MLDS-413 Introduction to Databases and Information Retrieval

Lecture 8
Having predicates, INNER Joins

Instructor: Nikos Hardavellas

Slides adapted from Steve Tarzia

Last Lecture

SELECT description and examples in SQLite

SELECT steps (abbreviated):

1. `FROM` chooses the table of interest
2. `WHERE` throws out irrelevant rows
3. `GROUP BY` identifies rows to combine
4. `SELECT` tells what values to return (math and aggregation on each group)
5. `HAVING` throws out irrelevant rows (after aggregation)
6. `ORDER BY` sorts
7. `LIMIT` throws out rows based on their position in the results

Predicates in more detail

- `WHERE` & `HAVING` filter rows according to conditions called *predicates*
- Any of the following can be combined, like an algebraic expression:
  - Binary operations (used between two things):
    - `=   ==   !=   <>   >   <   >=   <=   LIKE   AND   OR   REGEXP` ←*(coming soon!)*
    - `+   -   *   /   ||   %   <<   >>   &   |`
    - See https://www.sqlite.org/lang_expr.html
  - `NOT …`
  - `… IS NULL,  … IS NOT NULL` ←*(coming soon!)*
  - `… BETWEEN … AND …`
  - `… IN (…, …, …)`
  - `(…)`
- Can also use all of the above in the columns we print out, and inside aggregations like `SUM, MIN, MAX, AVG`

`HAVING` is like `WHERE`, but applied after aggregation

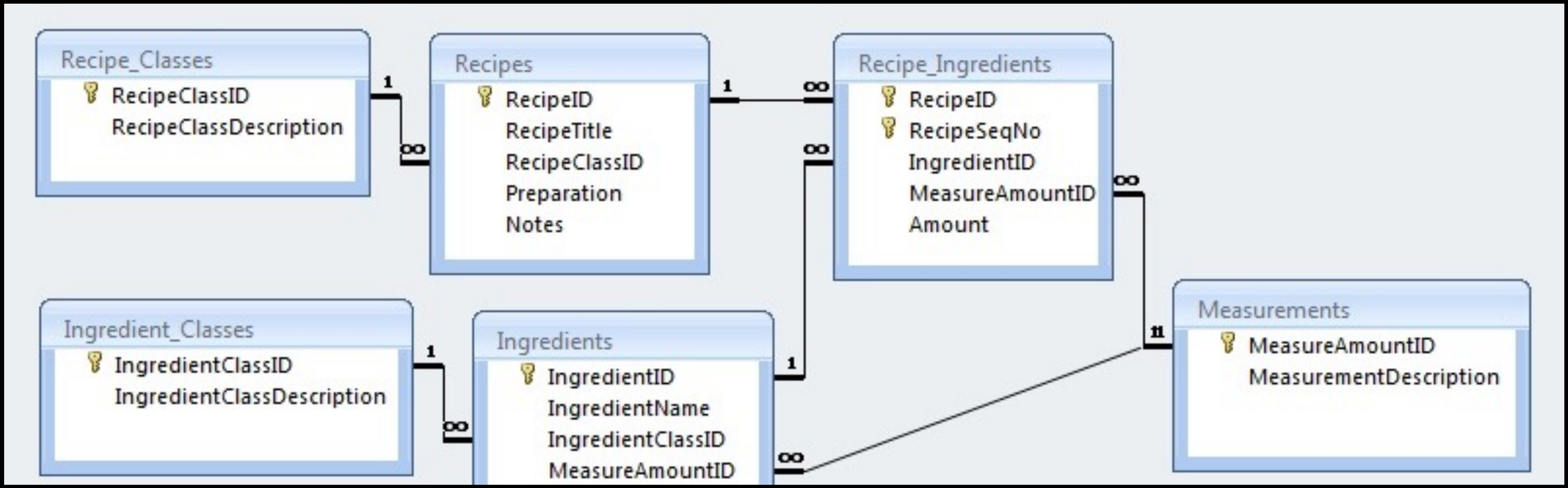
SELECT steps (abbreviated):

1. `FROM` chooses the table of interest
2. `WHERE` throws out irrelevant rows
3. `GROUP BY` identifies rows to combine
4. `SELECT` tells what values to return (allowing math and aggregation)
   - `COUNT`, `SUM`, `MAX`, `MIN`, `AVG`, etc, (i.e., all aggregators) are all evaluated at this step
5. `HAVING` throws out irrelevant rows (after aggregation)
   - `HAVING` can now use `COUNT`, `SUM`, `MAX`, `MIN`, `AVG`, etc, in conditionals
6. `ORDER BY` sorts
7. `LIMIT` throws out rows based on their position in the results

HAVING example

- Which recipes have 10 or more ingredients (ordered by #ingredients)?

```
SELECT recipeID, count(DISTINCT IngredientID) as NumIngredients
FROM Recipe_Ingredients
GROUP BY recipeID
HAVING NumIngredients >= 10
ORDER BY count(DISTINCT IngredientID);
```



What if you need to combine data from multiple tables?

