COMP9311:
DATABASE SYSTEMS

Term 1 2024

Week 3 – SQL

By Xiaoyang Wang, CSE UNSW

Disclaimer: the course materials are sourced from previous offerings of COMP9311 and COMP3311

Summary Week 2

Relational Model

- Relations, Tuples, Attributes
- Integrity Constraints
- ER to Relational Mapping

Relational Algebra

- Select, Project, Union, Intersection, Difference, Cartesian Product, Join, Divide, Rename
- Basic Operators vs Extended Operators
- Aggregation

SQL-99

- SQL = **Structured Query Language** (pronounced “sequel”).
- Developed at IBM (San Jose Lab) during the 1970’s, and standardised during the 1980’s.
- A **standard** language for querying and manipulating relational DBMSs.
- Interactive via GUI or command line or embedded in programs.
- **Declarative**, based on relational algebra.

SQL in Relational DBMS

In relational databases, what does SQL do?

A data definition language (DDL)

- CREATE TABLE, DROP TABLE, …

A data manipulation language (DML)

- SELECT: keywords relating to select, e.g., GROUP BY, HAVING, ORDER BY…
- INSERT, DELETE, UPDATE, ALTER, …

Other Commands

- indexes, constraints, views, triggers, transactions, authorization, …

Sample Database

To illustrate the features of SQL, we use a small example database below:

Beers(*name*, manf)

Drinkers(*name*, addr, phone)

Sells(*bar, beer*, price)

Bars(*name*, addr, license)

Likes(*drinker, beer*)

Frequents(*drinker, bar*)

keys are in ***italic*** font and highlighted by **underscore**.

Sample Database (cont)

Bars:

Name	Addr	License
Australia Hotel	The Rocks	123456
Coogee Bay Hotel	Coogee	966500
Lord Nelson	The Rocks	123888
Marble Bar	Sydney	122123
Regent Hotel	Kingsford	987654
Royal Hotel	Randwick	938500

Drinkers:

Name	Addr	Phone
Adam	Randwick	9385-4444
Gernot	Newtown	9415-3378
John	Clovelly	9665-1234
Justin	Mosman	9845-4321

Sample Database (cont)

Beers:

Name	Manf
80/-	Caledonian
Bigfoot Barley Wine	Sierra Nevada
Burragorang Bock	George IV Inn
Crown Lager	Carlton
Fosters Lager	Carlton
Invalid Stout	Carlton
Melbourne Bitter	Carlton
New	Toohey’s
Old	Toohey’s
Old Admiral	Lord Nelson
Pale Ale	Sierra Nevada
Premium Lager	Cascade
Red	Toohey’s
Sheaf Stout	Toohey’s
Sparkling Ale	Cooper’s
Stout	Cooper’s
Three Sheets	Lord Nelson
Victoria Bitter	Carlton

Sample Database (cont)

Frequents:

Drinker	Bar
Adam	Coogee Bay Hotel
Gernot	Lord Nelson
John	Coogee Bay Hotel
John	Lord Nelson
John	Australia Hotel
Justin	Regent Hotel
Justin	Marble Bar

Likes:

Drinker	Beer
Adam	Crown Lager
Adam	Fosters Lager
Adam	New
Gernot	Premium Lager
Gernot	Sparkling Ale
John	80/-
John	Bigfoot Barley Wine
John	Pale Ale
John	Three Sheets
Justin	Sparkling Ale
Justin	Victoria Bitter

Sample Database (cont)

Sells:

Bar	Beer	Price
Australia Hotel	Burragorang Bock	3.5
Coogee Bay Hotel	New	2.25
Coogee Bay Hotel	Old	2.5
Coogee Bay Hotel	Sparkling Ale	2.8
Coogee Bay Hotel	Victoria Bitter	2.3
Lord Nelson	Three Sheets	3.75
Lord Nelson	Old Admiral	3.75
Marble Bar	New	2.8
Marble Bar	Old	2.8
Marble Bar	Victoria Bitter	2.8
Regent Hotel	New	2.2
Regent Hotel	Victoria Bitter	2.2
Royal Hotel	New	2.3
Royal Hotel	Old	2.3
Royal Hotel	Victoria Bitter	2.3

Example:

Beers:

Name	Manf
80/-	Caledonian
Bigfoot Barley Wine	Sierra Nevada
Burragorang Bock	George IV Inn
Crown Lager	Carlton
Fosters Lager	Carlton
Invalid Stout	Carlton
Melbourne Bitter	Carlton
New	Toohey’s
Old	Toohey’s
Old Admiral	Lord Nelson
Pale Ale	Sierra Nevada
Premium Lager	Cascade
Red	Toohey’s
Sheaf Stout	Toohey’s
Sparkling Ale	Cooper’s
Stout	Cooper’s
Three Sheets	Lord Nelson
Victoria Bitter	Carlton

SQL Queries: What beers are made by Toohey’s?”

Example:

Beers:

Name	Manf
80/-	Caledonian
Bigfoot Barley Wine	Sierra Nevada
Burragorang Bock	George IV Inn
Crown Lager	Carlton
Fosters Lager	Carlton
Invalid Stout	Carlton
Melbourne Bitter	Carlton
New	Toohey’s
Old	Toohey’s
Old Admiral	Lord Nelson
Pale Ale	Sierra Nevada
Premium Lager	Cascade
Red	Toohey’s
Sheaf Stout	Toohey’s
Sparkling Ale	Cooper’s
Stout	Cooper’s
Three Sheets	Lord Nelson
Victoria Bitter	Carlton

SQL Queries: What beers are made by Toohey’s?”

