Chapter 3: Introduction to SQL

Database System Concepts, 7th Ed.

Outline

- Overview of The SQL Query Language
- SQL Data Definition
- Basic Query Structure of SQL Queries
- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions
- Nested Subqueries
- Modification of the Database
- 注意: 輸出結果請參考課本或上網執行

https://www.db-book.com/university-lab-dir/sqljs.html

History

- 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
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
  - Not all examples here may work on your particular system.

SQL Parts

- DML -- provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.
- integrity – the DDL includes commands for specifying integrity constraints.
- View definition -- The DDL includes commands for defining views.
- Transaction control –includes commands for specifying the beginning and ending of transactions.
- Embedded SQL and dynamic SQL -- define how SQL statements can be embedded within general-purpose programming languages.
- Authorization – includes commands for specifying access rights to relations and views.

Data Definition Language

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

- The schema for each relation.
- The type of values associated with each attribute.
- The Integrity constraints
- The set of indices to be maintained for each relation.
- Security and authorization information for each relation.
- The physical storage structure of each relation on disk.

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 to be stores exactly, but 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.
- More are covered in Chapter 4.

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$_1$),
  ...,
  (integrity-constraint$_k$))

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

- Example:

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

Integrity Constraints in Create Table

- Types of integrity constraints
  - **primary key** (A_1, ..., A_n)
  - **foreign key** (A_m, ..., A_n) **references** *r*
  - **not null**
- SQL prevents any update to the database that violates an integrity constraint.
- 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);
```

- 注意: primary key屬性也要求nonnull，大部分的軟體都有做到這一點。

And a Few More Relation Definitions

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

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

And more still

