Relational Database Management Systems for Epidemiologists:

SQL Part I

Outline

- SQL Basics
- Retrieving Data from a Table
- Operators and Functions

What is SQL?

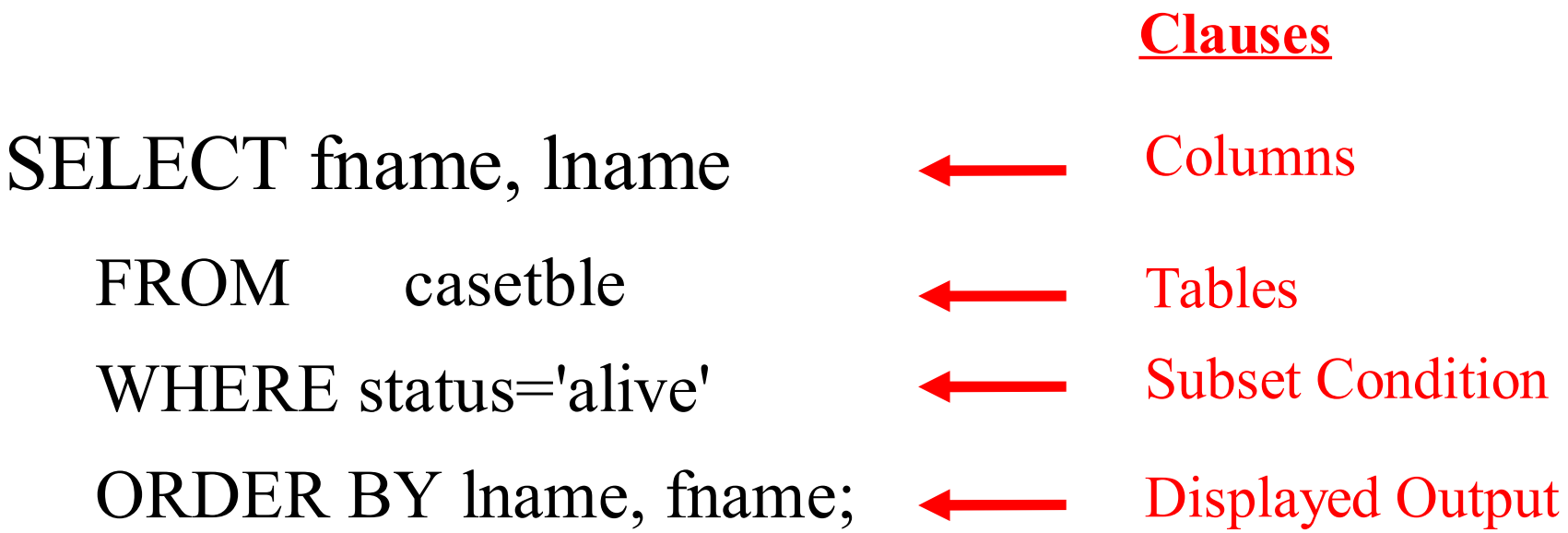
- SQL is the standard programming language to create, update, delete, and retrieve data stored in a RDMS.
- SQL is defined by rules of syntax (words and symbols allowed) and semantics (meaning).
- Three versions:
  - ANSI-89 SQL
  - ANSI-92 SQL
  - ANSI SQL-99 (newer)

SQL Syntax

SQL Statement

```
SELECT fname, lname
    FROM      casetble
    WHERE status='alive'
    ORDER BY lname, fname;
```

SQL Syntax


Clauses
SELECT fname, lname
Columns
FROM casetble
Tables
WHERE status='alive'
Subset Condition
ORDER BY lname, fname;
Displayed Output

SQL Syntax

```
SELECT fname, lname
    FROM      casetble
    WHERE status='alive'
    ORDER BY lname, fname;
```

Keywords

SQL Syntax

```
SELECT fname, lname
    FROM      casetble
    WHERE status='alive'
    ORDER BY lname, fname;
```

Identifiers

SQL Syntax

```
SELECT fname, lname
    FROM      casetble
    WHERE status='alive'
    ORDER BY lname, fname;
```

Terminating
semicolon

Elements of SQL Style

- SQL statements
  - can be in uppercase or lowercase (case insensitive).
  - can extend across multiple lines, so long as you do not split words or quoted strings in two.
- To improve readability and maintenance,
  - Begin each SQL statement on a new line.
  - Use uppercase for SQL keywords (eg, SELECT, NULL, CHARACTER).
  - Indent with a fixed number of spaces (eg, four).

