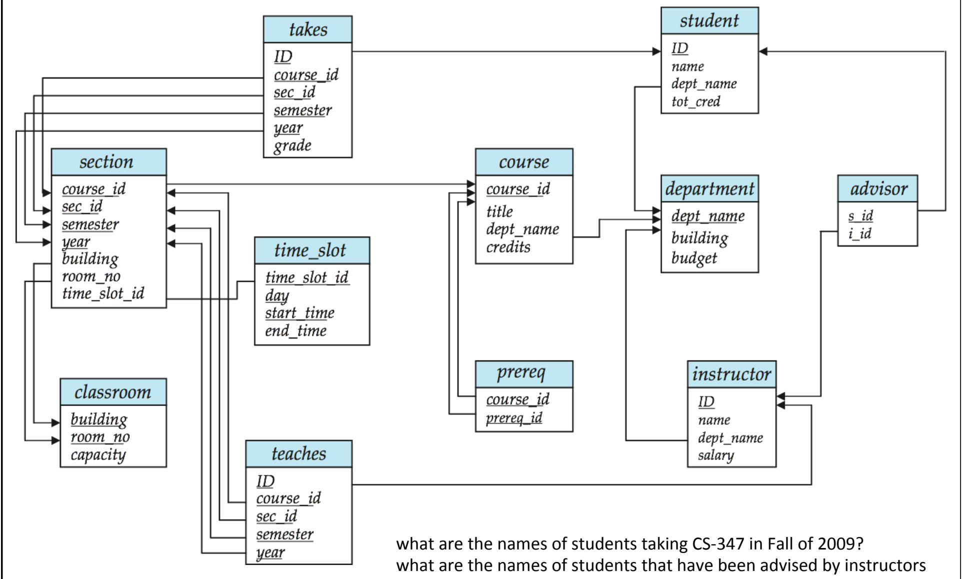


what are the names of students taking CS-347 in Fall of 2009?
what are the names of students that have been advised by instructors named Einstein?
what are the course_ids that are prerequisites for another course in Computer Science?
how many distinct students are taking courses taught by instructors in CS in the Fall of 2009?

JOIN Syntax

- When you want to join tables in a chain, NATURAL JOIN can fail to understand how to complete joins

```
SELECT DISTINCT student.ID
  FROM student S
       JOIN takes T ON S.ID=T.ID
       JOIN section C ON T.course_id=C.course_id
                        AND T.sec_id=C.sec_id
                        AND T.semester=C.semester
                        AND T.year=C.year
 WHERE building='Smith';
```

Types of Joins

- INNER JOIN a.k.a. JOIN
  - Default join
  - Only returns records with values matching in both tables
  - NATURAL JOIN is a special type of INNER JOIN
- OUTER JOINs
  - LEFT OUTER JOIN
    - Result includes everything from INNER JOIN, plus records in left table that have no match in right table
  - RIGHT OUTER JOIN
    - Result includes everything from INNER JOIN, plus records in right table that have no match in left table
  - FULL OUTER JOIN
    - Result includes everything from INNER JOIN, plus records from both left and right tables that have no match in the other table
    - MySQL doesn't have this

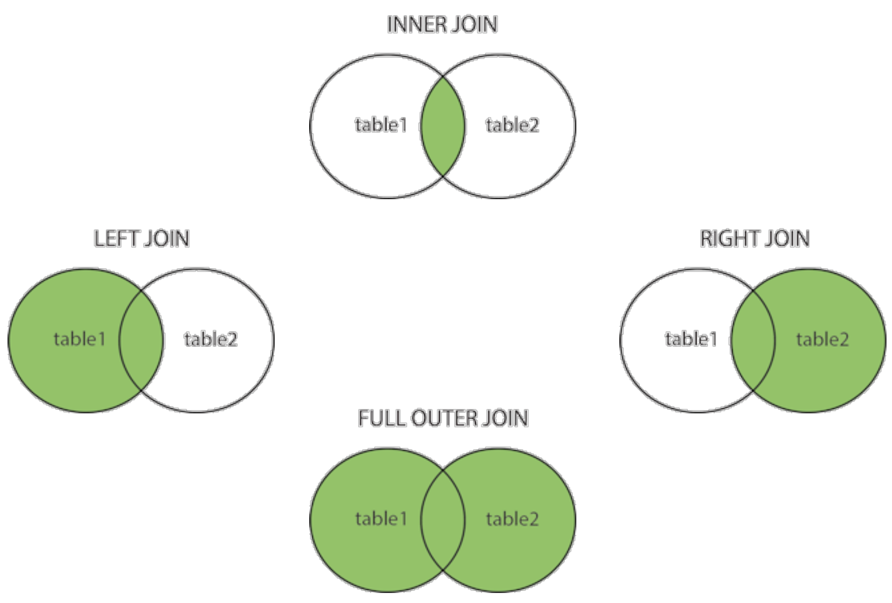
OUTER JOIN Example

- Find students who have not been assigned any advisor

```
SELECT *
  FROM student S
       LEFT OUTER JOIN advisor A ON S.ID=A.s_id
 WHERE A.i_id IS NULL
```

- Compare to answer using nested subquery:

```
SELECT *
  FROM student S
 WHERE S.id NOT IN (SELECT s_id FROM advisor);
```