1. `FROM` chooses the table of interest
2. `WHERE` throws out irrelevant rows
3. `GROUP BY` identifies rows to combine
4. `SELECT` tells what values to return (allowing math and aggregation)
5. `HAVING` throws out irrelevant rows (after aggregation)
6. `ORDER BY` sorts
7. `LIMIT` throws out rows based on their position in the results

A subquery can draw data from another table, but there is a better way …

JOINs create *virtual* tables from several tables

- Normalizing the staff directory left us with three tables
- Eliminated redundant information, but now we have to look in three different tables to answer some questions.

staff			
id	***name***	***room***	***departmentID***
11	Bob	100	1
20	Betsy	100	2
21	Fran	101	1
22	Frank	102	4
35	Sarah	200	5
40	Sam	10	7
54	Pat	102	2

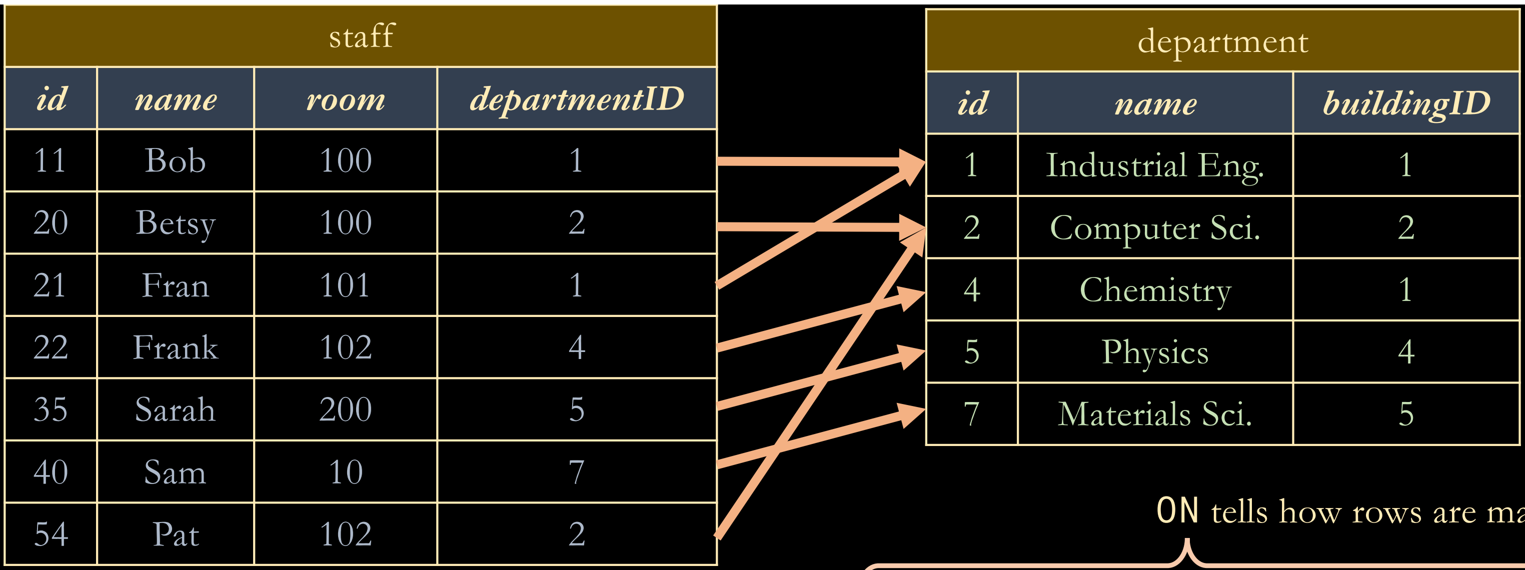
department		
id	***name***	***buildingID***
1	Industrial Eng.	1
2	Computer Sci.	2
4	Chemistry	1
5	Physics	4
7	Materials Sci.	5

building		
id	***name***	***facilitiesExt***
1	Tech	1-1000
2	Ford	1-5003
4	Mudd	1-2005
5	Cook	1-3004
6	Garage	1-6001

What if we want to print the staff directory?

staff					
id	***name***	***department***	***building***	***room***	***facilitiesExt***
11	Bob	Industrial Eng.	Tech	100	1-1000
20	Betsy	Computer Sci.	Ford	100	1-5003
21	Fran	Industrial Eng.	Tech	101	1-1000
22	Frank	Chemistry	Tech	102	1-1000
35	Sarah	Physics	Mudd	200	1-2005
40	Sam	Materials Sci.	Cook	10	1-3004
54	Pat	Computer Sci.	Ford	102	1-5003

