

Introduction to Database Systems: CS312

Advanced queries, joins and aggregates

Oliver Bonham-Carter

13 Feb 2019

Joins: Bringing Data Together

Joins

Terms
Cross Joins
Inner Join
Inner Joins

New
Database

Sets

Renaming
Attributes

Strings

Ordering



- The SQLite3 join-clause is used to combine records from two or more tables in a database.
- A **JOIN** is a means for combining fields from two tables by using values common to each.

Joins: Visual Definitions

Combining Tables

Joins

Terms
Cross Joins
Inner Join
Inner Joins

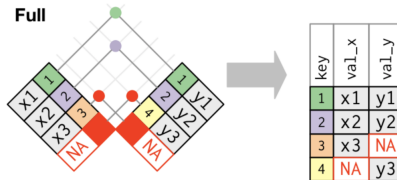
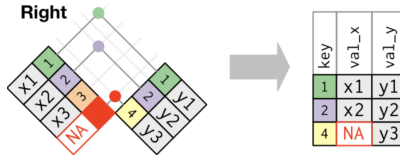
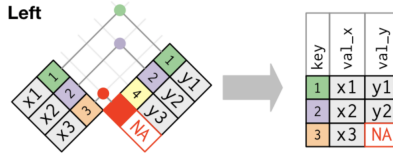
New
Database

Sets

Renaming
Attributes

Strings

Ordering



SQL Code and Venn Diagrams

Joins

Terms

Cross Joins
Inner Join
Inner Joins

New

Database

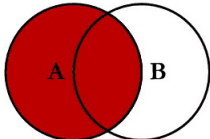
Sets

Renaming Attributes

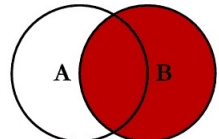
Strings

Ordering

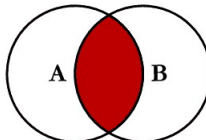
SQL JOINS



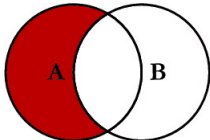
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



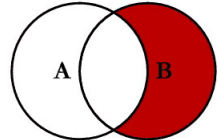
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



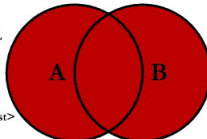
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



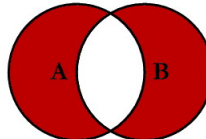
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

An explanation of terms

Joins

Terms

Cross Joins

Inner Join

Inner Joins

New

Database

Sets

Renaming
Attributes

Strings

Ordering

- SQL defines three major types of joins
 - The **CROSS JOIN**: Matches every row of the first table with every row of the second table. If the input tables have x and y columns, respectively, the resulting table will have $x * y$ columns.
 - The **INNER JOIN**: Creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate.

Joins

Cross joins

Joins

Terms

Cross Joins

Inner Join

Inner Joins

New

Database

Sets

Renaming

Attributes

Strings

Ordering

- Cross join: *SELECT ... FROM table1 CROSS JOIN table2 ...*
- Automatically testing for equality between the values of every column that exists in both tables

A practical example: Build a matrix of cards

```
CREATE TABLE ranks (  
    rank TEXT NOT NULL  
);  
  
CREATE TABLE suits (  
    suit TEXT NOT NULL  
);  
  
INSERT INTO ranks(rank)  
VALUES('2'),('3'),('4'),('5'),('6'),('7'),('8'),('9'),('10'),('J'),('Q'),('K'),('A');  
  
INSERT INTO suits(suit) VALUES('Clubs'),('Diamonds'),('Hearts'),('Spades');  
  
SELECT rank, suit  
FROM ranks  
    CROSS JOIN  
    suits  
ORDER BY suit;
```

Joins

Inner join

Joins

Terms

Cross Joins

Inner Join

Inner Joins

New

Database

Sets

Renaming
Attributes

Strings

Ordering

A

m	f
a1	1
a2	2
a3	3

```
SELECT m, A.f, B.f, n
FROM A
INNER JOIN B ON B.f = A.f
```

n	f
b1	1
b2	3
b3	5

B

m	A.f	B.f	n
a1	1	1	b1
a3	3	3	b2

- Inner join: *SELECT ... FROM table1 [INNER] JOIN table2 ON conditional_expression ...*
- Combines column values of two tables (table1 and table2) based upon the join-predicate

Create TableA and TableB

```
drop table tableA;  
create table tableA (  
    num VARCHAR,  
    letter VARCHAR);
```

```
drop table tableB;  
create table tableB (  
    num VARCHAR,  
    letter VARCHAR);
```