SELECT Name FROM Beers WHERE Manf = ‘Toohey’’s’;

SQL Queries

To answer the question “What beers are made by Toohey’s?”, we could ask:

```
SELECT Name FROM Beers WHERE Manf = ‘Toohey’’s’;
```

This gives a subset of the Beers relation, displayed as:

```
Name
--------------------
New
Old
Red
Sheaf Stout
```

Quotes are escaped by doubling them (‘ ‘)

Basic SELECT Structure

To retrieve information from a database, there is a basic query structure, known as the **select** statement

```
SELECT <Attribute list>
FROM <Table list>
WHERE <Condition>
```

- <attribute list>: list of attributes
- <table list>: list of relations
- <condition>: list of conditions (Boolean expression)

SELECT statement is also known as a **select-from-where block**. The result of this statement is a table, which is typically displayed on output.

The SELECT statement contains the functionality of **select, project and join** from the relational algebra.

SQL Identifiers

Names are used to identify objects such as tables, attributes, views, …

Identifiers in SQL use similar conventions to common programming languages:

- a sequence of **alpha-numerics**, starting with an alphabetic
- **not** case-sensitive
- reserve word **disallowed**, …
- Whether you can use sign (@), dollar sign ($), number sign (#), or underscore (_) depends on the standard

SQL Keywords

Some of the frequently-used ones

- ALTER AND CREATE
- FROM INSERT NOT OR
- SELECT TABLE WHERE

For PostgreSQL Keywords see https://www.postgresql.org/docs/current/sql-keywords-appendix.html .

SQL Data Types

All attributes in SQL relations have **domain** specified.

SQL supports a small set of useful built-in data types: strings, numbers, dates, bit-strings.

Self defined data type is allowed in PostgreSQL.

Various type conversions are available

- date to string, string to date, integer to real ...
- applied automatically “where they make sense”

SQL Data Types (cont.)

- Basic domain (type) checking is performed automatically
- Constraints can be used to “enforce” more complex domain membership conditions
- The NULL value is a member of all data types

SQL Data Types (cont.)

Comparison operators are defined on all types

< **>** **<=** **>=** **=** **!=**

Boolean operators **AND, OR, NOT** are available within WHERE expressions to combine results of comparisons

Comparison against NULL yields FALSE

Can explicitly test for NULL using:

***attr* IS NULL** ***attr* IS NOT NULL**

Most data types also have type-specific operations available (e.g., arithmetic for numbers)

Which operations are actually applied depends on the implementation

Data Types - Numeric

Some options for specifying the attributes for holding numeric values:

If you need integers

- **smallint** (2 byte integer)
- **int** (4 byte integer)
- **bigint** (8 byte integer)

If you need real numbers

- **real** (4 byte floating point)
- **double** (8 byte floating point)
- **numeric (<precision> , <scale>)**

  <precision>: specify significant digits in the whole number

  <scale>: specify digits after the decimal point

Data Types – String Literal

Example of a string literal: ‘John’

A string literal is **a sequence of zero or more characters**

In SQL, you specify a literal by enclosing it in single quotes.

Two kinds of string literals are available:

- **CHAR(n)** n length, left-justified blank-padded
- **VARCHAR(n)** can be between 0 and n length, no padding

String literals are case sensitive: ‘John’ != ‘JOHN ’

String Operators

- string || string … concatenate two strings
- LENGTH (string) ... return length of string
- SUBSTR (string, start, length) ... extract chars from within string

Example:

- ‘Post’|| ‘greSQL’ -> PostgreSQL
- substring('Thomas' from 2 for 3) -> hom

SQL Like Operator

str LIKE pattern ... matches string to pattern

Two kinds of string **pattern-matching**

- The symbol _ (underscore) matches any single characters
- The symbol % (percent) matches zero or more characters

Practice

String LIKE ‘Ja%’	Strings beginning with ‘Ja’
String LIKE ‘_i%’	Strings with ‘i’ as 2nd letter
String LIKE ‘%o%o%’	Strings contains two ‘o’s

SQL Dates

Dates are simply specially-formatted strings, with a range of operations to implement date semantics.

Format is typically **DD-Mon-YYYY**, e.g., ‘18-Aug-1998’

Accepts other formats

Comparison operators implement before (<) and after (>).

(start1, end1) OVERLAPS (start2, end2)

- This expression yields true when two time periods (defined by their endpoints) overlap, false when they do not overlap.
- SELECT (DATE '2001-02-16', DATE '2001-12-21') OVERLAPS (DATE '2001-10-30', DATE '2002-10-30'); -> Result: true

Converting Data Types

Conversions between data types are an important skill to know.

E.g., division of one integer with an integer

Various type conversions are available:

- integer to real ...
- string to integer …

SQL supports a small set of useful built-in data types, e.g., numbers, strings, dates...

- You can define your own type in SQL

Tuple and Set Literals

Tuple and set constants are both written as:

(val1, val2, val3, ...)

The correct interpretation is worked out from the context.

Examples:

```
Student(stude#, name, course)
( 2177364, ‘Jack Smith’, ‘BSc’)   -- tuple literal

SELECT name
FROM Employees
WHERE job IN (‘Lecturer’, ‘Tutor’, ‘Professor’);     -- set literal
```

Querying a Single Relation

Formal semantics (relational algebra):

- start with relation R in FROM clause
- apply σ using Condition in WHERE clause
- apply π using Attributes in SELECT clause

```
SELECT Attributes
FROM R
WHERE Conditions
```

Querying a Single Relation

Operationally, we think in terms of a tuple variable ranging over all tuples of the relation.

Operational semantics of SQL SELECT (single relation)

```
FOR EACH tuple T in R DO
        check whether T satisfies the condition in the WHERE clause
        IF it does THEN
                print the attributes of T that are
                specified in the SELECT clause
        END
END
```

Projection by 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: Give all the names of all drinkers

```
SELECT Name
FROM Drinkers;
```

Drinkers:

Name	Addr	Phone
Adam	Randwick	9385-4444
Gernot	Newtown	9415-3378
John	Clovelly	9665-1234
Justin	Mosman	9845-4321

Note: FROM is always necessary with SELECT, whereas WHERE is optional.

