NATURAL JOIN

- Way 1

select name, course id
from instructor, teaches
where instructor.ID = teaches.ID;

- Way 2

select name, course id
from instructor natural join teaches;

©LXD

CASES 2: THE NATURAL JOIN

1. **select** *name*, *title*
   from *instructor* **natural join** *teaches*, *course*
   where *teaches.course_id*= *course.course_id*;

2. **select** *name*, *title*
   from *instructor* **natural join** *teaches* **natural join** *course*;

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

©LXD

OBJECTIVES

- Overview of The SQL Query Language
- Data Definition
- Basic Query Structure
- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions
- Nested Subqueries
- Modification of the Database

©LXD

THE RENAME OPERATION

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

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

- 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*

©LXD

STRING OPERATIONS

- SQL includes a string-matching operator for comparisons on character strings. The operator **like** uses patterns that are described using two special characters:
  - percent (%). The % character matches any substring.
  - underscore (_). The _ character matches any character.

©LXD

STRING OPERATIONS

- Find the names of all instructors whose name includes the substring "dar".

  select *name*
  from *instructor*
  where *name* **like** '%dar%'
- Match the string "100%"

  like '100 \%' **escape** '\'

  in that above we use backslash (\) as the escape character.

©LXD

STRING OPERATIONS

- 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.

©LXD

STRING OPERATIONS

- SQL supports a variety of string operations such as
  - concatenation (using "||")
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, etc.

©LXD

ORDERING THE DISPLAY OF TUPLES

- List in alphabetic order the names of all instructors

  select distinct *name*
  from *instructor*
  order by *name*
- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default.
  - Example: **order by** *name* **desc**
- Can sort on multiple attributes
  - Example: **order by** *dept_name, name*

©LXD

WHERE CLAUSE PREDICATES

- SQL includes a **between** comparison operator
- Example: Find the names of all instructors with salary between $90,000 and $100,000 (that is, ≥ $90,000 and ≤ $100,000)
  - **select** *name*
    from *instructor*
    where *salary* **between** 90000 **and** 100000

©LXD

WHERE CLAUSE PREDICATES

- Way 1:
  - **select** *name*, *course_id*
    from *instructor*, *teaches*
    where (*instructor.ID= teaches.ID and dept_name* = 'Biology';
- Way 2:
  - **select** *name*, *course_id*
    from *instructor*, *teaches*
    where (*instructor.ID*, *dept_name*) = (*teaches.ID*, 'Biology');

©LXD

OBJECTIVES

- Overview of The SQL Query Language
- Data Definition
- Basic Query Structure
- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions
- Nested Subqueries
- Modification of the Database

©LXD

SET OPERATIONS

- union
- intersect
- except

©LXD

SET OPERATIONS - UNION

- 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)

©LXD

SET OPERATIONS - INTERSECT

- 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)

©LXD

SET OPERATIONS - EXCEPT

- 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)

©LXD

SET OPERATIONS

- 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*

©LXD

OBJECTIVES

- Overview of The SQL Query Language
- Data Definition
- Basic Query Structure
- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions
- Nested Subqueries
- Modification of the Database

©LXD

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
- 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**

©LXD

NULL VALUES AND THREE VALUED LOGIC

- Three values – *true*, *false*, *unknown*
- Any comparison with *null* returns *unknown*
  - Example: *5 < null or null <> null or null = null*

©LXD

NULL VALUES AND THREE VALUED LOGIC

- Three-valued logic using the 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*

©LXD

OBJECTIVES

- Overview of The SQL Query Language
- Data Definition
- Basic Query Structure
- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions
- Nested Subqueries
- Modification of the Database

©LXD

AGGREGATE FUNCTIONS聚集函数

- 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

©LXD

AGGREGATE FUNCTIONS

- 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;

©LXD

AGGREGATE FUNCTIONS – GROUP BY

- Find the average salary of instructors in each department
  - **select** *dept_name*, **avg** (*salary*) **as** *avg_salary*
    from *instructor*
    group by *dept_name*;

