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 instructors whose name includes the substring “dar”.

  select *name*
  from *instructor*
  where *name* **like** '%dar%'

- Match the string “100%”

  like ‘100 \%' **escape** '\'

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

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

SELECT column-names FROM table-name WHERE condition ORDER BY column-names

- List in alphabetic order the names of all instructors

  select distinct *name*
  from *instructor*
  order by *name*

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

String Operations (Cont.)

- The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
- •The BETWEEN operator is inclusive: begin and end values are included.
- •It is a 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

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 or signifies 'no value'.
- **is null** can be used to check for null values.
- **Is not null** can be used to check for not null values.
  - Example: Find all instructors whose salary is null.

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

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

Count(*): Returns total number of tuples

Count(column): Return number of non null values over the column

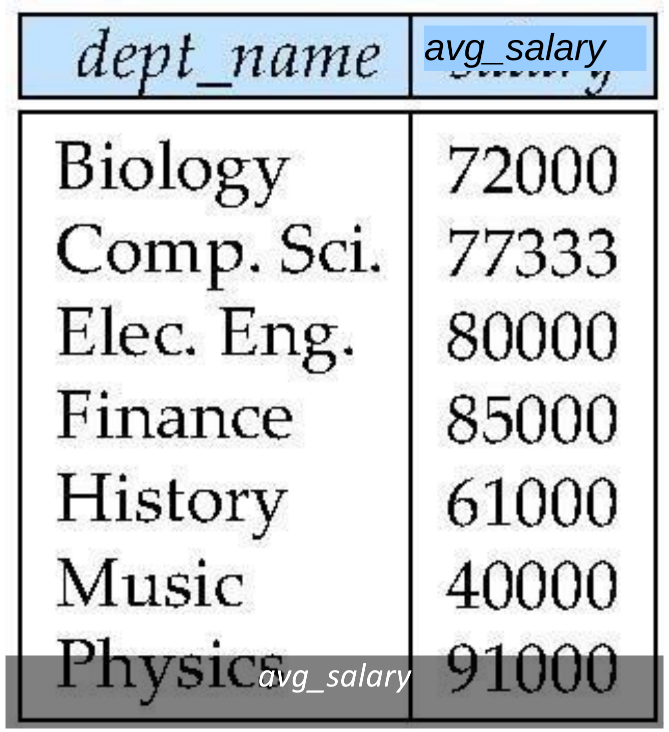
Count(Distinct column): Return number of distinct non null values over the column

Aggregate Functions (Cont.)

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

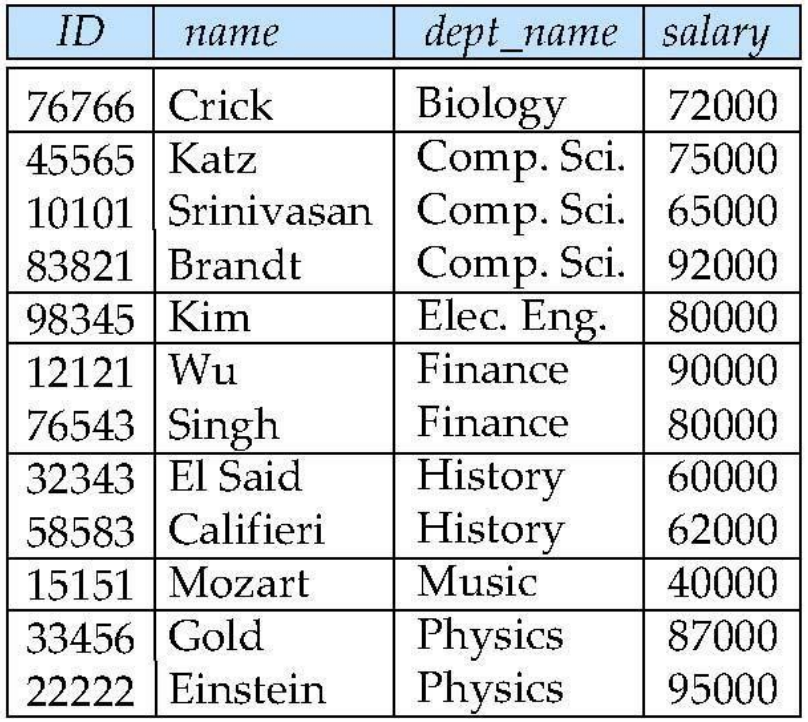
Aggregate Functions – Group By

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



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

avg_salary



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

Aggregation (Cont.)

- 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*;

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

```
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
- What if collection has only null values?
  - count returns 0
  - all other aggregates return null

CONSTRAINTS

To add check constraint:

ALTER TABLE *tablename* ADD CONSTRAINTS *constraint name* CHECK (SEX IN ('M','F’));

Example: ALTER TABLE *EMP* ADD CONSTRAINTS *SDF* CHECK (*SEX IN ('M','F'));*

To add foreign key

ALTER TABLE *tablename* ADD CONSTRAINT *constraintname* FOREIGN KEY (*attribute*) REFERENCES *Referenced_table*(*attribute*);

ALTER TABLE *MA* ADD CONSTRAINT *KL* FOREIGN KEY (*DEPID*) REFERENCES *V_DEP*(*DNO*);

To add unique key

ALTER TABLE *tablename* ADD *attributename domain* UNIQUE

ALTER TABLE *TEST* ADD *C3 INT* UNIQUE

CONSTRAINTS

The constraints from the table

SELECT *CONSTRAINT_NAME* FROM ALL_CONSTRAINTS WHERE TABLE_NAME = 'V_EMP';

To add primary key constraint

ALTER TABLE *tablename* ADD CONSTRAINT *constraint_name* PRIMARY KEY(*attribute*);

ALTER TABLE EMP ADD CONSTRAINT Con PRIMARY KEY(EMP_NO);

To drop constraint

ALTER TABLE *tablename* DROP CONSTRAINT *constraint_name*;

ALTER TABLE V_EMP DROP CONSTRAINT SYS_C0024414;

CONSTRAINTS

To Add constraint during table creation

```
create table dependents(
        DID number primary key,
        MID varchar2(20) references v_emp(emp_no),
        SEX char(1) check (sex in ('m','f'))
        NAME varchar2(20) not null);
```