Projection by 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: Give all the names of all drinkers

```
SELECT Name
FROM Drinkers;
```

Drinkers:

Name	Addr	Phone
Adam	Randwick	9385-4444
Gernot	Newtown	9415-3378
John	Clovelly	9665-1234
Justin	Mosman	9845-4321

Note: FROM is always necessary with SELECT, whereas WHERE is optional.

Projection by SELECT Clause

Example: Give me both names and addresses of drinkers

```
SELECT Name, Addr
FROM Drinkers;
```

Drinkers:

Name	Addr	Phone
Adam	Randwick	9385-4444
Gernot	Newtown	9415-3378
John	Clovelly	9665-1234
Justin	Mosman	9845-4321

An asterisk in the select clause denotes “all attributes”

```
SELECT *
FROM Drinkers;
```

Drinkers:

Name	Addr	Phone
Adam	Randwick	9385-4444
Gernot	Newtown	9415-3378
John	Clovelly	9665-1234
Justin	Mosman	9845-4321

DISTINCT

SQL allows duplicates in relations and in query results.

○ allows a table to have two or more tuples that are identical in all their attribute values.

In general, an SQL table can be a simple set of tuples, or a multiset of tuples.

Set: {a, b, c}

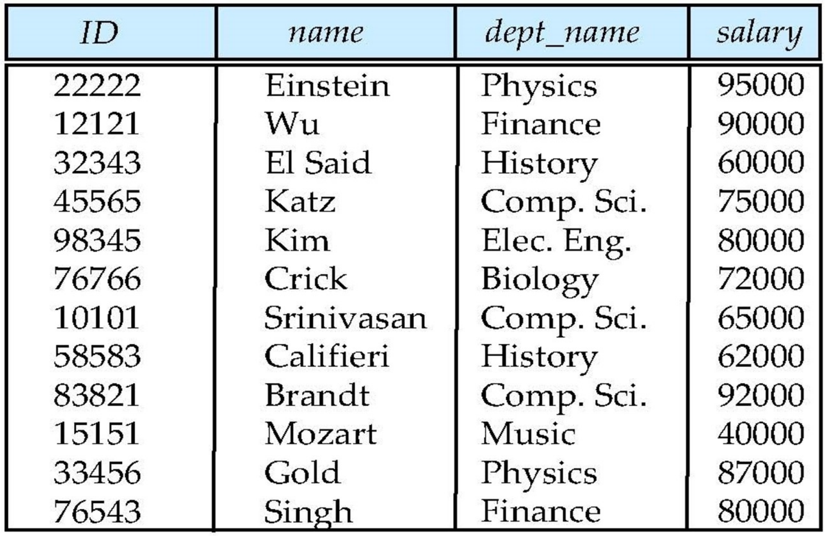
Multiset: {a, a, b, b, c, a, a, b, c, c …}

DISTINCT

To eliminate duplicates in the query results, insert the keyword **distinct** after select.

Example: Find the names of all departments and remove duplicates.

SELECT DISTINCT dept_name from instructor



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

(a) The *instructor* table

Question: When duplicates are useful?

Selection by Where Clause

Find the beers manufactured by Toohey’s

```
SELECT Name
FROM Beers
WHERE Manf = ‘Toohey’’s’;
```

Beers:

Name	Manf
80/-	Caledonian
Premium Lager	Cascade
Red	Toohey’s
Sheaf Stout	Toohey’s
Sparkling Ale	Cooper’s
Victoria Bitter	Carlton

The “typical” SELECT query:

```
    SELECT a1, a2, a3
    FROM Rel
    WHERE Cond
```

This corresponds to select followed by project

$$\pi_{\{a1,a2,a3\}}(\sigma_{Cond}(Rel))$$

Example

Find the price that Regent Hotel charges for New

```
SELECT price
FROM Sells
WHERE bar = ‘Regent Hotel’ AND beer = ‘New’;
```

```
PRICE
----------
2.2
```

The condition can be an arbitrarily complex boolean-valued expression using the operators mentioned previously.

Bar	Beer	Price
Australia Hotel	Burragorang Bock	3.5
Coogee Bay Hotel	New	2.25
Coogee Bay Hotel	Old	2.5
Coogee Bay Hotel	Sparkling Ale	2.8
Coogee Bay Hotel	Victoria Bitter	2.3
Lord Nelson	Three Sheets	3.75
Lord Nelson	Old Admiral	3.75
Marble Bar	New	2.8
Marble Bar	Old	2.8
Marble Bar	Victoria Bitter	2.8
Regent Hotel	New	2.2
Regent Hotel	Victoria Bitter	2.2
Royal Hotel	New	2.3
Royal Hotel	Old	2.3
Royal Hotel	Victoria Bitter	2.3

Null in SQL

What happens when the condition makes a comparison with a null value?

Comparisons with null returns unknown

- Example: 5 < null, null <> null, 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 as unknown

Result of where clause predicate is treated as **false** if it evaluates as unknown

Example: Null Values

```
SELECT A3
FROM R
WHERE A1 + 5 > A2 and A4 = ‘x’
```

A1	A2	A3	A4
5	9	alpha	x
	4	beta	
2	4	gamma	
3		delta	x

When it evaluates the second tuple:

- Null + 5 -> Null (for A1 + 5)
- Null > 4 -> unknown (for A1 + 5 > A2)
- Null = ‘x’ -> unknown (for A4 = ‘x’)
- unknown and unknown -> unknown (for A1 + 5 > A2 and A4 = ‘x’)

What about the following?

```
select  A3
from  R
where (A1 + 5 > A2 and A4 = ‘x’) is unknown
```

Renaming via AS

In relational algebra we have the renaming operator ρ to avoid name clashes.

Example: $\rho_{\text{Beers(Brand, Brewer)}}(\text{Beers})$

Gives a new relation, with same data as Beers, but with attribute names changed.

SQL provides AS to achieve this and it is used in the SELECT part.

Renaming via as (cont.)