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

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

©LXD

AGGREGATE FUNCTIONS – GROUP BY

- 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*;

©LXD

AGGREGATE FUNCTIONS – HAVING CLAUSE

- Find the names and average salaries of all departments whose average salary is greater than 42000

  select *dept_name*, **avg** (*salary*)
  from *instructor*
  group by *dept_name*
  having avg (*salary*) > 42000;

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

©LXD

AGGREGATE FUNCTIONS

- Total all salaries

  select sum (*salary*)
  from *instructor*
  - Above statement ignores null amounts
  - Result is *null* if there is no non-null amount

©LXD

AGGREGATE FUNCTIONS

- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes
- What if collection has only null values?
  - count returns 0
  - all other aggregates return null

©LXD

OBJECTIVES

- Overview of The SQL Query Language
- Data Definition
- Basic Query Structure
- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions
- Nested Subqueries
- Modification of the Database

©LXD

NESTED SUBQUERIES嵌套子查询

- SQL provides a mechanism for nesting subqueries.
- A subquery is a select-from-where expression that is nested within another query.

```
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);
```

©LXD

NESTED SUBQUERIES嵌套子查询

- The nesting can be done in the following SQL query

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

as follows:

- A_i can be replaced be a subquery that generates a single value.
- r_i can be replaced by any valid subquery
- P can be replaced with an expression of the form:

  B <operation> (subquery)

  Where *B* is an attribute and <operation> to be defined later.

©LXD

SUBQUERIES IN THE WHERE CLAUSE

- A common use of subqueries is to perform tests:
  - For set membership
  - For set comparisons
  - For set cardinality.

©LXD

NESTED SUBQUERIES - SET MEMBERSHIP集合成员测试

- in, not in

■ 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);
```

©LXD

NESTED SUBQUERIES - SET MEMBERSHIP

- 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);
```

©LXD

NESTED SUBQUERIES - SET COMPARISON集合比较

- 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');

©LXD

NESTED SUBQUERIES - SET COMPARISON集合比较

- > some
- < some
- <= some
- >= some
- <> some
  - (not same to not in)
- =some (same to in)

- 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 (read: 5 < some tuple in the relation)

(5 < **some** [0, 5]) = false

(5 = **some** [0, 5]) = true

(5 ≠ **some** [0, 5]) = true (since 0 ≠ 5)

(= **some**) ≡ **in**
However, (≠ **some**) ≢ **not in**

©LXD

NESTED SUBQUERIES - SET COMPARISON集合比较

- > all
- < all
- <=all
- >=all
- =all (not same to in)
- <> all (same to not in)

- 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)

(≠ **all**) ≡ **not in**
However, (= **all**) ≢ **in**

©LXD

NESTED SUBQUERIES - SET COMPARISON集合比较

?What meaning？

select *dept_name*
from *instructor*
group by *dept_name*
having avg (*salary*) >= **all** (**select avg** (*salary*)
from *instructor*
group by *dept_name*);

©LXD

TEST FOR EMPTY RELATIONS

- exists , not exists
- Yet another way of specifying the query "Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester"

select *course_id*
from *section* **as** *S*
where *semester* = 'Fall' **and** *year* = 2009 **and**
exists (**select** *
from *section* **as** *T*
where *semester* = 'Spring' **and** *year*= 2010 **and** *S.course_id* = *T.course_id*);

- **Correlation name** – variable S in the outer query
- **Correlated subquery** – the inner query

©LXD

TEST FOR EMPTY RELATIONS

- Find all students who have taken all courses offered in the Biology department.

select distinct *S.ID*, *S.name* **from** *student* **as** *S*
where not exists ((**select** *course_id* **from** *course*
where *dept_name* = 'Biology')
except
(**select** *T.course_id* **from** *takes* **as** *T*
where *S.ID* = *T.ID*));

- First nested query lists all courses offered in Biology
- Second nested query lists all courses a particular student took

- Note that $X - Y = \emptyset \Leftrightarrow X \subseteq Y$
- *Note:* Cannot write this query using = **all** and its variants

