Advanced queries, joins and aggregates

CMPSC 305 – Database Systems

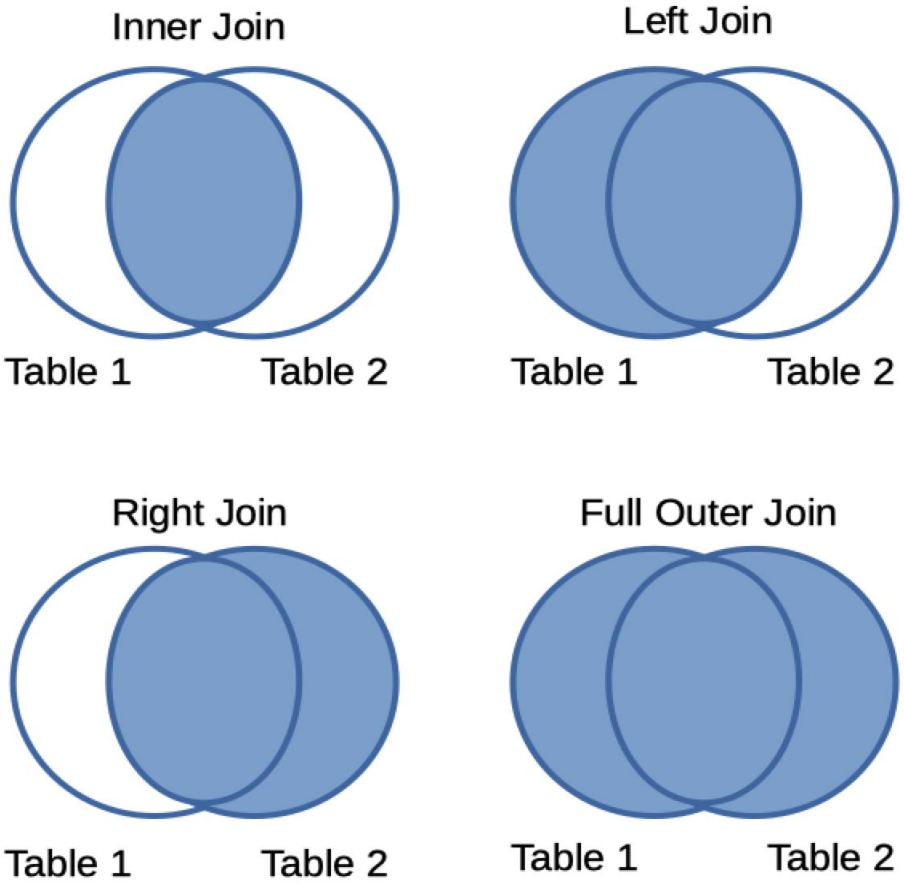
ALLEGHENY COLLEGE

Joins: Bringing Data Together

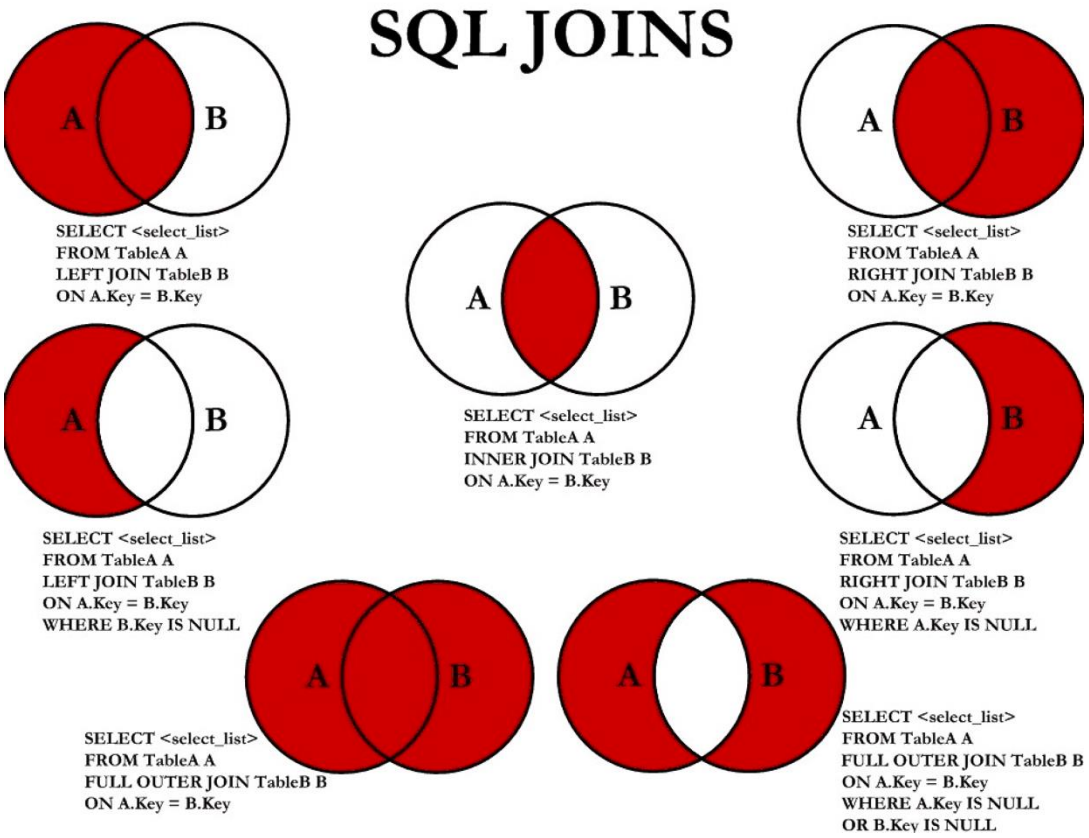
- The SQLite3 join-clause is used to combine records from two or more tables in a database.
- A JOIN is a means for combining fields from two tables by using values common to each.