Example:

- Beers(name, manf)

```
SELECT name AS Brand, manf AS Brewer FROM Beers;
```

```
BRAND                        BREWER
--------------------------   ---------------------
80/-                         Caledonian
Bigfoot Barley Wine          Sierra Nevada
Burragorang Bock             George IV Inn
Crown Lager                  Carlton
Fosters Lager                Carlton
Invalid Stout                Carlton
…
```

Expressions as Values in Columns

AS can also be used to introduce computed values

Example:

Sells(bar, beer, price)

```
SELECT bar, beer, price*120 AS PriceInYen
FROM Sells;
```

```
BAR                 BEER                       PRICEINYEN
------------------  -------------------------  ----------------
Australia Hotel     Burragorang Bock           420
Coogee Bay Hotel    New                        270
Coogee Bay Hotel    Old                        300
Coogee Bay Hotel    Sparkling Ale              336
Coogee Bay Hotel    Victoria Bitter            276
…
```

Just Display but no change to the database

Inserting Text in Result Table

Trick: to put text in output columns, use constant expression with AS.

Example:

Likes(drinker, beer)

```
SELECT drinker, ‘likes Cooper’’s’ AS WhoLikes
FROM Likes
WHERE beer = ‘Sparkling Ale’;
```

```
DRINKER            WHOLIKES
---------------     --------------
Gernot             likes Cooper’s
Justin             likes Cooper’s
```

Drinker	Beer
Adam	Crown Lager
Adam	Fosters Lager
Adam	New
Gernot	Premium Lager
Gernot	Sparkling Ale
John	80/-
John	Bigfoot Barley Wine
John	Pale Ale
John	Three Sheets
Justin	Sparkling Ale
Justin	Victoria Bitter

Querying Multi-relations

Question: Find the brewers whose beers John likes?

Likes:

Drinker	Beer
Adam	Crown Lager
Adam	Fosters Lager
Adam	New
Gernot	Premium Lager
Gernot	Sparkling Ale
John	80/-
John	Bigfoot Barley Wine
John	Pale Ale
John	Three Sheets
Justin	Sparkling Ale
Justin	Victoria Bitter

Beers:

Name	Manf
80/-	Caledonian
Bigfoot Barley Wine	Sierra Nevada
Burragorang Bock	George IV Inn
Crown Lager	Carlton
Fosters Lager	Carlton
Invalid Stout	Carlton
Melbourne Bitter	Carlton
New	Toohey's
Old	Toohey's
Old Admiral	Lord Nelson
Pale Ale	Sierra Nevada
Premium Lager	Cascade
Red	Toohey's
Sheaf Stout	Toohey's
Sparkling Ale	Cooper's
Stout	Cooper's
Three Sheets	Lord Nelson
Victoria Bitter	Carlton

Querying Multi-relations

Question: Find the brewers whose beers John likes?

Likes:

Drinker	Beer
Adam	Crown Lager
Adam	Fosters Lager
Adam	New
Gernot	Premium Lager
Gernot	Sparkling Ale
John	80/-
John	Bigfoot Barley Wine
John	Pale Ale
John	Three Sheets
Justin	Sparkling Ale
Justin	Victoria Bitter

Beers:

Name	Manf
80/-	Caledonian
Bigfoot Barley Wine	Sierra Nevada
Burragorang Bock	George IV Inn
Crown Lager	Carlton
Fosters Lager	Carlton
Invalid Stout	Carlton
Melbourne Bitter	Carlton
New	Toohey's
Old	Toohey's
Old Admiral	Lord Nelson
Pale Ale	Sierra Nevada
Premium Lager	Cascade
Red	Toohey's
Sheaf Stout	Toohey's
Sparkling Ale	Cooper's
Stout	Cooper's
Three Sheets	Lord Nelson
Victoria Bitter	Carlton

Querying Multi-relations

Example: Find the brewers whose beers John likes

- Likes(drinker, beer)
- Beers(name, manf)

```
 SELECT Manf
 FROM Likes, Beers
 WHERE drinker = ’John’ AND beer = name

 MANF
 -------------
Caledonian
Sierra Nevada
Sierra Nevada
Lord Nelson
```

Note: could eliminate the duplicates by using DISTINCT

Relation algebra: $\pi_{\mathrm{manf}}(\sigma_{\mathrm{drinker}='\mathrm{john}'}\mathrm{Likes} \bowtie_{\mathrm{beer}=\mathrm{name}} \mathrm{Beers})$

The From Clause

For SQL SELECT statement on several relations:

SELECT Attributes

FROM R1, R2, ...

WHERE Condition

Formal semantics (relational algebra):

- start with **Cartesian Product** R1 × R2 × ... in FROM clause
- apply σ using Condition in WHERE clause
- apply π using Attributes in SELECT clause

Querying Multi-relations

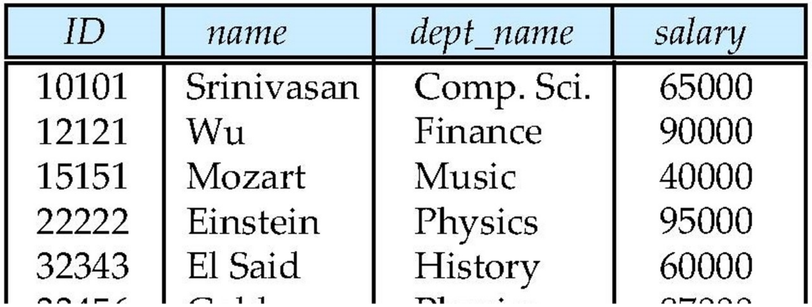
Operational semantics of SELECT (multi-relations):

```
FOR EACH tuple T1 in R1 DO
    FOR EACH tuple T2 in R2 DO
        ...
            check WHERE condition for current
            assignment of T1, T2, ... vars
            IF holds THEN
                print attributes of T1, T2, ...
                specified in SELECT
            END
        ...
    END
END
```

For efficiency reasons, it is not implemented in this way!