©LXD

TEST FOR ABSENCE OF DUPLICATE TUPLES

- The **unique** construct tests whether a subquery has any duplicate tuples in its result.
- The **unique** construct evaluates to "true" if a given subquery contains no duplicates .
- Find all courses that were offered at most once in 2009

```
select T.course_id
from course as T
where unique (select R.course_id
              from section as R
              where T.course_id= R.course_id
                    and R.year = 2009);
```

©LXD

SUBQUERIES IN THE FROM CLAUSE

- SQL allows a subquery expression to be used in the **from** clause
- Find the average instructors' salaries of those departments where the average salary is greater than $42,000."

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

- Note that we do not need to use the **having** clause

©LXD

SUBQUERIES IN THE FROM CLAUSE

- Find the average instructors' salaries of those departments where the average salary is greater than $42,000."
- 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;
```

©LXD

WITH CLAUSE

- The **with** clause provides a way of defining a temporary relation whose definition is available only to the query in which the **with** clause occurs
  - Introduced in SQL:1999
- Find all departments with the maximum budget

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

©LXD

COMPLEX QUERIES USING WITH CLAUSE

- Find all departments where the total salary is greater than the average of the total salary at all departments

```
with dept _total (dept_name, value) as
    (select dept_name, sum(salary)
     from instructor
     group by dept_name),
dept_total_avg(value) as
    (select avg(value)
     from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value > dept_total_avg.value;
```

©LXD

SCALAR SUBQUERIES标量子查询

- Scalar subqueries
  - SQL allows subqueries to occur wherever an expression returning a value is permitted, provided the subquery returns only one tuple containing a single attribute
  - List all departments along with the number of instructors in each department

```
select dept_name,
    (select count(*)
     from instructor where department.dept_name = instructor.dept_name)
    as num_instructors
from department;
```

- Runtime error if subquery returns more than one result tuple

©LXD

OBJECTIVES

- Overview of The SQL Query Language
- Data Definition
- Basic Query Structure
- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions
- Nested Subqueries
- Modification of the Database

©LXD

MODIFICATION OF THE DATABASE

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

©LXD

DELETION删除

- 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');

©LXD

DELETION

- 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)

©LXD

INSERTION插入

- 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*);

©LXD

INSERTION

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

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

- The **select from where** statement is evaluated fully before any of its results are inserted into the relation.

  Otherwise queries like

  insert into *table*1 **select** * **from** *table*1

  would cause problem

©LXD

UPDATES更新

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others by a 5%
  - Write two **update** statements:

    update *instructor*
    set *salary* = *salary* * 1.03
    where *salary* > 100000;
    update *instructor*
    set *salary* = *salary* * 1.05
    where *salary* <= 100000;

  - ***The order is important***
  - Can be done better using the **case** statement (next slide)

©LXD

UPDATE
- CASE STATEMENT FOR CONDITIONAL UPDATES

- Same query as before but with case statement

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

©LXD

UPDATE
- CASE STATEMENT FOR CONDITIONAL UPDATES

- The general form of the case statement is as follows

case
when $pred_1$ **then** $result_1$
when $pred_2$ **then** $result_2$
...
when $pred_n$ **then** $result_n$
else $result_0$
end

©LXD

UPDATES WITH SCALAR SUBQUERIES

- Recompute and update tot_creds value for all students

  update *student S*
  set *tot_cred* = (**select sum**(*credits*)
  from *takes, course*
  where *takes.course_id* = *course.course_id* **and**
  S.ID= *takes.ID* **and** *takes.grade* <> 'F' **and**
  takes.grade **is not null**);

- Sets *tot_creds* to null for students who have not taken any course

©LXD

UPDATES WITH SCALAR SUBQUERIES

- Recompute and update tot_creds value for all students
- Instead of **sum**(*credits*), use:

  update student S
  set tot_cred =
  case
  when sum*(credits)* ***is not null then sum****(credits)*
  else 0
  end

©LXD

SUMMARY

©LXD

Q&A?

THANKS!

leexudong@nankai.edu.cn