SELECT

- SELECT retrieves rows, columns, and derived values from one or more tables.
- Syntax:

SELECT *column(s)*
FROM *table(s)*
[JOIN *join(s)*]
[WHERE *search_condition(s)*]
[GROUP BY *grouping_column(s)*]
[HAVING *search_condition(s)*]
[ORDER BY *sort_column(s)*];

SELECT Example

```
SELECT fname, lname
  FROM patients;
```

SELECT Example

```
SELECT *
   FROM patients;
```

AS

- The AS statement can be used to create a column alias (an alternative name/identifier) that you specify to control how column headings are displayed in a result.
- Syntax:

SELECT *column1* AS *alias1,*
column2 AS *alias2,*
...
columnN AS *aliasN*
FROM *table;*

AS Example

```
SELECT fname AS “First Name”,
       lname AS “Last Name”
  FROM patients;
```

DISTINCT

- Results of queries oftentimes contain duplicate values for a particular column.
- The DISTINCT keyword eliminates duplicate rows from a result.
- Syntax:

SELECT DISTINCT *column(s)*

FROM *table(s);*

DISTINCT Example

```
SELECT DISTINCT fname, lname
  FROM patients;
```

Notes on DISTINCT

- If the SELECT DISTINCT clause contains more than one column, the values of all the columns specified determined the uniqueness of rows.
- All NULLS are considered duplicates of each other (even though they technically never equal each other since the values are unknown).
- DISTINCT can be used to find typographical errors in text fields.

ORDER BY

- Rows in a query result are unordered and should be viewed as arbitrary.
- The ORDER BY clause can be used to sort rows by a specified column (or columns) in ascending/descending order.
- Syntax:

```
SELECT DISTINCT column(s)
  FROM table
  ORDER BY sort_column1 [ASC | DESC],
                ...
                sort_columnN [ASC | DESC];
```

ORDER BY Example

```
SELECT fname, lname, city, state
   FROM patients
   ORDER BY state ASC,
                    city DESC;
```

ORDER BY Example

- Resulting queries can be sorted by relative column positions as in the following example.

```
SELECT fname, lname, city, state
  FROM patients
  ORDER BY 4 ASC,
           3 DESC;
```

ORDER BY Example

- Use column aliases in the ORDER BY clause when aliases are used.

```
SELECT fname AS “First Name”,
       lname AS “Last Name”,
       city,
       state
  FROM patients
  ORDER BY state,
           “Last Name” DESC,
           “First Name” ASC;
```

Notes on ORDER BY

- The default for sorting is ascending order.
- The sorting column(s) do not need to appear in the resulting query.
- If the ORDER BY columns do not identify each row uniquely, rows with duplicate values will be listed in arbitrary order.
- A DBMS uses a collating sequence (or collation) to determine the order in which characters are sorted.

WHERE

- The WHERE clause can be used to filter unwanted rows in a result (ie, yield a subset of all rows in the result with a specified condition).
- Syntax:

SELECT *column(s)*

FROM *table*

WHERE *test_column operator value;*

Types of Conditions

Condition	SQL Operators
Comparison	=, <>, <, <=, >, >=
Pattern Matching	LIKE
Range Filtering	BETWEEN
List Filtering	IN
Null Test	IS NULL

Comparison Operators

Operator	Descriptors
=	Equal to
<>	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

WHERE Example

```
SELECT caseid, fname, lname
    FROM patients
    WHERE lname = 'Johnson';
```

WHERE Example

```
SELECT caseid, fname, lname
    FROM patients
    WHERE lname <> 'Johnson';
```

WHERE Example

```
SELECT caseid, fname, lname
    FROM patients
    WHERE datevar >= '2001-01-10';
```

WHERE Example

```
SELECT caseid,
       fname,
       lname,
       MONTH(datevar) AS “Month”,
       DAY(datevar) AS “Day”
  FROM patients
  WHERE “Day” >= “Month”;
```

Notes on WHERE

- Occasionally, you may need to specify multiple conditions in a single WHERE clause.
- You can use the AND, OR or NOT operators to combine two or more conditions into a compound condition.
  - AND, OR, and NOT operators are known as Boolean operators; they are designed to work with “truth” values: true, false, and unknown.

