Chapter 3: Introduction to SQL
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.
Data Definition Language

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

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints
- And as we will see later, also other information such as
  - 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 $n$ digits to the right of decimal point.
- **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:

  ```sql
  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:

  ```sql
  create table instructor (ID char(5),
                          name varchar(20),
                          dept_name varchar(20),
                          salary numeric(8,2))
  ```
Integrity Constraints in Create Table

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

Example: Declare `branch_name` as the primary key for `branch`

```sql
create table instructor (
    ID char(5),
    name varchar(20) not null,
    dept_name varchar(20),
    salary numeric(8,2),
    primary key (ID),
    foreign key (dept_name) references department);
```

**primary key** declaration on an attribute automatically ensures **not null**
And a Few More Relation Definitions

- **create table** `student` (  
  
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>varchar(5)</td>
<td>primary key,</td>
</tr>
<tr>
<td>name</td>
<td>varchar(20)</td>
<td>not null,</td>
</tr>
<tr>
<td>dept_name</td>
<td>varchar(20),</td>
<td></td>
</tr>
<tr>
<td>tot_cred</td>
<td>numeric(3,0),</td>
<td></td>
</tr>
<tr>
<td>foreign key (dept_name) references department);</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **create table** `takes` (  
  
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>varchar(5),</td>
<td></td>
</tr>
<tr>
<td>course_id</td>
<td>varchar(8),</td>
<td></td>
</tr>
<tr>
<td>sec_id</td>
<td>varchar(8),</td>
<td></td>
</tr>
<tr>
<td>semester</td>
<td>varchar(6),</td>
<td></td>
</tr>
<tr>
<td>year</td>
<td>numeric(4,0),</td>
<td></td>
</tr>
<tr>
<td>grade</td>
<td>varchar(2),</td>
<td></td>
</tr>
<tr>
<td>primary key (ID, course_id, sec_id, semester, year) ,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>foreign key (ID) references student,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>foreign key (course_id, sec_id, semester, year) references section);</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: `sec_id` can be dropped from primary key above, to ensure a student cannot be registered for two sections of the same course in the same semester
Drop/Alter Table Constructs

- **Drop Table** – delete all information about a relation from the database
  - `drop table r`

- **Alter Table** – alter attributes of an existing relation
  - `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 tuples in the relation are assigned *null* as the value for the new attribute.
  - `alter table r drop A`
    - where $A$ is the name of an attribute of relation $r$
    - Dropping of attributes not supported by many databases.
Basic Structure of SQL Queries

- A typical SQL query has the form:

  ```sql
  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.
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
  
  ```sql
  select name
  from instructor
  ```
- An asterisk in the select clause denotes “all attributes”
  
  ```sql
  select *
  from instructor
  ```
- **NOTE:** SQL names are case insensitive (i.e., you may use upper- or lower-case letters.)
  - E.g., `Name ≡ NAME ≡ name`
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.
  
  ```sql
  select distinct dept_name
  from instructor
  ```

- The keyword `all` specifies that duplicates not be removed. (default)
  
  ```sql
  select all dept_name
  from instructor
  ```
The select Clause (Cont.)