We can generate a virtual table like this with INNER JOIN



staff			
id	*name*	*room*	*departmentID*
11	Bob	100	1
20	Betsy	100	2
21	Fran	101	1
22	Frank	102	4
35	Sarah	200	5
40	Sam	10	7
54	Pat	102	2

department		
id	*name*	*buildingID*
1	Industrial Eng.	1
2	Computer Sci.	2
4	Chemistry	1
5	Physics	4
7	Materials Sci.	5

`ON` tells how rows are matched

```
SELECT * FROM staff JOIN department ON staff.departmentId=department.id
```

staff.*id*	staff.*name*	*staff.room*	staff.*departmentId*	department.*id*	department.*name*	department.*buildingID*
11	Bob	100	1	1	Industrial Eng.	1
20	Betsy	100	2	2	Computer Sci.	2
21	Fran	101	1	1	Industrial Eng.	1
22	Frank	102	4	4	Chemistry	1
35	Sarah	200	5	5	Physics	4
40	Sam	10	7	7	Materials Sci.	5
54	Pat	102	2	2	Computer Sci.	2

How JOIN builds a composite table

```
SELECT * FROM staff JOIN department
                 ON staff.departmentId=department.id;
```

Start with the first table (staff)

Join with rows from the 2nd table (department) that match according to the `ON` columns

staff.*id*	staff.*name*	*staff.room*	staff.*departmentID*	department.*id*	department.*name*	department.*buildingID*
11	Bob	100	1	1	Industrial Eng.	1
20	Betsy	100	2	2	Computer Sci.	2
21	Fran	101	1	1	Industrial Eng.	1
22	Frank	102	4	4	Chemistry	1
35	Sarah	200	5	5	Physics	4
40	Sam	10	7	7	Materials Sci.	5
54	Pat	102	2	2	Computer Sci.	2

Just print the columns we need

```
SELECT staff.id, staff.name, room,
       department.name, department.buildingId
  FROM staff JOIN department
       ON staff.departmentId=department.id;
```

staff.*id*	staff.*name*	*staff.room*	department.*name*	department.*buildingID*
11	Bob	100	Industrial Eng.	1
20	Betsy	100	Computer Sci.	2
21	Fran	101	Industrial Eng.	1
22	Frank	102	Chemistry	1
35	Sarah	200	Physics	4
40	Sam	10	Materials Sci.	5
54	Pat	102	Computer Sci.	2

Reorder and rename the columns

```
SELECT staff.id AS staffID,
       staff.name AS name,
       department.name AS department,
       department.buildingId AS buildingId,
       staff.room AS room
  FROM staff JOIN department
       ON staff.departmentId=department.id;
```

staffId	*name*	*department*	*buildingID*	*room*
11	Bob	Industrial Eng.	1	100
20	Betsy	Computer Sci.	2	100
21	Fran	Industrial Eng.	1	101
22	Frank	Chemistry	1	102
35	Sarah	Physics	4	200
40	Sam	Materials Sci.	5	10
54	Pat	Computer Sci.	2	102

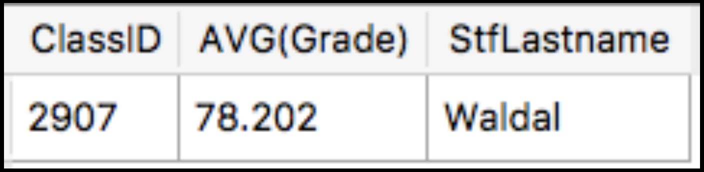
Add the third table

```
SELECT staff.id AS staffId, staff.name AS name,
       department.name AS department,
       building.name AS building, staff.room AS room,
       building.faxNumber AS faxNumber
  FROM staff JOIN department ON staff.departmentId    = department.id
             JOIN building   ON department.buildingId = building.id;
```

