

# SQL For Any IT Professional

- Overview of SQL and the Relational Database
- “Flying through the Basics”



# Course Overview

Welcome to a fast-paced overview of SQL!

The primary goal is to gain an understanding of SQL and the Relational Database Model, while learning some standard SQL syntax, to be able to:

- design a simple database
  - create simple database tables
  - get data into the database, and manage the data
  - effectively retrieve useful information from the database
- 
- It is more important to focus on the concepts than exact syntax at this point.
  - SQL concepts are universal (national and international), syntax varies.
  - If you understand the concepts, syntax is very easy to learn!

Oracle SQL\*Plus is used for examples in this course.

# Tell Me About You

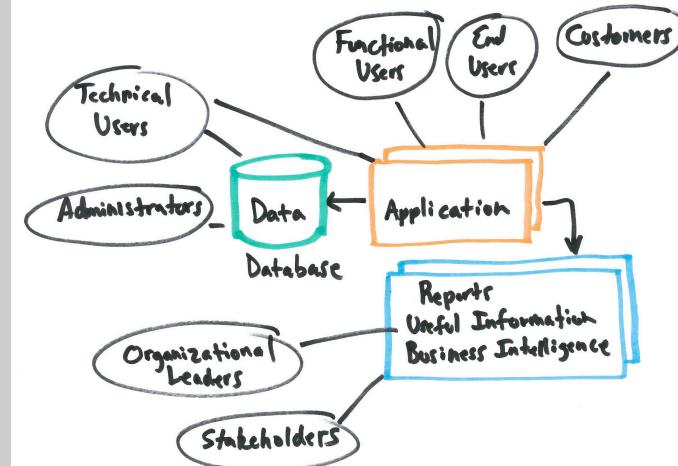
## Polling Question (radio buttons)

Which of the following best describes your profession related to SQL?

- Non-IT related
- Non-SQL related
- End User, Functional User
- Technical Manager
- Manager, Organization Leader, Stakeholder
- Developer, Programmer
- Database Administrator
- Business Analyst, Data Analyst
- Business Intelligence (BI) developer, analyst
- Other technical

# User Communities, Data, Applications

- Mountains of information must become “Usable Data.”
- Data is critical to the success of any organization.
  - End users, consumers, customers, customers’ customers
- Users depend on data every day.
  - Organizational leaders
  - Managers
  - Technical users
  - Administrators
  - Functional and end users
  - Stakeholders
  - Customers



# SQL: The Standard Language

- Structured Query Language (SQL)
  - the standard language used to communicate with any relational database.
- SQL is English-like.
- National and international standards exist that make SQL universal, easy to use, and adaptable to most modern databases and applications.
- Vendor implementations (flavors)
  - Oracle, Microsoft SQL Server, PostGreSQL, MySQL, IBM DB2, SAP Sybase, etc.
  - Oracle 11g Express and Oracle SQL\*Plus are used for examples in this class

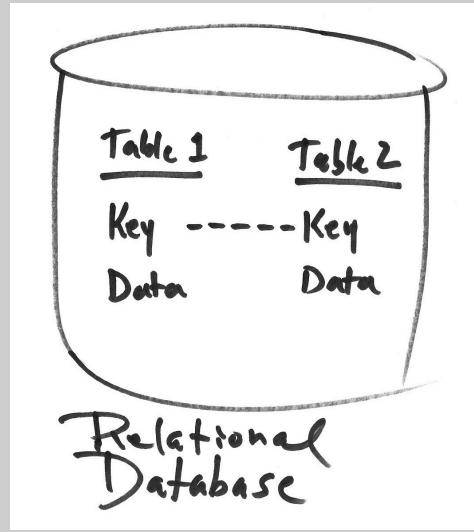
# Standard SQL Components

- **Data Definition Language (DDL)**
  - Used to design databases and organize data
  - Used to create and modify database structures (tables)
- **Data Manipulation Language (DML)**
  - Used to get data into a database tables
  - Used to manage data in database tables
- **Data Query Language (DQL)**
  - Used to get data out of the database
  - Used to generate useful reports
  - Used to return data into end user applications

# The Relational Database

- RDBMS

- Data sets divided into logical units called “tables.”
- Tables are related to one another directly or indirectly via one or more columns with like data.



# Basic Relational Database Design

- A table is comprised of rows and columns as shown below.

<u>Table : Employees</u>		
<u>Id</u>	<u>Last Name</u>	<u>First Name</u>
1	Smith	Mary
2	Jones	Bob
3	Williams	Steve
4	Mitchell	Kelly
5	Burk	Ron

# Basic Relational Database Design

- The following figure shows how two or more tables can be related in a relational database.
- Referential Integrity is promoted through the use of primary and foreign keys.

<u>Employees</u>			<u>Dependents</u>	
<u>Id</u>	<u>Last Name</u>	<u>First Name</u>	<u>Id</u>	<u>Name</u>
1	Smith	Mary	1	John
2	Jones	Bob	1	Mary
3	Williams	Steve	3	Ann
4	mitchell	Kelly	4	Mark
5	Burk	Ron	4	Laura

*Primary Key*

*Foreign Key*

4	4	4
5	4	5

# Basic Relational Database Design

Simply put...

- A relational database is a database that is logically organized into multiple tables, that may or may not be related to one another.
- A table consists of one or more columns, or fields, and one or more rows of data.
- Tables are related to one another through common columns, that are typically defined by constraints such as Primary Keys and Foreign Keys.

# Advantages of the Relational Database

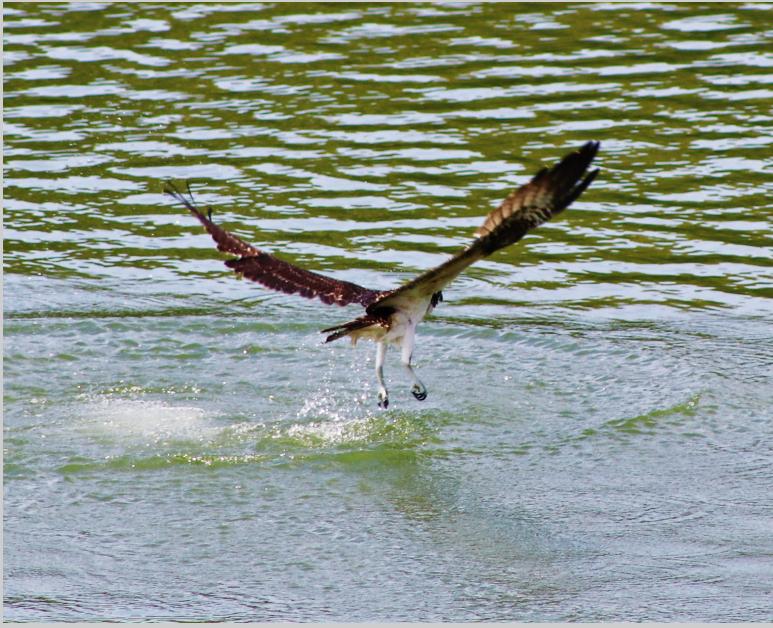
## RDBMS Advantages over Flat-file and Non-relational models:

- Better organization and more flexible
- Easily link information in multiple tables
- Much easier to maintain the integrity of data (referential integrity)
- Simplicity
- Data is easier to retrieve

That is why you use SQL. SQL is an English-like language that allows you to create and manage a relational database, and easily and effectively get data out of the database for a specific purpose.

# SQL For Any IT Professional

- Practical Application of Basic SQL Concepts



# SQL: Database Design Language (DDL)

- DDL is used to design and build database objects, such as:
  - Tables
  - Indexes
- Table definitions include:
  - Columns
  - Rows
  - Constraints (Primary keys, foreign keys, unique constraints)
- Each column in a table is defined:
  - By a specified “Data type”
  - Whether or not it requires a value (NULL, NOT NULL)
- Indexes can be created strategically on columns to improve the performance of data being returned.

# Thinking Through a Simple DB Design

- Before using DDL commands, you must think through your database design.
- In this class, we will use a database about “Birds.”
- Here is some basic information about the data:
  - Each bird will have some standard information, or data.
  - Each bird may have photographs, taken from different locations.
  - Information about each bird’s diet will be stored.
  - Information about each bird’s migration habits will be stored.
  - Information about the types of nests each bird builds will be stored.
  - Each bird may have one or more nickname.

# Thinking Through a Simple DB Design

We can begin to diagram out our design as follows.

## Birds

Name

Basic Info (size, wingspan, etc.)

Foods

Migration

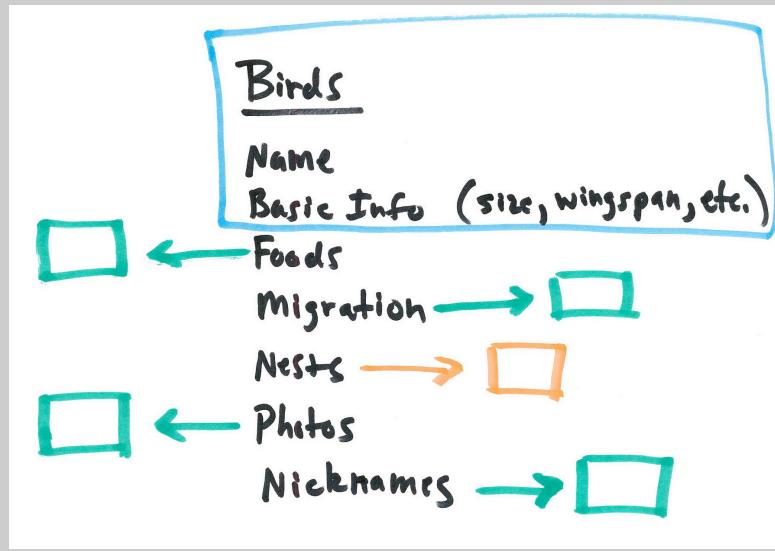
Nests

Photos

Nicknames

# Thinking Through a Simple DB Design

Now, in order to simply our design logically, we are going to separate some of the attributes of birds out into their own tables:



# Thinking Through a Simple DB Design

Here is why we are breaking “Birds” apart into multiple tables:

- Some information about each bird is simple, such as measurements, so we are leaving those in the “Birds” table. We only want one row of data in the “Birds” table for each bird. This will support referential integrity.
- WE DO NOT WANT DUPLICATE RECORDS ANYWHERE.
- “Foods” will be its own table, because each bird might eat many foods, and each food might have many birds that eat it.
- “Migration” will be its own table, because each bird might migrate to more than one location, and each location might have many birds that migrate to it.
- Likewise, each bird may have many “Photos” and “Nicknames.”
- Each bird only has one type of nest, but the reason we separate “Nest” into its own table is because each “Nest” type might have many different birds associated with it.

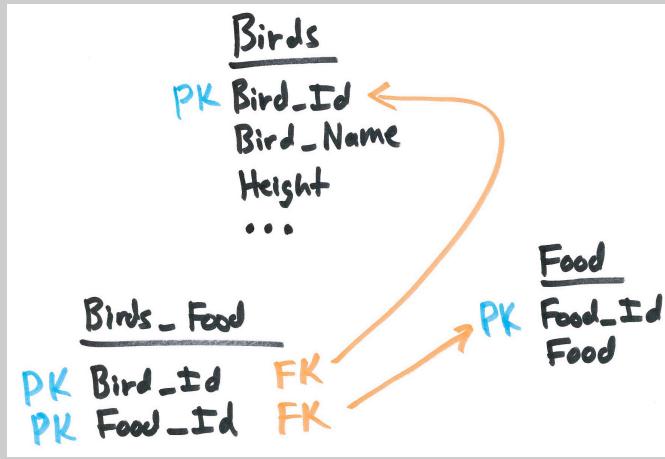
# A Simple Entity Relationship Diagram

- Separating data sets into multiple tables is called “Normalization.”
- The following figure shows how we begin to form a relationship between tables using keys.



# Normalizing Your Database Design

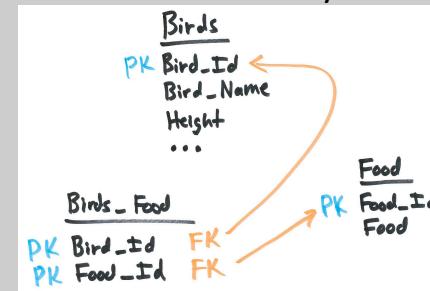
- The problem with the previous design is that we will end up with duplicate values for “Food” in the “Food” table.
- The following figure shows how we can resolve this problem by designing a table specifically to join the “Birds” and “Food” tables, ensuring not duplicate records exist in either “Birds” or “Food.”