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

- The query:

```sql
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 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`
  ```sql
  select *
  from instructor, teaches
  ```
  - generates every possible instructor – teaches pair, with all attributes from both relations.
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra).
# Cartesian Product

The Cartesian product of two tables can be used to combine the rows of one table with the rows of another table, where each pair of rows from the two tables is combined together.

### Table 1: Instructor Table

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
</tr>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
</tr>
<tr>
<td>32343</td>
<td>El Said</td>
<td>History</td>
<td>60000</td>
</tr>
<tr>
<td>22222</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
</tr>
</tbody>
</table>

### Table 2: Teaches Table

<table>
<thead>
<tr>
<th>ID</th>
<th>course_id</th>
<th>sec_id</th>
<th>semester</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>CS-101</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>10101</td>
<td>CS-315</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>10101</td>
<td>CS-347</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>12121</td>
<td>FIN-201</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>15151</td>
<td>MU-199</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>22222</td>
<td>PHY-101</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
</tbody>
</table>

### Combined Table

<table>
<thead>
<tr>
<th>Inst.ID</th>
<th>name</th>
<th>dept_name</th>
<th>salary</th>
<th>teaches.ID</th>
<th>course_id</th>
<th>sec_id</th>
<th>semester</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>10101</td>
<td>CS-101</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>10101</td>
<td>CS-315</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>10101</td>
<td>CS-347</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>12121</td>
<td>FIN-201</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>15151</td>
<td>MU-199</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>22222</td>
<td>PHY-101</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
<td>10101</td>
<td>CS-101</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
<td>10101</td>
<td>CS-315</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
<td>10101</td>
<td>CS-347</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
<td>12121</td>
<td>FIN-201</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
<td>15151</td>
<td>MU-199</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
<td>22222</td>
<td>PHY-101</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

*Database System Concepts - 6th Edition* 3.15 ©Silberschatz, Korth and Sudarshan
The where Clause

- The **where** clause specifies conditions that the result must satisfy
  - Corresponds to the selection predicate of the relational algebra.
- To find all instructors in Comp. Sci. dept with salary > 80000
  
  ```sql
  select name
  from instructor
  where dept_name = 'Comp. Sci.' and salary > 80000
  ```
- Comparison results can be combined using the logical connectives **and**, **or**, and **not**.
- Comparisons can be applied to results of arithmetic expressions.
Joins

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

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

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

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

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

- Find the IDs of all students who were taught by an instructor named Einstein; make sure there are no duplicates in the result.
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;`

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>salary</th>
<th>course_id</th>
<th>sec_id</th>
<th>semester</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>CS-101</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>CS-315</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>Comp. Sci.</td>
<td>65000</td>
<td>CS-347</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000</td>
<td>FIN-201</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>Music</td>
<td>40000</td>
<td>MU-199</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>Physics</td>
<td>95000</td>
<td>PHY-101</td>
<td>1</td>
<td>Fall</td>
<td>2009</td>
</tr>
<tr>
<td>32343</td>
<td>El Said</td>
<td>History</td>
<td>60000</td>
<td>HIS-351</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>45565</td>
<td>Katz</td>
<td>Comp. Sci.</td>
<td>75000</td>
<td>CS-101</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>45565</td>
<td>Katz</td>
<td>Comp. Sci.</td>
<td>75000</td>
<td>CS-319</td>
<td>1</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>76766</td>
<td>Crick</td>
<td>Biology</td>
<td>72000</td>
<td>BIO-101</td>
<td>1</td>
<td>Summer</td>
<td>2009</td>
</tr>
<tr>
<td>76766</td>
<td>Crick</td>
<td>Biology</td>
<td>72000</td>
<td>BIO-301</td>
<td>1</td>
<td>Summer</td>
<td>2010</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>Comp. Sci.</td>
<td>92000</td>
<td>CS-190</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>Comp. Sci.</td>
<td>92000</td>
<td>CS-190</td>
<td>2</td>
<td>Spring</td>
<td>2009</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>Comp. Sci.</td>
<td>92000</td>
<td>CS-319</td>
<td>2</td>
