SQL
Part 1

History

- SQL was developed in the 1970s at IBM
- It is an ISO Standard since 1987
  - The standard consists of 16 parts, each costs about $200
- Existing relational database systems
  - generally try to follow the standard
  - but also deviate from it significantly

Parts of SQL

- **Data Definition Language** (**DDL**)
  - Define schemas, integrity constraints, views

- **Data Manipulation Language** (**DML**)
  - Query (select), insert, delete and update tuples in the database
  - Specify transactions

- **Data Control Language** (**DCL**)
  - Control access to data stored in a database
  - Grant and revoke privileges.

DATA DEFINITION LANGUAGE (DDL)

Domains: Basic Data Types

- **char(n)** Fixed length character string, with length *n*
- **varchar(n)** Variable length character string, with maximum length *n*
- **integer** Integer, size is machine-dependent
- **real** Floating point number, with machine-dependent precision.
- **numeric(p,d)** Fixed point number, with *p* digits before and *n* digits after the decimal point.

Domains: Large-Object Data Types

- Objects that are large (several kilobytes up to several gigabytes) are stored as:
  - **blob**, binary large object: uninterpreted binary data (interpretation is left to an application outside of the database system)
    - a photo or video
  - **clob**, character large object: a large string.
    - an XML/HTML/Markdown/JSON document

- When a query returns a large object, a pointer is returned rather than the large object itself

Create Table

- Define a relation:

  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) not null,
    dept_name   varchar(20),
    salary      numeric(8,2))
```

Drop Table and Alter Table

- **drop table** *student*
  - Deletes the table and its contents
  - This is different from **"delete from** student" (which just deletes the contents, and is part of the DML)

- **alter table** *student* **add** *name* **varchar***(20)*
  - Adds attribute with domain
  - Existing tuples in the relation are extended with *null* as the value for the new attribute.

- **alter table** *student* **drop** *name*
  - Removes attribute
  - Many databases do not support it

Integrity Constraints

- Examples
  - An instructor name cannot be *null*.
  - No two instructors can have the same instructor ID.
  - Every department name in the *course* relation must have a matching department name in the *department* relation.
  - A semester must be either *Spring* or *Fall*

- When you specify such constraints then the database rejects changes that violate them

Integrity Constraints

```
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)
```

- **not null**: null values will be rejected

- **unique** (A_1, A_2, …, A_n): states that the attributes A_1, …, A_n form a superkey. In other words, a change will be rejected if it leads to two tuples with the same values on A_1, ..., A_n

- **primary key** (A_1, ..., A_n): denotes the primary key, and implies both:
  - **unique** (A_1, …, A_n) and
  - **not null** for every A_i

Integrity Constraints

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

- The primary key declaration can be combined with the attribute declaration as shown above
- **foreign key** (A_1, ..., A_n) **references** *r* :
  - a change will be rejected if it leads to a tuple for which there is no tuple in *r* with the same values on A_1, ..., A_n

Integrity Constraints in SQLite

- By default, SQLite does **not** follow the SQL standard:
  - SQLite does allow *null* columns in primary keys
  - SQLite does not check foreign key constraints at all

- Why do you think that's the case?

- To enable foreign key constraint checking, do the following:

```
sqlite> PRAGMA foreign_keys = ON;
```

- To ensure primary key columns are not null, do the following:

```
create table instructor (
   ID              char(5),
    …
   primary key (ID) not null,
    …
    )
```

Integrity Constraints: Understanding the Primary Key Constraint

-
  ```
  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 );
  ```

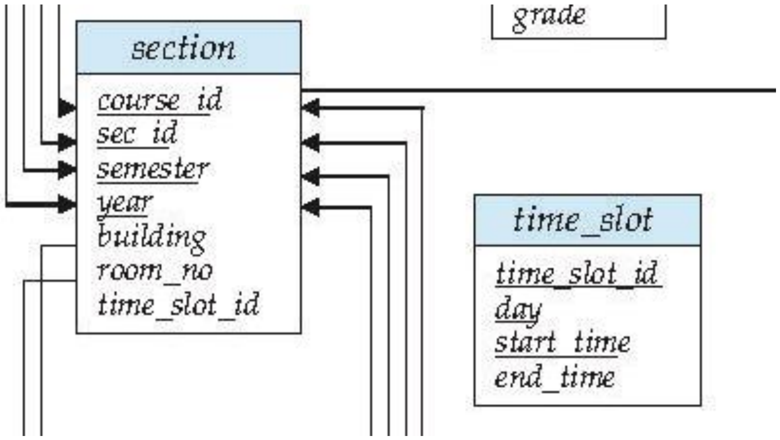
- Question: what happens if *sec_id* is dropped from the primary key above?

Integrity Constraint: check(P)

- **check** (P), where P is a predicate:

```
create table section (
    course_id varchar (8),
    semester varchar (6),
        …
    primary key (course_id, sec_id, semester, year),
    check (semester in (’Fall’, ’Spring’))
);
```

Complex Check Clauses



- We want to add a constraint to section:
  - Each *time_slot_id* should appear in *time_slot*
- But we cannot use a foreign key constraint
- The SQL standard allows subqueries in the check clause:
  check (*time_slot_id* **in** (**select** *time_slot_id* **from** *time_slot*))
- But most databases do not support subqueries in the check clause
- It can be done using triggers (which we'll see later)

DATA MANIPULATION LANGUAGE (DML)

The select Clause

- By default SQL lists duplicate tupels:

  select *dept_name*
  from *instructor*

  - To force the elimination of duplicates**:**

    select distinct *dept_name*
    from *instructor*

  - To retain duplicates (default):

    select all *dept_name*
    from *instructor*

- An asterisk denotes “all attributes”

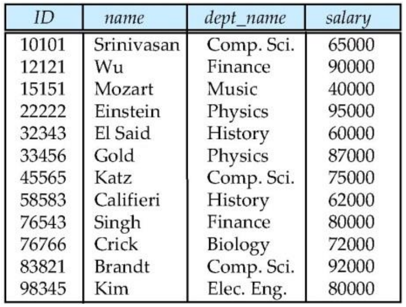
  select *
  from *instructor*

- Can contain arithmetic expressions:

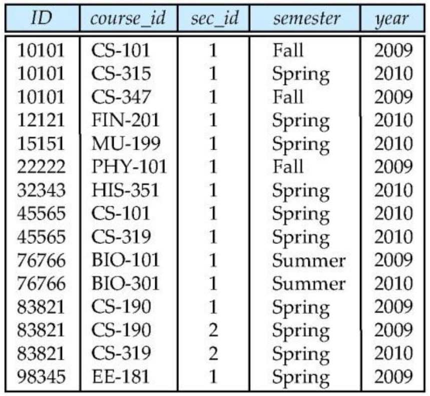
  select *ID, name, salary / 12*
  from *instructor*

Natural Join

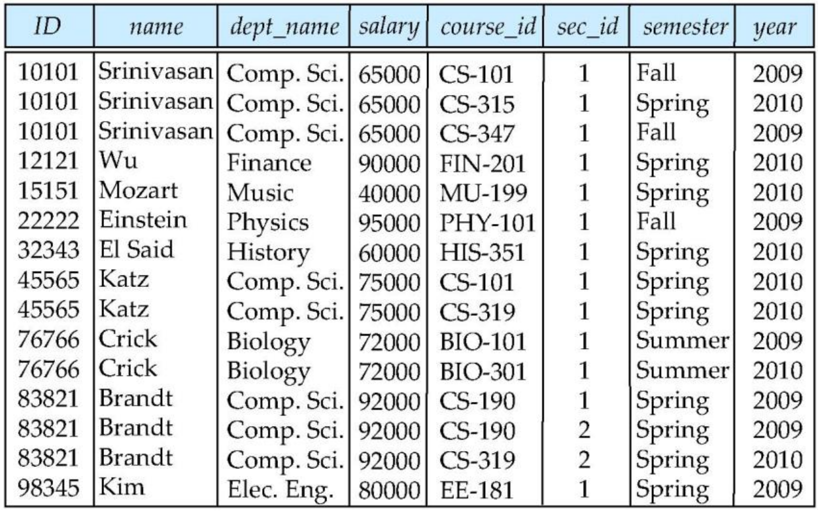
- **select** * **from** *instructor* **natural join** *teaches*;



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



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
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	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

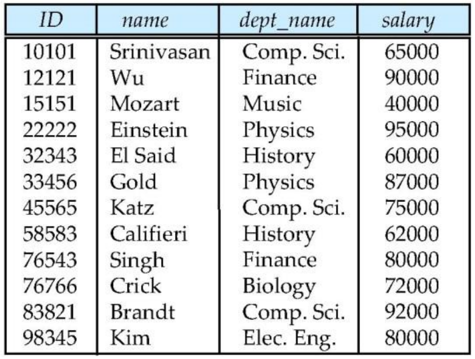
Natural Join Example

- List the names of instructors along with the course ID of the courses that they taught.

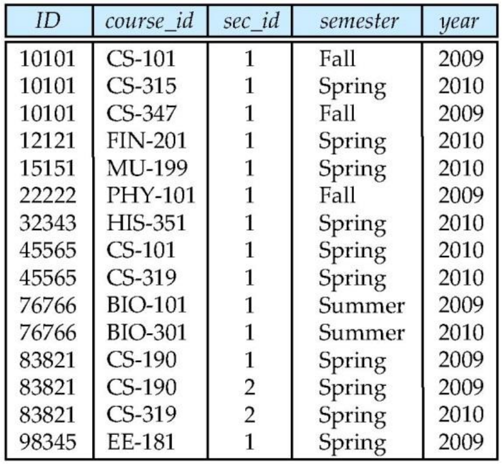