-
  ```
  create table takes (
          ID              varchar(5),
          course_id       varchar(8),
          sec_id          varchar(8),
          semester        varchar(6),
          year            numeric(4,0),
          grade           varchar(2),
          primary key (ID, course_id, sec_id, semester, year) ,
          foreign key (ID) references  student,
          foreign key (course_id, sec_id, semester, year) references section);
  ```

- 注意: 1. A primary key can consist of many attributes.

  2. A table can have many foreign keys.

Updates to tables

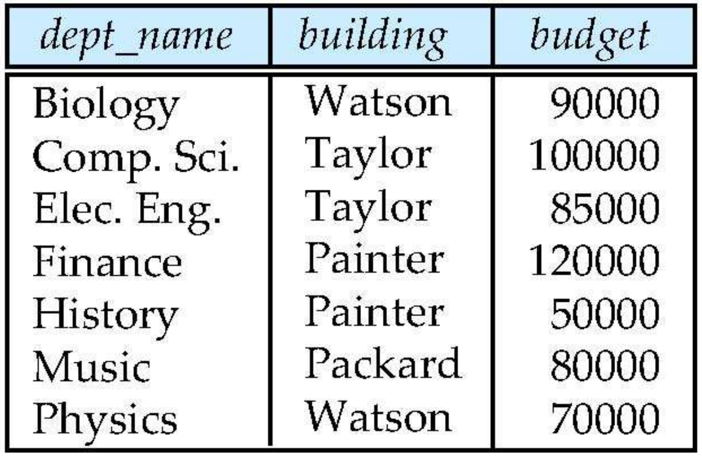
- **Insert**
  - **insert into** *instructor* **values** (‘10211’, ‘Smith’, ‘Biology’, 66000);->字串加單引號
  - **insert into** *instructor* **values** (‘10211’, *null*, ’Biology’, 66000); ->出現錯誤訊息
  - **insert into** *instructor* **values** (‘10211’, ’Smith’, ’Biology’, *null*); ->可執行
- **Delete**
  - Remove all tuples from the *student* relation
    - **delete from** *student*
- **Drop Table: 刪掉資料和全部定義**
  - **drop table** *r*
- **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 existing 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*

資料庫綱要 (課本圖2.8)

- 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)

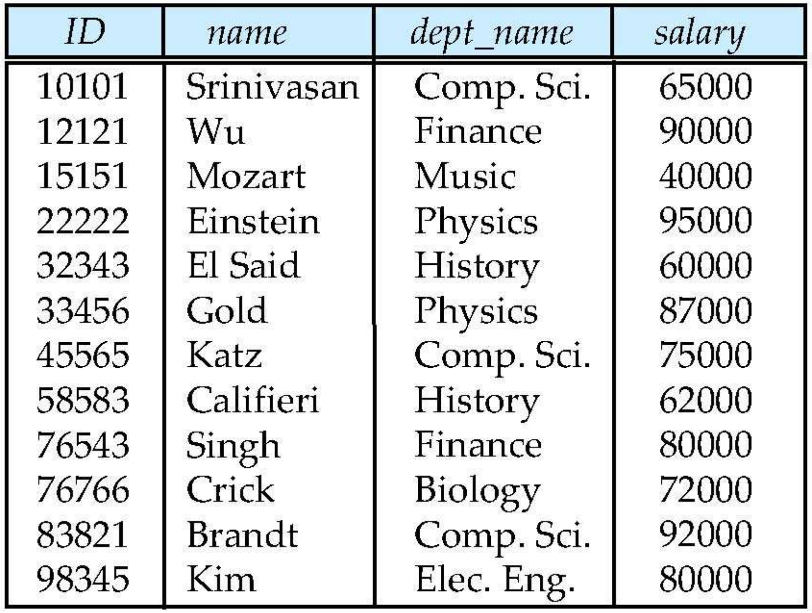
Relational Instances (課本圖2.5, 2.1, 2.7)

- department



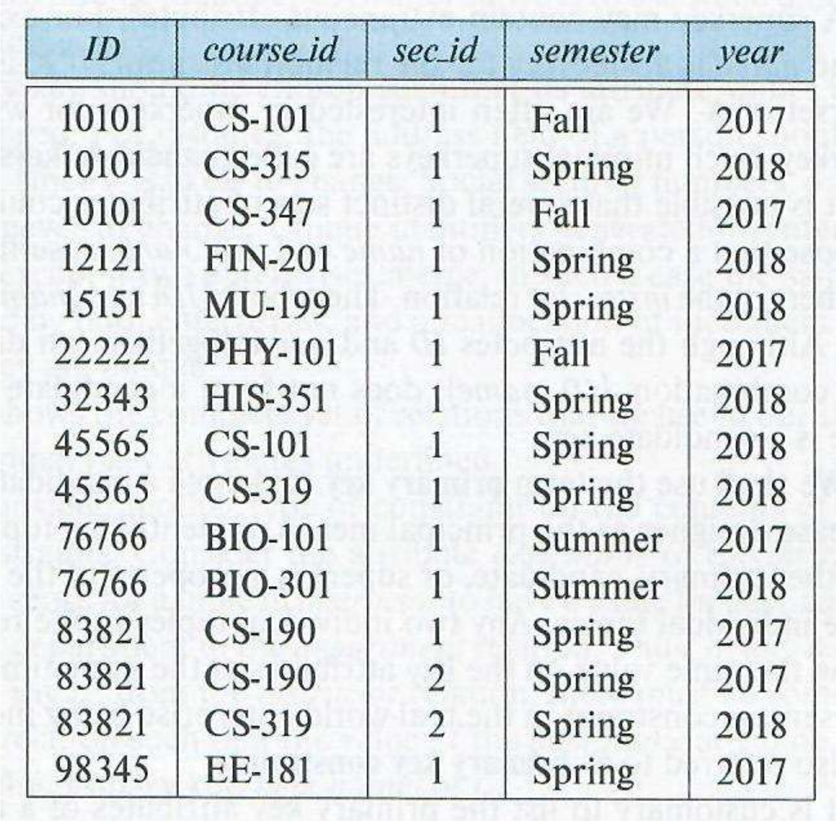
dept_name	*building*	*budget*
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

- instructor



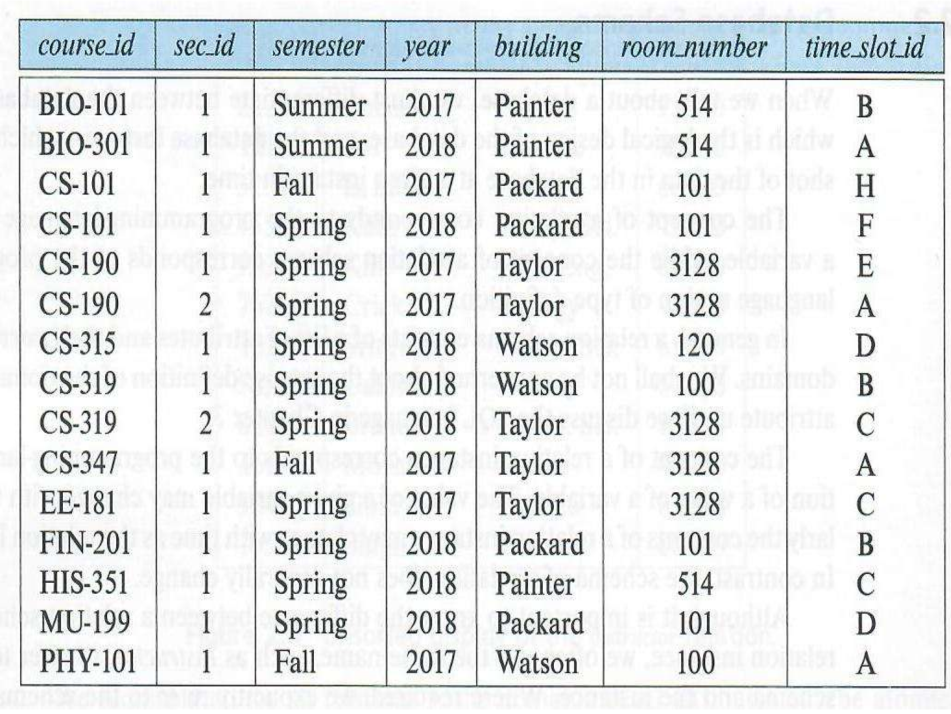
ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000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

- teaches



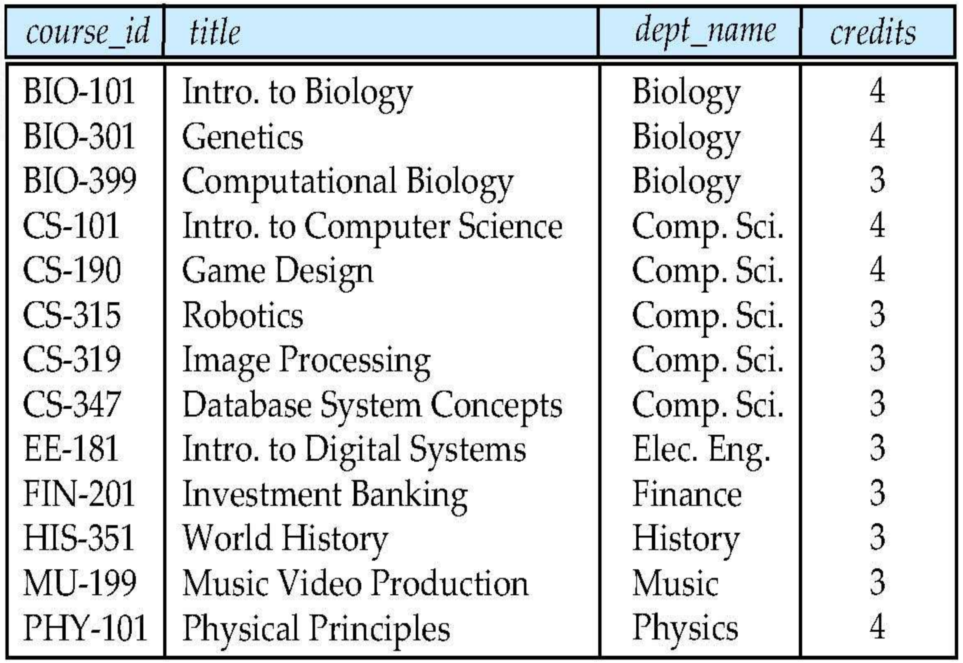
ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2017
10101	CS-315	1	Spring	2018
10101	CS-347	1	Fall	2017
12121	FIN-201	1	Spring	2018
15151	MU-199	1	Spring	2018
22222	PHY-101	1	Fall	2017
32343	HIS-351	1	Spring	2018
45565	CS-101	1	Spring	2018
45565	CS-319	1	Spring	2018
76766	BIO-101	1	Summer	2017
76766	BIO-301	1	Summer	2018
83821	CS-190	1	Spring	2017
83821	CS-190	2	Spring	2017
83821	CS-319	2	Spring	2018
98345	EE-181	1	Spring	2017

More Relational Instances (課本圖2.6, 2.2)