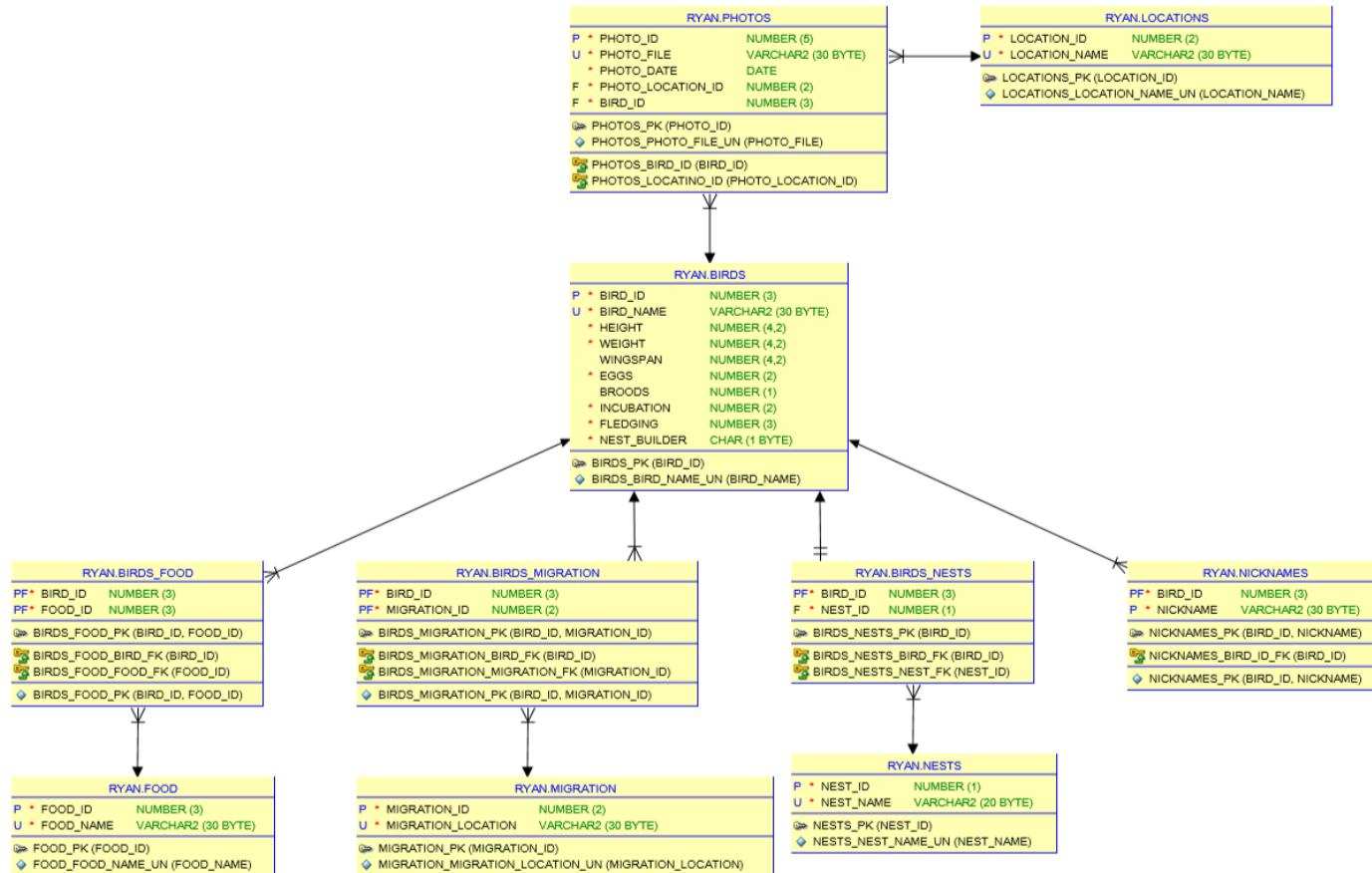
# Normalizing Your Database Design

What this Entity Relationship Diagram (ERD) says:

- “Bird\_Id” is the primary key in “Birds”, ensuring each “Bird\_Name” is unique.
- “Food\_Id” is the primary key in “Foods”, ensuring each “Food\_Name” is unique.
- The combination of “Bird\_Id” and “Food\_Id” is the primary key in “Birds\_Food”, ensuring each combination in the table is unique.
- “Bird\_Id” and “Food\_Id” in “Birds\_Food” are foreign keys, referencing their corresponding keys in the other two tables. There cannot be a “Bird\_Id” in “Birds\_Food” unless there is a corresponding “Bird\_Id” in “Birds.” There must be a record for each bird in the “Birds” table before it exists anywhere else in the database.
- This is REFERENTIAL INTEGRITY.



# An Entity Relationship Diagram for “Birds”



# Summarizing the ERD

You will notice the following details in the ERD:

- Each table is related to one or more tables through keys.
- There are lines between tables specifying the type of relationship.
- Primary keys, foreign keys, and unique key constraints are identified.
- Each table has multiple columns.
- Each column has been specified to adhere to a certain data type (we will discuss that next).
- Note that the database user “RYAN” owns these table, identified by RYAN.BIRDS, etc.
- This group of tables owned by “RYAN” is called a database “Schema.”
- This ERD was created using Oracle SQL Developer.

Next, we will use this database design model to create our tables!

# Creating Your Tables

- To create database objects such as tables, you must have a database user account, and have been granted permission by the database administrator to create objects.
- In SQL, tables are created using the CREATE TABLE statement, in its most simplistic state, as follows:

```
CREATE TABLE table_name
( column1    data_type    [NOT NULL] ,
  column2    data_type    [NOT NULL] ,
  column3    data_type    [NOT NULL] );
```

# Creating Your Tables

- Data Types: for the purpose of this class, we will use the following basic SQL data types:
  - NUMBER (X,Y) numeric value with a precision of X and scale of Y
  - CHAR (X) fixed length character value of X
  - VARCHAR (X) variable length character value, with a max length of X
  - DATE a date value
- You specify whether a column in a row of data must or must not contain data using:
  - NULL (means the value of the data can be left blank, or NULL)
  - NOT NULL (means a data value must be inserted into the associated column for every row of data)

# Creating Your Tables

Standard SQL/Oracle syntax to create two related tables:

```
create table birds
(bird_id          number(3)          not null    primary key,
bird_name        varchar(30)         not null    unique,
height           number(4,2)         not null,
weight           number(4,2)         not null,
wingspan         number(4,2)         null,
eggs             number(2)          not null,
broods            number(1)          null,
incubation       number(2)          not null,
fledging          number(3)          not null,
nest_builder     char(1)           not null);

create table nicknames
(bird_id          number(3)          not null,
nickname         varchar(30)         not null,
constraint nicknames_pk primary key (bird_id, nickname),
constraint nicknames_bird_id_fk foreign key (bird_id) references birds (bird_id));
```

# More Basic DDL Statements to Manage Tables

## Dropping Tables

```
DROP TABLE table_name [restrict | cascade];
```

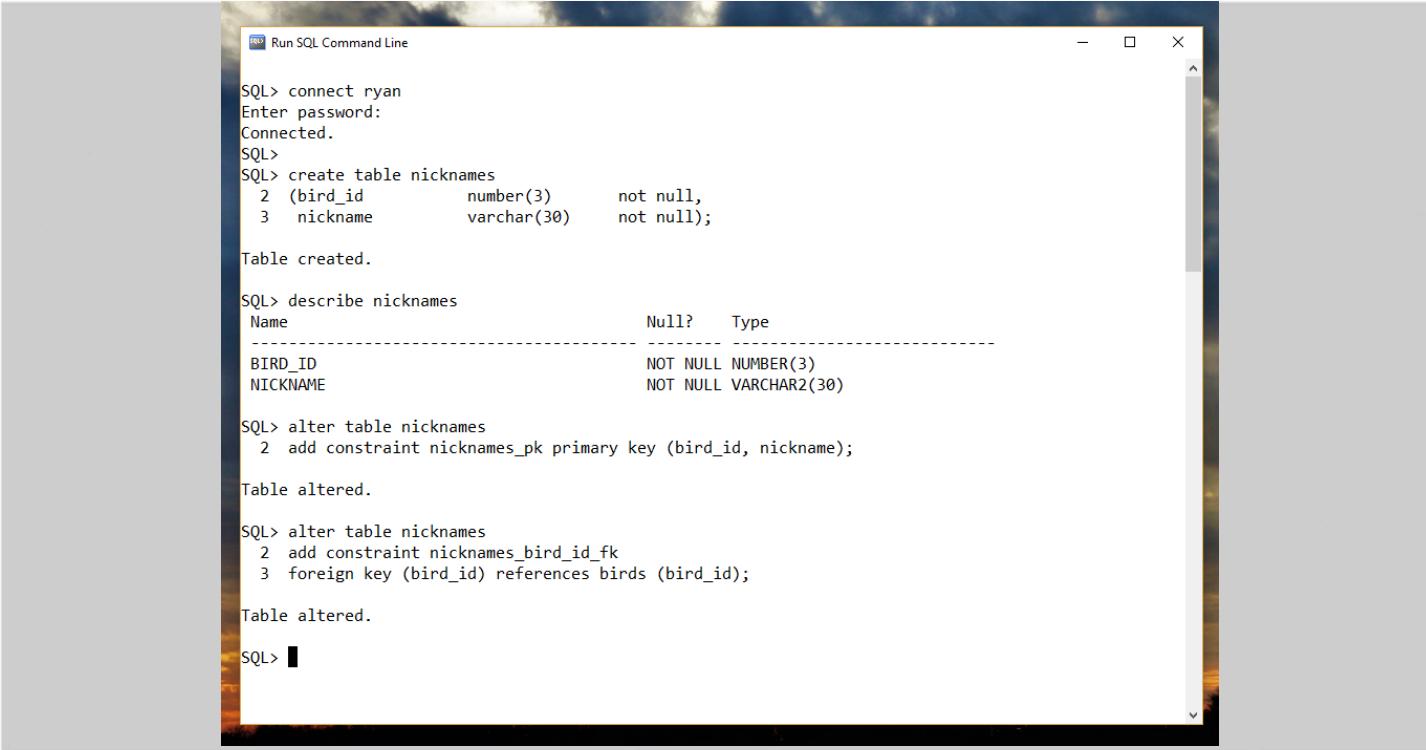
## Modifying Tables

```
ALTER TABLE table_name [modify] [column column_name] [datatype |NULL NOT NULL]  
[RESTRICT | CASCADE]  
[ADD | DROP] [constraint constraint_name]  
[PRIMARY KEY (column1, column2, ...)]  
[FOREIGN KEY (column_name) REFERENCES table_name (column_name)]  
[ADD] [column column_name definition];
```

