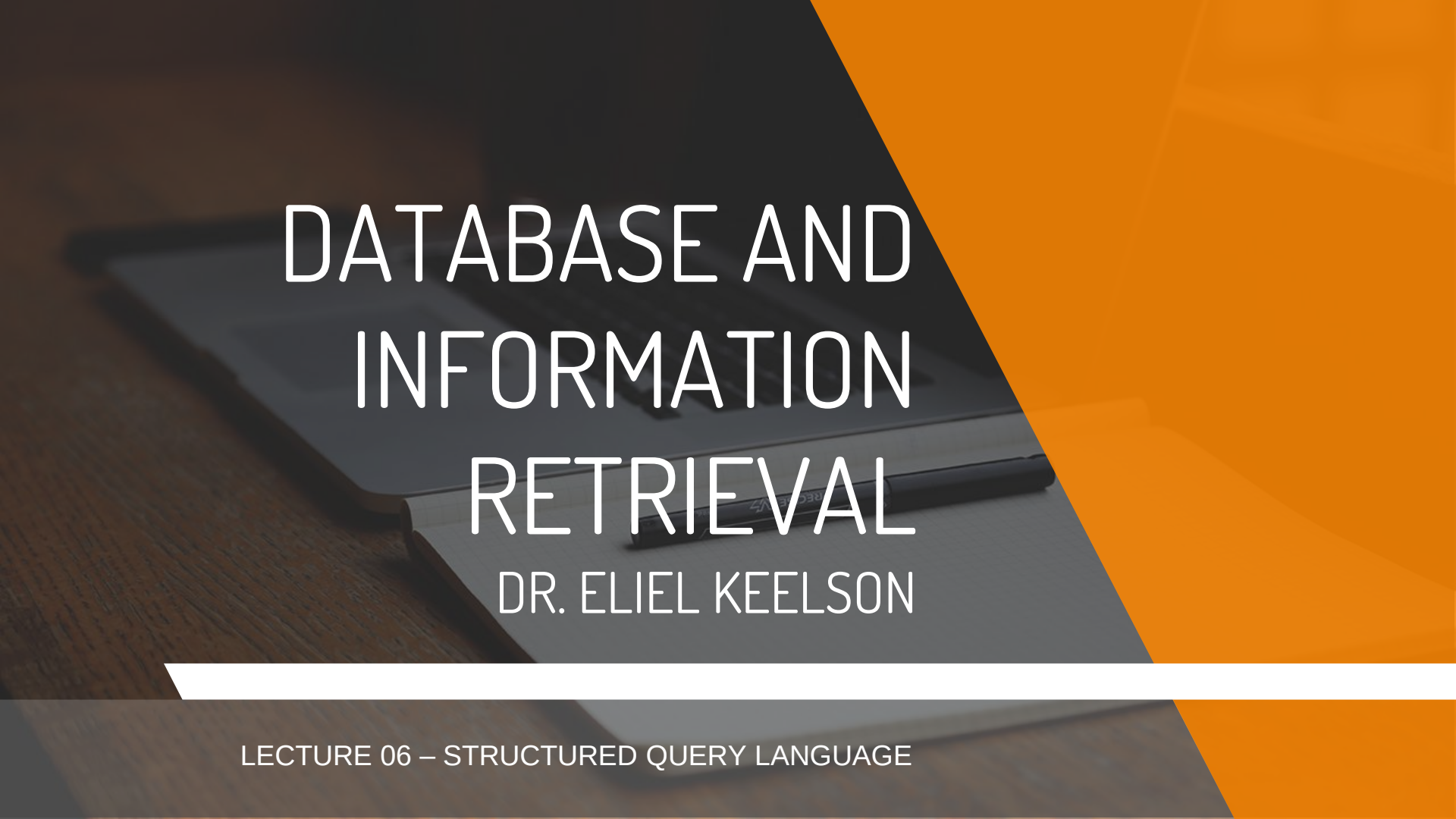

DATABASE AND INFORMATION RETRIEVAL

DR. ELIEL KEELSON

LECTURE 06 – STRUCTURED QUERY LANGUAGE

SQL

- SQL is by far the most important standard within the modern-day relational database market.
- SQL is a declarative language that enables you to perform a range of operations on relational tables.
- These operations are traditionally divided into three categories

