CSc 484
Database Management Systems

Ken Gamradt

Spring 2024

Introduction to SQL (IV)

Relation
(instructor)

id	name	dept_name	salary
abc Filter...	abc Filter...	abc Filter...	abc Filter...
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

SELECT statement
(calculated fields)

- The SELECT clause may contain arithmetic expressions
  - Involving the operators: +, - , *, / , parentheses
    - Build complex expressions
  - Operating on constants or attributes of tuples
  - Attributes used in arithmetic expression must have a numeric type
- E.g., list the name and monthly salary of all instructors

select *name,* ***salary / 12***

from *instructor;*

name	?column?
Filter...	Filter...
Srinivasan	5416.66666666666666...
Wu	7500.00000000000000...
Mozart	3333.33333333333333...
Einstein	7916.66666666666666...
El Said	5000.00000000000000...
Gold	7250.00000000000000...
Katz	6250.00000000000000...
Califieri	5166.66666666666666...
Singh	6666.66666666666666...
Crick	6000.00000000000000...
Brandt	7666.66666666666666...
Kim	6666.66666666666666...

SELECT statement
(calculated fields)

- **Note**: arithmetic expressions on attributes **does not** result in any change to the original (input) tables(s)
- The **SELECT** operation is **closed**:
  - The result of a query on a table is another table

SELECT statement
(rename operation)

- The names of the attributes in the result are derived from the names of attributes in the relations in the **FROM** clause
- The result attribute from the arithmetic expression doesn't have a
- name
- SQL provides a way of renaming the attributes of a result relation
  - By using **AS** clause, take the form
  - old_name **AS** new_name

```
select name, salary / 12 as MonthlySalary
    from instructor;

select name, salary / 12 as 'Monthly Salary'
    from instructor;
```

name	monthlysalary
Srinivasan	5416.66666666666666…
Wu	7500.00000000000000…
Mozart	3333.33333333333333…
Einstein	7916.66666666666666…
El Said	5000.00000000000000…
Gold	7250.00000000000000…
Katz	6250.00000000000000…
Califieri	5166.66666666666666…
Singh	6666.66666666666666…
Crick	6000.00000000000000…
Brandt	7666.66666666666666…
Kim	6666.66666666666666…

SELECT statement
(AS clause – rename operation)

- Why **AS**?
  - Result attributes from an arithmetic expression doesn't have a name
  - Two relations in the from clause may have attributes with the same name
  - We may want to change the attribute name in the result

```
select name, dept_name
    from instructor;

select name as instructor_name, dept_name
    from instructor;
```

name	dept_name
Filter...	Filter...
Srinivasan	Comp. Sci.
Wu	Finance
Mozart	Music

instructor_name	dept_name
Filter...	Filter...
Srinivasan	Comp. Sci.
Wu	Finance
Mozart	Music

SELECT statement
(AS clause – rename operation)

- The **AS** clause is particularly useful in renaming relations
  - To replace a long relation name with a shortened version
    - Which is more convenient to use elsewhere in the statement
  - Can be used to compare tuples in the same relation

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

select name, course_id
    -- Table alias
    from instructor as T, teaches as S
    where T.ID = S.ID
```

SELECT statement
(ORDER BY clause – sorting results)

- In general, the rows is an SQL query result table are not arranged in any particular order
- **ORDER BY**: used to sort the result relation
  - Followed by a list of column identifiers that the result is to be sorted on
  - **ASC**: order by ascending
    - Default sorting order
  - **DESC**: order by descending

SELECT statement
(ORDER BY clause – sorting results)

- Example: list all instructors, arranged in descending order of salary

```
select *
    from instructor
    order by salary desc;
```

id	name	dept_name	salary
Filter...	Filter...	Filter...	Filter...
22222	Einstein	Physics	95000.00
83821	Brandt	Comp. Sci.	92000.00
12121	Wu	Finance	90000.00
33456	Gold	Physics	87000.00
76543	Singh	Finance	80000.00
98345	Kim	Elec. Eng.	80000.00
45565	Katz	Comp. Sci.	75000.00
76766	Crick	Biology	72000.00
10101	Srinivasan	Comp. Sci.	65000.00
58583	Califieri	History	62000.00
32343	El Said	History	60000.00
15151	Mozart	Music	40000.00

SELECT statement
(ORDER BY clause – sorting results)

- Multiple column ordering:
  - Followed columns used to order the result table with the same value for the major sort key
  - Set (**ASC** | **DESC**) separately