## Truncating the Data from Tables

```
TRUCATE TABLE table_name;
```

# Example Using Oracle SQL\*Plus



The screenshot shows a Windows command-line interface window titled "Run SQL Command Line". The session starts with connecting to a user named "ryan". Then, a table named "nicknames" is created with two columns: "bird\_id" (number(3)) and "nickname" (varchar(30)). Both columns are defined as NOT NULL. After creating the table, its structure is described, showing the column names, nullability, and data types. Next, the table is altered by adding a primary key constraint named "nicknames\_pk" that covers both columns. Finally, a foreign key constraint named "nicknames\_bird\_id\_fk" is added to the "bird\_id" column, referencing the "bird\_id" column in another table. The session ends with a prompt for the next command.

```
SQL> connect ryan
Enter password:
Connected.
SQL>
SQL> create table nicknames
  2  (bird_id          number(3)      not null,
  3    nickname        varchar(30)    not null);

Table created.

SQL> describe nicknames
Name                  Null?    Type
-----              -----
BIRD_ID                NOT NULL NUMBER(3)
NICKNAME               NOT NULL VARCHAR2(30)

SQL> alter table nicknames
  2  add constraint nicknames_pk primary key (bird_id, nickname);

Table altered.

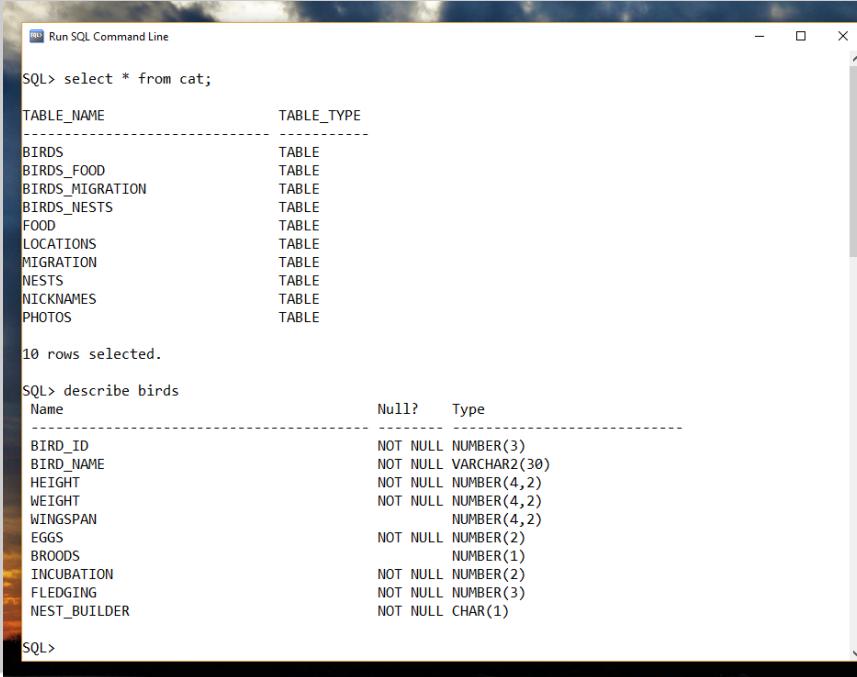
SQL> alter table nicknames
  2  add constraint nicknames_bird_id_fk
  3  foreign key (bird_id) references birds (bird_id);

Table altered.

SQL> █
```

# Finding Information About Your Tables

Getting a list of your tables from the Oracle catalog:



The screenshot shows a Windows command-line interface titled "Run SQL Command Line". The user has run two SQL commands: "select \* from cat;" and "describe birds".

```
SQL> select * from cat;
-----+-----+
TABLE_NAME      TABLE_TYPE
-----+-----+
BIRDS           TABLE
BIRDS_FOOD       TABLE
BIRDS_MIGRATION TABLE
BIRDS_NESTS      TABLE
FOOD             TABLE
LOCATIONS        TABLE
MIGRATION        TABLE
NESTS            TABLE
NICKNAMES        TABLE
PHOTOS           TABLE
-----+-----+
10 rows selected.

SQL> describe birds
Name          Null?    Type
-----+-----+-----+
BIRD_ID        NOT NULL NUMBER(3)
BIRD_NAME       NOT NULL VARCHAR2(30)
HEIGHT          NOT NULL NUMBER(4,2)
WEIGHT          NOT NULL NUMBER(4,2)
WINGSPAN        NUMBER(4,2)
EGGS            NOT NULL NUMBER(2)
BROODS           NUMBER(1)
INCUBATION      NOT NULL NUMBER(2)
FLEDGING         NOT NULL NUMBER(3)
NEST_BUILDER    NOT NULL CHAR(1)
-----+-----+
SQL>
```

# Summary and Q&A

- Good, clean data is critical to any organization.
- The RDBMS has been one of the most effective database models for decades.
- SQL is the standard language to communicate with a RDBMS.
- There are many vendor implementations of standard SQL.
- The main components of SQL are:
  - Data Definition Language (DDL)
  - Data Manipulation Language (DML)
  - Data Query Language (DQL)
- Databases can be designed and normalized using an Entity Relationship Diagram (ERD).
- There are many vendor implementations of standard SQL.
- The basic DDL statements to create and manage database objects are:
  - CREATE TABLE
  - ALTER TABLE
  - DROP TABLE
- Remember that the syntax of each vendor's SQL implementation my vary slightly.

# SQL For Any IT Professional

- Transactions: Overview of Managing Data in SQL



# Data Manipulation Language (DML)

- Once your database is designed and tables and other objects are created using DDL, Data Manipulation Language (DML) statements are used to load and manage data.
- In this section, you will learn how to use:
  - DML statements
    - INSERT
    - UPDATE
    - DELETE
  - Transactional control commands
    - COMMIT
    - ROLLBACK

# Inserting Data into Tables

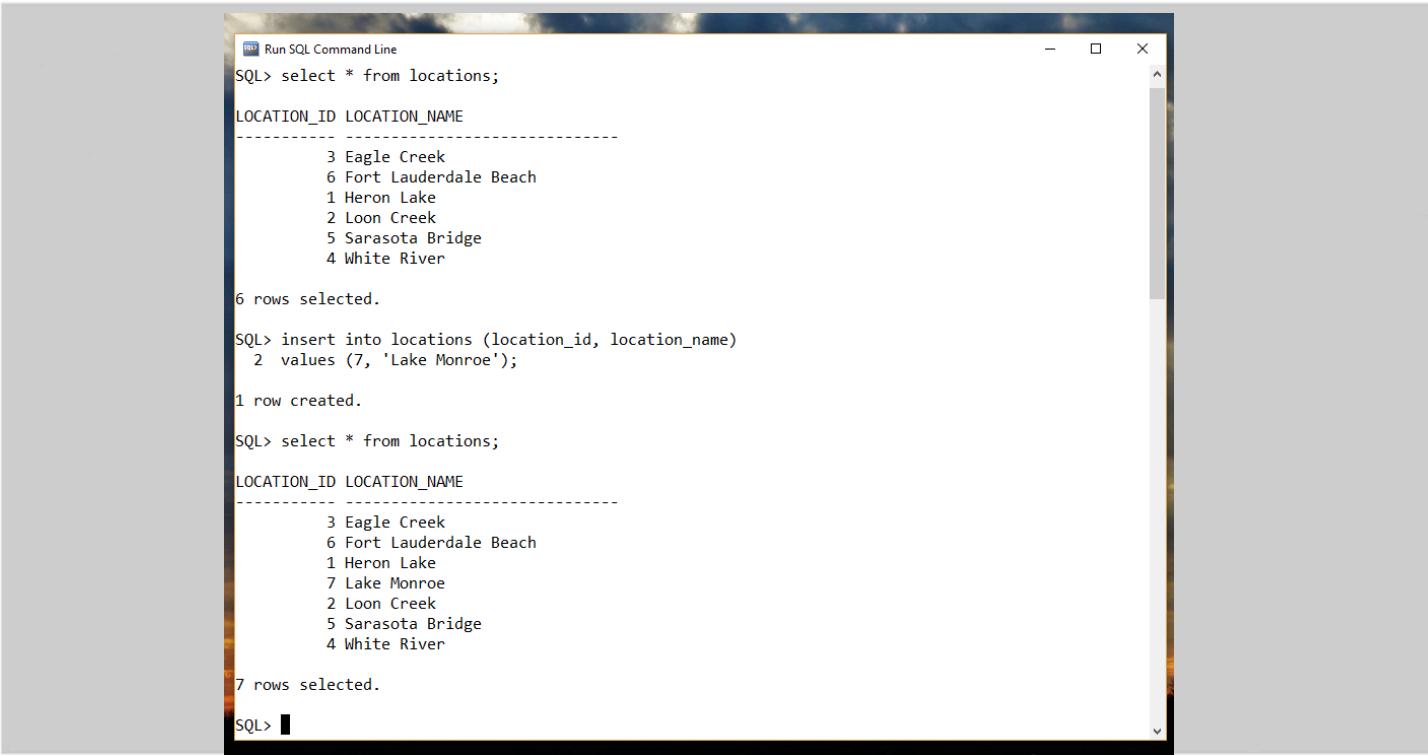
## INSERT Statement

- Used to insert a single row of data into a table
- Commands in SQL are typically not case-sensitive
- Case matters when inserting data values into a table

## Syntax:

```
INSERT INTO table_name (column1, column2, ...)  
VALUES ('value1, value2, ...);
```

# Inserting Data into Tables



The screenshot shows a Windows-style application window titled "Run SQL Command Line". Inside, a SQL session is displayed:

```
SQL> select * from locations;
LOCATION_ID LOCATION_NAME
-----
3 Eagle Creek
6 Fort Lauderdale Beach
1 Heron Lake
2 Loon Creek
5 Sarasota Bridge
4 White River

6 rows selected.

SQL> insert into locations (location_id, location_name)
  2 values (7, 'Lake Monroe');

1 row created.

SQL> select * from locations;
LOCATION_ID LOCATION_NAME
-----
3 Eagle Creek
6 Fort Lauderdale Beach
1 Heron Lake
7 Lake Monroe
2 Loon Creek
5 Sarasota Bridge
4 White River

7 rows selected.

SQL> █
```

The session starts by selecting all rows from the "locations" table, displaying six rows. Then, it inserts a new row with location\_id 7 and location\_name 'Lake Monroe', resulting in one row being created. Finally, it selects all rows again, showing a total of seven rows, including the newly inserted one.

# Updating/Modifying Data in Tables

## UPDATE Statement

- Used to update the data values of one or more columns in one or more rows of data in a table
- Commands in SQL are typically not case-sensitive
- Case matters when inserting data values into a table

## Syntax:

```
UPDATE table_name
    SET column1 = 'value1', column2 = 'value2'
        [ WHERE condition ];
```

# Updating/Modifying Data in Tables



The screenshot shows a Windows command-line interface window titled "Run SQL Command Line". The window displays the following SQL session:

```
Run SQL Command Line
LOCATION_ID LOCATION_NAME
-----
3 Eagle Creek
6 Fort Lauderdale Beach
1 Heron Lake
7 Lake Monroe
2 Loon Creek
5 Sarasota Bridge
4 White River

7 rows selected.

SQL> update locations
2 set location_name = 'Lake Michigan'
3 where location_id = 7;

1 row updated.

SQL> select * from locations;
LOCATION_ID LOCATION_NAME
-----
3 Eagle Creek
6 Fort Lauderdale Beach
1 Heron Lake
7 Lake Michigan
2 Loon Creek
5 Sarasota Bridge
4 White River

7 rows selected.

SQL>
```

The session starts by displaying the current data in the locations table. An UPDATE statement is then executed to change the location name for location ID 7 to 'Lake Michigan'. A SELECT \* FROM locations statement is run again to verify that the update was successful, showing that the location name has been updated.

# Deleting Data from Tables

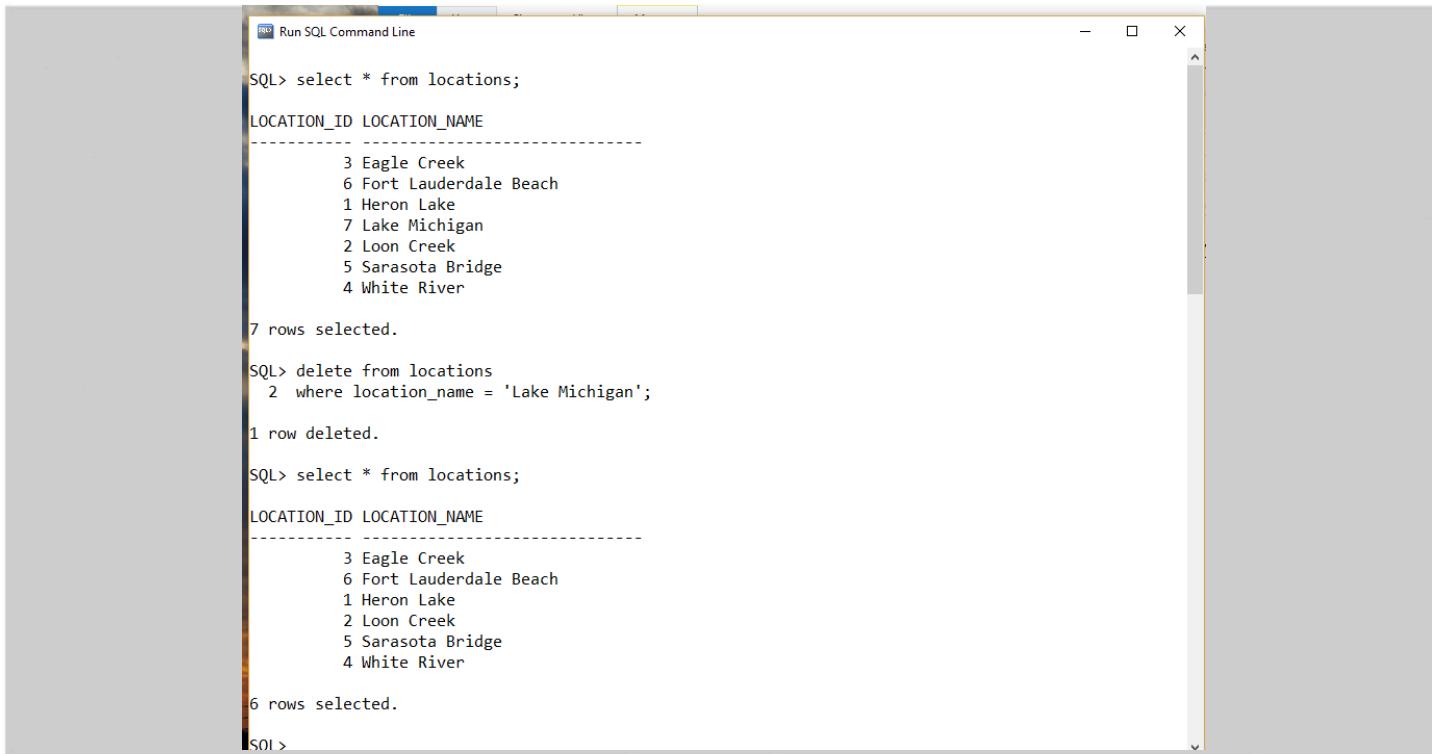
## DELETE Statement

- Used to delete one or more rows of data from a table
- Commands in SQL are typically not case-sensitive
- Case matters when inserting data values into a table