SQL

The 3 categories of SQL are

1. Data Definition Language (DDL)
2. Data Manipulation Language (DML)
3. Data Control Language (DCL)

Data Definition Language (DDL)

- These are a set of SQL commands used to define (create, alter and drop) database schema as well as perform the following operations

1. Create a database
2. Drop a database
3. Create a table
4. Alter a table
5. Specify integrity checks
6. Delete a table
7. Build an index for a table
8. Define a virtual table (view)

Data Manipulation Language (DML)

- DML modifies the database instance by inserting, updating, and deleting its data.

- DML is responsible for all forms data modification in a database.

These manipulations include the following operations:

1. Query the database to show selected records
2. Insert, delete and update rows of the table
3. Control transactions when updating a database

Data Control Language (DCL)

- These set of SQL commands control access rights to parts of the database.

SQL

SQL

- Creating a database table requires that you assign a name to a database as well as the tables it contains and that you define the names and datatypes of each of the columns of each table.

- Each column description may include other supplementary clauses concerned with violation and integrity.

- Views and indexes may also be manually created.

SQL

- After having successfully installed a suitable DBMS (e.g. MySQL) and all its dependencies we can proceed further to creating a database.

- In these slides we would be using MySQL for all our commands.

SQL

- After having successfully installed a suitable DBMS (e.g. MySQL) and all its dependencies we can proceed further to creating a database.

- In these slides we would be using MySQL for all our commands.

- To see a list of options provided by mysql, invoke it with the --help option

SQL – Connecting to and Disconnecting from the Server

- To connect to the server, you will usually need to provide a MySQL user name when you invoke mysql and, most likely, a password.

- If the server runs on a machine other than the one where you log in, you will also need to specify a host name.

shell> mysql -h *host* -u user -p
Enter password: ********

SQL – Connecting to and Disconnecting from the Server

- *host* and *user* represent the host name where your MySQL server is running and the user name of your MySQL account.

- Substitute appropriate values for your setup. The ******** represents your password; enter it when mysql displays the Enter password: prompt.

SQL – Connecting to and Disconnecting from the Server

- If you are logging in on the same machine that MySQL is running on, you can omit the *host*, and simply use the following:

```
shell> mysql -u user –p
```

- The mysql> prompt tells you that mysql is ready for you to enter SQL statements.

SQL – Connecting to and Disconnecting from the Server

- After you have connected successfully, you can disconnect any time by typing QUIT (or \q) at the mysql> prompt:

- On Unix, you can also disconnect by pressing Control+D.

SQL – Creating and Using a Database

- Before creating a database it is important to check if the name of the database already exists in previously created databases.

- To show existing databases run the query:

```
mysql> SHOW DATABASES;
```

SQL – Creating and Using a Database

- If the database you want to create already exists you can try to access it by using the query:

mysql> USE *database_name*;

- After that you can show all the tables that are in that database

mysql> SHOW TABLES;

SQL – Creating and Using a Database

- If the database you want to create does **not** exist you can go ahead and create the new database.

mysql> CREATE DATABASE *database_name*;

- After that you can go ahead and use that database.

mysql> USE *database_name*;

SQL – Creating and Using a Database

- If the database you want to create does exist and you wish to delete it first and then recreate it.

mysql> DROP DATABASE IF EXISTS *database_name*;
mysql> CREATE DATABASE *database_name*;

- After that you can go ahead and use that database.

mysql> USE *database_name*;

SQL – Creating and Using a Database

- If you only want to create the database when it does **not** exists you can run the following query.

mysql> CREATE DATABASE IF NOT EXISTS *database_name*;

- After that you can go ahead and use that database.

mysql> USE *database_name*;

SQL – Creating a Table

- Creating the database is the easy part, but at this point it is empty, as SHOW TABLES tells you:

```
mysql> SHOW TABLES;
```

- The harder part is deciding what the structure of your database should be: what tables you need and what columns should be in each of them.