Cartesian Product

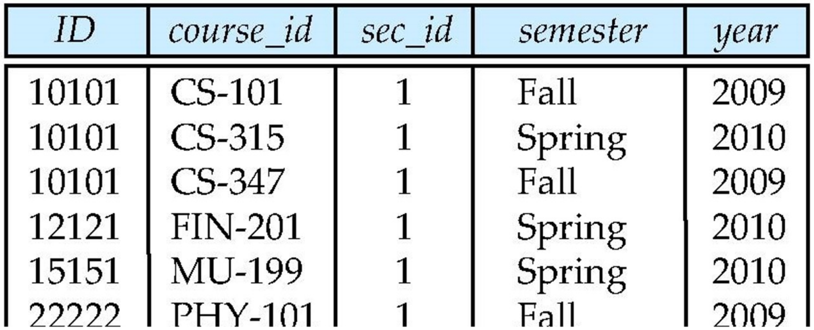
instructor



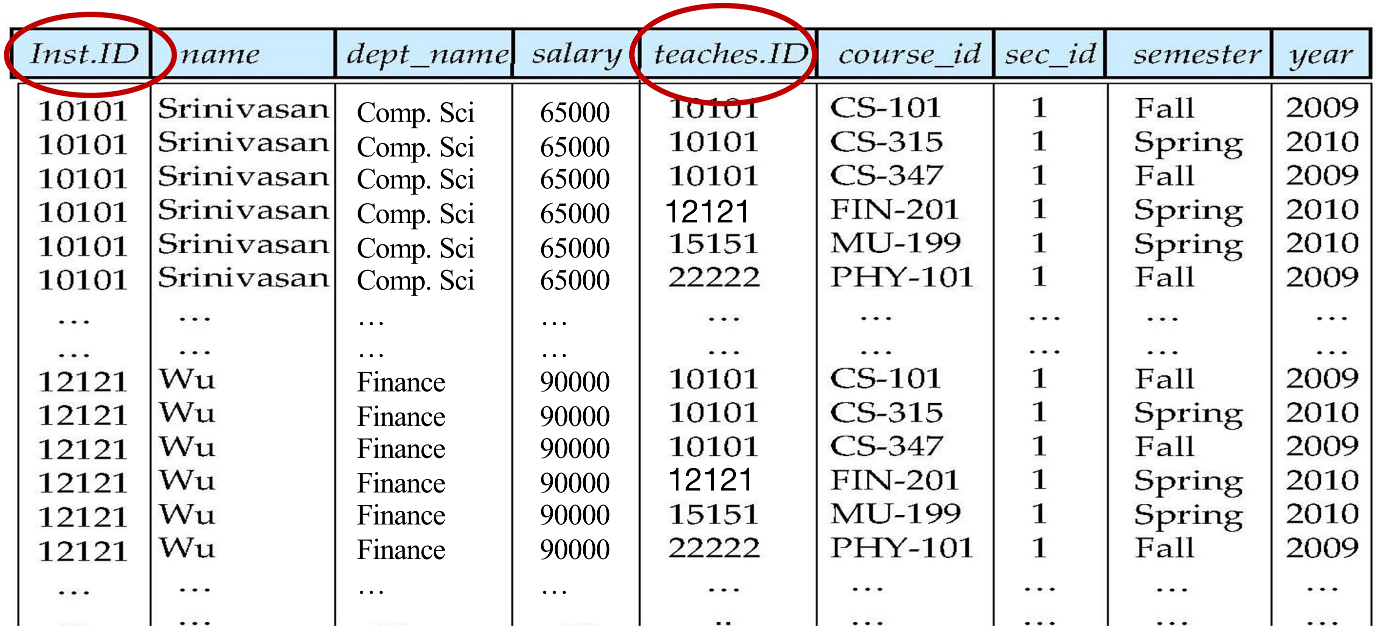
ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000

×

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
…	…	…	…	…	…	…	…	…
…	…	…	…	..	…	…	…	…

Attribute Name Clashes

Two tables can have attributes with the same name.

Beers (**name**, manf)
Bars (**name**, addr, license)

Problem: this ambiguity can lead to confusion if you write a query involving two tables with common column names:

if two same names appear in the WHERE clause

```
SELECT Bars.name
FROM Bars, Beers
WHERE name = name;
```

SQL: “**ERROR**: Ambiguous name column”.

Qualified Column Names

Solution: **disambiguate** attributes by specifying the relation name (giving a **qualified name** of a column)

- e.g., Bars.name means the column name from table Bars

We typically qualify a column name to specify the table which the column comes from. (see previous example below)

```
SELECT Bars.name
FROM Bars, Beers
WHERE Bars.name = Beers.name;
```

Qualified Column Names

Question: can I use **qualified names** even if there is no ambiguity?

```
SELECT Sells.beer
FROM Sells
WHERE Sells.price > 3.00;
```

Table Name Clashes

The **relation-dot-attribute** convention doesn't help if we use the same relation twice in SELECT.

To handle this, we need to define new names for each "instance" of the relation in the FROM clause.

Example: Find pairs of beers by the same manufacturer.

Note: we should avoid:

- pairing a beer with itself e.g., (New, New)
- same pairs with different order e.g., (New, Old) (Old, New)

Table Name Clashes

Beers:

Name	Manf
80/-	Caledonian
Bigfoot Barley Wine	Sierra Nevada
Burragorang Bock	George IV Inn
Crown Lager	Carlton
Fosters Lager	Carlton
Invalid Stout	Carlton
Melbourne Bitter	Carlton
New	Toohey’s
Old	Toohey’s
Old Admiral	Lord Nelson
Pale Ale	Sierra Nevada
Premium Lager	Cascade
Red	Toohey’s
Sheaf Stout	Toohey’s
Sparkling Ale	Cooper’s
Stout	Cooper’s
Three Sheets	Lord Nelson
Victoria Bitter	Carlton