## Syntax:

```
DELETE FROM table_name
[ WHERE condition ];
```

# Deleting Data from Tables



The screenshot shows a Windows command-line interface window titled "Run SQL Command Line". The window displays the following SQL session:

```
SQL> select * from locations;
LOCATION_ID LOCATION_NAME
-----
3 Eagle Creek
6 Fort Lauderdale Beach
1 Heron Lake
7 Lake Michigan
2 Loon Creek
5 Sarasota Bridge
4 White River

7 rows selected.

SQL> delete from locations
  2 where location_name = 'Lake Michigan';

1 row deleted.

SQL> select * from locations;
LOCATION_ID LOCATION_NAME
-----
3 Eagle Creek
6 Fort Lauderdale Beach
1 Heron Lake
2 Loon Creek
5 Sarasota Bridge
4 White River

6 rows selected.

SQL >
```

The session starts by selecting all rows from the "locations" table, displaying 7 rows. Then, a single row is deleted where the "location\_name" is "Lake Michigan". After the delete operation, the session runs a select query again, showing only 6 rows remain in the table.

# Deleting Data from Tables

Caution: deleting records from a table without using criteria (WHERE clause) will remove all rows of data!



```
Run SQL Command Line

SQL> select * from nests;

  NEST_ID NEST_NAME
  -----
    5 Cavity Nest
    3 Cup Nest
    7 Floating Platform
    1 Ground Nest
    6 None/Minimal
    4 Pendulous Nest
    2 Platform Nest

7 rows selected.

SQL> delete from nests;
delete from nests
*
ERROR at line 1:
ORA-02292: integrity constraint (RYAN.BIRDS_NESTS_NEST_FK) violated - child
record found

SQL>
```

The reason this statement returned an error and did not delete any rows is because there is a foreign key in the database that references “Nest\_Id” in the “Nests” tables. The data in the foreign keys would have to be deleted first. Constraints help protect REFERENTIAL INTEGRITY.

# Transactional Control Commands

- **Commit: Saving a Transaction**
  - After issuing one or more DML statements, COMMIT is used to save work to the database.
  - COMMIT will save all transactions since the last COMMIT or ROLLBACK, or since the user database session began.
- **Rollback: Undoing a Transaction**
  - After issuing one or more DML statements, ROLLBACK is used to undo recent transactions to the database.
  - ROLLBACK will undo all transactions since the last COMMIT or ROLLBACK, or since the user database session began.

# Live Transactional (DML) Examples

1. Show all records in the “Migration” table.
2. Insert a new migration location into the “Migration” table for “Canada.”
3. Show all records in the “Migration” table.
4. Change the value from “Canada” to “Alaska.”
5. Show all records in the “Migration” table.
6. Delete the record “Alaska” from the “Migration” table.
7. Show all records in the “Migration” table.
8. Insert a new migration location into the “Migration” table for “Iceland,” then issue a ROLLBACK.
9. Show all records in the “Migration” table.
10. Insert a new migration location into the “Migration” table for “Iceland,” then issue a COMMIT.
11. Show all records in the “Migration” table.
12. Delete the record “Mexico” from the “Migration” table.
13. Show all records in the “Migration” table.
14. Issue the ROLLBACK command.
15. Show all records in the “Migration” table.

# SQL For Any IT Professional

- Getting Data Out of a Database Using SQL



# Intro to the SQL Query

- The ability to effectively and simply retrieve useful data is what SQL and the RDBMS are all about.
- Data Query Language (DQL) is the SQL component that allows you perform queries.
- Writing SQL queries are:
  - Simple
  - Fast
  - Powerful
  - Easy to view deeper perspectives of data

# “Asking” for Information with SQL

Writing queries in SQL is very “English-like.” For example:

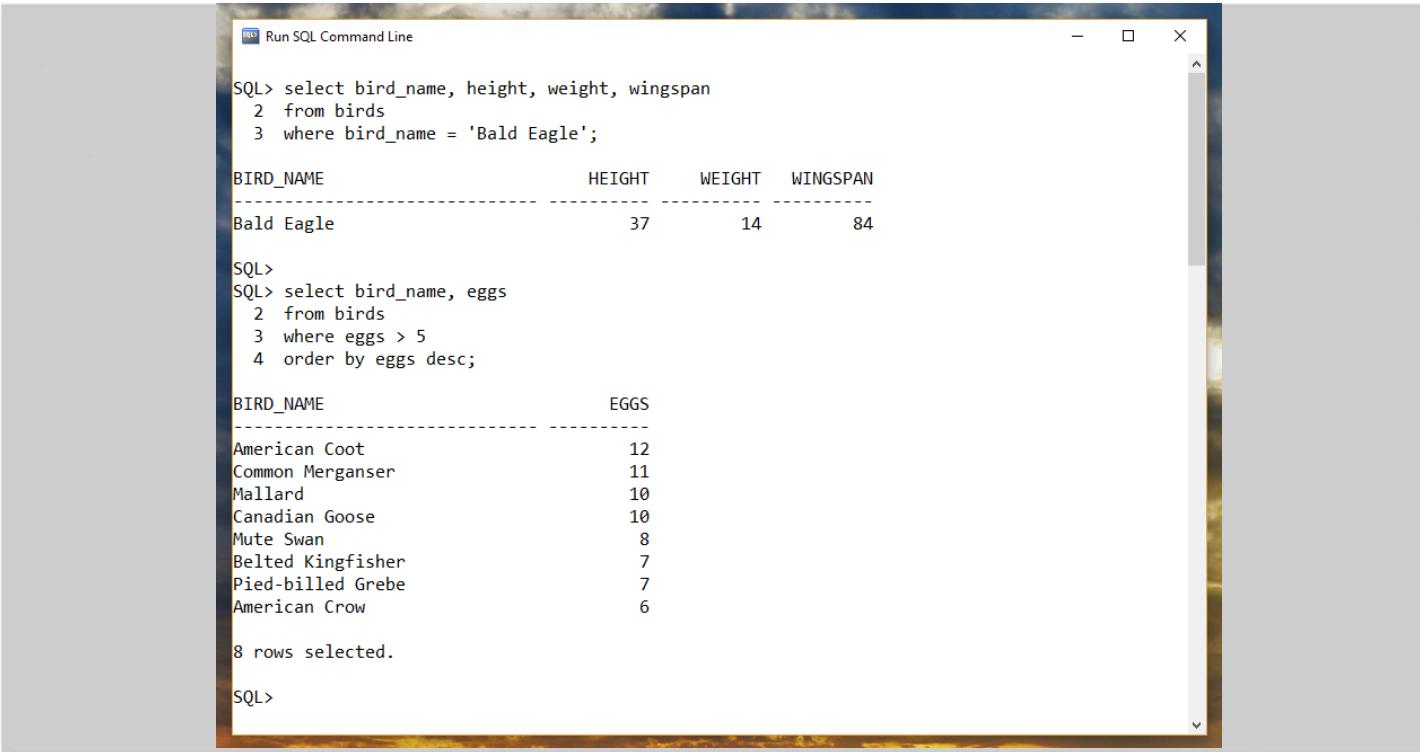
- How many types of birds are in the database?
- Which bird has the most diverse diet?
- Which is the largest bird?
- Which birds have a greater than average wingspan?
- With which birds do both male and female contribute toward the nest?
- List all birds in order of their size.
- Which bird spends the most total time with its young?
- What percentage of birds eat fish?
- How many eggs does a “War Eagle” (nickname) typically have?

# SELECT: Writing a Simple Query

- It is easy to begin writing queries using the SELECT statement.
- Basic syntax:

```
SELECT column1, column2, ...
FROM table1, table2, ...
where column 1 = 'value1'
      [AND | OR] column2 = 'value2' ...
order by column1, column2, ...
```

# SELECT: Writing a Simple Query



The screenshot shows a window titled "Run SQL Command Line". It contains two separate SQL queries and their results.

```
SQL> select bird_name, height, weight, wingspan
  2  from birds
  3  where bird_name = 'Bald Eagle';

BIRD_NAME           HEIGHT    WEIGHT   WINGSPAN
-----            -----      -----      -----
Bald Eagle          37        14        84

SQL>
SQL> select bird_name, eggs
  2  from birds
  3  where eggs > 5
  4  order by eggs desc;

BIRD_NAME           EGGS
-----            -----
American Coot       12
Common Merganser    11
Mallard              10
Canadian Goose      10
Mute Swan            8
Belted Kingfisher    7
Pied-billed Grebe    7
American Crow         6

8 rows selected.

SQL>
```

The first query selects the columns `bird_name`, `height`, `weight`, and `wingspan` from the `birds` table, filtering for the row where `bird_name` is 'Bald Eagle'. The result is a single row for the Bald Eagle with values 37, 14, and 84 respectively.

The second query selects the columns `bird_name` and `eggs` from the `birds` table, filtering for rows where `eggs` is greater than 5, and ordering the results by `eggs` in descending order. The result is eight rows listing various birds and their egg counts, starting with the American Coot at 12 eggs and ending with the American Crow at 6 eggs.