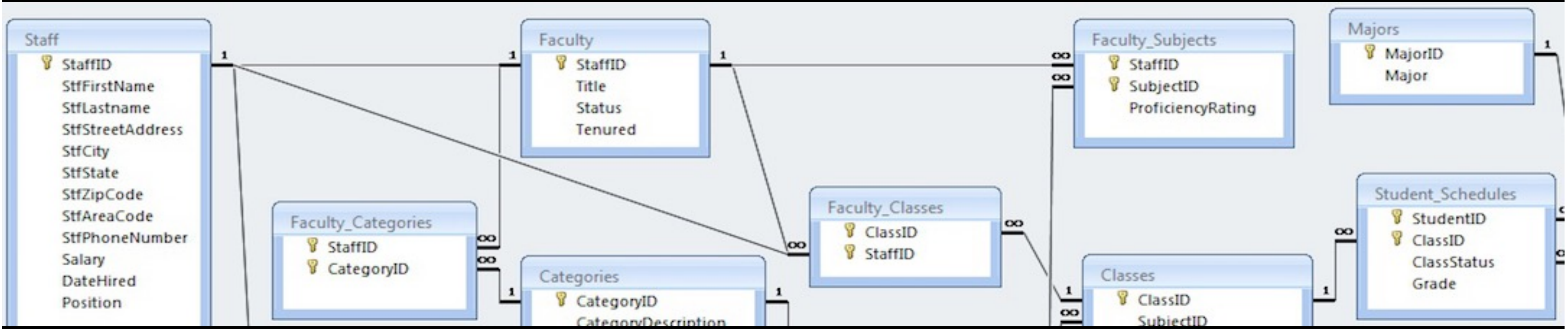
staffId	*name*	*department*	*building*	*room*	*faxNumber*
11	Bob	Industrial Eng.	Tech	100	1-1000
20	Betsy	Computer Sci.	Ford	100	1-5003
21	Fran	Industrial Eng.	Tech	101	1-1000
22	Frank	Chemistry	Tech	102	1-1000
35	Sarah	Physics	Mudd	200	1-2005
40	Sam	Materials Sci.	Cook	10	1-3004
54	Pat	Computer Sci.	Ford	102	1-5003

Who teaches the largest class and what is the average grade?

```
SELECT Student_Schedules.ClassID,
       AVG(Grade),
       StfLastname
FROM Student_Schedules JOIN Faculty_Classes
     ON Student_Schedules.ClassID=Faculty_Classes.ClassID
     JOIN Staff ON Faculty_Classes.StaffID = Staff.StaffID
GROUP BY Student_Schedules.ClassID
ORDER BY COUNT(*) DESC LIMIT 1;
```



ClassID	AVG(Grade)	StfLastname
2907	78.202	Waldal



How JOIN builds a composite table

```
SELECT * FROM staff JOIN department
              ON staff.departmentId=department.id
```

What if there are multiple matches in the second table?

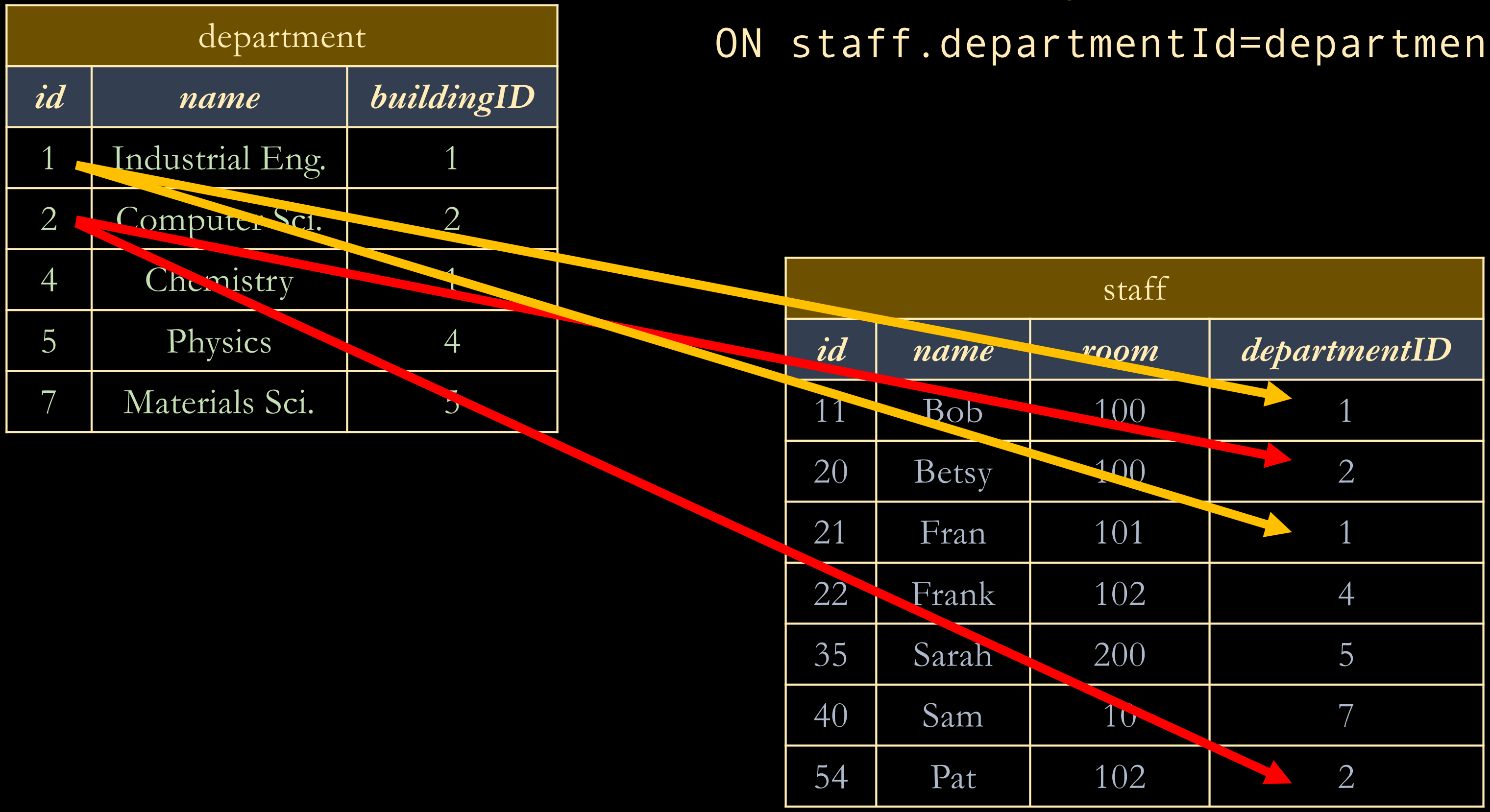
Start with the first table (staff)