Joins

Terms

Cross Joins

Inner Join

Inner Joins

New

Database

Sets

Renaming
Attributes

Strings

Ordering

Populate your TableA and TableB

```
INSERT INTO tableA VALUES (1,"a");  
INSERT INTO tableA VALUES (2,"b");  
INSERT INTO tableA VALUES (4,"d");  
INSERT INTO tableA VALUES (6,"f");
```

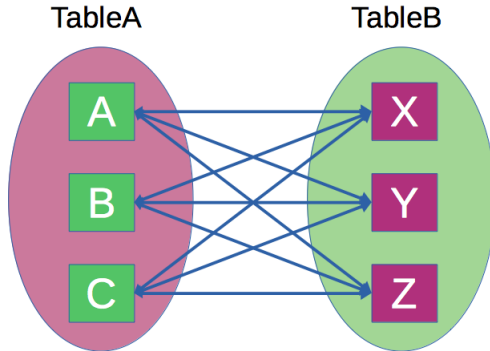
```
INSERT INTO tableB VALUES (1,"a");  
INSERT INTO tableB VALUES (2,"b");  
INSERT INTO tableB VALUES (3,"c");  
INSERT INTO tableB VALUES (5,"e");  
INSERT INTO tableB VALUES (6,"f");  
INSERT INTO tableB VALUES (7,"g");  
INSERT INTO tableB VALUES (8,"h");  
INSERT INTO tableB VALUES (9,"i");  
INSERT INTO tableB VALUES (1,"a");
```

Joins

CROSS JOIN

Joins
Terms
Cross Joins
Inner Join
Inner Joins

New
Database
Sets
Renaming
Attributes
Strings
Ordering



`SELECT * FROM tableA CROSS JOIN tableB`

```
select * from TableA CROSS Join TableB;
select * from tableA, TableB;
```

Inner Joins

Joins
Terms
Cross Joins
Inner Join
Inner Joins

New
Database

Sets

Renaming
Attributes

Strings

Ordering

Left

```
/* inner (left) join */  
SELECT tableA.num  
FROM tableB  
LEFT JOIN tableA  
ON tableA.num == tableB.num;
```

Right

```
/* inner (right) join */  
SELECT tableB.num  
FROM tableA  
LEFT JOIN tableB  
ON tableA.num == tableB.num;
```

- How many spaces did you count from each query?
- What do the spaces tell you?

Joins

New
Database

Sets

Renaming
Attributes

Strings

Ordering



(A New Database!)



Schema: Red boxes are the tables of today's database study

Schema: Red boxes are the tables of today's database study

New Database

Renaming Attributes

Ordering

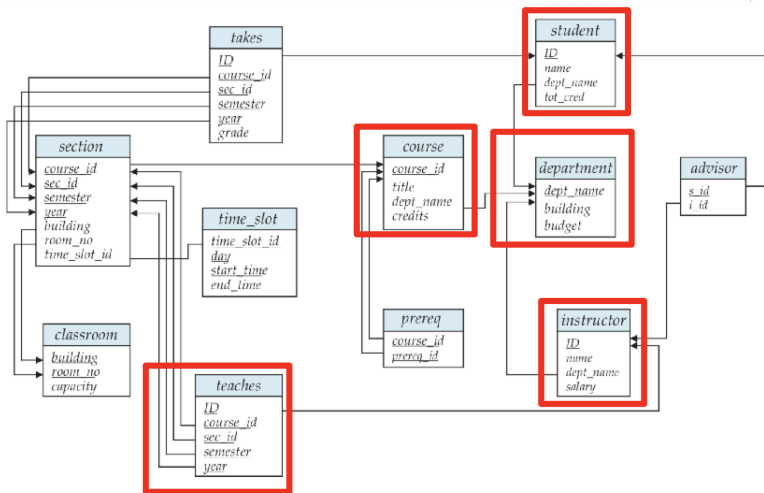


Figure 2.8 Schema diagram for the university database.

New database

Joins

New
Database

Sets

Renaming
Attributes

Strings

Ordering

- Find the database maker file, *campusDB_build.txt*, in your sandbox directory

```
cat campusDB_build.txt | sqlite3 myCampusDB.sqlite3
```

Set operations

Joins

New
Database

Sets

Renaming
Attributes

Strings

Ordering

- Find all deptNames common to both the Instructor and Course

- `select deptName from Instructor UNION select deptName from course;`
- `select distinct(deptName) from Instructor;`

- Find all deptNames common to both the Instructor and Course

- `select deptName from Instructor INTERSECT select deptName from Course;`
- `select distinct(Instructor.deptName) from Instructor, Course where Instructor.deptName == Course.deptName;`

Set operations

Joins

New
Database

Sets

Renaming
Attributes

Strings

Ordering

- The EXCEPT operator compares the result sets of two queries and returns distinct rows from the left query that are not in the output by the right query.
- Find all deptNames different to both the *Instructor* and *Course*
- Check these two queries below. Why is the output different?