  - **select** *name*, *course_id*
    from *instructor, teaches*
    where *instructor.ID* = *teaches.ID*;

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

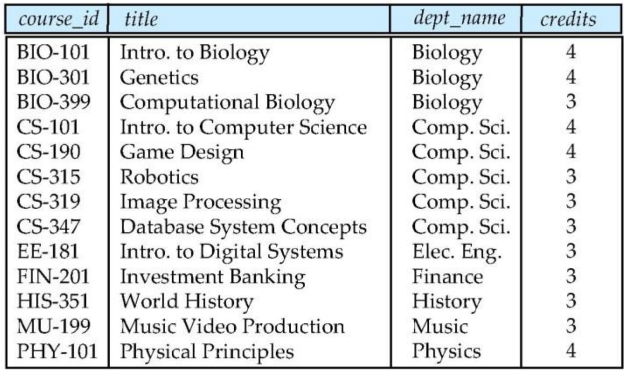
Natural Join



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



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
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

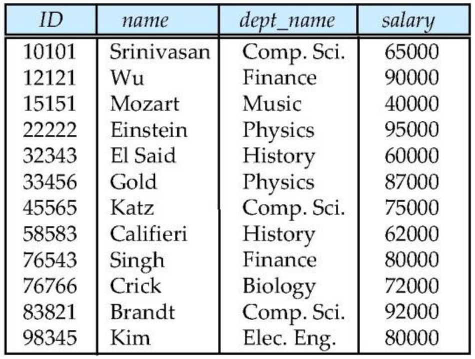


course_id	*title*	*dept_name*	*credits*
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database 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

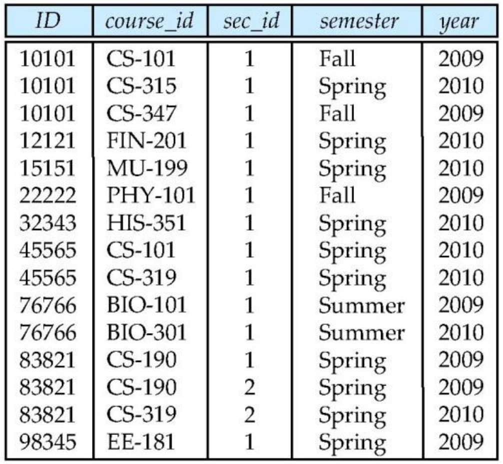
- What is the intended result of the following query?

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

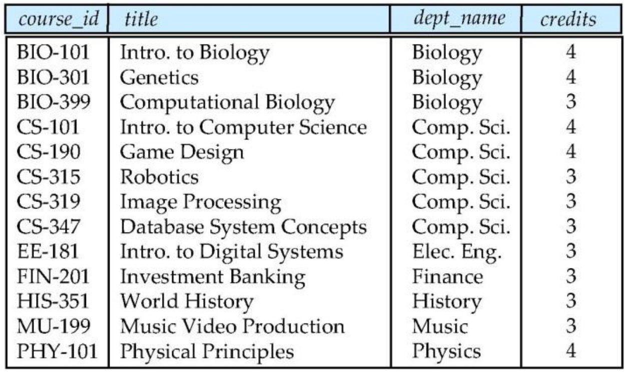
Natural Join



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



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
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009



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

- Correct query to find names of instructors with courses that they teach:

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

The Rename Operation – as clause

- Renaming attributes:
  - **select** *ID, name, salary/12* **as** *monthly_salary*
    from *instructor*

- Renaming relations:
  - Find all pairs of instructors who have the same name:

    select *T.ID, S.ID*
    from *instructor* **as** *T, instructor* **as** *S*
    where *T.name = S.name*

    - Keyword **as** is optional and may be omitted
    - Keyword **as** must be omitted in Oracle

String Matching – like clause

- Patterns are strings containing:
  - percent (%). Matches any substring.
  - underscore (_). Matches any character.
- Example:

```
select name
from instructor
where name like '%stein%'
```

  - ‘Intro%’ matches any string beginning with “Intro”.
  - ‘_ _ _’ matches any string of exactly three characters.
  - ‘_ _ _ %’ matches any string of at least three characters.

- Escaping the characters % and _:
  - Match the string “100%”:

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

Ordering – order by clause

- List names in alphabetic order:

```
select distinct name
from  instructor
order by name
```

- Specify **desc**ending order, **asc**ending order is default:

```
order by name desc
```

- Sort on multiple attributes:

```
order by  dept_name asc, name desc
```

Set Operations: union, intersect, except

Keyword in SQL	Relational Algebra
union	$\cup$
intersect	$\cap$
except	$-$

- Find courses that ran in 2009 or in 2010 or both:

```
select course_id  from section where year = 2009
  union
select course_id  from section where year = 2010;
```

- ...and similarly for the set operations **intersect** and **except**
- Set operations eliminate duplicates!
- To retain duplicates use **all**:
    - **union all**
    - **intersect all**
    - **except all**

Null Values

- Null values work just like in relational algebra
  - The result of any arithmetic expression involving *null* is *null*
  - comparison with *null* returns special boolean value *unknown*
  - **where** clause: treats *unknown* predicate as *false*

- What's the result of this?

  select *name*
  from *instructor*
  where *salary* = *null*

name	salary
Einstein	80000
Katz	null
Mozart	0

- The predicates **is null** and **is not null** need to be used to check for null values

More Problems with Nulls

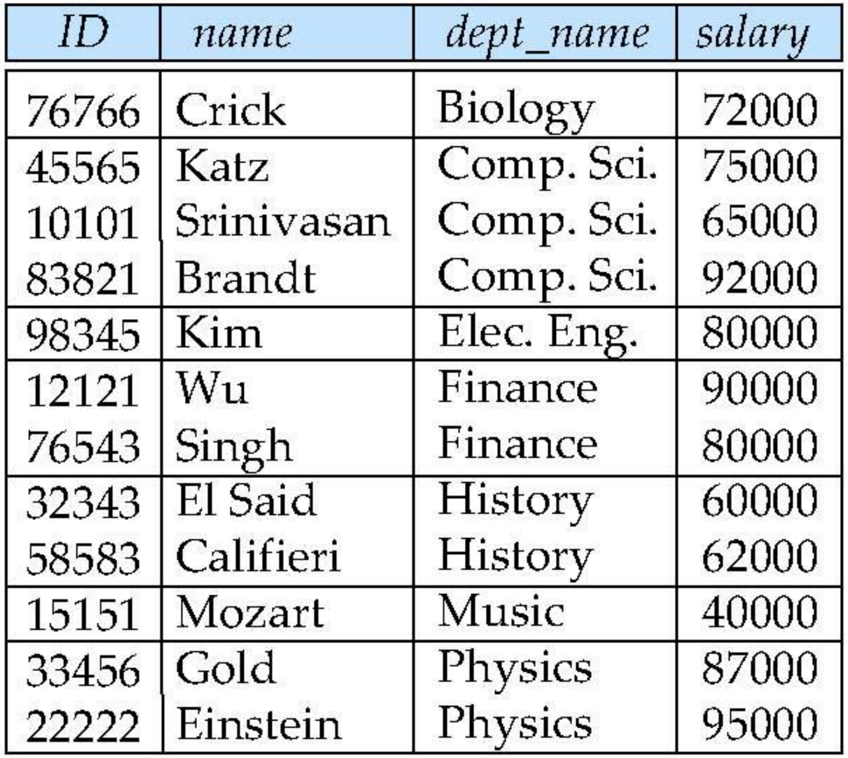
- While the **where** clause treats *unknown* as *false*, the **check** clause treats *unknown* as *true*
- …so null = null is sometimes treated as true (in **check** clauses) and sometimes as false (in **where** clauses)
- Set operations, Aggregate grouping and the **distinct** clause treat different nulls as equal
- How nulls are sorted is implementation-specific
- The SQL standard has two **unique** constraints: one that treats nulls as equal and one that treats nulls as different
    - In SQLite **unique** treats nulls as different

Aggregate Functions

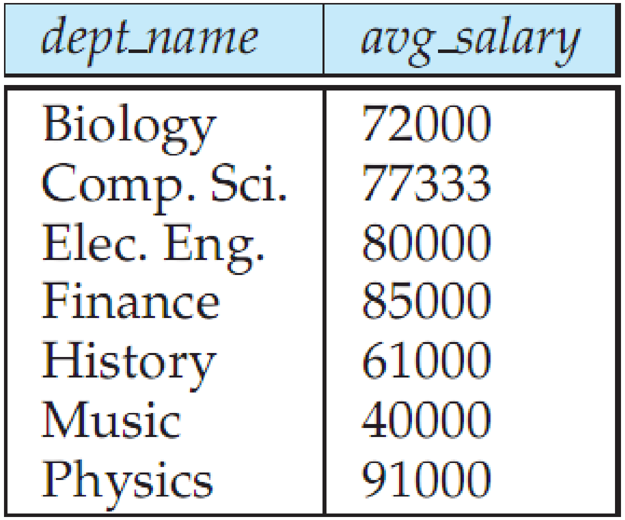
- Find the average salary of instructors in the Computer Science department
  - **select avg** (*salary*)
    from *instructor*
    where *dept_name*= 'Comp. Sci.';
- Find the number of tuples in the *course* relation
  - **select count** (*)
    from *course*;
- Find the number of instructors who taught a course in 2010
  - **select count** (**distinct** *ID*)
    from *teaches*
    where *year* = 2010

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

- The attributes in the **select** clause outside of the aggregate functions must appear in the **group by** list
- So, for example, the following is not allowed:

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

- Actually, since 2011 the SQL standard does allow this ... but does not specify the semantics so it's interpreted differently by different databases. Try not to use it.

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;

- Predicates in the **where** clause are applied before forming groups
- Predicates in the **having** clause are applied after forming groups

Null Values and Aggregates

- All aggregate operations except count(*) ignore null values
  - Example: Summing all salaries is really summing all *known* salaries

    select sum (*salary*)
    from *instructor*

  - Same for **count** (<Attribute>)
  - Exception: **count(*)** does count null values

- On empty collections:
  - **count** returns 0
  - all other aggregates return null (!)

Nested Subqueries

- A **subquery** is a query inside another query.
- There are three kinds of subqueries:
  - in the where-clause,
  - in the from-clause,
  - scalar subqueries (that can occur anywhere).

Subquery in the Where-Clause – in

- Find courses offered in 2009 and in 2010