Join with rows from the 2nd table (department) that match according to the `ON` columns

staff.*id*	staff.*name*	*staff.room*	staff.*departmentID*	department.*id*	department.*name*	department.*buildingID*
11	Bob	100	1	1	Industrial Eng.	1
20	Betsy	100	2	2	Computer Sci.	2
21	Fran	101	1	1	Industrial Eng.	1
22	Frank	102	4	4	Chemistry	1
35	Sarah	200	5	5	Physics	4
40	Sam	10	7	7	Materials Sci.	5
54	Pat	102	2	2	Computer Sci.	2

How JOIN deals with multiple matches

```
SELECT * FROM department JOIN staff
  ON staff.departmentId=department.id
```



department

id	*name*	*buildingID*
1	Industrial Eng.	1
2	Computer Sci.	2
4	Chemistry	1
5	Physics	4
7	Materials Sci.	5

staff

id	*name*	*room*	*departmentID*
11	Bob	100	1
20	Betsy	100	2
21	Fran	101	1
22	Frank	102	4
35	Sarah	200	5
40	Sam	10	7
54	Pat	102	2

How JOIN deals with multiple matches

```
SELECT * FROM department JOIN staff
  ON staff.departmentId=department.id
```

What if there are multiple matches in the second table?

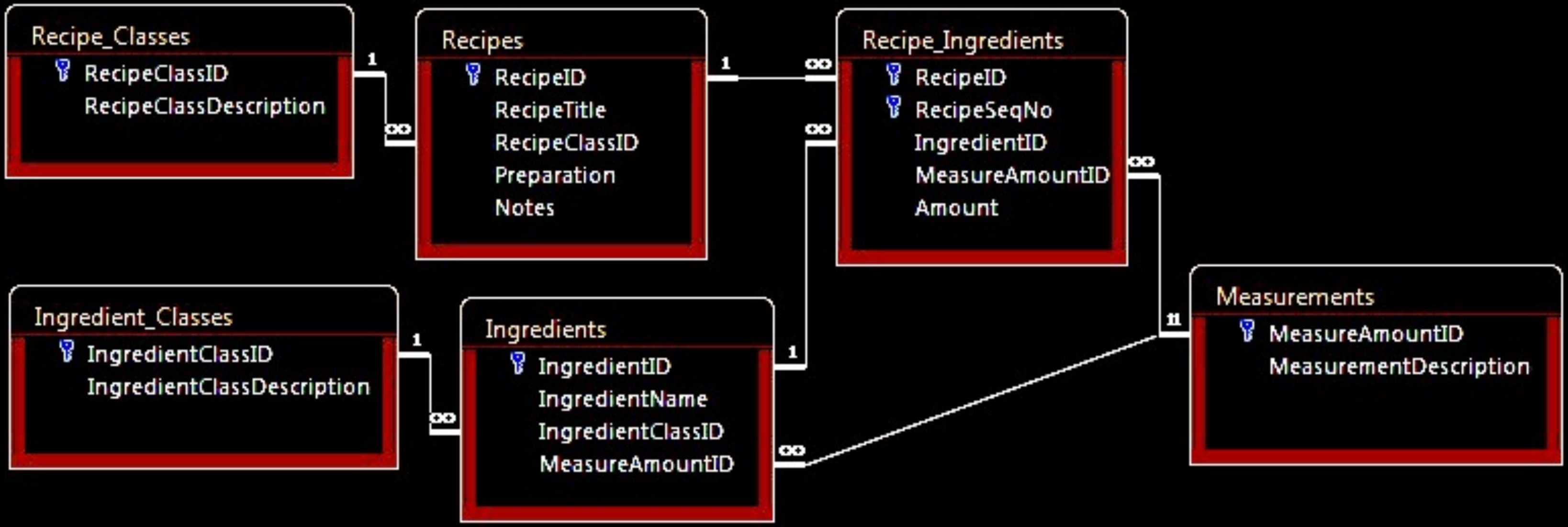
Create a row for *every pair* of matches