course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2017	Painter	514	B
BIO-301	1	Summer	2018	Painter	514	A
CS-101	1	Fall	2017	Packard	101	H
CS-101	1	Spring	2018	Packard	101	F
CS-190	1	Spring	2017	Taylor	3128	E
CS-190	2	Spring	2017	Taylor	3128	A
CS-315	1	Spring	2018	Watson	120	D
CS-319	1	Spring	2018	Watson	100	B
CS-319	2	Spring	2018	Taylor	3128	C
CS-347	1	Fall	2017	Taylor	3128	A
EE-181	1	Spring	2017	Taylor	3128	C
FIN-201	1	Spring	2018	Packard	101	B
HIS-351	1	Spring	2018	Painter	514	C
MU-199	1	Spring	2018	Packard	101	D
PHY-101	1	Fall	2017	Watson	100	A

- section



course_id	*title*	*dept_name*	*credits*
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

- course

Basic Query Structure

- A typical SQL query has the form:

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

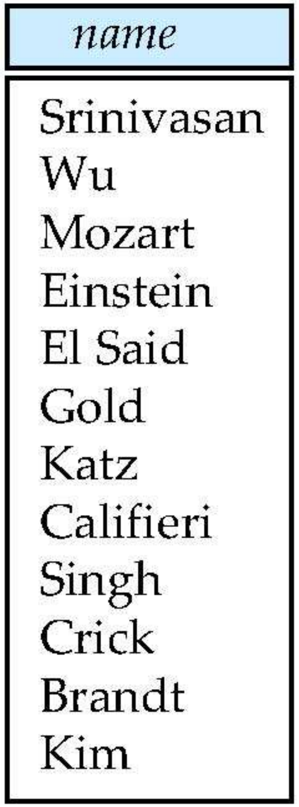
  - A_i represents an attribute
  - r_i represents a relation
  - P is a predicate.
- The result of an SQL query is a relation.
- 注意: An SQL query is usually ended with the semicolon “;” ，實作上各軟體不同，考試和作業加不加都OK

The select Clause

- 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 (i.e., you may use upper- or lower-case letters.)
  - E.g., *Name* ≡ *NAME* ≡ *name*
  - Some people use upper case wherever we use bold font.



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

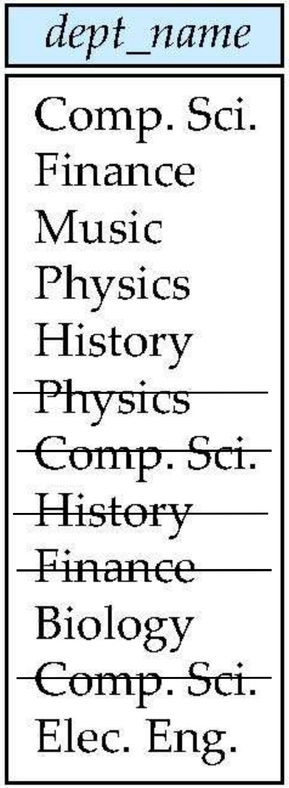
The select Clause (Cont.)

- 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. (因為all是default，所以習慣上都不寫)

  select all *dept_name*
  from *instructor*



dept_name
Comp. Sci.
Finance
Music
Physics
History
~~Physics~~
~~Comp. Sci.~~
~~History~~
~~Finance~~
Biology
~~Comp. Sci.~~
Elec. Eng.

The select Clause (Cont.)

- An asterisk in the select clause denotes “all attributes”

  select *
  from *instructor*

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

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

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

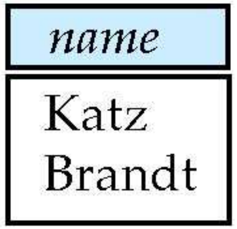
The where Clause

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

  select *name*
  from *instructor*
  where *dept_name* = ‘Comp. Sci.’