```
select *
    from instructor
    -- salary – major sorting key
    -- name – minor sorting key
    order by salary desc, name;
```

id	name	dept_name	salary
Filter...	Filter...	Filter...	Filter...
22222	Einstein	Physics	95000.00
83821	Brandt	Comp. Sci.	92000.00
12121	Wu	Finance	90000.00
33456	Gold	Physics	87000.00
98345	Kim	Elec. Eng.	80000.00
76543	Singh	Finance	80000.00
45565	Katz	Comp. Sci.	75000.00
76766	Crick	Biology	72000.00
10101	Srinivasan	Comp. Sci.	65000.00
58583	Califieri	History	62000.00
32343	El Said	History	60000.00
15151	Mozart	Music	40000.00

SELECT statement
(ORDER BY)

- **Null** value in order by:
  - ISO standard specifies to treat it either less than all non-null values, or greater than all non-null values
  - The choice is left to the DBMS implementor
    - **MySQL**, **SQL Server**: treat the null value **less than** all nonnull value
    - **PostgreSQL**: treat the null values **greater than** all nonnull value

SELECT statement (ORDER BY)

- **Null** value in order by

```
insert into instructor      -- PostgreSQL
    values ('11111', 'Hanks', 'Music', null);

select *
    from instructor
    order by salary;
```

id	name	dept_name	salary
15151	Mozart	Music	40000.00
32343	El Said	History	60000.00
58583	Califieri	History	62000.00
10101	Srinivasan	Comp. Sci.	65000.00
76766	Crick	Biology	72000.00
45565	Katz	Comp. Sci.	75000.00
98345	Kim	Elec. Eng.	80000.00
76543	Singh	Finance	80000.00
33456	Gold	Physics	87000.00
12121	Wu	Finance	90000.00
83821	Brandt	Comp. Sci.	92000.00
22222	Einstein	Physics	95000.00
11111	Hanks	Music	NULL

SELECT statement (ORDER BY)

- **Null** value in order by

```
insert into instructor      -- MySQL
    values ('11111', 'Hanks', 'Music', null);

select *
    from instructor
    order by salary;
```

ID	name	dept_name	salary
11111	Hanks	Music	NULL
15151	Mozart	Music	40000.00
32343	El Said	History	60000.00
58583	Califieri	History	62000.00
10101	Srinivasan	Comp. Sci.	65000.00
76766	Crick	Biology	72000.00
45565	Katz	Comp. Sci.	75000.00
76543	Singh	Finance	80000.00
98345	Kim	Elec. Eng.	80000.00
33456	Gold	Physics	87000.00
12121	Wu	Finance	90000.00
83821	Brandt	Comp. Sci.	92000.00
22222	Einstein	Physics	95000.00

SELECT statement
(aggregate functions)

- **Green** – stored in the database
- **Red** – desired information

Name	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	Average
Hans	301	300	299	299	296	294	293	301	295				**298**
Henry	683	689	686	697	717	707	714	717	707				**702**
John	72	71	51	52	51	49	49	53	52				**56**
Mary	97	93	94	94	95	95	93	97	95				**95**
Messi	6	20	40	42	43	46	48	49	52				**38**
Rolando	13	13	14	15	15	15	14	15	15				**14**
Sam	252	251	249	251	254	259	262	260	266				**256**
Samuel	160	153	152	153	159	160	160	157	159				**157**
Zaltan	48	47	47	51	48	49	49	51	54				**49**
Total	**1632**	**1637**	**1632**	**1654**	**1678**	**1674**	**1682**	**1700**	**1695**				**185**

SELECT statement
(aggregate functions)

- Takes a collection of values (single column of a table) as input and returns a single value
- ISO standard defines 5 aggregate functions
  - **AVG** (average): return the average of the values in a specified column (*)
  - **SUM** (total): return the sum of the values in a specified column (*)
  - **MIN** (minimum): return the smallest value in a specified column (+)
  - **MAX** (maximum): return the largest value in a specified column (+)
  - **COUNT** (count): return the number of values in a specified column (+)

- (*) used for numeric values
- (+) used for both numeric and non-numeric values

SELECT statement
(aggregate functions)

- Example 1: find the average salary of all the instructors