SQL – Creating a Table

- Use a CREATE TABLE statement to specify the layout of your table:

```
mysql> CREATE TABLE tablename
       (column-definition-list);
```

where column-definition-list consists of a comma-separated list of column definition each with the format:
column-name type[additional clauses]

- The square brackets indicate optionality.
- The additional clauses are concerned with violation and integrity

SQL – Creating a Table – Example

- Let’s create a dummy table called pet

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner
VARCHAR(20),  species VARCHAR(20), sex CHAR(1), birth DATE,
death DATE);
```

- Once you have created a table, SHOW TABLES should produce some output:

```
mysql> SHOW TABLES;
```

SQL – Creating a Table – Example

- To verify that your table was created the way you expected, use a DESCRIBE statement:

```
mysql> DESCRIBE pet;
```

- You can use DESCRIBE any time, for example, if you forget the names of the columns in your table or what types they have.

SQL – Loading Data into a Table

- After creating your table, you need to populate it. The LOAD DATA and INSERT statements are useful for this.

SQL – Loading Data into a Table

- Suppose that your pet records can be described as shown below. (Observe that MySQL expects dates in 'YYYY-MM-DD' format; this may be different from what you are used to.)

```
mysql> INSERT INTO pet VALUES
('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

SQL – Loading Data into a Table

- To see the data in the table you can run the query:

```
mysql>SELECT * FROM pet;
```

(We would look at Select commands in detail later)

SQL – Loading Data into a Table

- You could create a text file dummy_data.txt containing one record per line, with values separated by tabs, and given in the order in which the columns were listed in the CREATE TABLE statement.

- For missing values (such as unknown sexes or death dates for animals that are still living), you can use NULL values.

- To represent these in your text file, use \N (backslash, capital-N).

Whistler Gwen bird \N 1997-12-09 \N

SQL – Loading Data into a Table

- To load the text file dummy_data.txt into the pet table, use this statement:

```
mysql> LOAD DATA LOCAL INFILE '/path/dummy_data.txt'
INTO TABLE pet;
```

- Note that the path is the same as the location of where the file is stored. Also note that it contains forward slashes and not back slashes.

SQL – Loading Data into a Table

- To see the all the data in the table you can run the SELECT query again:

```
mysql>SELECT * FROM pet;
```

SQL – Retrieving Information from a Table

- The SELECT statement is used to pull information from a table.

- The general form of the statement is:

```
mysql>SELECT what_to_select
        FROM which_table
        WHERE conditions_to_satisfy;
```

SQL – Retrieving Information from a Table

- what_to_select indicates what you want to see. This can be a list of columns, or * to indicate “all columns.”
- which_table indicates the table from which you want to retrieve data.
- The WHERE clause is optional. If it is present, conditions_to_satisfy specifies one or more conditions that rows must satisfy to qualify for retrieval.

SQL – Retrieving Information from a Table

- The simplest form of SELECT retrieves everything from a table:

```
mysql> SELECT * FROM pet;
```

- This form of SELECT is useful if you want to review your entire table, for example, after you've just loaded it with your initial data set.

SQL – Editing Information in a Table

- After viewing all records in the Pet Table you may happen to think that the birth date for Buffy doesn't seem quite right.
- Consulting your original pedigree papers, you find that the correct birth year should be 2001, not 2000.
- You can edit this in two ways

SQL – Editing Information in a Table

- One way is to delete all records and then edit the dummy_data.txt file and re-upload it.

```
mysql> DELETE FROM pet;
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO
TABLE pet;
```

SQL – Editing Information in a Table

- Another way is to fix only the erroneous record with an UPDATE statement:

```
mysql> UPDATE pet SET birth = ‘2000-08-08' WHERE
name = 'Buffy';
```

- The UPDATE changes only the record in question and does not require you to reload the table

SQL – Selecting Particular Rows

- As shown in the preceding examples, it is easy to retrieve an entire table by just omitting the WHERE clause from the SELECT statement.
- But typically you don't want to see the entire table, particularly when it becomes large.
- Instead, you're usually more interested in answering a particular question, in which case you specify some constraints on the information you want.