Joins: Visual Definitions

As Venn Diagrams



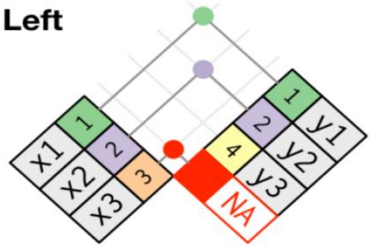
SQL Code and Venn Diagrams



Joins: Visual Definitions

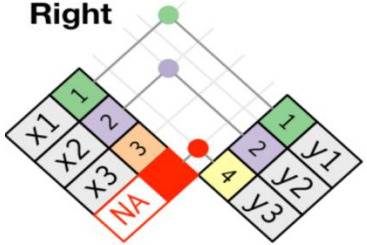
Combining Tables

Left



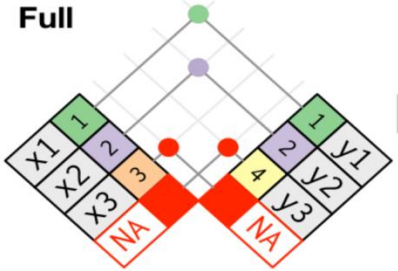
key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

Right



key	val_x	val_y
1	x1	y1
2	x2	y2
4	NA	y3

Full



key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

An Explanation of Terms

SQL joins

- An inner join will return records that have matching values in both tables.
- A left outer join will return all records from the left table and the matched records from the right table.
- A right outer join will return all records from the right table and the matched records from the left table.
- A full outer join will return all records when there is a match from either table.

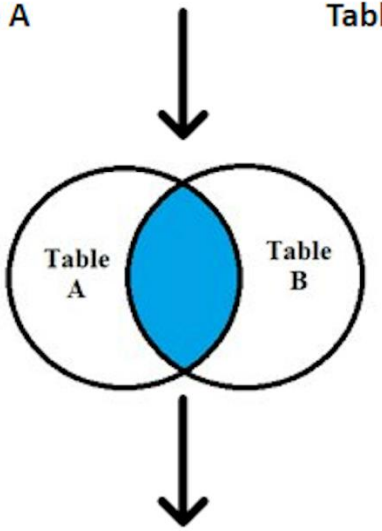
Inner Joins

Student ID	Name
1001	A
1002	B
1003	C
1004	D

Table A

Student ID	Department
1004	Mathematics
1005	Mathematics
1006	History
1007	Physics
1008	Computer Science

Table B



Student ID	Name	Department
1004	D	Mathematics

Inner Joins

File: /sandbox/fruitJoin.txt

Create two tables

```
DROP TABLE IF EXISTS TableA;
CREATE TABLE TableA (
fruit VARCHAR,
colour VARCHAR);

DROP TABLE IF EXISTS TableB;
CREATE TABLE TableB (
fruit VARCHAR,
colour VARCHAR);
```

Inner joins

File: /sandbox/fruitJoin.txt

Populate the tables

```
INSERT INTO TableA VALUES ("Lemons_A","Yellow");
INSERT INTO TableA VALUES ("Apples_A","Red");
INSERT INTO TableA VALUES ("Grapes_A","Purple");

INSERT INTO TableB VALUES ("Lemons_B","Yellow");
INSERT INTO TableB VALUES ("Apples_B","Red");
INSERT INTO TableB VALUES ("Oranges_B", "Orange");
INSERT INTO TableB VALUES ("Grapes_B","Purple");
```