```
select distinct course_id
from section
where year = 2009 and
course_id in (      select course_id
                    from section
                    where year = 2010 );
```

Subquery in the Where-Clause – exists

- **exists** *r* returns **true** iff *r* is nonempty.
- **not exists** *r* returns **true** iff *r* is empty

- Example: Find all courses taught in both 2009 and 2010

```
select course_id
from section as S
where year = 2009 and
        exists (select *
                from section as T
                where year = 2010 and S.course_id = T.course_id);
```

  - S is a correlation variable
  - the inner query is a correlated subquery

Subquery in the Where-Clause – exists

- 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 );
```

- Note: X – Y = Ø   iff   X is a subset of Y

Subqueries in the From Clause

- Find the average salary of the departments where the average salary is greater than $42,000.

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

- Similar result with subquery instead of having-clause:

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

Subqueries in the From Clause

- Find the maximum of the total salaries at each department:

```
select max (tot_salary)
from      (select dept_name, sum(salary) as tot_salary
           from instructor
           group by dept_name);
```

- We cannot write this using the having-clause

With Clause

- The with clause defines a temporary relation only available to the query in which it occurs
- Useful for writing complex queries
- Find the departments with the maximum budget:

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

Complex Queries using With Clause

- Find the departments which have an above-average total salary:

```
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;
```

Scalar Subqueries

- A **scalar subquery** is a subquery which is used where a single value is expected
- A scalar subquery that returns more than one tuple gives a runtime error
- Example: Scalar subquery in the select-clause
  - Find the number of instructors per department:

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

Scalar Subqueries

- Example of a scalar subquery in the where-clause:
  - Find the instructors that cost more than 10% of their departments budget:

```
select name
from instructor
where  salary * 10 >
          (select budget  from department
             where department.dept_name = instructor.dept_name)
```