SQL – Selecting Particular Rows

- You can select only particular rows from your table.
- For example, if you want to verify the change that you made to Buffy's birth date, select Buffy’s record like this:

```
mysql> SELECT * FROM pet WHERE name = 'Buffy';
```

- The output confirms that the year is correctly recorded as 2000, not 2001.

SQL – Selecting Particular Rows

- String comparisons normally are case-insensitive, so you can specify the name as 'buffy', 'BUFFY', and so forth. The query result would still be the same.

SQL – Selecting Particular Rows

- You can specify conditions on any column, not just name.

- For example, if you want to know which animals were born during or after 2003, test the birth column:

```
mysql> SELECT * FROM pet WHERE birth >= '2003-01-01';
```

SQL – Selecting Particular Rows

- You can combine conditions, for example, to locate female dogs:

```
mysql> SELECT * FROM pet WHERE species = 'dog' AND
sex = 'f';
```

SQL – Selecting Particular Rows

- The preceding query uses the AND logical operator. There is also an OR operator:

```
mysql> SELECT * FROM pet WHERE species = 'hamster'
OR species = 'bird';
```

SQL – Selecting Particular Rows

- AND and OR may be intermixed, although AND has higher precedence than OR.

- If you use both operators, it is a good idea to use parentheses to indicate explicitly how conditions should be grouped:

```
mysql> SELECT * FROM pet WHERE (species = 'cat' AND sex = 'm') OR (species = 'dog' AND sex = 'f');
```

SQL – Selecting Particular Columns

- If you do not want to see entire columns from your table, just name the columns in which you are interested, separated by commas.

- For example, if you want to know when your animals were born, select the name and birth columns:

```
mysql> SELECT name, birth FROM pet;
```

SQL – Selecting Particular Columns

- To find out who owns the pets, use this query:

```
mysql> SELECT owner FROM pet;
```

- Notice that the query simply retrieves the owner column from each record, and some of them appear more than once.

- To minimize the output, retrieve each unique output record just once by adding the keyword DISTINCT:

```
mysql> SELECT DISTINCT owner FROM pet;
```

SQL – Selecting Particular Columns

- You can use a WHERE clause to combine row selection with column selection.

- For example, to get birth dates for dogs and cats only, use this query:

```
mysql> SELECT name, species, birth FROM pet WHERE
species = 'dog' OR species = 'cat';
```

SQL – Sorting Rows

- You may have noticed in the preceding examples that the result rows are displayed in no particular order.
- It is often easier to examine query output when the rows are sorted in some meaningful way.
- To sort a result, use an ORDER BY clause.

SQL – Sorting Rows

- To sort the animal birthdays by date you can run the query:

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
```

SQL – Sorting Rows

- On character type columns, sorting—like all other comparison operations—is normally performed in a case-insensitive fashion.
- This means that the order is undefined for columns that are identical except for their case.
- You can force a case-sensitive sort for a column by using BINARY like so: ORDER BY BINARY col_name.

SQL – Sorting Rows

- The default sort order is ascending, with smallest values first.

- To sort in reverse (descending) order, add the DESC keyword to the name of the column you are sorting by:

```
mysql> SELECT name, birth FROM pet ORDER BY birth DESC;
```

SQL – Sorting Rows

- You can sort on multiple columns, and you can sort different columns in different directions.

- For example, to sort by type of animal in ascending order, then by birth date within animal type in descending order (youngest animals first), use the following query:

```
mysql> SELECT name, species, birth FROM pet ORDER BY
species, birth DESC;
```

SQL – Date Calculations

- MySQL provides several functions that you can use to perform calculations on dates, for example, to calculate ages or extract parts of dates.
- To determine how many years old each of your pets is, use the TIMESTAMPDIFF() function.
- Its arguments are the unit in which you want the result expressed, and the two date for which to take the difference.

SQL – Date Calculations

- The following query shows, for each pet, the birth date, the current date, and the age in years.

- An *alias* (age) is used to make the final output column label more meaningful.