department.***id***	department.***name***	department.***buildingId***	staff.***id***	staff.***name***	staff.***room***	department.***id***
1	Industrial Eng.	1	11	Bob	100	1
1	Industrial Eng.	1	21	Fran	101	1
2	Computer Sci.	2	20	Betsy	100	2
2	Computer Sci.	2	54	Pat	102	2
4	Chemistry	1	22	Frank	102	4
5	Physics	4	35	Sarah	200	5
7	Materials Sci.	5	40	Sam	10	7

Summary of INNER JOINs

- Introduced INNER JOIN
  - `table1 INNER JOIN table2 ON table1.col1 = table2.col2`
  - Creates a virtual table
  - Rows are matched according to columns specified with "ON"
    - Usually this is a foreign key
    - If "ON" is omitted, all columns with identical names are checked for a match
  - Joined table has all the columns from both tables
- NOTE:
  - The "`INNER`" keyword is optional
  - If a matching row is not found in the second table, the row is omitted
  - In other words, a row must exist in both tables to produce a row in the joined table

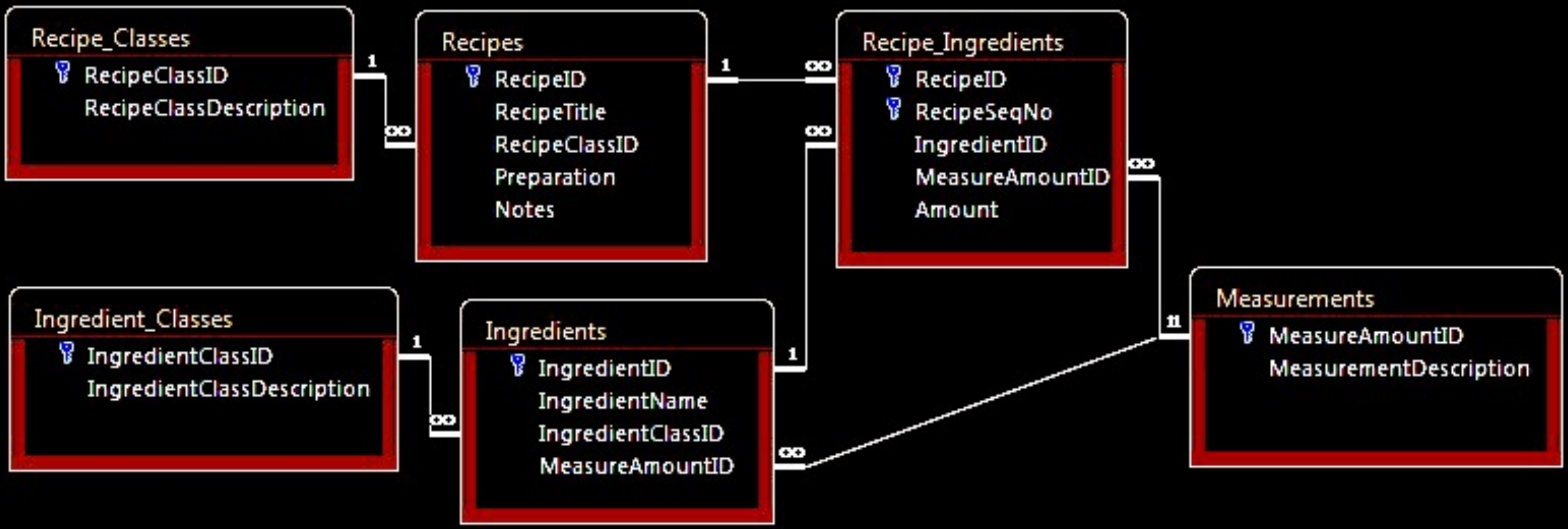
Print the recipe (ingredients, amount, measure) for Irish Stew in recipe sequence order (RecipeID = 1) *(Recipes.sqlite)*