- SQL allows the use of the logical connectives **and, or,** and **not**
- The operands of the logical connectives can be expressions involving the comparison operators <, <=, >, >=, =, and <> (不等於).
- Comparisons can be applied to results of arithmetic expressions
- To find all instructors in Comp. Sci. dept with salary > 70000

  select *name*
  from *instructor*
  where *dept_name* = 'Comp. Sci.' **and** *salary* > 70000



name
Katz
Brandt

The from Clause

- 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*

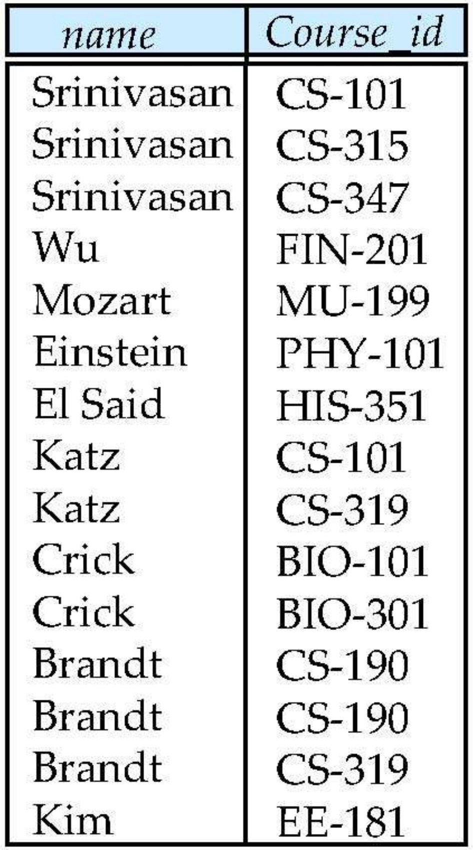
```
select *
from instructor, teaches
```

  - generates every possible instructor – teaches pair, with all attributes from both relations. (參考第二章投影片或課本圖3.6)
  - For common attributes (e.g., *ID*), the attributes in the resulting table are renamed using the relation name (e.g., *instructor.ID*)
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra).

Examples

- 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'



name	*Course_id*
Srinivasan	CS-101
Srinivasan	CS-315
Srinivasan	CS-347
Wu	FIN-201
Mozart	MU-199
Einstein	PHY-101
El Said	HIS-351
Katz	CS-101
Katz	CS-319
Crick	BIO-101
Crick	BIO-301
Brandt	CS-190
Brandt	CS-190
Brandt	CS-319
Kim	EE-181

練習

- Find the titles of courses in the Comp. Sci. department that have 3 credits.

Answer:

- Output the ID and building of the instructor named “Einstein”.

  (Einstein老師的編號和辦公室所在的建築物名稱)

Answer:

The Rename Operation

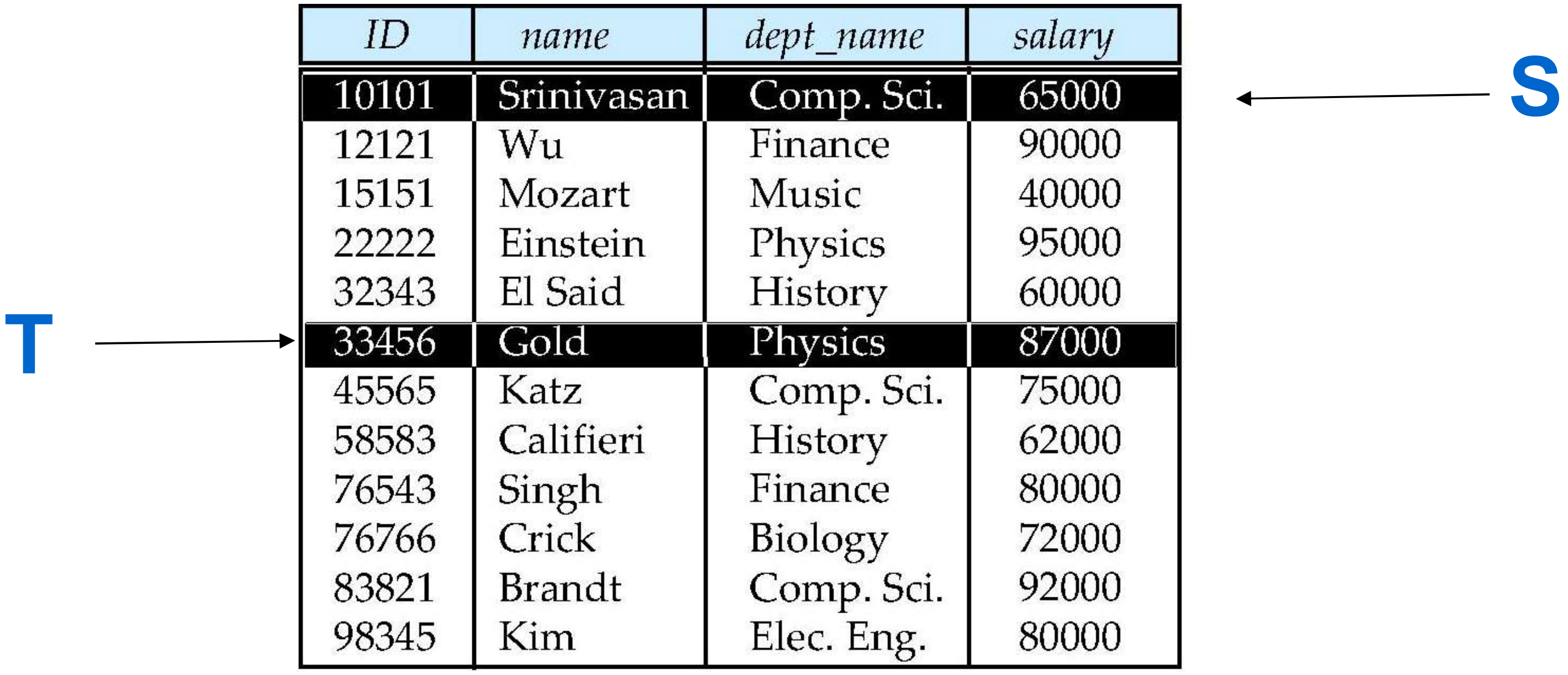
- The SQL allows renaming relations and attributes using the **as** clause:

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

  - **as** is optional, so “*instructor* **as** *T”* ≡ *“instructor T”*
- attribute的例子在講aggregate function時會提到
- relation的例子見下一頁

The Rename Operation (Cont.)

- 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* =>*特別稱呼為tuple variable*
    where *T.salary* > *S.salary* **and** *S.dept_name* = *'Comp. Sci.'*



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
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

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.
- Find the names of all departments whose building name includes the substring “Watson”.