Table Name Clashes

```
SELECT b1.name, b2.name
FROM Beers b1, Beers b2
WHERE b1.manf = b2.manf AND b1.name <
b2.name;
```

```
NAME                NAME
---------------     ---------------
Crown Lager         Fosters Lager
Crown Lager         Invalid Stout
Fosters Lager       Invalid Stout
Fosters Lager       Melbourne Bitter
….
```

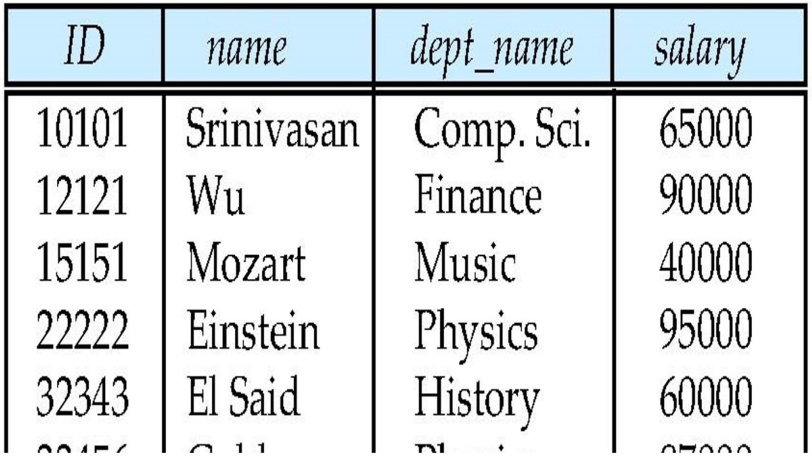
Beers:

Name	Manf
80/-	Caledonian
Bigfoot Barley Wine	Sierra Nevada
Burragorang Bock	George IV Inn
Crown Lager	Carlton
Fosters Lager	Carlton
Invalid Stout	Carlton
Melbourne Bitter	Carlton
New	Toohey’s
Old	Toohey’s
Old Admiral	Lord Nelson
Pale Ale	Sierra Nevada
Premium Lager	Cascade
Red	Toohey’s
Sheaf Stout	Toohey’s
Sparkling Ale	Cooper’s
Stout	Cooper’s
Three Sheets	Lord Nelson
Victoria Bitter	Carlton

Joins (1)

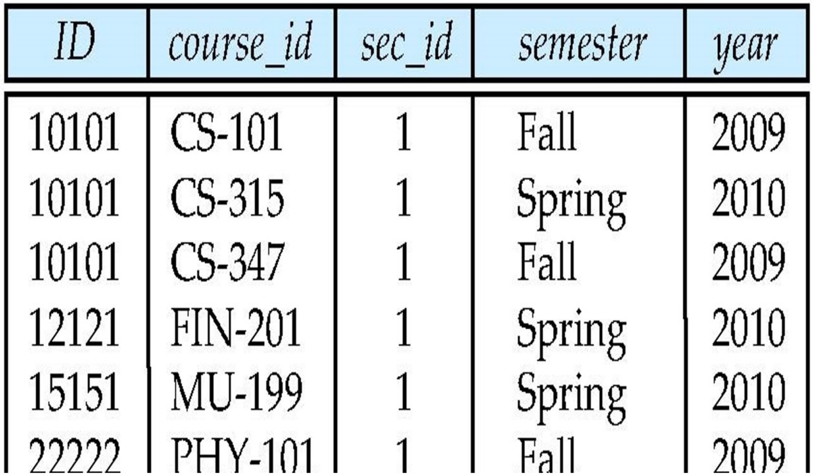
For all instructors who have taught courses, find their names and the course ID of the courses they taught.

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

Joins (1)

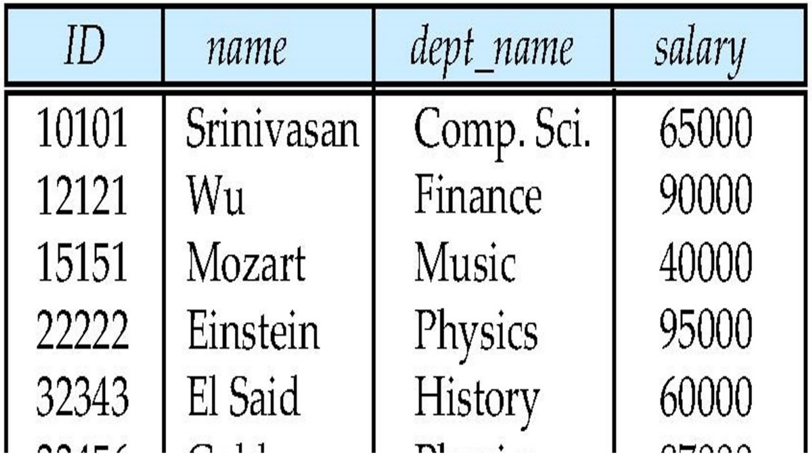
For all instructors who have taught courses, find their names and the course ID of the courses they taught.