AND Example

```
SELECT caseid,
       fname,
       lname,
       MONTH(datevar) AS “Month”
  FROM patients
  WHERE “Month”=12 AND lname='Johnson';
```

Truth Table for Two Conditions

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

OR Example

```
SELECT caseid,
       fname,
       lname,
       MONTH(datevar) AS “Month”
  FROM patients
  WHERE “Month”=10
        OR “Month”=11
        OR “Month”=12;
```

Truth Table for Two Conditions

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

NOT Example

```
SELECT caseid,
       fname,
       lname,
       MONTH(datevar) AS “Month”
  FROM patients
  WHERE NOT (“Month”=10);
```

NOT Example

```
SELECT caseid,
       fname,
       lname,
       MONTH(datevar) AS “Month”
  FROM patients
  WHERE NOT (“Month”=10 OR “Month”=11
                OR “Month”=12);
```

NOT Truth Table

Condition	NOT Condition
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

Equivalent Conditions

This condition	Is equivalent to:
NOT (p AND q)	(NOT p) AND (NOT q)
NOT (p OR q)	(NOT p) OR (NOT q)
NOT (NOT p)	p

Order of Evaluation

- When you use multiple logical operators in a compound condition, the order of evaluation is:

  (1) NOT

  (2) AND

  (3) OR.

- According to these rules, the following condition

  x AND NOT y OR z

  is equivalent to

  (x AND (NOT y)) OR z.

- Use parentheses to make the evaluation order more clear (whatever is in the parenthesis gets evaluated first)!

LIKE

- You can use the LIKE operator to retrieve partial information for a character string (not numbers or date/times) rather than an exact value.
- LIKE uses a pattern that values are matched against.
  - Pattern: quoted string that contains the literal characters to match and any combination of wildcards.
  - Wildcards: special characters used to match parts of a value.
- You can negate a LIKE condition with NOT LIKE.

Wildcard Operators

Operator	Matches
%	Any string of zero or more characters
_	Any one character

LIKE Example

- Syntax:

  SELECT *columns*

  FROM *table*

  WHERE *test_column* [NOT] LIKE *'pattern';*

- *Example:*

  SELECT fname, lname

  FROM patients

  WHERE lname LIKE 'John%';

BETWEEN

- Use the BETWEEN clause to determine whether a given value falls within a specified range.
- BETWEEN works with character strings, numbers, and date/times.
- The range contains a low and high value, separated by AND (inclusive).
- You can negate a BETWEEN condition with NOT BETWEEN.

BETWEEN Example

- Syntax:

SELECT *columns*

FROM *table*

WHERE *test_column* BETWEEN
low_value AND *high value;*

- *Example:*

SELECT fname, lname

FROM patients

WHERE zip BETWEEN 94510 AND 94515;

Equivalent Statements

- SELECT fname, lname

  FROM patients

  WHERE zip BETWEEN 94510 AND 94515;
- SELECT fname, lname

  FROM patients

  WHERE (zip >= 94510) AND (zip<=94515);

List Filtering with IN

- The IN clause can be used to display records with any value in a specified list for a particular column.
- Syntax:

SELECT *columns*

FROM *table*

WHERE *test_column* [NOT] IN

(value1, value2, ...);

IN Example

- Example:

```
SELECT fname, lname, state
    FROM patients
    WHERE state IN ('CA', 'NY');
```

IS NULL

- Recall: NULLs represent missing or unknown values.
- IS NULL can be used to find records with NULL values
- Syntax:

SELECT *columns*

FROM *table*

WHERE *test_column* IS [NOT] NULL;

IS NULL Example

```
SELECT caseid,
       fname,
       lname
  FROM patients
 WHERE occupation IS NULL;
```

Additional Operators & Functions

- Arithmetic Operations (+. -, *, /)
- Concatenation (||)
- Extracting Text (SUBSTRING())
- Changing Case (UPPER() and LOWER())
- Trimming Characters (TRIM())
- Length of a String (CHARACTER_LENGTH() or LEN())
- Position of a Substring (POSITION())

Next Time

- Summarizing and Grouping Data
  - Aggregate functions (MIN, MAX, SUM, AVG, COUNT)
  - Grouping rows with GROUP BY
  - Filtering groups with HAVING
- Joins
  - Cross, natural, inner, left outer, right outer, full outer, self-join