```
select dept_name
from department
where building like ‘%Watson%'
```

- Match the string “100%”

```
like '100 \%'  escape  '\'
```

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

※ String Operations (Cont.)

- Patterns are case sensitive.
- Pattern matching examples:
  - 'Intro%' matches any string beginning with “Intro”.
  - '%Comp%' matches any string containing “Comp” as a substring.
  - '_ _ _' matches any string of exactly three characters.
  - '_ _ _ %' matches any string of at least three characters.
- SQL supports a variety of string operations such as
  - concatenation (using “||”)
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, etc.

Ordering the Display of Tuples

- List in alphabetic order the names of all instructors in the Physics department.

```
select name
from   instructor

where dept_name = `Physics'
order by name
```

name
Einstein
Gold

- 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**

name
Gold
Einstein

- Can sort on multiple attributes
  - Example:

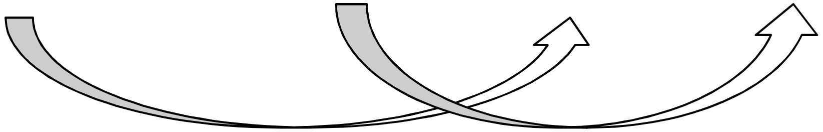
```
select *
from   instructor

order by  salary desc, name asc
```

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
  - 另一種寫法如下:

    select *name*
    from *instructor*
    where *salary* >= 90000 **and** *salary* <= 100000;
- Tuple comparison
  - **select** *name*, *course_id*
    from *instructor*, *teaches*
    where (*instructor*.*ID*, *dept_name*) = (*teaches*.*ID*, 'Biology');



Set Operations

- Find courses that ran in Fall 2017 or in Spring 2018

  (**select** *course_id* **from** *section* **where** *semester* = 'Fall' **and** *year* = 2017)
  union
  (**select** *course_id* **from** *section* **where** *semester* = 'Spring' **and** *year* = 2018)

- Find courses that ran in Fall 2017 and in Spring 2018

  (**select** *course_id* **from** *section* **where** *semester* = 'Fall' **and** *year* = 2017)
  intersect
  (**select** *course_id* **from** *section* **where** *semester* = 'Spring' **and** *year* = 2018)

- Find courses that ran in Fall 2017 but not in Spring 2018

  (**select** *course_id* **from** *section* **where** *semester* = 'Fall' **and** *year* = 2017)
  except
  (**select** *course_id* **from** *section* **where** *semester* = 'Spring' **and** *year* = 2018)

Set Operation Example (課本圖3.8-3.12)

course_id
CS-101
CS-347
PHY-101

course_id
CS-101
CS-315
CS-319
CS-319
FIN-201
HIS-351
MU-199

union

course_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

intersect

course_id
CS-101

except

course_id
CS-347
PHY-101

※ Set Operations (Cont.)

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

- 例子: A = {p, p, q}, B = {p, r}
  - A union all B = {p, p, p, q, r}; A union B = {p, q, r}
  - A intersect all B = {p}
  - A except all B = {p, q}; B except all A = {r}

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**
- The predicate **is not null** succeeds if the value on which it is applied is not null.

※ Null Values (Cont.)

- SQL treats as **unknown** the result of any comparison involving a null value (other than predicates **is null** and **is not null**).
  - Example*: 5* < **null** or **null** <> **null** or **null** = **null**
- The predicate in a **where** clause can involve Boolean operations (**and**, **or**, **not**); thus the definitions of the Boolean operations need to be extended to deal with the value **unknown**.
  - **and** : *(true* **and** *unknown) = unknown,*
    (false **and** *unknown) = false,*
    (unknown **and** *unknown) = unknown*
  - **or:** (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*
- *例子:*
  - *Select C from T1 where A = 'α' and B = 'β' => {7}*
  - *Select C from T1 where A = 'α' or B = 'β' => {8, 7}*

A	B	C
α		8
α	β	7

T1

Aggregate Functions

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

avg:	average value	平均
min:	minimum value	極小
max:	maximum value	極大
sum:	sum of values	總和
count:	number of values	個數

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

Aggregate Functions Examples

- Find the average salary of instructors in the Computer Science department
  - **select avg** (*salary*) **as** *avg_salary*
    from *instructor*
    where *dept_name*= 'Comp. Sci.';

avg_salary
77333

- Find the total number of instructors who teach a course in the Spring 2018 semester
  - **select count** (**distinct** *ID*)
    from *teaches*
    where *semester* = 'Spring' **and** *year* = 2018;

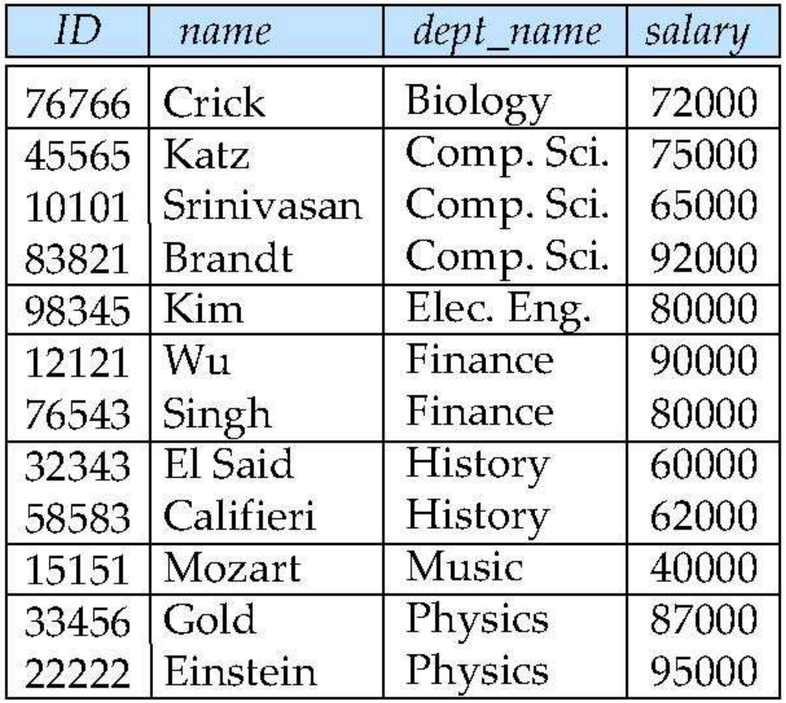
count (distinct *ID*)
6

- Find the number of tuples in the *course* relation
  - **select count** (*)
    from *course*;

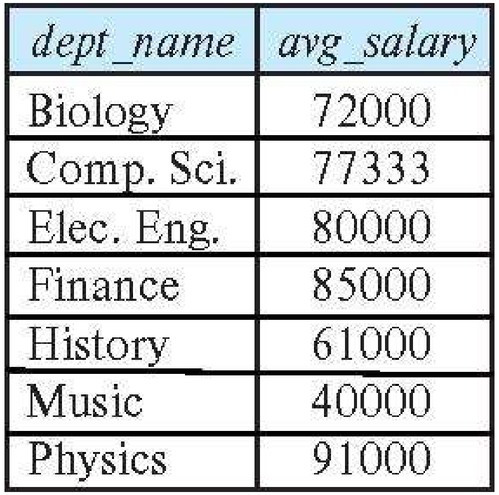
count (*)
13

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

Aggregation (Cont.)

- Attributes in **select** clause outside of aggregate functions must appear in **group by** list
  - 否則會出現multi-valued的狀況，
    違反atomic的限制
  - /* erroneous query */

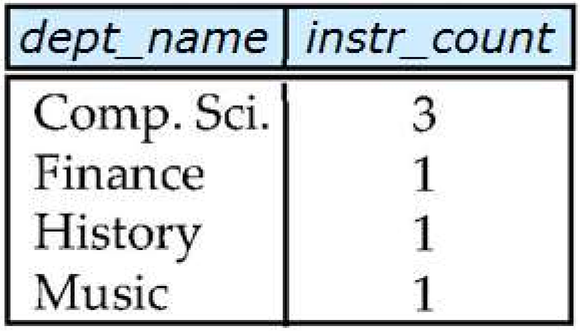
```
select dept_name, ID, avg (salary)
from instructor
group by dept_name;
```

dept_name	*ID*	*avg(salary)*
biology	*76766*	72000
Comp. Sci	*45565*	77333
	10101	
	83821	
…..	…..	……….

(使用到where和group by的例子)

- Find the number of instructors in each department who teach a course in the Spring 2018 semester.

```
select dept_name, count (distinct ID) as instr_count
from instructor, teaches
where instructor.ID = teaches.ID and
          semester= 'Spring' and year = 2018
group by dept_name;
```



dept_name	*instr_count*
Comp. Sci.	3
Finance	1
History	1
Music	1

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) as 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
  - aggregate function 不能直接使用在where clause裡.
  - 放在having和select clause裡的aggregate function 可不同

(使用到where, group by, having的例子)

- For each course section offered in 2017, find the average total credits (tot_cred) of all students enrolled in the section, if the section had at least 2 students.