```
mysql> SELECT name, birth, CURDATE(),
TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age FROM pet;
```

SQL – Date Calculations

- The query works, but the result could be scanned more easily if the rows were presented in some order.

- This can be done by adding an ORDER BY name clause to sort the output by name:

```
mysql> SELECT name, birth, CURDATE(),
TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age FROM pet
ORDER BY name;
```

SQL – Date Calculations

- To sort the output by age rather than name, just use a different ORDER BY clause:

```
mysql> SELECT name, birth, CURDATE(),
TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age FROM pet
ORDER BY age;
```

SQL – Date Calculations

- A similar query can be used to determine age at death for animals that have died.
- You determine which animals these are by checking whether the death value is NULL.
- Then, for those with non-NULL values, compute the difference between the death and birth values:

```
mysql> SELECT name, birth, death,
TIMESTAMPDIFF(YEAR,birth,death) AS age FROM pet
WHERE death IS NOT NULL ORDER BY age;
```

SQL – Date Calculations

- The previous query, used death IS NOT NULL rather than death <> NULL because NULL is a special value that cannot be compared using the usual comparison operators. This is discussed later.

SQL – Date Calculations

- What if you want to know which animals have birthdays next month?
- For this type of calculation, year and day are irrelevant; you simply want to extract the month part of the birth column.
- MySQL provides several functions for extracting parts of dates, such as YEAR(), MONTH(), and DAYOFMONTH().
- MONTH() is the appropriate function here.

SQL – Date Calculations

- To see how it works, run a simple query that displays the value of both birth and MONTH(birth):

```
mysql> SELECT name, birth, MONTH(birth) FROM pet;
```

- Finding animals with birthdays in the upcoming month is also simple. Suppose that the current month is July. Then the month value is 7 and you can look for animals born in August (month 8) like this:

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 8;
```

SQL – Date Calculations

- There is a small complication if the current month is December. You cannot merely add one to the month number (12) and look for animals born in month 13, because there is no such month.

- Instead, you look for animals born in January (month 1).

SQL – Date Calculations

- You can write the query so that it works no matter what the current month is, so that you do not have to use the number for a particular month.

- DATE_ADD() enables you to add a time interval to a given date. If you add a month to the value of CURDATE(), then extract the month part with MONTH(), the result produces the month in which to look for birthdays:

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) =
MONTH(DATE_ADD(CURDATE(),INTERVAL 1 MONTH));
```

SQL – Date Calculations

- A different way to accomplish the same task is to add 1 to get the next month after the current one after using the modulo function (MOD) to wrap the month value to 0 if it is currently 12:

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) =
MOD(MONTH(CURDATE()), 12) + 1;
```

SQL – Date Calculations

- In the last query MONTH() returns a number between 1 and 12. And MOD(*something*,12) returns a number between 0 and 11.

- So the addition has to be after the MOD(), otherwise we would go from November (11) to January (1).

SQL – Working with NULL Values

- The NULL value can be surprising until you get used to it. Conceptually, NULL means “a missing unknown value” and it is treated somewhat differently from other values.