SELECT name, course_id

FROM instructor, teaches

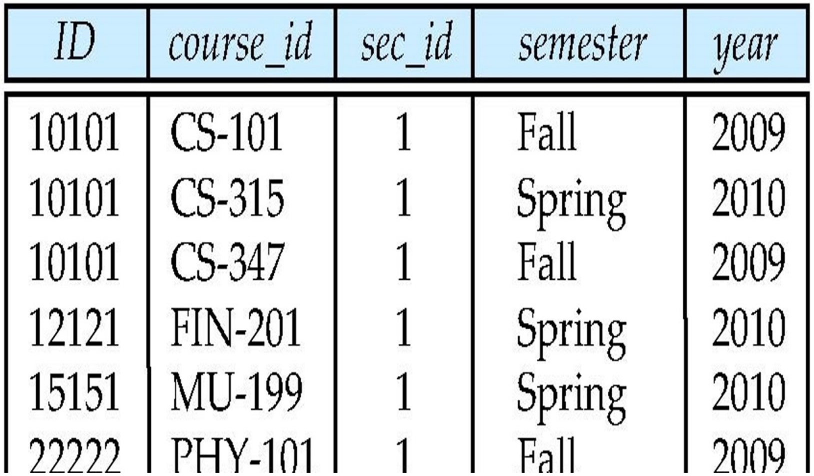
WHERE instructor.ID = teaches.ID

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

teaches

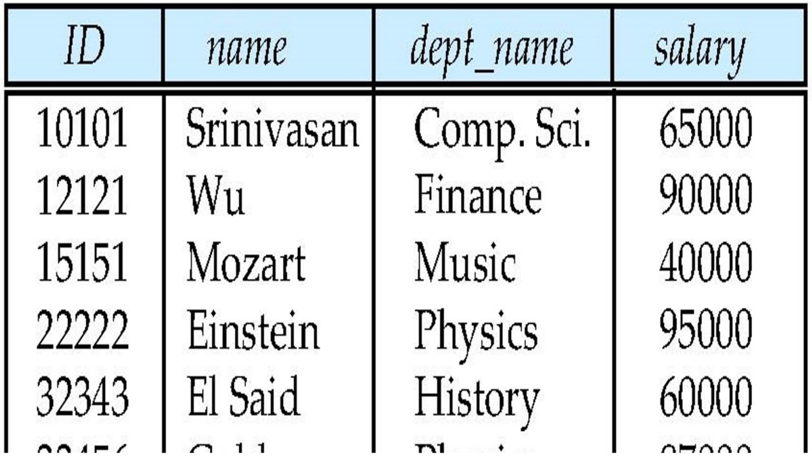


ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009

Joins (2)

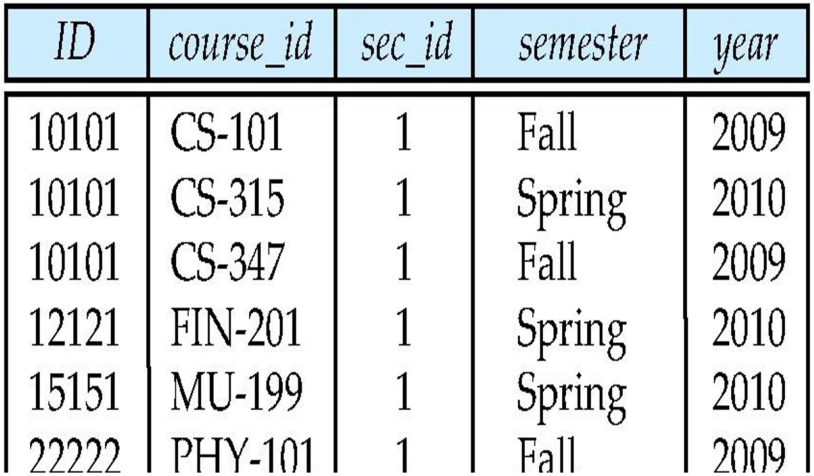
Find instructor names and the courses they taught in 2010.

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

Joins (2)

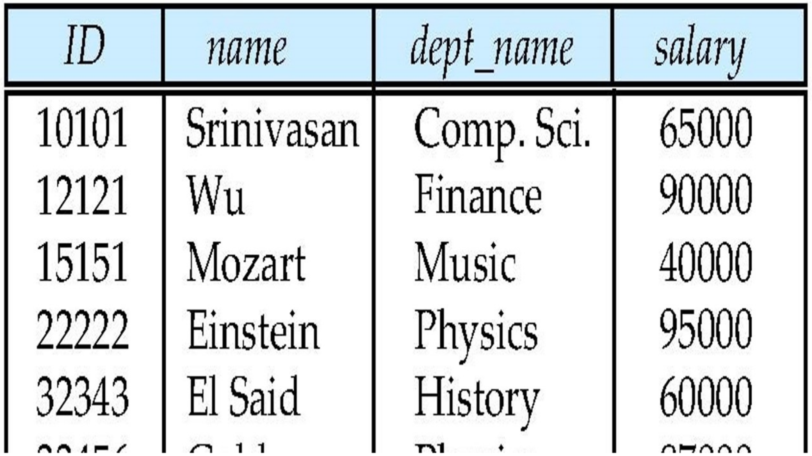
Find instructor names and the courses they taught in 2010.

SELECT name, course_id

FROM instructor, teaches

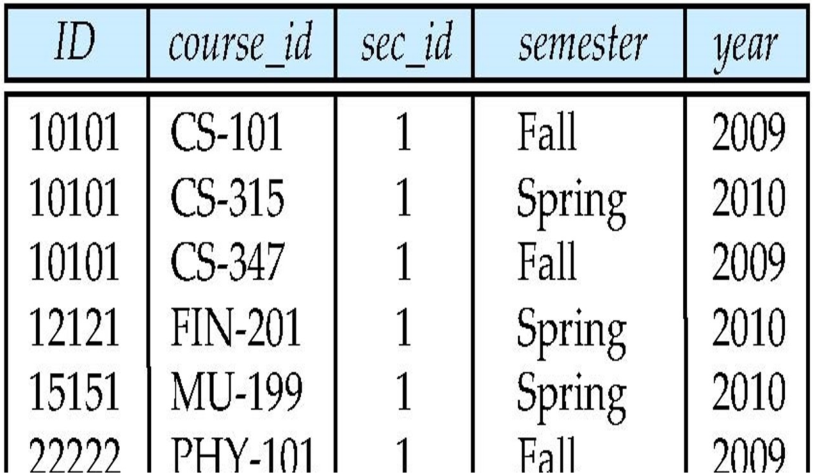
WHERE instructor.ID = teaches.ID **AND** year = 2010