Inner joins

File: /sandbox/fruit– innerJoin.txt

Use INNER JOIN to query

```
.tables

SELECT * from TableA;
SELECT* from TableB;

SELECT
        TableA.fruit,
        TableA.colour,
        TableB.colour,
        TableB.fruit
FROM
        TableA
INNER JOIN
        TableB ON TableB.colour == TableA.colour;
```

Inner joins

Output

```
Lemons_A|Yellow|Yellow|Lemons_B
Apples_A|Red|Red|Apples_B
Grapes_A|Purple|Purple|Grapes_B
```

Where vs Inner Joins

```
SELECT
        TableA.fruit,
        TableA.colour,
        TableB.colour,
        TableB.fruit
FROM
        TableA, TableB
WhERE
        TableB.colour = TableA.colour;
```

Left Join

Matches entries from LEFT table to the other table

Setup Tables

```
DROP TABLE IF EXISTS Employees;

CREATE TABLE Employees (
        EmployeeID INT PRIMARY KEY, LastName VARCHAR, DepartmentID INT,
        FirstName VARCHAR
);

DROP TABLE IF EXISTS Departments;

CREATE TABLE Departments (
        DepartmentID INT PRIMARY KEY, DepartmentName VARCHAR
);
```

Left Join

```
INSERT INTO Employees (
        EmployeeID, FirstName,
        LastName, DepartmentID)
VALUES
        (1, 'John', 'Doe', 1),
        (2, 'Jane', 'Smith', 2),
        (3, 'Bob', 'Johnson', 1),
        (4, 'Alice', 'Williams', NULL);

INSERT INTO Departments (DepartmentID, DepartmentName)
VALUES
        (1, 'HR'),
        (2, 'IT'),
        (3, 'Finance');
```

Left Join

```
/*Perform a LEFT JOIN to retrieve a list of all employees and their departments*/
SELECT
        e.EmployeeID, e.FirstName, e.LastName, d.DepartmentName
FROM
        Employees e
LEFT JOIN
        Departments d
ON
        e.DepartmentID = d.DepartmentID;
```

```
EmployeeID | FirstName | LastName | DepartmentName
-----------------------------------------------
1       |      John |      Doe |       HR
2       |      Jane |    Smith |       IT
3       |       Bob |  Johnson |       HR
4       |     Alice | Williams |       NULL
```

Right Join

Matches entries from RIGHT table to the other table

```
/*Perform a RIGHT JOIN to retrieve a list of all departments, even if they have no employees.*/
SELECT
        e.EmployeeID, e.FirstName, e.LastName, d.DepartmentName
FROM
        Employees e
RIGHT JOIN
        Departments d
ON
        e.DepartmentID = d.DepartmentID;
```

```
EmployeeID | FirstName | LastName | DepartmentName
-------------------------------------------
1          |      John |      Doe |     HR
2          |      Jane |    Smith |     IT
3          |       Bob |  Johnson |     HR
NULL       |      NULL |     NULL |     Finance
```

Cross joins

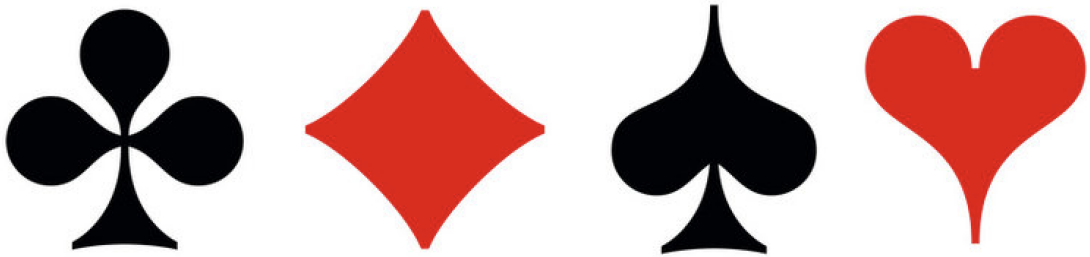
Cross product derivied from both tables

```
DROP TABLE IF EXISTS ranks;
CREATE TABLE ranks (
                rank TEXT NOT NULL
);
DROP TABLE IF EXISTS suits;
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;
```

Cross joins: All Card Pairs

Cross join: output

```
2|Clubs
...
J|Clubs
Q|Clubs
K|Clubs
A|Clubs
...
A|Spades
```



New Database

(A New Database!)