- To test for NULL, use the IS NULL and IS NOT NULL operators, as shown here:

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
```

SQL – Working with NULL Values

- You cannot use arithmetic comparison operators such as =, <, or <> to test for NULL.
- To demonstrate this for yourself, try the following query:

```
mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
```

- Because the result of any arithmetic comparison with NULL is also NULL, you cannot obtain any meaningful results from such comparisons.

SQL – Working with NULL Values

- In MySQL, 0 or NULL means false and anything else means true.
- The default truth value from a boolean operation is 1.

SQL – Working with NULL Values

- This special treatment of NULL is why, in the previous section, it was necessary to determine which animals are no longer alive using death IS NOT NULL instead of death <> NULL.
- Two NULL values are regarded as equal in a GROUP BY (to be discussed later)
- When doing an ORDER BY, NULL values are presented first if you do ORDER BY ... ASC and last if you do ORDER BY ... DESC.

SQL – Working with NULL Values

- A common error when working with NULL is to assume that it is not possible to insert a zero or an empty string into a column defined as NOT NULL, but this is not the case.

- These are in fact values, whereas NULL means “not having a value.” You can test this easily enough by using IS [NOT] NULL as shown:

```
mysql> SELECT 0 IS NULL, 0 IS NOT NULL, ' ' IS NULL, ' ' IS NOT NULL;
```

- Thus it is entirely possible to insert a zero or empty string into a NOT NULL column, as these are in fact NOT NULL.

SQL – Pattern Matching

- MySQL provides standard SQL pattern matching as well as a form of pattern matching based on extended regular expressions similar to those used by Unix utilities.
- SQL pattern matching enables you to use _ to match any single character and % to match an arbitrary number of characters (including zero characters).
- In MySQL, SQL patterns are case-insensitive by default.

SQL – Pattern Matching

- Some examples are shown here.
- You do not use = or <> when you use SQL patterns; use the LIKE or NOT LIKE comparison operators instead.
- To find names beginning with b:

```
mysql> SELECT * FROM pet WHERE name LIKE 'b%';
```

SQL – Pattern Matching

- To find names ending with fy

```
mysql> SELECT * FROM pet WHERE name LIKE '%fy‘;
```

SQL – Pattern Matching

- To find names containing a w:

```
mysql> SELECT * FROM pet WHERE name LIKE '%w%';
```

SQL - Pattern Matching

- To find names containing exactly five characters, use five instances of the _ pattern character:

```
mysql> SELECT * FROM pet WHERE name LIKE '_____';
```

SQL – Pattern Matching

- The other type of pattern matching provided by MySQL uses extended regular expressions.
- When you test for a match for this type of pattern, use the REGEXP and NOT REGEXP operators (or RLIKE and NOT RLIKE, which are synonyms).

SQL – Pattern Matching

The following list describes some characteristics of extended regular expressions:

- . matches any single character.

- A character class [...] matches any character within the brackets. For example, [abc] matches a, b, or c. To name a range of characters, use a dash. [a-z] matches any letter, whereas [0-9] matches any digit.

SQL – Pattern Matching

The list continues…..:

- * matches zero or more instances of the thing preceding it. For example, x* matches any number of x characters, [0-9]* matches any number of digits, and .* matches any number of anything.

- A REGEXP pattern match succeeds if the pattern matches anywhere in the value being tested. (This differs from a LIKE pattern match, which succeeds only if the pattern matches the entire value.)

SQL – Pattern Matching

The list continues…..:

- To anchor a pattern so that it must match the beginning or end of the value being tested, use ^ at the beginning or $ at the end of the pattern.

SQL – Pattern Matching

- To demonstrate how extended regular expressions work, the LIKE queries shown previously are rewritten here to use REGEXP.

- To find names beginning with b, use ^ to match the beginning of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^b';
```

SQL – Pattern Matching

- If you really want to force a REGEXP comparison to be case sensitive, use the BINARY keyword to make one of the strings a binary string.