```
select AVG(salary)
    from instructor;
```

	(No column name)
1	74833.333333

- Example 2: find the average salary of instructors in the Computer Science dept

```
select AVG(salary)
    from instructor
    where dept_name = 'Comp. Sci.';
```

	(No column name)
1	77333.333333

- The result of this query is a relation with a single attribute containing a single tuple with a numerical value corresponding to the aggregation functions
  - The DB system may provide an awkward name to it

SELECT statement
(aggregate functions)

- Give a meaningful name to the attribute of result relation generated by aggregate function by using **AS** clause

```
select AVG (salary) as averageSalary
    from instructor
    where dept_name = 'Comp. Sci.';
```

	averageSalary
1	74250.000000

SELECT statement
(aggregate functions)

- Example

```
select
    COUNT (ID)      as totalNum,
    SUM   (salary) as sumSalary,
    AVG   (salary) as averageSalary,
    MIN   (salary) as minSalary,
    MAX   (salary) as maxSalary
    from instructor
    where dept_name = 'Comp. Sci.';
```

	totalNum	sumSalary	averageSalary	minSalary	maxSalary
1	4	297000.00	74250.000000	65000.00	92000.00

SELECT statement
(aggregate functions)

- If the column include null value
  - **COUNT**(*): special use of **COUNT** that counts all the rows of a table, regardless of whether nulls or duplicate values occur
  - Other functions eliminate nulls first and operate only on the remaining nonnull values

```
insert into instructor
    values ('10002', 'Gamradt', 'Comp. Sci', null);
select COUNT (*) as countAll
    from instructor
    where dept_name = 'Comp. Sci.';
select COUNT (salary) as countSalary
    from instructor
    where dept_name = 'Comp. Sci.';
```

	countAll
1	5

	countSalary
1	4

SELECT statement
(aggregate functions)

- Example

```
insert into instructor
    values ('10002', 'Gamradt', 'Comp. Sci', null);
select
    COUNT (ID)     as totalNum,
    SUM   (salary) as sumSalary,
    AVG   (salary) as averageSalary,
    MIN   (salary) as minSalary,
    MAX   (salary) as maxSalary
    from  instructor
    where dept_name = 'Comp. Sci.';
```

	totalNum	sumSalary	averageSalary	minSalary	maxSalary
1	5	297000.00	74250.000000	65000.00	92000.00

SELECT statement
(aggregate functions)

- Use of **COUNT(DISTINCT)**
- E.g., find the total number of instructors who teach a course in the Spring 2018 semester

```
select *
    from teaches
    where semester = 'Spring' and year = '2018';

select COUNT (distinct ID) as totalNum
    from teaches
    where semester = 'Spring' and year = '2018';
```

	ID	course_id	sec_id	semester	year
1	10101	CS-315	1	Spring	2018
2	12121	FIN-201	1	Spring	2018
3	15151	MU-199	1	Spring	2018
4	32343	HIS-351	1	Spring	2018
5	45565	CS-101	1	Spring	2018
6	45565	CS-319	1	Spring	2018
7	83821	CS-319	2	Spring	2018

	totalNum
1	6

- SQL does not allow the use of DISTINCT with COUNT(*)

Relation
(teaches)

id	course_id	sec_id	semester	year
abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...
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

SELECT statement
(aggregation with grouping)

- Example: find the average salary in each department?
- **GROUP BY**:
  - The attributes given in the **GROUP BY** clause are used to form groups
  - Tuples with the same value on all attributes in the **GROUP BY** clause are placed in one group

```
select [distinct | all] {* | [columnExpression [as newName]] [,...]}
    from TableName [alias] [,...]
    [where condition]
    -- Grouping columns
    [group by columnList] [having condition]
    [order by columnList]
```

SELECT statement
(aggregation with grouping)

- Example: find the average salary in each department?