- `select deptName from Instructor EXCEPT select deptName from Course;`
- `select deptName from Course EXCEPT select deptName from Instructor;`

Renaming an attribute

Joins

New
Database

Sets

Renaming
Attributes

Strings

Ordering

- SQL allows renaming relations and attributes using the **as** clause: *old-name as new-name*

- `select distinct(Instructor.name) as newName from Instructor, teaches where Instructor.ID = teaches.ID and newName == "Thompson";`

- Find the names of all Instructors whose salary is greater than at least one Instructor in the Biology department.
- `select distinct(T.name) from Instructor as T, Instructor as S where T.salary > S.salary and S.deptName == "Math";`
- `select distinct T.name, T.salary from Instructor as T, Instructor as S where T.salary > S.salary and S.deptName == "Math";`
- Compare to: `select * from Instructor;`

Regular Expression-ish

Joins

New
Database

Sets

Renaming
Attributes

Strings

Ordering

- Textual *wildcards* to recover information from partial knowledge.
- Finding substrings using the % and _ operators.

- `select name from Instructor where name like "%ille%";`
 - Selects *Miller* from a substring
- `select name from Instructor where name like "%son";`
 - Selects all names followed by "son" substring
- Compare to: *Select * from Instructor;*
- `select name from Instructor where name like "__ll__";`
- `select name from Instructor where name like "__ll___";`
 - Selects "Miller" or "William" from the number of spaces after the "ll";.

Regular Expression-ish

Joins

New
Database

Sets

Renaming
Attributes

Strings

Ordering

- Find special pattern characters (i.e., "%" and "_") in strings
- SQL even allows the specification of an escape character.

- **like 'ab\%cd%' escape '\'** matches all strings beginning with "ab%cd".
- **like 'ab\\cd%' escape '\'** matches all strings beginning with "ab\cd".

Ordering results

Joins

New
Database

Sets

Renaming
Attributes

Strings

Ordering

- SQL allows for sorting the output.
- Output is sorted alphabetically

- `select name from Instructor order by name;`
- `select name,salary from Instructor order by salary;`
 - Provides numerical values in an interval

Greater than, less than

Joins

New
Database

Sets

Renaming
Attributes

Strings

Ordering

- `select * from Instructor group by deptName HAVING salary < 100000;`
- `select deptName, avg(salary) from Instructor group by deptName;`
 - Groups deptName attributes by their average salaries.

Ordering result using BETWEEN

Joins

New
Database

Sets

Renaming
Attributes

Strings

Ordering

- SQL allows for sorting the output by criteria
- Output is sorted for values in an interval

- `select name, salary from Instructor where salary <= 100000 and salary >= 90000;`
- `select name, salary from Instructor where salary between 70000 and 100000;`
 - Query values in their intervals.

“Intermediate” results using **HAVING**

Joins

New
Database

Sets

Renaming
Attributes

Strings

Ordering

- The **HAVING** clause enables you to specify conditions that filter which group results appear in the final results.
- The **HAVING** clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used.

Pseudo-code

```
SELECT column1, column2  
FROM table1, table2  
WHERE [ conditions ]  
GROUP BY column1, column2  
HAVING [ conditions ]  
ORDER BY column1, column2
```



Greater than or equal to, less than...

Joins

New
Database

Sets

Renaming
Attributes

Strings

Ordering

- `select name, deptName, salary from Instructor group by deptName having avg(salary) >= (select avg(salary) from Instructor group by deptName);`
 - `select avg(salary), deptName from Instructor group by deptName;`
- `select name, deptName, salary from Instructor group by deptName having avg(salary) >= (select avg(salary) from Instructor group by deptName);`
 - Query salaries which are above or below average for their departments.

Use avg to query

Joins

New
Database

Sets

Renaming
Attributes

Strings

Ordering

- `select deptName, avg(salary) as avgSalary from Instructor group by deptName;`
 - Report average salaries for departments
- `select deptName, avgSalary from (select deptName, avg(salary) as avgSalary from Instructor group by deptName) where avgSalary > 97000;`
 - This query is similar to one using the **HAVING** clause. Here we use the **FROM** clause.