3.3 Basic Structure of SQL Queries

Database System Concepts
Abraham Silberschatz, Henry F. Korth, S. Sudarshan

Content

1. Basic Clauses
2. Example Queries using 1 Relation
3. Example Queries using Multiple Relations
4. Cartesian Product
5. Summary

Typical SQL Query

$$\begin{aligned}&\textbf{select } A_1,\ A_2,\ \ldots,\ A_n\\&\textbf{from } r_1,\ r_2,\ \ldots,\ r_m\\&\textbf{where } P;\end{aligned}$$

- Ai represents an attribute (columns)
- Ri represents a relation (tables)
- P is a predicate (conditions)

Clauses: Select, From, Where

- From: specify the relations to be used for evaluation of the query
- Select: specify the attributes desired in the results of the query
- Where: specifies predicates involving attributes of relations (narrowing down results)

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

Figure 2.1 The *instructor* relation.

Queries on a Single Relation

Example: “Find the names of all instructors”

SQL Query:

```
select name
from instructor;
```

name
Srinivasan
Wu
Mozart
Einstein
El Said
Gold
Katz
Califieri
Singh
Crick
Brandt
Kim

Figure 3.2 Result of “**select** *name* **from** *instructor*”.

Queries on a Single Relation

Example: “Find the department names of all instructors”

SQL Query:

```
select dept_name
from instructor;
```

SQL Query:

```
select distinct dept_name
from instructor;
```

SQL allows duplicates in database relations and results of SQL queries

dept_name
Comp. Sci.
Finance
Music
Physics
History
Physics
Comp. Sci.
History
Finance
Biology
Comp. Sci.
Elec. Eng.

Figure 3.3 Result of “**select** *dept_name* **from** *instructor*”.

Select: Arithmetic Expressions (+, -, *, /)

Example: “Find the expected salary of each instructor after a 10% raise.

SQL Query:

```
select name, salary*1.1
from instructor;
```

This does NOT make changes to the instructor relation

Where: Clause Predicates

name
Katz
Brandt

Example: “Find the names of all instructors in the CS department who have a salary greater than $70,000”

SQL Query:

```
select name
from instructor
where dept_name='Comp. Sci' and salary>70000;
```

Predicates can use Comparison Operators (<, <=, >, >= , = , <>) and Logical Connectives (and, or , not)

Queries on Multiple Relations

dept_name	building	budget
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

Figure 2.5 The *department* relation.

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

Figure 2.1 The *instructor* relation.

Example: “Find the names of all instructors, along with their department names and department building names”

SQL Query:

```
select name, instructor.dept_name, building
from instructor, department
where instructor.dept_name=department.dept_name;
```

name	dept_name	building
Srinivasan	Comp. Sci.	Taylor
Wu	Finance	Painter
Mozart	Music	Packard
Einstein	Physics	Watson
El Said	History	Painter
Gold	Physics	Watson
Katz	Comp. Sci.	Taylor
Califieri	History	Painter
Singh	Finance	Painter
Crick	Biology	Watson
Brandt	Comp. Sci.	Taylor
Kim	Elec. Eng.	Taylor

Figure 3.5 The result of “Retrieve the names of all instructors, along with their department names and department building name.”

Cartesian Product (multiple relations)

Definition: “An iterative process that generates tuples for the resulting relation of the **from** clause”

from instructor, teaches;

↓

instructor.ID, instructor.name, instructor.dept name, instructor.salary, teaches.ID, teaches.course id, teaches.sec id, teaches.semester, teaches.year

↓

Each tuple from instructor is combined with *every* tuple from teaches, and vice versa.

ID	*course_id*	*sec_id*	*semester*	*year*
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

Figure 2.7 The *teaches* relation.

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

Figure 2.1 The *instructor* relation.

Cartesian Product

The Cartesian Product combines ALL tuples from instructor and teaches, even those that are unrelated and refer to different instructors.

Use **where** to only combine tuples that have the same ID value.

```
from instructors, teaches
where instructors.id=teaches.id;
```

instructor.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2017
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2018
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2018
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2017
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2018
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2017
15151	Mozart	Music	40000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
15151	Mozart	Music	40000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2017
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2018
22222	Einstein	Physics	95000	10101	CS-347	1	Fall	2017
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2018
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Figure 3.6 The Cartesian product of the *instructor* relation with the *teaches* relation.

Summary of Understanding SQL Queries

1. Generate a Cartesian Product of all the relations listed in the **from** clause.
2. Apply predicates specified in the **where** clause onto the results of step 1.
3. For each tuple in the results of step 2, output the attributes (or arithmetic expressions) that are specified in the **select** clause.

$$\begin{aligned}&\textbf{select } A_1,\ A_2,\ \ldots,\ A_n\\&\textbf{from } r_1,\ r_2,\ \ldots,\ r_m\\&\textbf{where } P;\end{aligned}$$