```
select dept_name, AVG (salary) as averageSalary
    from instructor
    group by dept_name;
```

dept_name	averagesalary
abc Filter...	abc Filter...
Finance	85000.000000000000
History	61000.000000000000
Physics	91000.000000000000
Music	40000.000000000000
Comp. Sci.	77333.333333333333
Biology	72000.000000000000
Elec. Eng.	80000.000000000000

SELECT statement
(aggregation with grouping)

- SELECT and GROUP BY are closely integrated
  - Attributes appearing in SELECT without being aggregated must appear in GROUP BY – the contrary is not true
  - Each item in SELECT must be single-valued per group

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

Column 'instructor.ID' is invalid in the select list because it is not contained within either an aggregate function or a GROUP BY clause

- *If WHERE is used with GROUP BY, WHERE is applied first, GROUP BY second*

SELECT statement
(aggregation with grouping)

```
select dept_name,
    AVG (salary) as averageSalary
    from instructor
    group by dept_name;
```

id	name	dept_name	salary
76766	Crick	Biology	72000.00
10101	Srinivasan	Comp. Sci.	65000.00
45565	Katz	Comp. Sci.	75000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00
12121	Wu	Finance	90000.00
76543	Singh	Finance	80000.00
32343	El Said	History	60000.00
58583	Califieri	History	62000.00
15151	Mozart	Music	40000.00
33456	Gold	Physics	87000.00
22222	Einstein	Physics	95000.00

- Step 1: get the instructor relation
- Step 2: divide the instructors into groups according to their dept_name
- Step 3: for each group, compute the average salary, and generate **a single row** in the result for each group

dept_name	averagesalary
Finance	85000.000000000000
History	61000.000000000000
Physics	91000.000000000000
Music	40000.000000000000
Comp. Sci.	77333.333333333333
Biology	72000.000000000000
Elec. Eng.	80000.000000000000

Relation
(section)

course_id	sec_id	semester	year	building	room_number	time_slot_id
abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...	abc Filter...
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

SELECT statement
(aggregation with grouping)

- Example: list the number of courses provided each semester in 2017

```
select semester, year, COUNT(distinct course_id) as countCourse
    from section
    where year = 2017
    group by semester, year;
```

semester	year	countcourse
abc Filter...	abc Filter...	abc Filter...
Fall	2017	3
Spring	2017	2
Summer	2017	1

SELECT statement
(HAVING clause)

- Designed for use with the **GROUP BY** clause to restrict the groups that appear in the final result table
  - E.g., find the average salary in departments where the average salary of the instructors is more than $42,000

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

dept_name	avg
abc Filter...	abc Filter...
Finance	85000.000000000000
History	61000.000000000000
Physics	91000.000000000000
Comp. Sci.	77333.333333333333
Biology	72000.000000000000
Elec. Eng.	80000.000000000000

SELECT statement
(HAVING clause)

- **HAVING** vs. **WHERE**
  - **WHERE** clause filters individual rows going into the final result relation
  - **HAVING** clause filters groups into the final relation
- Column names used in the **having** clause must also appear in the **GROUP BY** clause, or be contained within an aggregate function
- In practice, the search condition in the **having** clause always includes at least one aggregate function
  - Otherwise, the search condition can be moved to the **where** clause and applied to the individual rows

SELECT statement
(HAVING clause)

- The sequence of operations of queries containing aggregation, **group by**, or **having** clause
  - The **FROM** clause is first evaluated to get a relation
  - If the **WHERE** clause is present, it trims down the data set based on the specified logic/predicate
  - Tuples are placed in groups based on **GROUP BY** clause if present, otherwise treat the relation as one group
  - Apply the **HAVING** clause to each group
  - The **SELECT** clause applies the aggregate functions to obtain a single resulting tuple for each group
  - If **ORDER BY** is present, it merely formats the output

SELECT statement
(aggregate function)

- Aggregation and group by with **null**
  - All aggregate functions ignore **null** value in their input collection
    - Except count(*)
  - The collection of values can be empty
    - The count of an empty collection is defined to be 0
    - All other aggregate operations return a **null** value when operating on an empty collection

SELECT statement
(aggregation operation)

- Example: information for the math department instructors is added with the salary being unknown
  - List the average salary in each department

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

3	10010	Cogswell	Math	NULL
4	10011	Flint	Math	NULL
5	10012	Vestal	Math	NULL
6	10013	Abraham	Math	NULL
7	10101	Srinivasan	Comp. Sci.	65000.00

	dept_name	(No column name)
1	Biology	72000.000000
2	Comp. Sci.	74250.000000
3	Elec. Eng.	80000.000000
4	Finance	85000.000000
5	History	61000.000000
6	Math	NULL
7	Music	40000.000000
8	Physics	91000.000000

Acknowledgements

- WIKIPEDIA
  - https://en.wikipedia.org/wiki/SQL