New Database

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

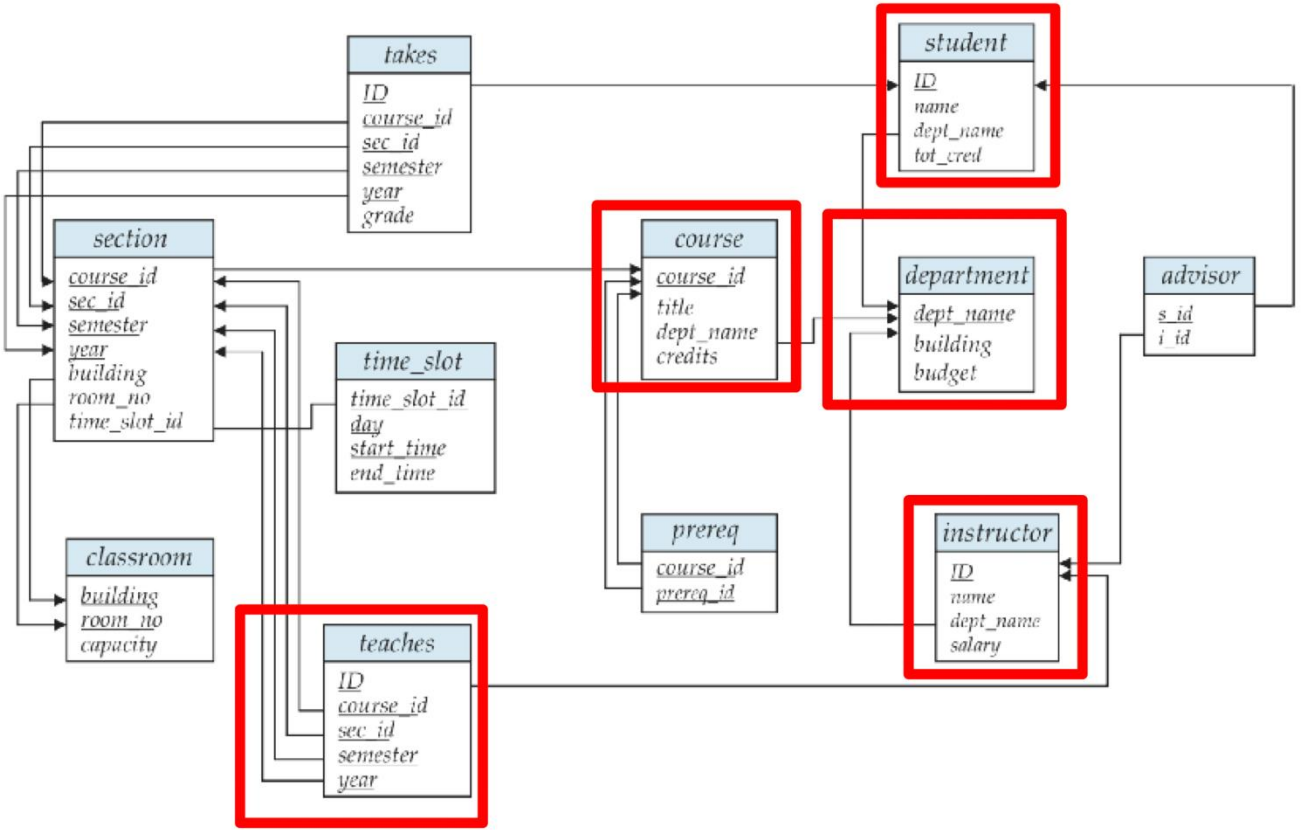


Figure 2.8 Schema diagram for the university database.

New Database

- Find the database maker file, campusDB build.txt, in your sandbox directory

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

Set Operations
OR & AND

- OR: Find all deptNames in the UNION of Instructor and Course

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

- AND: Find all deptNames in the INTERSECT of 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

- select distinct(deptName) from Instructor;
- select distinct(deptName) from Course;

- 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;

AS clauses

- The AS clause is used to rename relations; useful for reducing necessary code in queries
- Ex: For all instructors in the university who have taught some course, find their names and the course ID of all their taught courses

```
Select I.name, T.courseID
FROM Instructor AS I , Teaches AS T
WHERE I.ID= T.ID;
```

- On the second line:
- the Instructor table is renamed to I
- the Teaches table is renamed to T.

AS clauses

- Another reason to rename a relation is a case where we wish to compare tuples in the same relation.
- We then need to take the Cartesian product of a relation with itself and, without renaming, it becomes impossible to distinguish one tuple from the other.
- Suppose that we want to write the query, find the names of all instructors whose salary is greater than at least one instructor in the Math department.
-
```
SELECT DISTINCT T.name
FROM Instructor as T ,
Instructor AS S
WHERE T.salary> S.salary and S.deptName == "Math";
```

AS clauses

- Find all names of common teachers in Instructor and Teaches tables

Use AS to implement variables attributes to hold places

```
select distinct(Instructor.name) as newName
From Instructor, teaches
Where Instructor.ID = teaches.ID and newName == "Thompson";
```

AS clauses

- Find the names of all Instructors whose salary is greater than at least one Instructor in the Math 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";
  ```

- Reference: select * from Instructor;