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
[illegible]	[illegible]	[illegible]	[illegible]

teaches



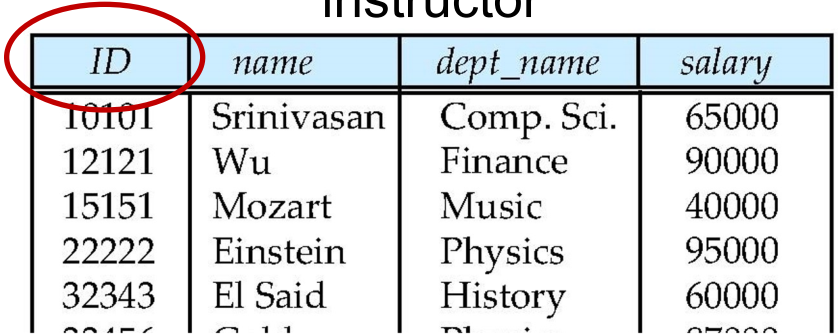
ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009

Natural Join

Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column

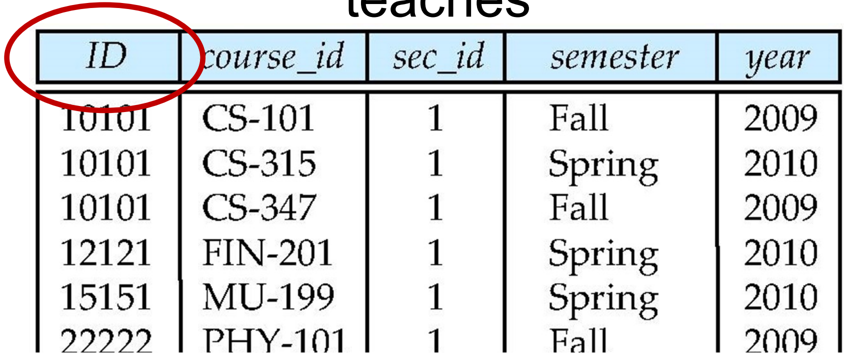
SELECT * **FROM** instructor **NATURAL JOIN** teaches;

instructor

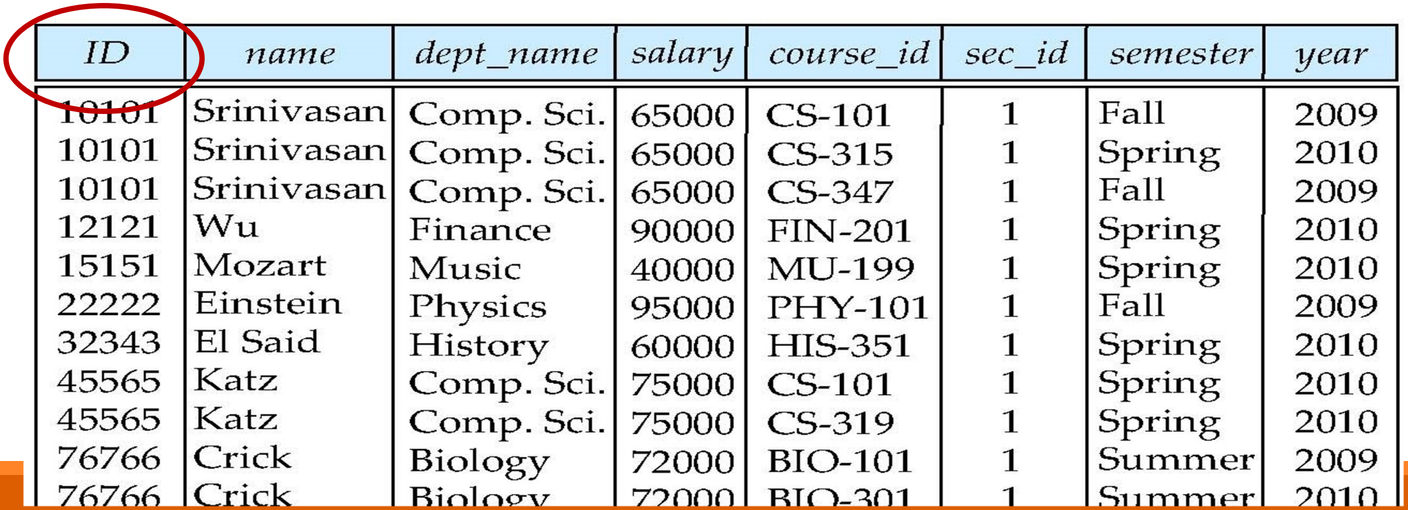


ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000

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



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

Danger of Natural Join

List the names of instructors along with the titles of courses that they teach. This is an incorrect version:

```
SELECT name, title
FROM instructor
NATURAL JOIN teaches
NATURAL JOIN course;
```

instructor

instructor_id	name	dept_name	salary
8	ABC	SEEM	100
7	XYZ	SEEM	120

teaches

instruct_id	course_id	s_id	semester	year
7	3550	1	1	2018
8	2100	1	2	2018

course

course_id	title	dept_name	credits
3550	DB	SEEM	3
2100	Algo	CSE	3

Danger of Natural Join

- Course.dept_name and instructor.dept_name are not related
- Therefore, cannot be assumed to be the same.

instructor

instructor_id	name	dept_name	salary
8	ABC	SEEM	100
7	XYZ	SEEM	120

teaches

instruct_id	course_id	s_id	semester	year
7	3550	1	1	2018
8	2100	1	2	2018

course

course_id	title	dept_name	credits
3550	DB	SEEM	3
2100	Algo	CSE	3

Correct Natural Join

List the names of instructors along with the titles of courses that they teach. This is a correct version:

SELECT name, title
FROM instructor, teaches, course
WHERE instructor.ID = teaches.ID **AND** teaches.course_id = course.course_id;

instructor

instructor_id	name	dept_name	salary
8	ABC	SEEM	100
7	XYZ	SEEM	120

teaches

instruct_id	course_id	s_id	semester	year
7	3550	1	1	2018
8	2100	1	2	2018

course

course_id	title	dept_name	credits
3550	DB	SEEM	3
2100	Algo	CSE	3

JOIN ON

books(id, title, translator_id)

translators(id, name)

```
SELECT books.id, books.title, translators.name AS translator
FROM books
JOIN translators ON books.translator_id = translators.id；
```