```
select course_id, semester, year, sec_id, avg (tot_cred)
 from takes, student
where takes.ID = student.ID and year = 2017
group by course_id, sec_id, semester, year
 having count (student.ID) >= 2;
```

練習

- Find the number of instructors in each department.

- Find the names of all departments whose instructors are more than 10.

Nested Subqueries

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

  - **From clause:** r_i can be replaced by any valid subquery
  - **Where clause:** P can be replaced with an expression of the form:

    B <operation> (subquery)

    B is an attribute and <operation> to be defined later.
  - **Select clause:**

    A_i can be replaced by a subquery that generates a single value.

Set Membership

Set Membership

- Find courses offered in Fall 2017 and in Spring 2018

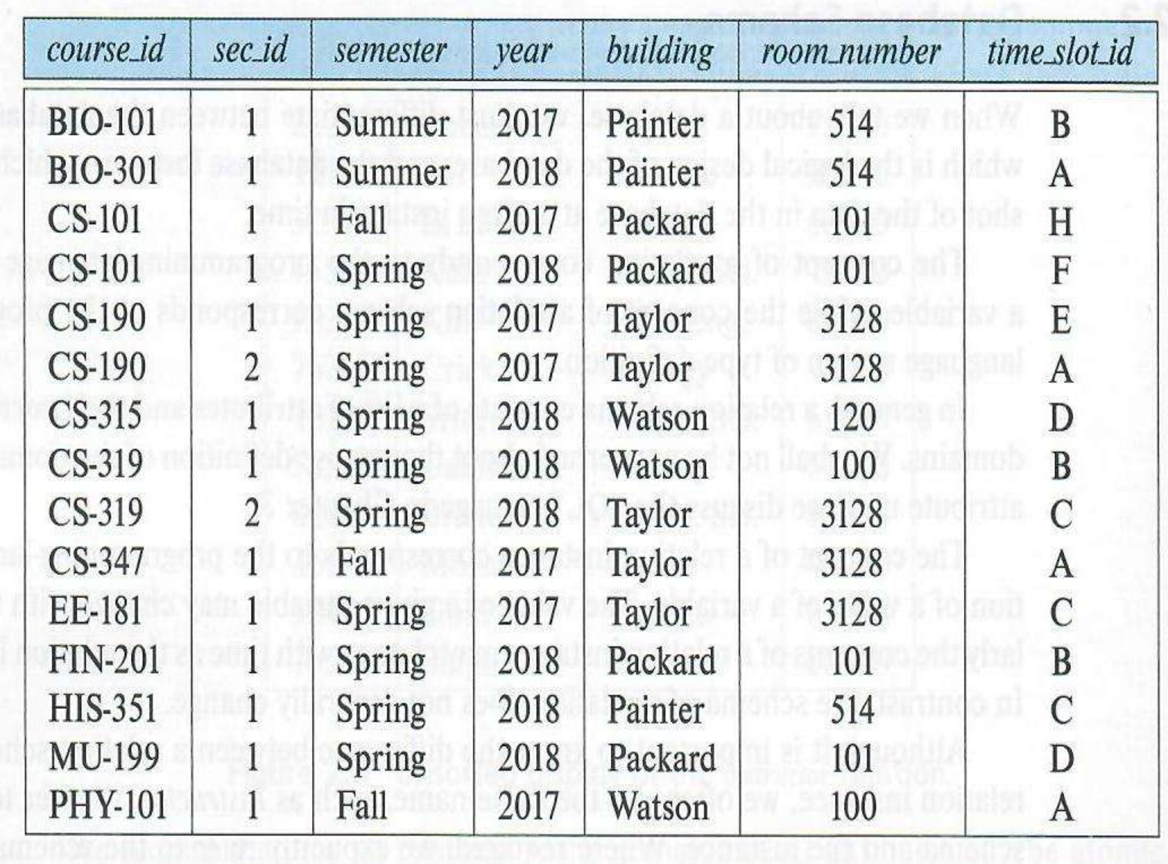
```
select distinct course_id
from section
where semester = 'Fall' and year= 2017 and
          course_id in (select course_id
                             from section
                             where semester = 'Spring' and year= 2018);
```

- Find courses offered in Fall 2017 but not in Spring 2018

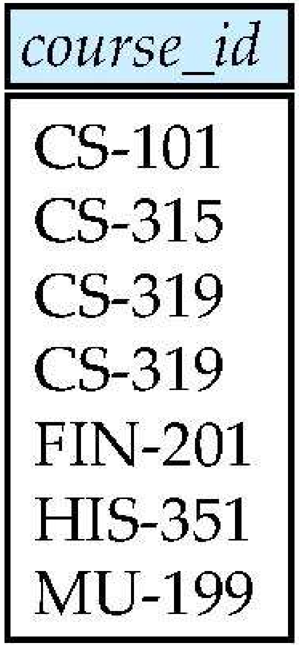
```
select distinct course_id
from section
where semester = 'Fall' and year= 2017 and
          course_id  not in (select course_id
                                   from section
                                   where semester = 'Spring' and year= 2018);
```

- 練習題：Find courses offered in Fall 2017 **or** in Spring 2018

Section 表格 和 Result of the subquery



course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2017	Painter	514	B
BIO-301	1	Summer	2018	Painter	514	A
CS-101	1	Fall	2017	Packard	101	H
CS-101	1	Spring	2018	Packard	101	F
CS-190	1	Spring	2017	Taylor	3128	E
CS-190	2	Spring	2017	Taylor	3128	A
CS-315	1	Spring	2018	Watson	120	D
CS-319	1	Spring	2018	Watson	100	B
CS-319	2	Spring	2018	Taylor	3128	C
CS-347	1	Fall	2017	Taylor	3128	A
EE-181	1	Spring	2017	Taylor	3128	C
FIN-201	1	Spring	2018	Packard	101	B
HIS-351	1	Spring	2018	Painter	514	C
MU-199	1	Spring	2018	Packard	101	D
PHY-101	1	Fall	2017	Watson	100	A



course_id
CS-101
CS-315
CS-319
CS-319
FIN-201
HIS-351
MU-199

Set Membership (Cont.)

- Select the names of instructors whose names are neither “Mozart” nor Einstein”

```
select distinct name
from instructor
where  name not in ('Mozart', 'Einstein')
```

- Find the total number of (distinct) students who have taken course sections taught by the instructor with *ID* 10101

```
select count (distinct ID)
from takes
where (course_id, sec_id, semester, year) in
                    (select course_id, sec_id, semester, year
                     from teaches
                     where teaches.ID= 10101);
```

- Note: Above query can be written in a much simpler manner, 而且有些軟體不支援此種寫法。 The formulation above is simply to illustrate SQL features

Set Comparison

Set Comparison – “some” Clause

- 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');
```

- 確定sub-query只輸出一筆資料則可省略量詞some或all.

