Intro to Yelp & SQL

CS 1270 Database Management Systems
Slides by Jon
Yelp DB schema:

<table>
<thead>
<tr>
<th>business</th>
<th>review</th>
<th>user</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>business id</td>
<td>id</td>
</tr>
<tr>
<td>name</td>
<td>user id</td>
<td>name</td>
</tr>
<tr>
<td>full address</td>
<td>text</td>
<td>review count</td>
</tr>
<tr>
<td>city</td>
<td>useful votes</td>
<td>useful votes</td>
</tr>
<tr>
<td>state</td>
<td>funny votes</td>
<td>funny votes</td>
</tr>
<tr>
<td>latitude</td>
<td>cool votes</td>
<td>cool votes</td>
</tr>
<tr>
<td>longitude</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stars</td>
<td></td>
<td></td>
</tr>
<tr>
<td>review count</td>
<td></td>
<td></td>
</tr>
<tr>
<td>open</td>
<td></td>
<td></td>
</tr>
<tr>
<td>photo url</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Your task:

1. Connect to Yelp DB via JDBC
2. Run SQL Queries that answer the questions
3. Retrieve query results and return them
1. Connect to JDBC

To connect, you will need to establish a connection:

```java
Class.forName("org.sqlite.JDBC"); // not necessary
conn = DriverManager.getConnection(
    "jdbc:sqlite:/course/cs127/pub/yelp/yelp.db");
```
2. Run a SQL query

Once connected, you will write a string as a SQL query that looks something like this:

```java
String query = "SELECT ... 
FROM ... 
WHERE att1 = ?;"
```

Note the use of variable parameters given by "?". You will set those parameters in the next step.
2. Run a SQL query

Then compile the query with a "prepared statement":

```java
PreparedStatement stat =
    conn.prepareStatement(query);
```

**Why not just use 'Statement'?**

*Statement* will be used for executing static SQL *statements* and it can’t accept input parameters. *PreparedStatement* will be used for executing SQL *statements* many times dynamically. It will accept input parameters.

From
https://stackoverflow.com/questions/3271249/difference-between-statement-and-preparedstatement
2. Run SQL Query

Set your parameters:

```java
stat.setString(1, my_variable);
```

Note: 1 corresponds to the first "?" in your sql query, 2 corresponds to the second "?", 3 corresponds to the third "?", etc.

Now, execute the query and save the results:

```java
ResultSet results = stat.executeQuery();
```
3. Retrieve results and return them

Create a list of objects to hold the values in the result set (see stencil code classes for useful objects)

```java
List<My_Object> my_objectList = new ArrayList<My_Object>();
```

Loop through tuples of the result set, creating a new object for each tuple and adding values of each tuple to the object:

```java
while(results.next()){
    My_Object my_object_name = new My_Object();
    my_object_name.setAtt(results.getString("<att>");
    my_objectList.add(object_name);
}
3. Retrieve results and return them

Look up JDBC docs to see the different `getObject()` methods for extracting values from the `resultSet`. Here are a few:

```java
getString("att")
getDouble("att")
getInt("att")
```

Finally, close the result set and return the list:

```java
results.close();
return objectList;
```
Overview of SQL

Basic Structure:

```
SELECT att1, att2/2 AS 'half_att2', DISTINCT(att3) 'my_att'
FROM rel1, rel2
WHERE att1 = 'x' AND att2 < 'y' OR (att3, att4) >= (z, w)
ORDER BY half_att2
LIMIT 5
```
JOINS

Instead of cross product, you can use JOINs

...FROM rel1 JOIN rel2 ON rel1.att1 = rel2.att1 AND rel1.att2 = rel2.att2

Alternative that automatically matches on similarly named attributes:

   FROM rel1 NATURAL JOIN rel2

Alternative that matches on the attributes in the using clause if they are in both relations:

   FROM rel1 JOIN rel2 USING (att1, att2)

Note: You can do a theta join on attributes with different names using the ON clause with a comparison. You can use any of =, <, <=, >, >=, <>
AGGREGATE FUNCTIONS: COUNT(), SUM(), MAX(), MIN(), AVG()

SELECT att1, SUM(att1) sum_1, MAX(att2) max_2,
FROM some_rel
WHERE <condition>
GROUP BY att1, att2,
HAVING (sum_1, max_2) = (4, 5)

Should avoid ambiguity by only including in SELECT clause att's that are either aggregates or that are mentioned in the GROUP BY clause.
SQLite does support select statement on non-aggregate att's not in the group by clause, but this is not good practice.

HAVING clause must come after GROUP BY clause, and is the equivalent of WHERE clause but on aggregate values.
AGGREGATE FUNCTIONS: COUNT(), SUM(), MAX(), MIN(), AVG()

SELECT att1, SUM(att1) sum_1, MAX(att2) max_2, 
FROM some_rel 
GROUP BY att1, att2, 
HAVING (sum_1, max_2) = (6, 5)

result: 3|6|5
SUBQUERIES

A subquery returns a relation that can then be operated on by other queries:

Structures:

One after the other

\[
\text{subq}_1 \ \text{SET\_OPERATION} \ \text{subq}_2
\]

Or, the subquery can be a preface to the main query via the WITH clause:

\[
\text{WITH} \ \text{subq} \ \text{AS} \ (\text{SELECT} \ \ldots) \\
\text{SELECT} \ * \ \text{FROM} \ \text{subq} \\
\text{WHERE} \ \ldots
\]

Or, the subquery can be nested in outer queries in the FROM clause:

\[
\text{SELECT} \ a \ \text{FROM} \ ( \\
\text{SELECT} \ \text{att2} \ \text{as} \ a \ \text{FROM} \ \text{rel} \\
\text{WHERE} \ \ldots
\)
SET OPERATORS ON SUBQUERIES

subquery1 \textbf{UNION} subquery2 (deletes all duplicates)

subquery1 \textbf{UNION ALL} subquery2 (retains all duplicates)

subquery1 \textbf{INTERSECT} subquery2 (deletes duplicates in the resultant intersection)

subquery1 \textbf{INTERSECT ALL} subquery2 (retains duplicates in the resultant intersection. Ex. 5 repeats in first relation, and 2 in second: retains 2)

subquery1 \textbf{EXCEPT} subquery2 (deletes duplicates that appear in the second relation)

subquery1 \textbf{EXCEPT ALL} subquery2 (retains duplicates. Ex. 5 repeats in first relation, and 2 in second: retains 3)
SUBQUERIES in the WHERE CLAUSE

Note: att1, att2, and att3 could all be the same att, or different

SELECT att1
FROM rel...

WHERE att2 IN (SELECT att3
FROM rel
WHERE <condition>)
returns true if at least one member of subquery result is equal to att2.

WHERE att > SOME (SELECT att
FROM rel
WHERE <condition>)
returns true if comparison holds true for at least one member of subquery result

WHERE EXISTS (SELECT att
FROM rel
WHERE <condition>)
returns true if subquery is not the empty set

WHERE UNIQUE (SELECT att
FROM rel
WHERE <condition>)
returns true if subquery contains no duplicates
SUBQUERIES in the SELECT CLAUSE

A "scalar subquery" is a subquery that returns no more than one value (i.e. one tuple with one attribute).

Technically, it’s a relation, but SQL extracts the value and turns it into a scalar. Therefore you may write scalar subqueries wherever a scalar is allowed.

This means that you can have subqueries in the SELECT clause:

```
SELECT att,
(SELECT COUNT(*) FROM rel2) FROM rel1
```
CORRELATED SUBQUERIES

A nested subquery may reference attributes of the outer query. This is called a 'correlated subquery'.

In the WHERE clause:
```
SELECT att1
FROM rel1
WHERE EXISTS (
    SELECT att2
    FROM rel2
    WHERE rel1.att1 >= rel2.att2
)
```

From the textbook p. 96: We note that nested subqueries in the from clause cannot use correlation variables from other relations in the from clause. However, SQL:2003 allows a subquery in the from clause that is prefixed by the lateral keyword to access attributes of preceding tables or subqueries in the from clause. For example, if we wish to print the names of each instructor, along with their salary and the average salary in their department, we could write the query as follows:
```
select name, salary, avg salary
from instructor I1, lateral (select avg(salary)
    as avg salary
    from instructor I2
    where I2.dept_name= I1.dept_name):
```

In the oSELECT clause (if scalar):
```
SELECT att1,(  
    SELECT MAX(att2)
    FROM rel2
    WHERE rel1.att1 >= rel2.att2
) 
FROM rel1
```

Without the lateral clause, the subquery cannot access the correlation variable I1 from the outer query. Currently, only a few SQL implementations, such as IBM DB2, support the lateral clause.
DEALING WITH NULL COMPARISONS

The comparison of anything with null is unknown, therefore:

```
SELECT NULL and TRUE;
    -> UNKNOWN (NULL)
SELECT NULL and FALSE;
    -> FALSE (0)
SELECT NULL OR TRUE;
    -> TRUE (1)
SELECT NULL OR FALSE;
    -> UNKNOWN (NULL)
```

According to the SQL standard:

If the `WHERE` clause evaluates to false or unknown for a tuple, that tuple is not returned.

If `UNIQUE` is used, then nulls evaluate to unequal tuples, so evaluate as unique

If the `SELECT DISTINCT` is used, then all null values are considered identical, and are returned as a single distinct value; similarly for set operations when duplicates are eliminated.
One more thing: CASE

Cases allow you to break down conditionals easily:

```
SELECT
  CASE
    WHEN att1 = x THEN a
    WHEN att2 = y THEN b
    ELSE z
  END
FROM rel
```
Practice problem 1

DB Schema:

employee(emp_id, name, dept_id, mgr_id, salary)

department(dept_id, name)

1. For each department with an average salary of greater than $70,000, find the number of its employees, using the HAVING clause
Practice problem 1 solution

1. For each department with an average salary of greater than $70,000, find the number of its employees, using the HAVING clause

```sql
select count(emp_id)
from employee
group by dept_id
having avg(salary) > $70,000
```
Practice problem 2

employee(emp_id, name, dept_id, mgr_id, salary)

department(dept_id, name)

2. Find managers with a salary greater than the average manager's salary, using the WITH clause
Practice problem 2 solution

2. Find managers with a salary greater than the average manager's salary, using the WITH clause

```sql
with avg_mgr_salary as (
    select avg(salary) as average
    from employees
    where manager_id is null)
select name
from employee
where mgr_id is null
  and salary > (select average
              from avg_mgr_salary)
```
Practice problems 3 and 4: for you to answer

```
employee(emp_id, name, dept_id, mgr_id, salary)

department(dept_id, name)
```

3. Find all departments with more than 1 employee, using the UNIQUE clause

(Hint: see textbook p. 95)

4. Find all employees making at least 3 times more than some other employee in their department using the SOME clause.
Questions?

Happy coding :)