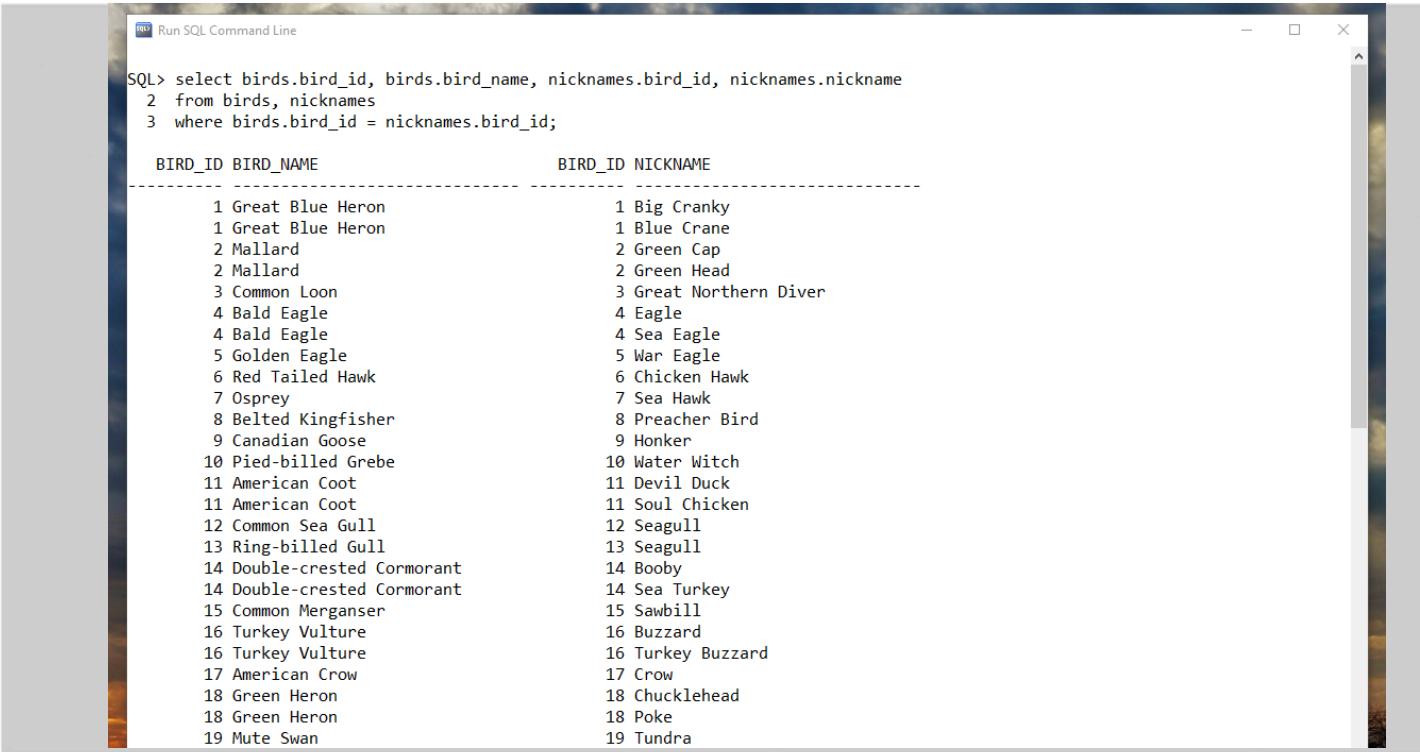
# Joining Tables to Query Multiple Tables

- More times than you, you will need to get data from more than one table.
- If you have a solid, normalized design, it is simple to join tables using keys (typically primary and foreign keys).

Syntax:

```
SELECT table1.column, table2.column  
FROM table1, table2  
WHERE table1.key = table2.key;
```

# Joining Tables to Query Multiple Tables



The screenshot shows a Windows-style application window titled "Run SQL Command Line". Inside, an SQL query is run against two tables, "birds" and "nicknames". The query selects bird IDs and names from the "birds" table and nicknames from the "nicknames" table, joining them where the bird ID matches the nickname ID.

```
SQL> select birds.bird_id, birds.bird_name, nicknames.bird_id, nicknames.nickname
  2  from birds, nicknames
  3 where birds.bird_id = nicknames.bird_id;
```

BIRD_ID	BIRD_NAME	BIRD_ID	NICKNAME
1	Great Blue Heron	1	Big Cranky
1	Great Blue Heron	1	Blue Crane
2	Mallard	2	Green Cap
2	Mallard	2	Green Head
3	Common Loon	3	Great Northern Diver
4	Bald Eagle	4	Eagle
4	Bald Eagle	4	Sea Eagle
5	Golden Eagle	5	War Eagle
6	Red Tailed Hawk	6	Chicken Hawk
7	Osprey	7	Sea Hawk
8	Belted Kingfisher	8	Preacher Bird
9	Canadian Goose	9	Honker
10	Pied-billed Grebe	10	Water Witch
11	American Coot	11	Devil Duck
11	American Coot	11	Soul Chicken
12	Common Sea Gull	12	Seagull
13	Ring-billed Gull	13	Seagull
14	Double-crested Cormorant	14	Booby
14	Double-crested Cormorant	14	Sea Turkey
15	Common Merganser	15	Sawbill
16	Turkey Vulture	16	Buzzard
16	Turkey Vulture	16	Turkey Buzzard
17	American Crow	17	Crow
18	Green Heron	18	Chucklehead
18	Green Heron	18	Poke
19	Mute Swan	19	Tundra

# Live Query Examples

1. Show all records in the “Food” table.
2. Show a list of birds that eat fish.
3. Show a list of birds that migrate to Mexico.
4. Show all list of all birds and their eggs, for those that build a platform nest. Sort the results by the number of eggs layed.
5. List all of the foods eaten by the “War Eagle.”
6. Delete the record “Alaska” from the “Migration” table.

# Summary and Q&A

- DML statements are used to load and manage data in a relational database.
  - INSERT
  - UPDATE
  - DELETE
- Transaction control commands are used to control when and how transactions are saved or undone.
- DQL statements (SELECT) are used to retrieve data from a relational database.

```
SELECT  
FROM  
WHERE  
ORDER BY
```

- Multiple tables can easily be joined in a query to derive more data, using keys that are established during the database design (normalization) phase.

# SQL For Any IT Professional

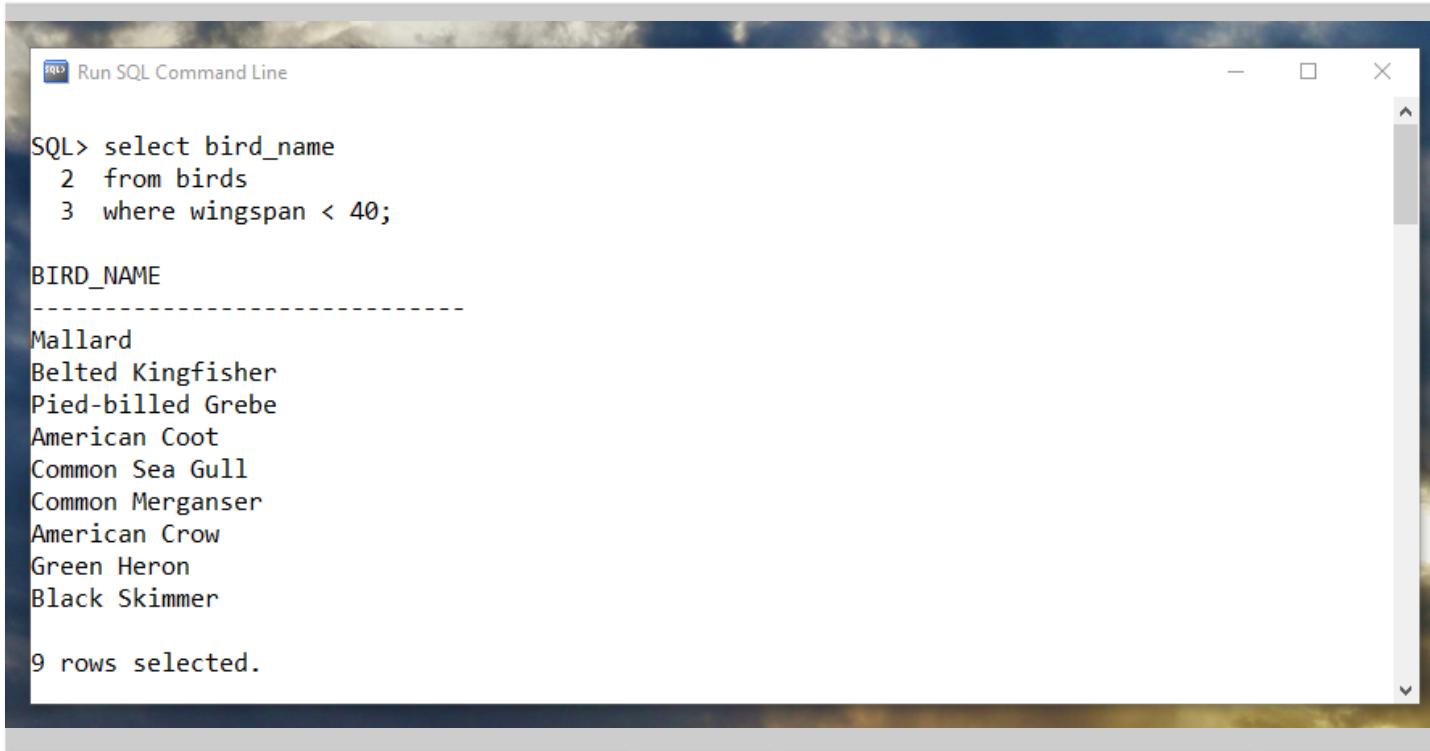
- Getting More Out of Your SQL Queries



# Comparison Operators

- Equality                =
- Non-equality        !=, <>
- Less than              <
- Greater than         >
- Combining logical operators
  - !=
  - <>
  - >=
  - <=

# Comparison Operators

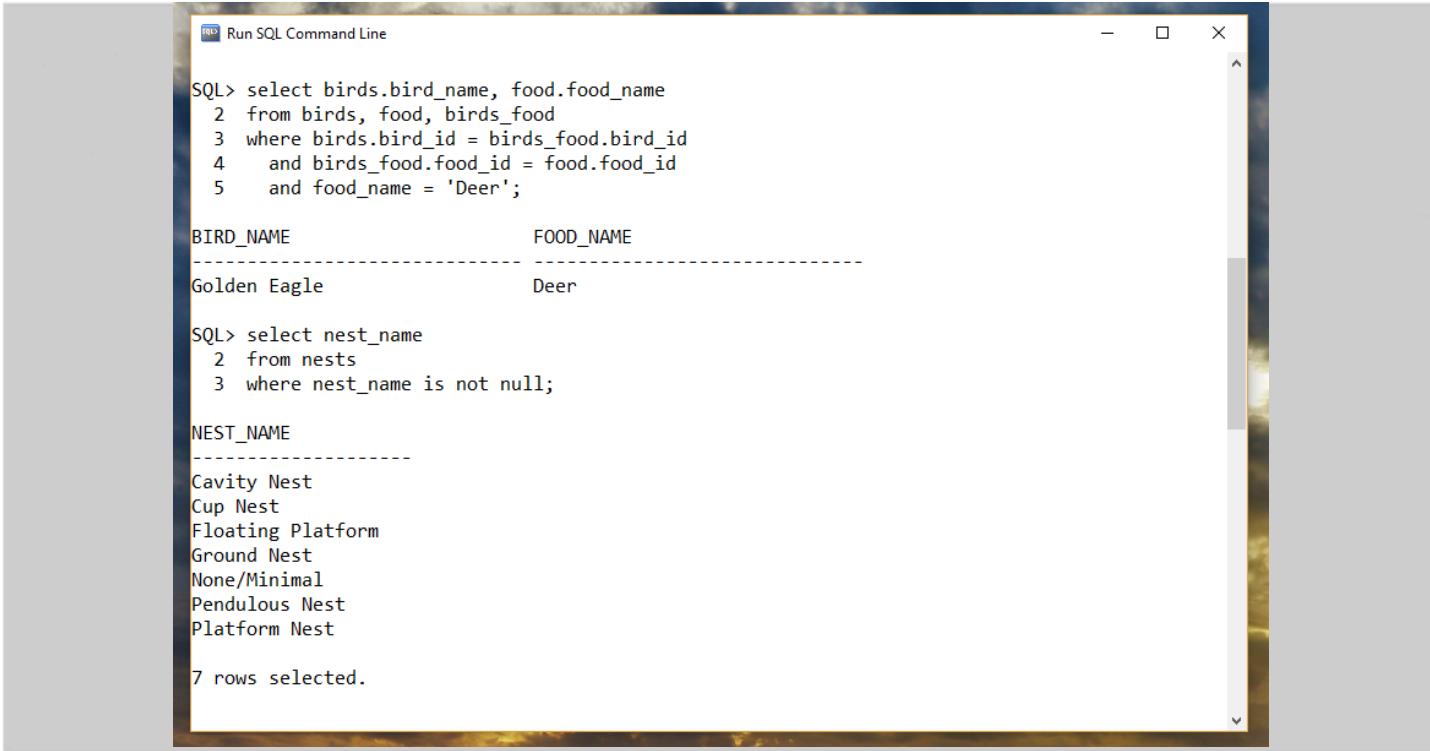


The screenshot shows a Windows-style application window titled "Run SQL Command Line". Inside, an SQL query is run against a table named "birds". The query selects bird names where the wingspan is less than 40. The results are displayed in a table format with a header "BIRD\_NAME" and nine rows of bird names.

BIRD_NAME
Mallard
Belted Kingfisher
Pied-billed Grebe
American Coot
Common Sea Gull
Common Merganser
American Crow
Green Heron
Black Skimmer

9 rows selected.