Definition of “some” Clause

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

$(5 < \textbf{some} \begin{array}{|c|}\hline 0\\\hline 5\\\hline 6\\\hline\end{array}) = \text{true}$ (read: 5 < some tuple in the relation)

$(5 < \textbf{some} \begin{array}{|c|}\hline 0\\\hline 5\\\hline\end{array}) = \text{false}$

$(5 = \textbf{some} \begin{array}{|c|}\hline 0\\\hline 5\\\hline\end{array}) = \text{true}$

$(5 \neq \textbf{some} \begin{array}{|c|}\hline 0\\\hline 5\\\hline\end{array}) = \text{true (since } 0 \neq 5)$

(= **some**) $\equiv$ **in**
However, ($\neq$ **some**) $\not\equiv$ **not in**

Set Comparison – “all” Clause

- Find the names of all instructors whose salary is greater than the salary of all instructors in the Biology department.

```
select name
from instructor
where salary > all (select salary
                    from instructor
                    where dept_name = 'Biology');
```

- 練習題：Find the names of all instructors who earn the most salary in the Biology department.

Answer:

Definition of "all" Clause

- F <comp> **all** *r* ⇔ ∀ *t* ∈ *r* (F <comp> *t)*

$$(5 < \textbf{all} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{false}$$

$$(5 < \textbf{all} \begin{array}{|c|}\hline 6 \\ \hline 10 \\ \hline \end{array}) = \text{true}$$

$$(5 = \textbf{all} \begin{array}{|c|}\hline 4 \\ \hline 5 \\ \hline \end{array}) = \text{false}$$

$$(5 \neq \textbf{all} \begin{array}{|c|}\hline 4 \\ \hline 6 \\ \hline \end{array}) = \text{true (since } 5 \neq 4 \text{ and } 5 \neq 6)$$

($\neq$ **all**) ≡ **not in**

However, (= **all**) $\not\equiv$ **in**

Test for Empty Relations

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

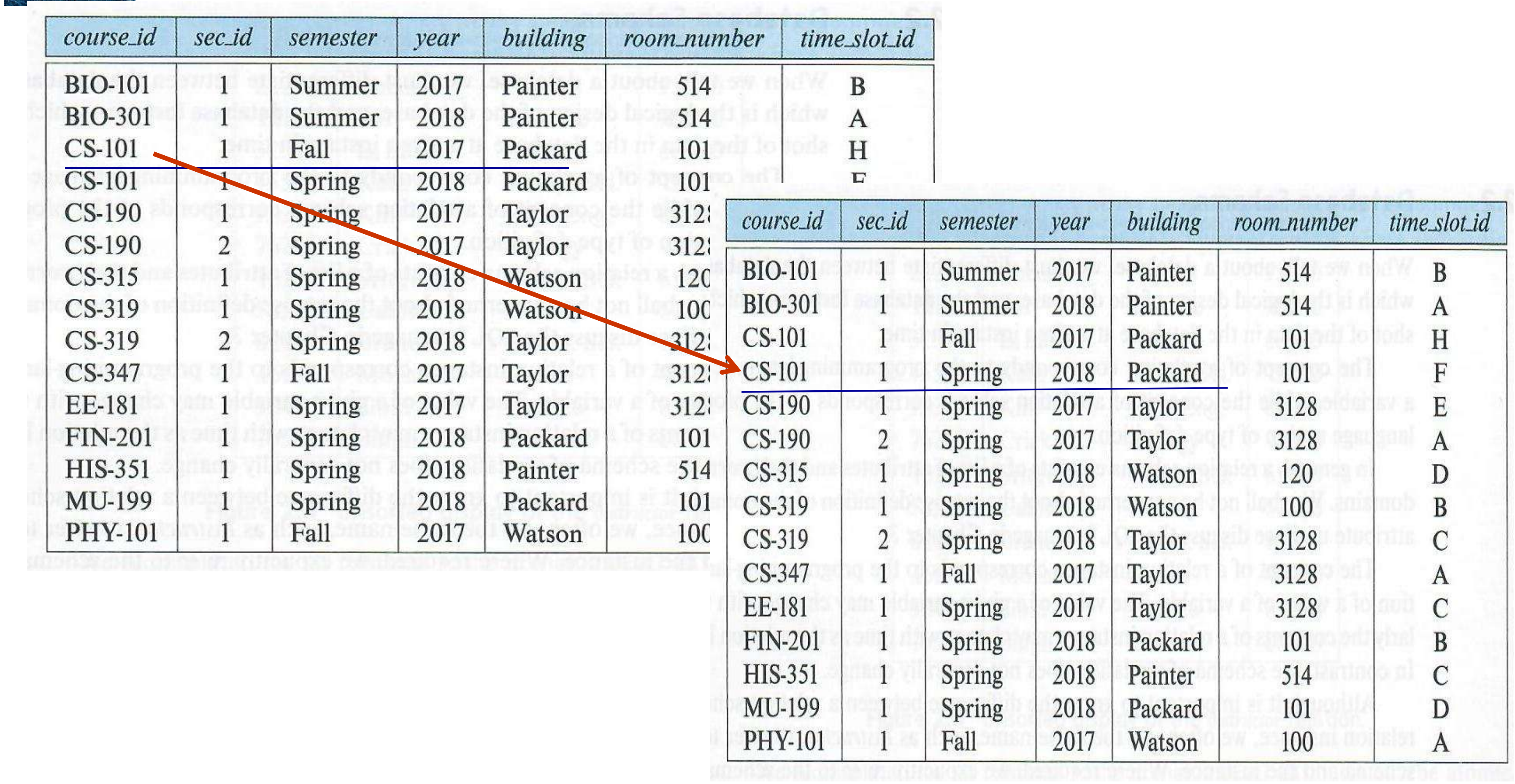
Use of “exists” Clause

- Another way of specifying the query “Find all courses taught in both the Fall 2017 semester and in the Spring 2018 semester”

```
select course_id
from section as S
where semester = 'Fall' and year = 2017 and
          exists  (select *
                   from section as T
                   where semester = 'Spring' and year= 2018
                               and S.course_id = T.course_id);
```