- This query matches only lowercase b at the beginning of a name:

```
mysql> SELECT * FROM pet WHERE name REGEXP BINARY '^b';
```

SQL – Pattern Matching

- To find names ending with fy, use $ to match the end of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP 'fy$';
```

SQL – Pattern Matching

- To find names containing a w, use this query:

```
mysql> SELECT * FROM pet WHERE name REGEXP 'w';
```

- Because a regular expression pattern matches if it occurs anywhere in the value, it is not necessary in the previous query to put a wildcard on either side of the pattern to get it to match the entire value like it would be if you used an SQL pattern.

SQL – Pattern Matching

- To find names containing exactly five characters, use ^ and $ to match the beginning and end of the name, and five instances of . in between:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^.....$';
```

SQL – Pattern Matching

- You could also write the previous query using the {n} (“repeat-n-times”) operator:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^.{5}$';
```

SQL – Grouping using the GROUP BY Clause

- GROUP BY notionally subdivides a table into groups based on a nominated column or columns.

- Members of each group have the same value of the nominated column.

- Find out the highest salary in each Department

mysql> SELECT owner, MAX(birth) FROM pet GROUP BY owner;

SQL – Grouping using the GROUP BY Clause

- When the GROUP BY clause is used, the data that is selected must be one of the following:
  - A column or columns value that is constant within the group; this must be the GROUP BY column(s).
  - A value computed over the whole group; i.e., aggregate functions such as AVG,SUM, COUNT (to be discussed later)
  - Expressions involving combinations of the above

SQL – Grouping using the GROUP BY Clause

- When the GROUP BY clause is used, the data that is selected must be one of the following:
  - A column or columns value that is constant within the group; this must be the GROUP BY column(s).
  - A value computed over the whole group; i.e., aggregate functions such as AVG,SUM, COUNT (to be discussed later)
  - Expressions involving combinations of the above

SQL – Counting Rows

- Databases are often used to answer the question, “How often does a certain type of data occur in a table?”

- For example, you might want to know how many pets you have, or how many pets each owner has, or you might want to perform various kinds of census operations on your animals.

SQL – Counting Rows

- Counting the total number of animals you have is the same question as “How many rows are in the pet table?” because there is one record per pet.

- COUNT(*) counts the number of rows, so the query to count your animals looks like this:

```
mysql> SELECT COUNT(*) FROM pet;
```

SQL – Counting Rows

- Earlier, you retrieved the names of the people who owned pets.

- You can use COUNT() if you want to find out how many pets each owner has:

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
```

SQL – Counting Rows

- The preceding query uses GROUP BY to group all records for each owner.
- The use of COUNT() in conjunction with GROUP BY is useful for characterizing your data under various groupings.
- The following examples show different ways to perform animal census operations.

SQL – Counting Rows

- Number of animals per species:

```
mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
```

SQL – Counting Rows

- Number of animals per sex:

```
mysql> SELECT sex, COUNT(*) FROM pet GROUP BY sex;
```

SQL – Counting Rows

- Number of animals per combination of species and sex:

```
mysql> SELECT species, sex, COUNT(*) FROM pet GROUP BY species, sex;
```

SQL – Counting Rows

- You need not retrieve an entire table when you use COUNT().
- For example, the previous query, when performed just on dogs and cats, looks like this:

```
mysql> SELECT species, sex, COUNT(*) FROM pet WHERE
species = 'dog' OR species = 'cat' GROUP BY species, sex;
```

SQL – Counting Rows

- Or, if you wanted the number of animals per sex only for animals whose sex is known:

- mysql> SELECT species, sex, COUNT(*) FROM pet WHERE sex IS NOT NULL GROUP BY species, sex;

SQL – Counting Rows

- If you name columns to select in addition to the COUNT() value, a GROUP BY clause should be present that names those same columns. Otherwise an error would occur

- If the ONLY_FULL_GROUP_BY SQL mode is enabled, an error occurs:

```
mysql> SET sql_mode = 'ONLY_FULL_GROUP_BY';
mysql> SELECT owner, COUNT(*) FROM pet;
```

SQL – Counting Rows

- If ONLY_FULL_GROUP_BY is not enabled, the query is processed by treating all rows as a single group, but the value selected for each named column is **indeterminate**.

- The server is free to select the value from any row:

```
mysql> SET sql_mode = ' ';
mysql> SELECT owner, COUNT(*) FROM pet;
```

END

THANKS!

Any questions?

You can find me at elielkeelson@gmail.com &
ekeelson@knust.edu.gh