Print the recipe (ingredients, amount, measure) for Irish Stew in recipe sequence order (RecipeID = 1) *(Recipes.sqlite)*

```
SELECT IngredientName,
       Amount,
       Measurements.MeasurementDescription
FROM Recipe_Ingredients
  JOIN Ingredients
    ON Recipe_Ingredients.IngredientId
       = Ingredients.IngredientID
  JOIN Measurements
    ON Recipe_Ingredients.MeasureAmountID
       = Measurements.MeasureAmountID
WHERE RecipeId=1
ORDER BY RecipeSeqNo;
```

What is the name of the recipe with the most ingredients, and how many ingredients does it have? *(Recipes.sqlite)* (Can be done with either a subquery or a JOIN)

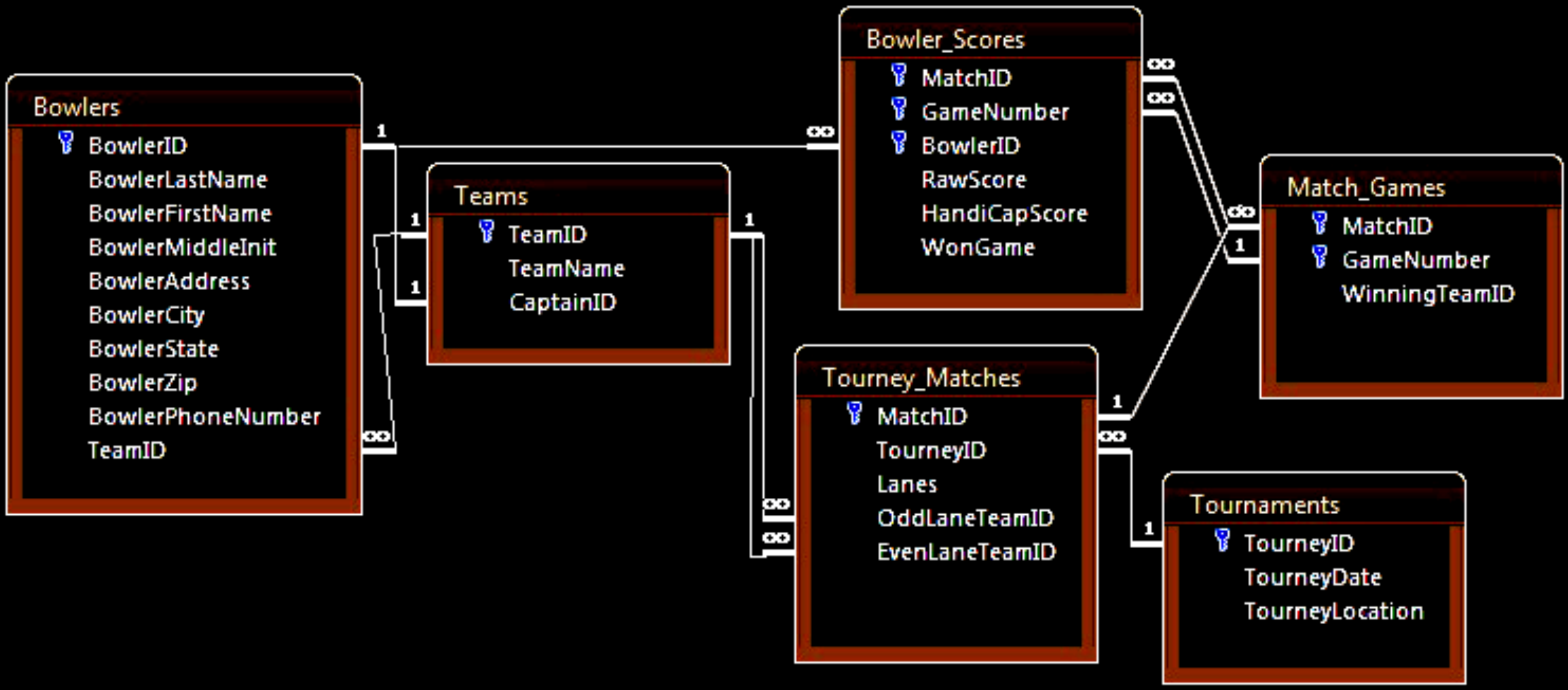


What is the name of the recipe with the most ingredients, and how many ingredients does it have? *(Recipes.sqlite)*

```
SELECT Recipes.RecipeTitle,
       COUNT(DISTINCT IngredientID) AS NumIngredients
FROM Recipe_Ingredients
  JOIN Recipes
    ON Recipes.RecipeID = Recipe_Ingredients.RecipeID
GROUP BY Recipes.RecipeID
ORDER BY NumIngredients DESC
LIMIT 1;
```

Print a schedule of all the team matchups over the whole season (Date, Location, OddTeamName, EvenTeamName)