# Comparison Operators



```
Run SQL Command Line

SQL> select birds.bird_name, food.food_name
  2  from birds, food, birds_food
  3  where birds.bird_id = birds_food.bird_id
  4    and birds_food.food_id = food.food_id
  5    and food_name = 'Deer';

BIRD_NAME          FOOD_NAME
-----
Golden Eagle        Deer

SQL> select nest_name
  2  from nests
  3  where nest_name is not null;

NEST_NAME
-----
Cavity Nest
Cup Nest
Floating Platform
Ground Nest
None/Minimal
Pendulous Nest
Platform Nest

7 rows selected.
```

# Logical Operators

- IS NULL        the value in the column is missing or NULL
- BETWEEN        the results are between two values
- IN              the results are in a list of values
- LIKE            wildcard operator (similarity)
- EXISTS         searches for a row that meets a certain criteria (used in subqueries)

# Logical Operators

```
Run SQL Command Line

SQL> select bird_name
  2  from birds
  3  where bird_name like '%Eagle%';

BIRD_NAME
-----
Bald Eagle
Golden Eagle

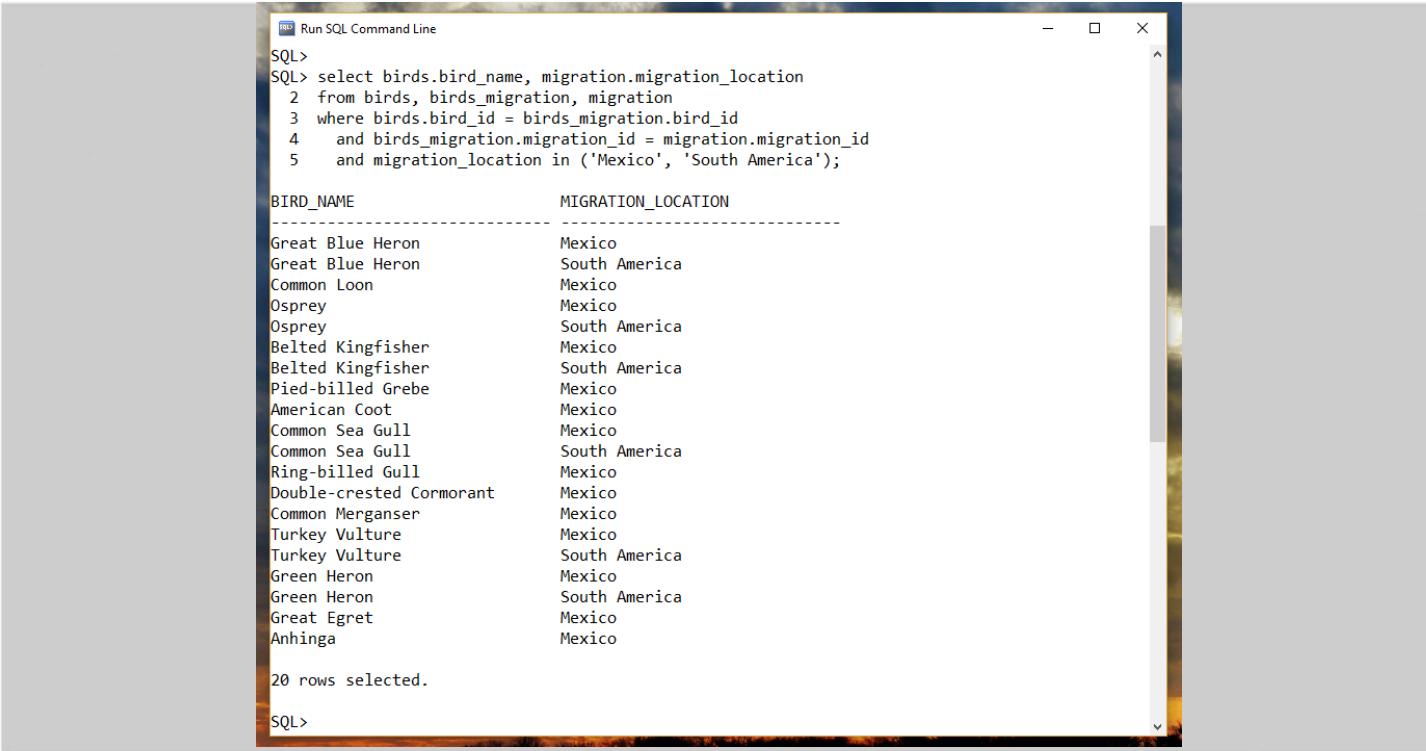
SQL> select bird_name, wingspan
  2  from birds
  3  where wingspan between 40 and 70;

BIRD_NAME          WINGSPAN
-----
Common Loon           54
Red Tailed Hawk       48
Ring-billed Gull       50
Double-crested Cormorant   54
Great Egret            67.2
Anhinga                 42

6 rows selected.

SQL> ■
```

# Logical Operators



The screenshot shows a Windows application window titled "Run SQL Command Line". Inside, an SQL query is run against three tables: birds, birds\_migration, and migration. The query retrieves bird names and migration locations where the bird ID matches the migration ID and the migration location is either Mexico or South America. The results are displayed in a tabular format with columns "BIRD\_NAME" and "MIGRATION\_LOCATION". The output shows 20 rows selected, all of which have "Mexico" listed under "MIGRATION\_LOCATION".

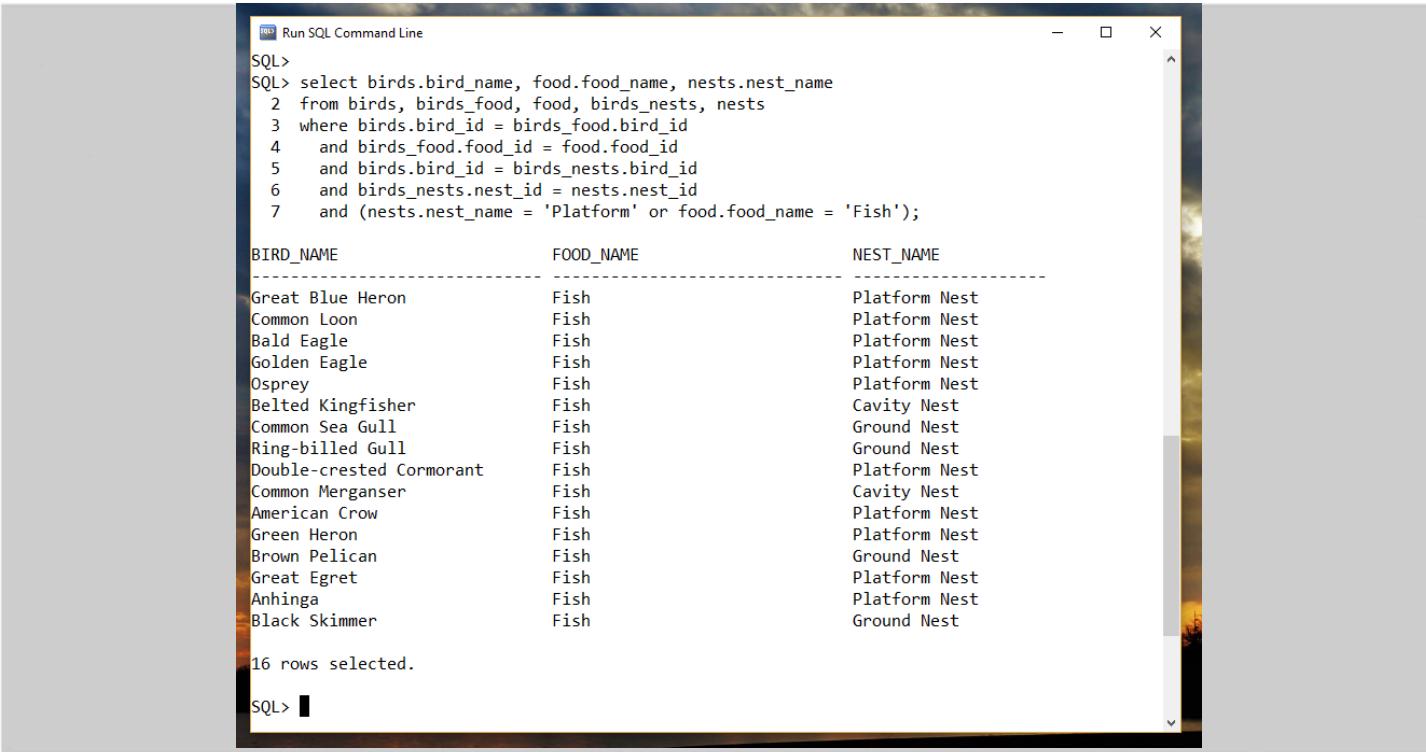
BIRD_NAME	MIGRATION_LOCATION
Great Blue Heron	Mexico
Great Blue Heron	South America
Common Loon	Mexico
Osprey	Mexico
Osprey	South America
Belted Kingfisher	Mexico
Belted Kingfisher	South America
Pied-billed Grebe	Mexico
American Coot	Mexico
Common Sea Gull	Mexico
Common Sea Gull	South America
Ring-billed Gull	Mexico
Double-crested Cormorant	Mexico
Common Merganser	Mexico
Turkey Vulture	Mexico
Turkey Vulture	South America
Green Heron	Mexico
Green Heron	South America
Great Egret	Mexico
Anhinga	Mexico

20 rows selected.

# Conjunctive Operators

- AND
  - Both conditions must be true in order to return data
  - WHERE nest = 'Platform' AND eggs > 3
- OR
  - Either condition can be true to return data
  - WHERE food = 'Fish' OR food = 'Insects'

# Conjunctive Operators



The screenshot shows a Windows-style application window titled "Run SQL Command Line". Inside, an SQL query is run against a database containing tables for birds, food, and nests. The query retrieves bird names, food names, and nest names where the bird's food ID matches the food ID, and the bird's nest ID matches the nest ID. It also filters for either a platform nest or fish as food.

```
SQL> select birds.bird_name, food.food_name, nests.nest_name
  2  from birds, birds_food, food, birds_nests, nests
  3 where birds.bird_id = birds_food.bird_id
  4   and birds_food.food_id = food.food_id
  5   and birds.bird_id = birds_nests.bird_id
  6   and birds_nests.nest_id = nests.nest_id
  7   and (nests.nest_name = 'Platform' or food.food_name = 'Fish');

BIRD_NAME          FOOD_NAME        NEST_NAME
-----          -----
Great Blue Heron      Fish      Platform Nest
Common Loon          Fish      Platform Nest
Bald Eagle           Fish      Platform Nest
Golden Eagle         Fish      Platform Nest
Osprey                Fish      Platform Nest
Belted Kingfisher    Fish      Cavity Nest
Common Sea Gull       Fish      Ground Nest
Ring-billed Gull     Fish      Ground Nest
Double-crested Cormorant  Fish      Platform Nest
Common Merganser     Fish      Cavity Nest
American Crow         Fish      Platform Nest
Green Heron           Fish      Platform Nest
Brown Pelican         Fish      Ground Nest
Great Egret            Fish      Platform Nest
Anhinga                Fish      Platform Nest
Black Skimmer          Fish      Ground Nest

16 rows selected.

SQL>
```