前面不要有屬性

- **Correlation name** – variable S in the outer query
- **Correlated subquery** – the inner query
- 注意：有的subquery需要額外宣告變數，有的不需要



course_id	sec_id	semester	year	building	room_number	time_slot_id
BIO-101	1	Summer	2017	Painter	514	B
BIO-301	1	Summer	2018	Painter	514	A
CS-101	1	Fall	2017	Packard	101	H
CS-101	1	Spring	2018	Packard	101	
CS-190	1	Spring	2017	Taylor	312	
CS-190	2	Spring	2017	Taylor	312	
CS-315	1	Spring	2018	Watson	120	
CS-319	1	Spring	2018	Watson	100	
CS-319	2	Spring	2018	Taylor	312	
CS-347	1	Fall	2017	Taylor	312	
EE-181	1	Spring	2017	Taylor	312	
FIN-201	1	Spring	2018	Packard	101	
HIS-351	1	Spring	2018	Painter	514	
MU-199	1	Spring	2018	Packard	101	
PHY-101	1	Fall	2017	Watson	100	

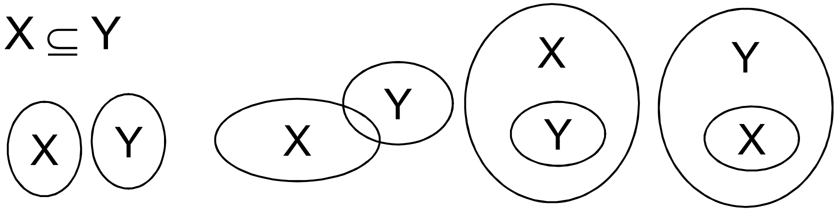
course_id	sec_id	semester	year	building	room_number	time_slot_id
BIO-101	1	Summer	2017	Painter	514	B
BIO-301	1	Summer	2018	Painter	514	A
CS-101	1	Fall	2017	Packard	101	H
CS-101	1	Spring	2018	Packard	101	F
CS-190	1	Spring	2017	Taylor	3128	E
CS-190	2	Spring	2017	Taylor	3128	A
CS-315	1	Spring	2018	Watson	120	D
CS-319	1	Spring	2018	Watson	100	B
CS-319	2	Spring	2018	Taylor	3128	C
CS-347	1	Fall	2017	Taylor	3128	A
EE-181	1	Spring	2017	Taylor	3128	C
FIN-201	1	Spring	2018	Packard	101	B
HIS-351	1	Spring	2018	Painter	514	C
MU-199	1	Spring	2018	Packard	101	D
PHY-101	1	Fall	2017	Watson	100	A

※ Use of “not exists” Clause

- 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')        X
                  except
                    (select T.course_id
                     from takes as T                    Y
                     where S.ID = T.ID));
```

  - First nested query lists all courses offered in Biology => X
  - Second nested query lists all courses a particular student took => Y
- Note that $X – Y = Ø \iff X \subseteq Y$



- Note: Cannot write this query using = all and its variants

※ 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 2017

```
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 = 2017);
```

Subqueries in the From Clause

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;
```

更改屬性的名稱

- Note that we do not need to use the **having** clause
- Another way to write above query

更改表格的名稱

```
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; <-只能用新定義的relation裡的屬性
```

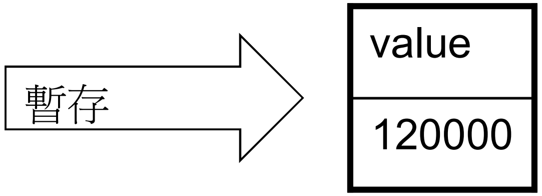
- 注意： (1) 寫在from的subquery不能用到同層from的其他relation.

  (2) 有的軟體要求一定要把derived relation改名

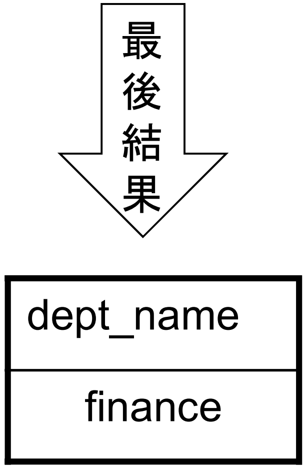
With Clause

- 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.
- 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;
```



value
120000



dept_name
finance

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;
```

- 注意:
  - 有的軟體沒支援with語法
  - 定義的順序，後面的可用到前面的定義，整個query的輸出為最後select-from-where所定義的。

Scalar Subquery

- Scalar subquery is one which is used where a single value is expected
- 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

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

Deletion

- Delete all instructors

```
delete from instructor
```

- Delete all instructors from the Finance department

```
delete from instructor
where dept_name= 'Finance’;
```

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

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

Deletion (Cont.)

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

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

  - Problem: as we delete tuples from *instructor*, 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)

Insertion

- 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*);

Insertion (Cont.)

- Make each student in the Music department who has earned more than 144 credit hours an instructor in the Music department with a salary of $18,000.

  insert into *instructor*
  select *ID, name, dept_name, 18000*
  from *student*
  where *dept_name* = 'Music' **and** *total_cred* > 144;

- 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

Updates

- Give a 5% salary raise to all instructors

新的欄位值

欲修改的欄位名稱

```
update instructor
    set salary = salary * 1.05
```

注意: 若同時修改多個欄位，每組中間用逗號隔開

- Give a 5% salary raise to those instructors who earn less than 70000

```
update instructor
    set salary = salary * 1.05
   where salary < 70000;
```

- Give a 5% salary raise to instructors whose salary is less than average

```
update instructor
set salary = salary * 1.05
where salary <  (select avg (salary)
                  from instructor);
```

Updates (Cont.)

- 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

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

注意: 若有case沒被涵蓋到，其值會被設為0。

Updates with Scalar Subqueries

- Recompute and update tot_creds value for all students

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

- The above statement sets *tot_creds* to null for students who have not taken any course. The following statement will set *tot_creds* to “0”.
  - Instead of **sum**(*credits*), use:

```
select case
    when sum(credits) is not null then sum(credits)
    else 0
 end
```