<td>Spring</td>
<td>2010</td>
</tr>
<tr>
<td>98345</td>
<td>Kim</td>
<td>Elec. Eng.</td>
<td>80000</td>
<td>EE-181</td>
<td>1</td>
<td>Spring</td>
<td>2009</td>
</tr>
</tbody>
</table>
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;`
The Rename Operation

- The SQL allows renaming relations and attributes using the `as` clause:
  
  \[
  \text{old-name} \text{ as } \text{new-name}
  \]

- E.g.,

  - select `ID`, `name`, `salary/12` as `monthly_salary`  
  from `instructor`

- Find the names of all instructors who have a higher salary than some instructor in ‘Comp. Sci.’

  - select `distinct T. name`  
    from `instructor` as `T`, `instructor` as `S`  
    where `T.salary > S.salary` and `S.dept_name` = ‘Comp. Sci.’

- Keyword `as` is optional and may be omitted

  \[
  \text{instructor as } T \equiv \text{instructor } T
  \]
String Operations

- **like**: a string-matching operator for comparisons on character strings

- 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 instructors whose name includes the substring “dar”.
  
  ```sql
  select name
  from instructor
  where name like '%dar%'
  ```

- Match the string “100%”
  
  ```sql
  like '100\%' escape '\'
  ```

  in that above we use backslash (\) as the escape character.
Ordering the Display of Tuples

- List in alphabetic order the names of all instructors
  
  ```sql
  select distinct name
  from instructor
  order by name
  ```

- We may specify `desc` for descending order or `asc` for ascending order, for each attribute; ascending order is the default.
  
  - Example: `order by name desc`

- Can sort on multiple attributes
  
  - Example: `order by dept_name, name`
Set Operations – Example

- Find courses that ran in Fall 2009 or in Spring 2010
  \[(\text{select } \text{course\_id} \text{ from section where } \text{sem} = \text{‘Fall’ and year} = 2009) \text{ union} \ (\text{select } \text{course\_id} \text{ from section where } \text{sem} = \text{‘Spring’ and year} = 2010)\]

- Find courses that ran in Fall 2009 and in Spring 2010
  \[(\text{select } \text{course\_id} \text{ from section where } \text{sem} = \text{‘Fall’ and year} = 2009) \text{ intersect} \ (\text{select } \text{course\_id} \text{ from section where } \text{sem} = \text{‘Spring’ and year} = 2010)\]

- Find courses that ran in Fall 2009 but not in Spring 2010
  \[(\text{select } \text{course\_id} \text{ from section where } \text{sem} = \text{‘Fall’ and year} = 2009) \text{ except} \ (\text{select } \text{course\_id} \text{ from section where } \text{sem} = \text{‘Spring’ and year} = 2010)\]
Set Operations

- Set operations **union**, **intersect**, and **except**
  - Each of the above operations automatically eliminates duplicates

- To retain all duplicates use the corresponding multiset versions **union all**, **intersect all** and **except all**.

- Suppose a tuple occurs $m$ times in $r$ and $n$ times in $s$, then, it occurs:
  - $m + n$ times in $r$ **union all** $s$
  - $\min(m,n)$ times in $r$ **intersect all** $s$
  - $\max(0, m - n)$ times in $r$ **except all** $s$
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.
    ```sql
    select name
    from instructor
    where salary is null
    ```
Null Values and Three Valued Logic

- Any comparison with `null` returns `unknown`
  - Example: `5 < null` or `null <> null` or `null = null`
- Three-valued logic using the truth value `unknown`:
  - OR: `(unknown or true) = true`,
    `(unknown or false) = unknown`
  - (unknown or unknown) = unknown
  - AND: `(true and unknown) = unknown`,
    `(false and unknown) = false`,
    `(unknown and unknown) = unknown`
  - NOT: `(not unknown) = unknown`
  - “P is unknown” evaluates to true if predicate P evaluates to unknown
- Result of `where` clause predicate is treated as `false` if it evaluates to unknown
Aggregate Functions

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

  - **avg**: average value
  - **min**: minimum value
  - **max**: maximum value
  - **sum**: sum of values
  - **count**: number of values
Aggregate Functions (Cont.)

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

- Find the total number of instructors who teach a course in the Spring 2010 semester
  
  ```sql
  select count(distinct ID)
  from teaches
  where semester = 'Spring' and year = 2010;
  ```

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

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

<table>
<thead>
<tr>
<th>dept_name</th>
<th>avg_salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biology</td>
<td>72000</td>
</tr>
<tr>
<td>Comp. Sci.</td>
<td>77333</td>
</tr>
<tr>
<td>Elec. Eng.</td>
<td>80000</td>
</tr>
<tr>
<td>Finance</td>
<td>85000</td>
</tr>
<tr>
<td>History</td>
<td>61000</td>
</tr>
<tr>
<td>Music</td>
<td>40000</td>
</tr>
<tr>
<td>Physics</td>
<td>91000</td>
</tr>
</tbody>
</table>
Attributes in select clause outside of aggregate functions must appear in group by list

/* erroneous query */
select dept_name, ID, avg(salary)
from instructor
group by dept_name;
Exercise

- Find the highest salary of any instructor.

- Find the enrollment of each section that was offered in Autumn 2009.
Aggregate Functions – Having Clause

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

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

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups.
Null Values and Aggregates

- Total all salaries