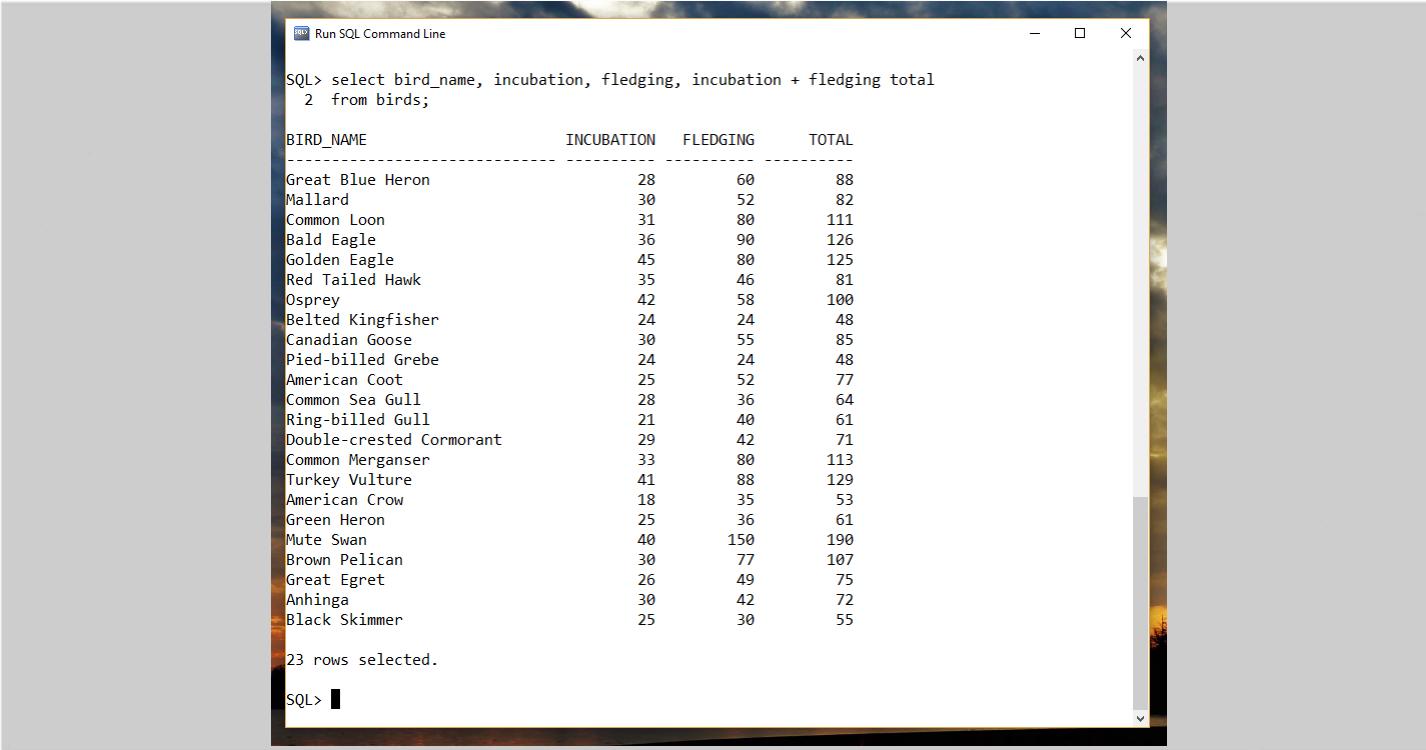
# Negative Operators

- NOT EQUAL
- NOT BETWEEN
- NOT IN
- NOT LIKE
- IS NOT NULL
- NOT EXISTS
- NOT UNIQUE

# Arithmetic Operators

- Addition +
- Subtraction -
- Multiplication \*
- Division /
- Combinations of arithmetic operators
- *Ensure your math is correct! If not, incorrect or incomplete data results will be returned.*

# Arithmetic Operators



The screenshot shows a Windows command-line interface window titled "Run SQL Command Line". The window contains the following SQL query and its results:

```
SQL> select bird_name, incubation, fledging, incubation + fledging total
  2 from birds;
```

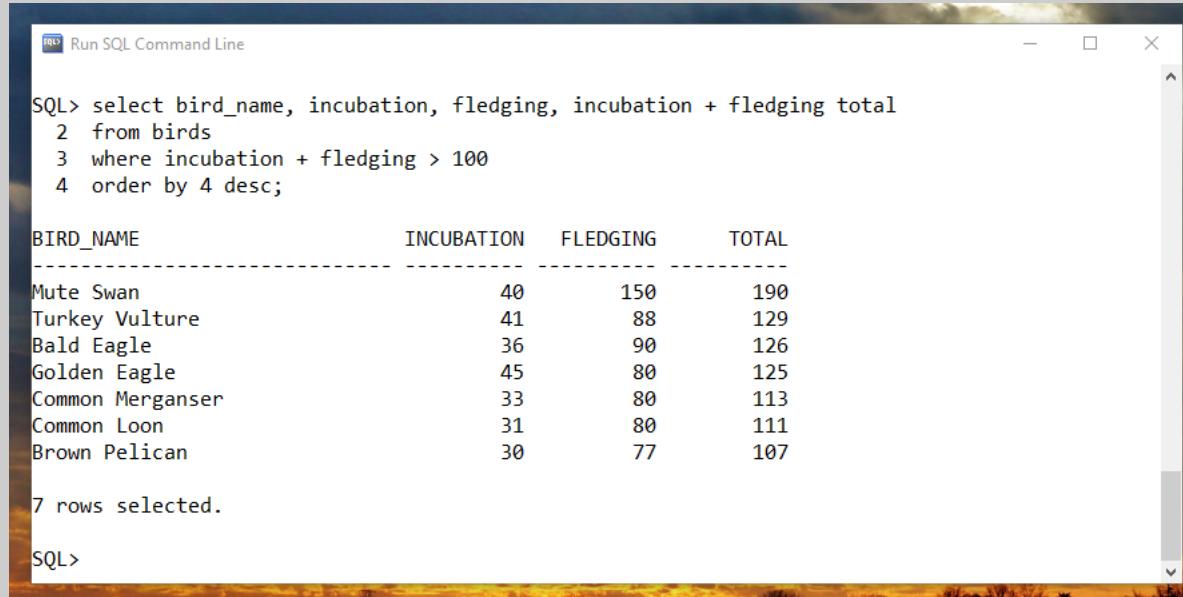
BIRD_NAME	INCUBATION	FLEDGING	TOTAL
Great Blue Heron	28	60	88
Mallard	30	52	82
Common Loon	31	80	111
Bald Eagle	36	90	126
Golden Eagle	45	80	125
Red Tailed Hawk	35	46	81
Osprey	42	58	100
Belted Kingfisher	24	24	48
Canadian Goose	30	55	85
Pied-billed Grebe	24	24	48
American Coot	25	52	77
Common Sea Gull	28	36	64
Ring-billed Gull	21	40	61
Double-crested Cormorant	29	42	71
Common Merganser	33	80	113
Turkey Vulture	41	88	129
American Crow	18	35	53
Green Heron	25	36	61
Mute Swan	40	150	190
Brown Pelican	30	77	107
Great Egret	26	49	75
Anhinga	30	42	72
Black Skimmer	25	30	55

23 rows selected.

```
SQL> ■
```

# Arithmetic Operators

Using simple arithmetic, let's retrieve a list of all birds where the total time they spend with their young (incubation + fledging) is greater than 100 days. We will sort the results from most days spent with their young to least.



```
SQL> select bird_name, incubation, fledging, incubation + fledging total
  2  from birds
  3  where incubation + fledging > 100
  4  order by 4 desc;

BIRD_NAME           INCUBATION    FLEDGING      TOTAL
-----              -----
Mute Swan                  40        150        190
Turkey Vulture             41         88        129
Bald Eagle                 36         90        126
Golden Eagle                45         80        125
Common Merganser            33         80        113
Common Loon                  31         80        111
Brown Pelican                30         77        107

7 rows selected.

SQL>
```

BIRD_NAME	INCUBATION	FLEDGING	TOTAL
Mute Swan	40	150	190
Turkey Vulture	41	88	129
Bald Eagle	36	90	126
Golden Eagle	45	80	125
Common Merganser	33	80	113
Common Loon	31	80	111
Brown Pelican	30	77	107

# Live Examples of SQL Operators

1. Create a list of birds and their associated nicknames for those who have the word “Turkey” in their nickname.
2. Who has biggest ratio of wingspan to size (height)?
3. Which types of nests do males help build?
4. For each bird, what is the average time parents spend with each egg? In other words, does it appear that incubation and fledging periods are longer with birds that lay more eggs? Retrieve data that shows the bird, number of eggs, incubation, fledging, total time spent with young, and total time spent per egg.
5. Create a list of birds and their nicknames, where the nicknames have a primary color in their name.

# SQL For Any IT Professional

- Scratching the Surface of Advanced Queries



# SQL Functions: Reading More into Data

SQL functions allow you to look at data, and represent data in results, differently than how it is natively stored in the database.

Common SQL functions include:

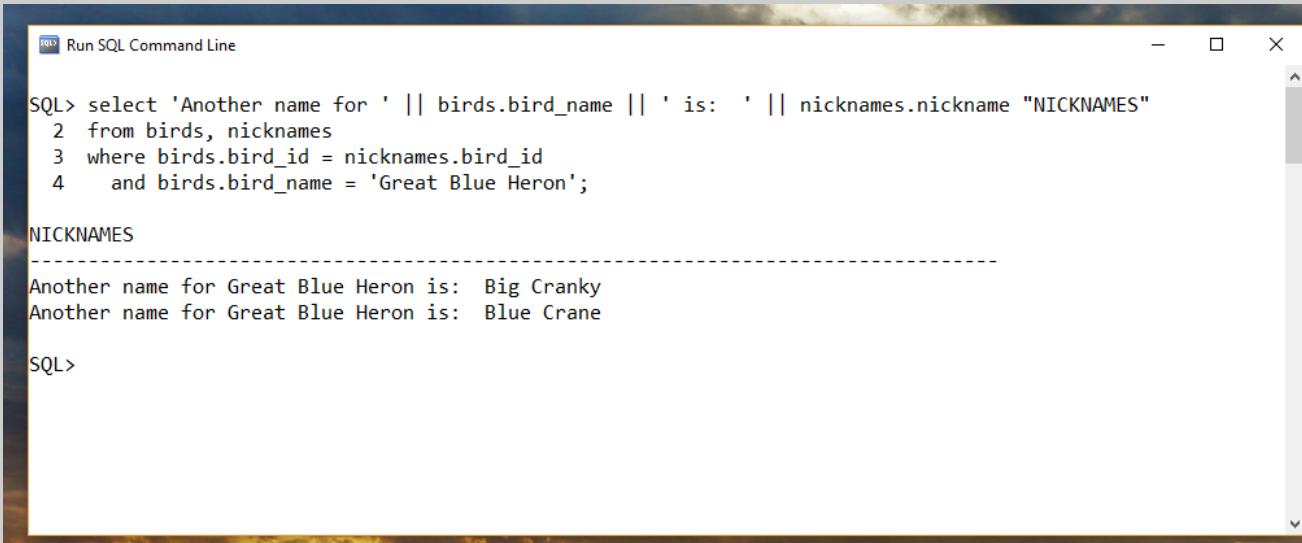
- Character Functions
- Mathematical Functions
- Date Functions
- Aggregate (allow you to group data retrieved)

# Character Functions

- **CONCAT**    ||    `string1 || string2'
- **UPPER, LOWER**    UPPER (value)
- **SUBSTR**            SUBSTR (column, *start\_position*, *end\_position*)
- **TRANSLATE**        TRANSLATE (column, 'string\_in\_value', 'new string')
- **REPLACE**           REPLACE (column, 'value', 'new\_value')
- **LTRIM, RTRIM**     LTRIM (column, 'value')
- **LENGTH**            LENGTH (value)
- **LPAD, RPAD**       LPAD (column, *num\_characters*, 'character')
- **DECODE**           DECODE (column, *value1*, *new1*, *value2*, *new2*, *default\_value*)
- **DISTINCT**          DISTINCT (column)

# Character Functions

Concatenation example:



The screenshot shows a Windows-style application window titled "Run SQL Command Line". Inside the window, an SQL query is displayed and executed. The query concatenates the name from the "birds" table with the nickname from the "nicknames" table where the bird ID matches. The results show two nicknames for the Great Blue Heron.