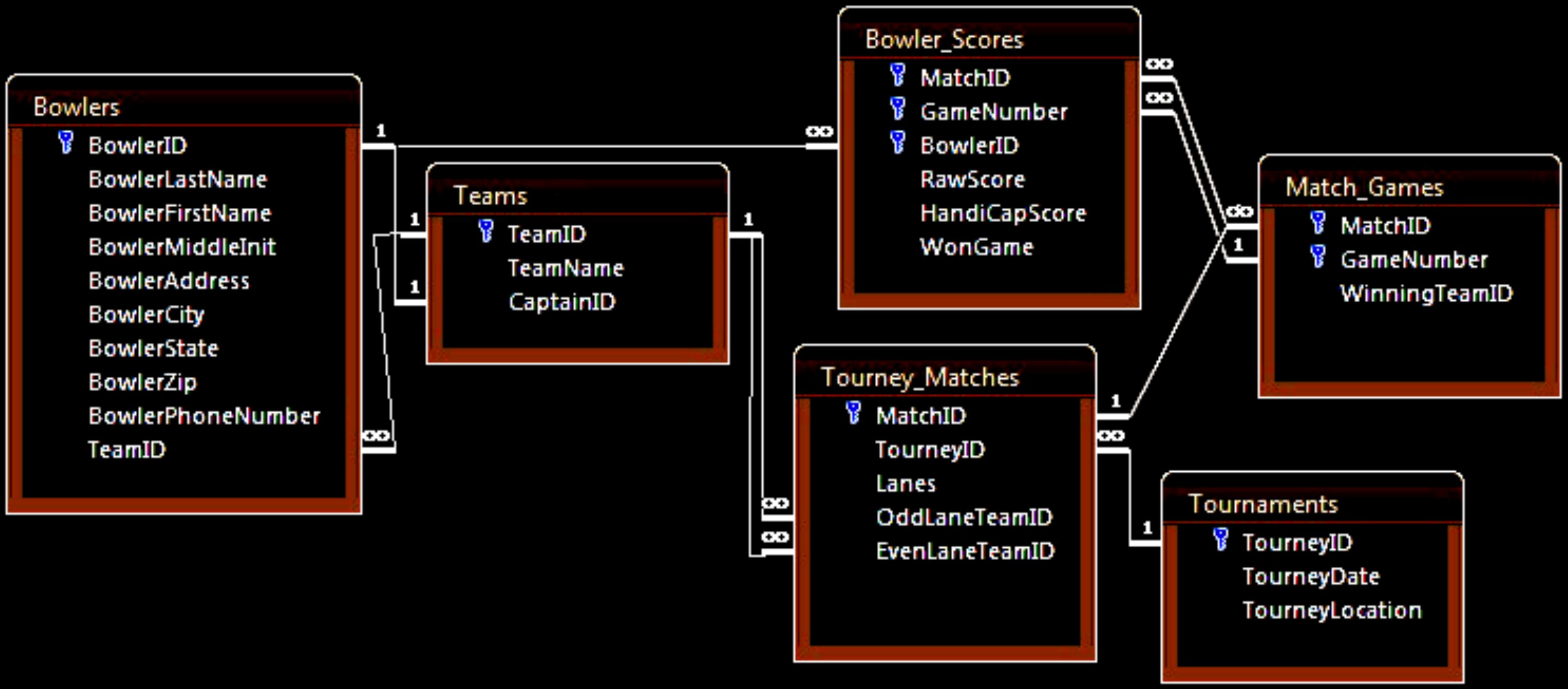
(BowlingLeague.sqlite)



Print a schedule of all the team matchups over the whole season (Date, Location, OddTeamName, EvenTeamName) *(BowlingLeague.sqlite)*

```
SELECT TourneyDate,
       TourneyLocation,
       OddTeam.TeamName,
       EvenTeam.TeamName
FROM Tourney_Matches
  JOIN Tournaments
    ON Tourney_Matches.TourneyID = Tournaments.TourneyID
  JOIN Teams AS OddTeam
    ON OddLaneTeamID = OddTeam.TeamID
  JOIN Teams AS EvenTeam
    ON EvenLaneTeamID = EvenTeam.TeamID;
```

Print game results for Tournament #1, including bowler names, team names, & raw score *(BowlingLeague.sqlite)*



Print game results for Tournament #1, including bowler names, team names, & raw score *(BowlingLeague.sqlite)*

```
SELECT
  Bowler_Scores.MatchID, GameNumber, TeamName,
  BowlerFirstName || " " || BowlerLastName AS Bowler,
  RawScore
FROM Bowler_Scores
  JOIN Tourney_Matches
    ON Bowler_Scores.MatchID = Tourney_Matches.MatchID
  JOIN Bowlers
    ON Bowlers.BowlerID = Bowler_Scores.BowlerID
  JOIN Teams
    ON Bowlers.TeamID = Teams.TeamID
WHERE TourneyId=1;
```