  ```sql
  select sum(salary)
  from instructor
  ```

  - Above statement ignores null amounts
  - Result is `null` if there is no non-null amount

- All aggregate operations except `count(*)` ignore tuples with null values on the aggregated attributes
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.
- A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality.
Example Query

- Find courses offered in Fall 2009 and in Spring 2010

```sql
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
    course_id in (select course_id
                   from section
                   where semester = 'Spring' and year= 2010);
```

- Find courses offered in Fall 2009 but not in Spring 2010

```sql
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
    course_id not in (select course_id
                       from section
                       where semester = 'Spring' and year= 2010);
```
Example Query

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

```sql
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. The formulation above is simply to illustrate SQL features.
Set Comparison

- Find names of instructors with salary greater than that of some (at least one) instructor in the Biology department.

```sql
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

```sql
select name
from instructor
where salary > some (select salary
    from instructor
    where dept_name = 'Biology');
```
Definition of some Clause

F <comp> some r ⇔ ∃ t ∈ r such that (F <comp> t )
Where <comp> can be: <, ≤, >, =, ≠

(5 < some 5 ) = true
(5 < some 5 ) = false
(5 = some 5 ) = true
(5 ≠ some 5 ) = true (since 0 ≠ 5)
(= some)≡ in
However, (≠ some)≠ not in
Example Query

- 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');
```
Definition of \textit{all} Clause

\[ F <\text{comp}> \text{all} \ r \iff \forall t \in r \ (F <\text{comp}> t) \]

\[
\begin{array}{c|c}
5 & 0 \\
6 & 0 \\
\end{array}
\]

\((5 < \text{all} \ 5 \ 6) = \text{false}\)

\[
\begin{array}{c|c}
10 & 6 \\
\end{array}
\]

\((5 < \text{all} \ 10 \ 6) = \text{true}\)

\[
\begin{array}{c|c}
5 & 4 \\
\end{array}
\]

\((5 = \text{all} \ 5 \ 4) = \text{false}\)

\[
\begin{array}{c|c}
4 & 6 \\
\end{array}
\]

\((5 \neq \text{all} \ 4 \ 6) = \text{true} \ (\text{since} \ 5 \neq 4 \ \text{and} \ 5 \neq 6)\)

\((\neq \text{all}) \equiv \text{not in}\)

However, \((= \text{all}) \neq \text{in}\)
Test for Empty Relations

- The `exists` construct returns the value `true` if the argument subquery is nonempty.
- `exists` \( r \iff r \neq \emptyset \)
- `not exists` \( r \iff r = \emptyset \)
Yet another way of specifying the query “Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester”

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

- **Correlated subquery**
- **Correlation name** or **correlation variable**
Find all students who have taken all courses offered in the Biology department.

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

Note that $X - Y = \emptyset \iff X \subseteq Y$

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

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

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

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

```sql
as dept_avg (dept_name, avg_salary)
where avg_salary > 42000;
```
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');
Delete all instructors whose salary is less than the average salary of instructors

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

- Problem: as we delete tuples from deposit, the average salary changes
- Solution used in SQL:
  1. First, compute `avg` salary and find all tuples to delete
  2. Next, delete all tuples found above
     (without recomputing `avg` or retesting the tuples)
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.)

- Add all instructors to the student relation with tot_creds set to 0
  
  ```sql
  insert into student
  select ID, name, dept_name, 0
  from instructor
  ```

- The select from where statement is evaluated fully before any of its results are inserted into the relation.
  - Otherwise queries like the following would cause problem
    
    ```sql
    insert into table1 select * from table1
    ```
Updates

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others receive a 5% raise
  - Write two update statements:
    ```sql
    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
    ```
    ```sql
    update instructor
    set salary = case
        when salary <= 100000 then salary * 1.05
        else salary * 1.03
    end
    ```
Exercise

- Increase the salary of each instructor in the Comp. Sci. department by 10%.

- Delete all courses that have never been offered (that is, do not occur in the section relation).

- Insert every student whose tot_cred attribute is greater than 100 as an instructor in the same department, with a salary of 10,000.
End of Chapter 3