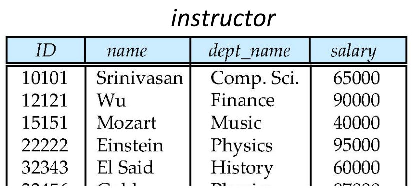
Aggregation

```
SELECT field1, FUNCTION(field2), …
  FROM Table
 WHERE condition
 GROUP BY field1
```

- Calculate aggregate functions for values of `field2` within *groups* of records by values of `field1`
  - `AVG`, `MIN`, `MAX`, `SUM`, `COUNT` are typical aggregation functions
- `FUNCTION()` computes something from values of `field2`
- `GROUP BY` restricts function to values of `field2` in records with the same value of `field1`

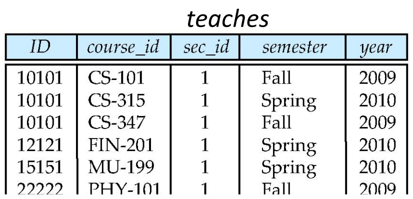
Aggregation

- `SELECT … GROUP BY …`



instructor

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
[illegible]	[illegible]	[illegible]	[illegible]



teaches

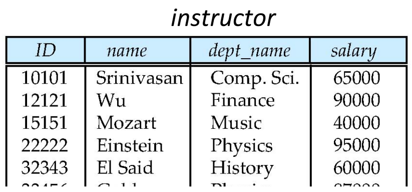
ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009

More on Aggregation

- `SELECT COUNT(DISTINCT field1) FROM Table`
  - Return the number of unique values of `field1`
  - Different from `SELECT DISTINCT COUNT(field1)`!
- `HAVING` operator is like `WHERE`, but used to restrict by aggregated fields
  - `SELECT field1, AVG(field2) AS mean FROM Table GROUP BY field1 HAVING mean > 3`
  - Only return groups where the average value of `field2` is more than three

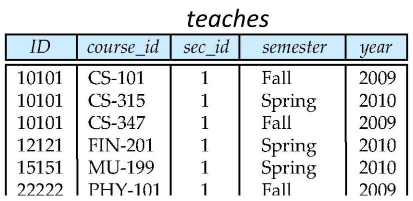
Aggregation

- `SELECT .. GROUP BY .. HAVING ..`



instructor

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
[illegible]	[illegible]	[illegible]	[illegible]



teaches

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009

Departments with average monthly salary over $10,000?

Semesters with more than 100 sections offered?

Sorting

```
SELECT *
FROM Table1, Table2, …
WHERE …
ORDER BY Table1.field1
```

- Return results sorted in increasing order of `Table1.field1`
- Specify multiple fields to break ties
  - `ORDER BY Table1.field1, Table2.field1, …`
- Use DESC or ASC to specify decreasing or increasing
  - `ORDER BY Table1.field1 DESC, Table2.field1 ASC, …`
- Sorting can be expensive!

Restricting Results by Row Number

```
SELECT *
FROM Table1, Table2, …
WHERE Table1.fKey = Table2.pKey AND …
LIMIT n
```

- Returns only the first n matching records
- `LIMIT x, y` to select a range
  - `LIMIT 0, 10` equivalent to `LIMIT 10`
  - `LIMIT 5, 10` returns records 6, 7, 8, 9, and 10

Conceptual Order of Operations

- First complete JOINs and WHEREs
  - If there is a chain of JOINs, MySQL decides the best way to complete them
  - Sometimes a WHERE is applied before a JOIN, other times the JOIN comes first—MySQL decides best order
- Then `GROUP BY` to form groups of those records with common field values
- Then `HAVING` to select a subset of groups
- Then `ORDER BY` to sort the result set
- Finally, if there is a DISTINCT or LIMIT, apply it last

UNION

```
(SELECT field1, field2 FROM Table1 WHERE …)
UNION
(SELECT field1, field2 FROM Table2 WHERE …)
```

- The two tables must be **union-compatible**
  - Same number of fields, same domains
- Use `UNION ALL` to keep duplicates

INTERSECTion

```
(SELECT field1, field2 FROM Table1 WHERE …)
INTERSECT
(SELECT field1, field2 FROM Table2 WHERE …)
```

- The two tables must be union-compatible
  - Same number of fields, same domains
- Use `INTERSECT ALL` to keep duplicates
- MySQL does not support `INTERSECT`

Set Intersection With Nested Subquery

- Implement INTERSECT using nested subqueries

```
SELECT course_id
FROM Teaches
WHERE semester=‘Spring’ AND year=2015 AND
      course_id IN (SELECT course_id
                    FROM Teaches
                    WHERE semester=‘Fall’ AND year=2014)
```

EXCEPT (Set Difference)

```
(SELECT field1, field2 FROM Table1 WHERE …)
EXCEPT
(SELECT field1, field2 FROM Table2 WHERE …)
```

- The two tables must be union-compatible
  - Same number of fields, same domains
- Use EXCEPT ALL to keep duplicates
- MySQL does not support EXCEPT

Set Difference With Nested Subquery

- Implement EXCEPT using nested subqueries

```
SELECT course_id
FROM Teaches
WHERE semester=‘Spring’ AND year=2015 AND
   course_id NOT IN (SELECT course_id
                    FROM Teaches
                    WHERE semester=‘Fall’ AND year=2014)
```

Summary

- SQL is an extensive and deep query language for relational databases
  - Allows you to find records matching specific conditions across multiple tables
  - “Quickly find answers to arbitrary questions”
- Most important features:
  1. Joining tables
  2. Grouping rows and aggregating values
  3. Sorting
  4. Set operations