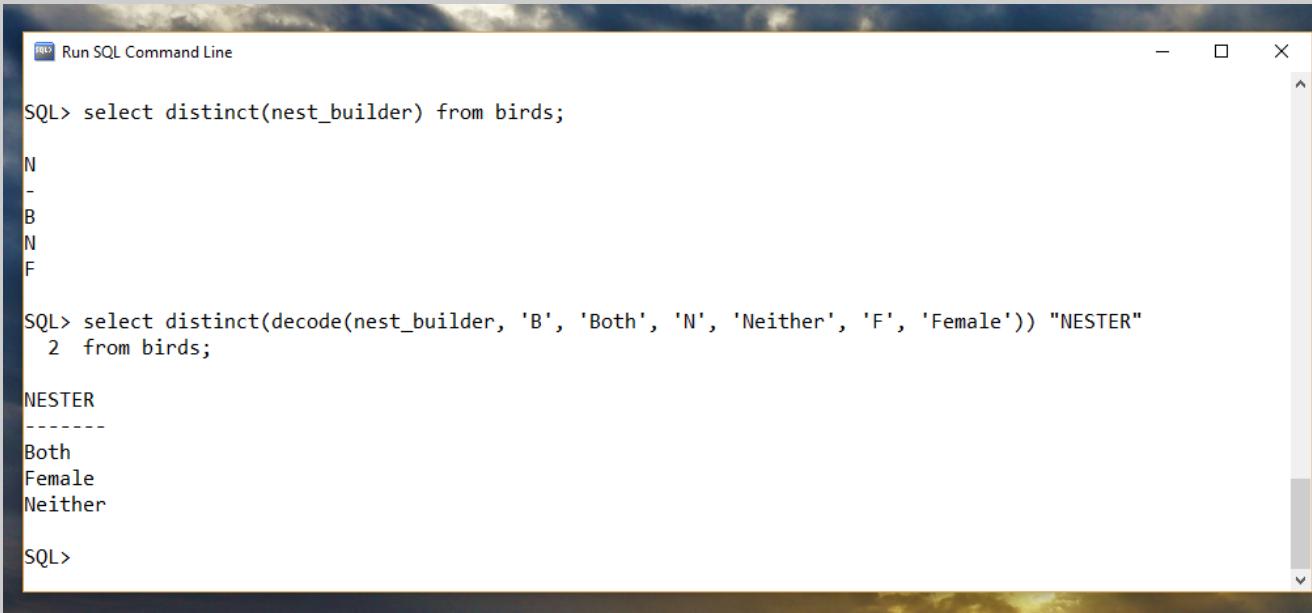
```
SQL> select 'Another name for ' || birds.bird_name || ' is: ' || nicknames.nickname "NICKNAMES"
  2  from birds, nicknames
  3  where birds.bird_id = nicknames.bird_id
  4  and birds.bird_name = 'Great Blue Heron';

NICKNAMES
-----
Another name for Great Blue Heron is: Big Cranky
Another name for Great Blue Heron is: Blue Crane

SQL>
```

# Character Functions

DISTINCT (UNIQUE) and DECODE example:



```
Run SQL Command Line

SQL> select distinct(nest_builder) from birds;

N
-
B
N
F

SQL> select distinct(decode(nest_builder, 'B', 'Both', 'N', 'Neither', 'F', 'Female')) "NESTER"
  2  from birds;

NESTER
-----
Both
Female
Neither

SQL>
```

# Mathematical Functions

- ABS (absolute value)       $\text{ABS}(-11.3) = 11.3$
- ROUND (round a value)       $\text{ROUND}(11.3) = 11$
- SQRT (square root)       $\text{SQRT}(16) = 4$
- SIGN (sign values)       $\text{SIGN}(-11.3) = -1$
- POWER (power)       $\text{POWER}(2, 2) = 4$
- CEIL (ceiling value)       $\text{CEIL}(11.3) = 12$
- FLOOR (floor value)       $\text{FLOOR}(11.3) = 11$
- EXP (exponential value)       $\text{EXP}(1) = 2.7182$
- SIN, COS, TAN (SIN, COS, TAN)

# Date Functions

Standard Date Data Type	DATE
Standard Date Format (Oracle)	DD-MON-YY
Current Date (Oracle)	SYSDATE
Convert Date to Character String	TO_CHAR(date, ' <i>date picture</i> )

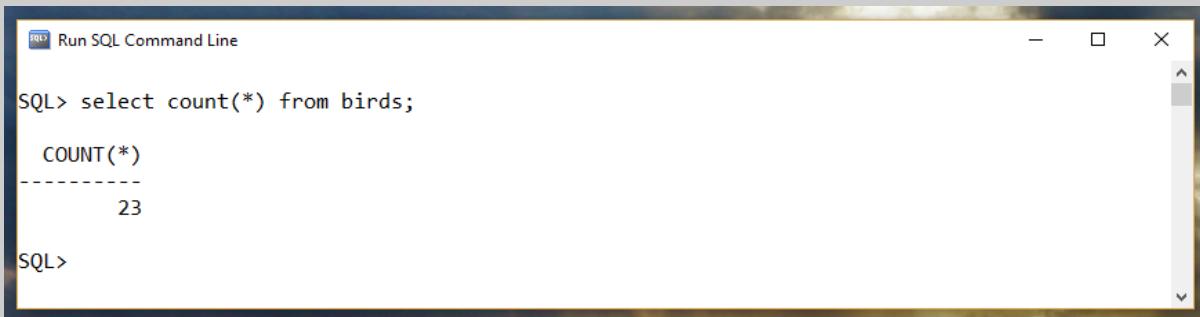
## Example Date Pictures

SYSDATE	02-OCT-18
TO_CHAR(SYSDATE, 'MONTH,DD,YYYY')	October , 02, 2018
TO_CHAR(SYSDATE, 'DDMONTHYYYY')	02 OCTOBER 2018
TO_CHAR(SYSDATE, 'DAY')	Tuesday
TO_CHAR(SYSDATE, 'HH:MM:SS')	10:10:39

# Aggregate Functions

Aggregate functions are useful in summarizing and analyzing data.

- COUNT (counts rows of data, or values in columns)
- SUM (returns the SUM of values in a column)
- MAX (returns the MAXIMUM value in a column)
- MIN (returns the MINIMUM value in a column)
- AVG (returns the AVERAGE value in a column)



```
Run SQL Command Line

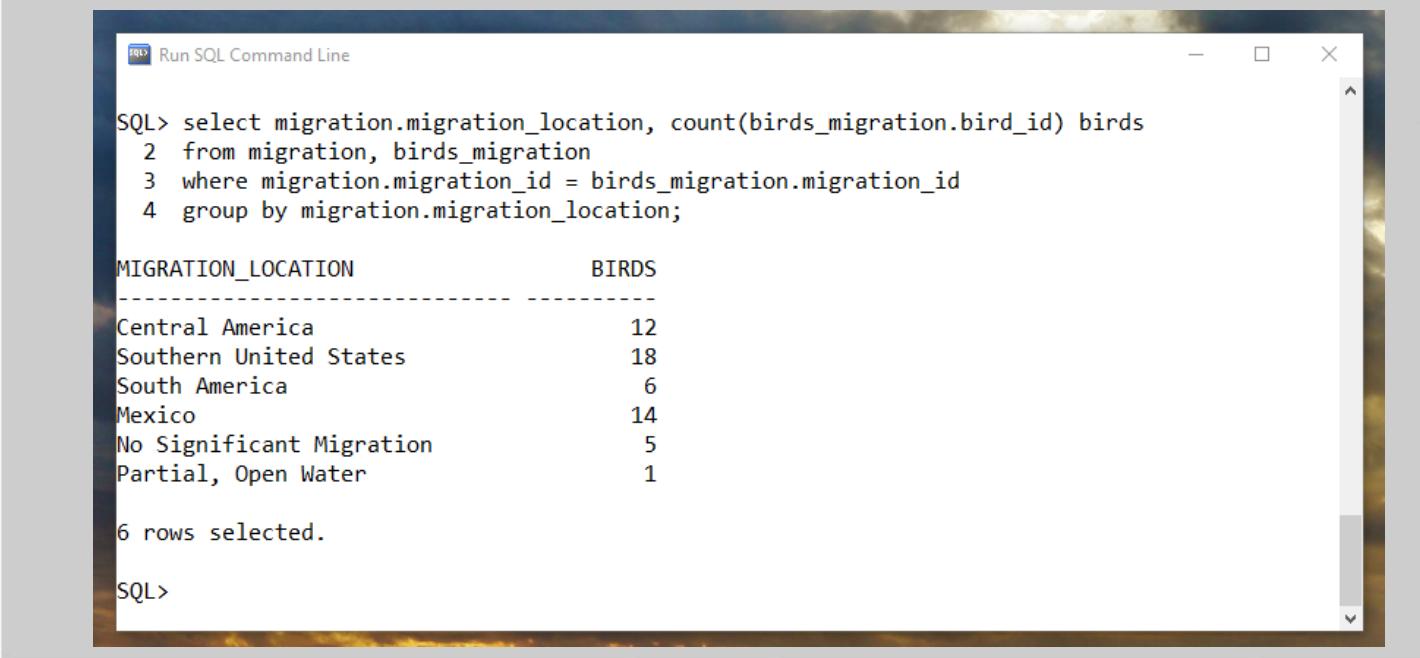
SQL> select count(*) from birds;

COUNT(*)
-----
23

SQL>
```

# Aggregate Functions

Counting birds for each “Migration Location”:



The screenshot shows a Windows-style application window titled "Run SQL Command Line". Inside, an SQL query is run against a database named "migration". The query counts the number of birds for each migration location by joining the "migration" table with the "birds\_migration" table on their common "migration\_id" column, and then grouping the results by "migration\_location". The output displays six rows of data, each showing a migration location and its corresponding count of birds.

```
SQL> select migration.migration_location, count(birds_migration.bird_id) birds
  2  from migration, birds_migration
  3  where migration.migration_id = birds_migration.migration_id
  4  group by migration.migration_location;

MIGRATION_LOCATION          BIRDS
-----
Central America                  12
Southern United States            18
South America                     6
Mexico                            14
No Significant Migration           5
Partial, Open Water                 1

6 rows selected.

SQL>
```

MIGRATION_LOCATION	BIRDS
Central America	12
Southern United States	18
South America	6
Mexico	14
No Significant Migration	5
Partial, Open Water	1

# Grouping Data

When using aggregate functions in a query, you can group the results of the query for further analysis.

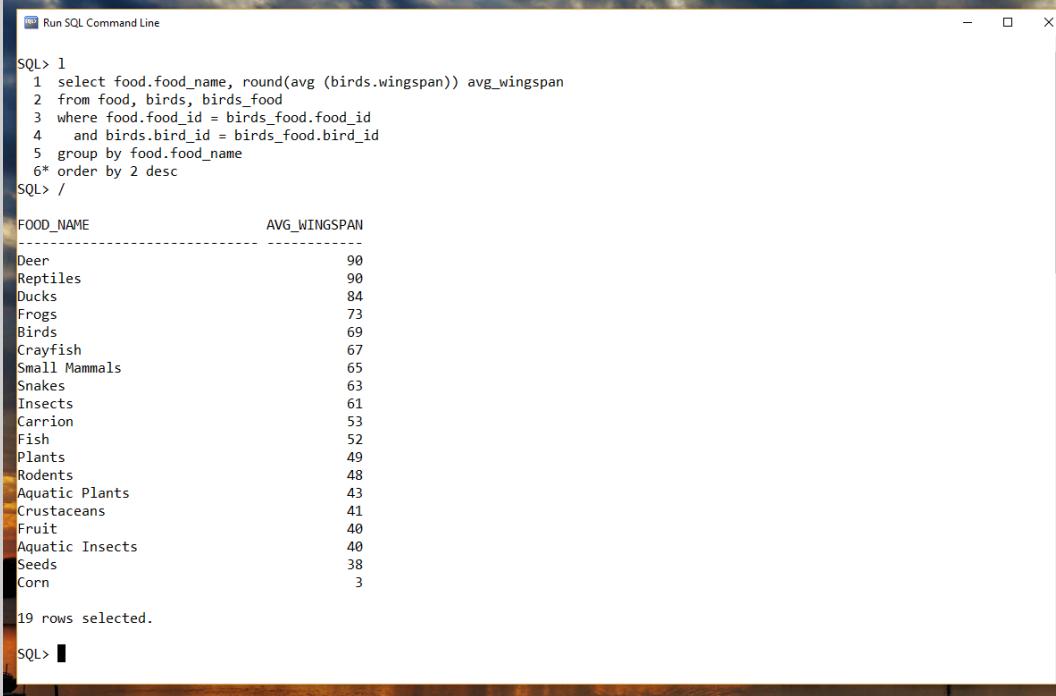
- **GROUP BY**      allows summarized data to be grouped
- **HAVING**          allows conditions to be placed on grouped data before final query results are returned

SQL Query Syntax:

```
SELECT column1, column2, ...
  FROM table1, table2, ...
 WHERE column 1 = 'value1'
   [AND | OR] column2 = 'value2' ...
 GROUP BY column1, column2, ...
 HAVING [conditions]
 ORDER BY column1, column2, ...
```

# Grouping Data

Let's generate a report showing the average wingspan of birds that consume each type of food.



The screenshot shows a Windows command-line interface titled "Run SQL Command Line". The user has run the following SQL query:

```
SQL> 1
  1 select food.food_name, round(avg (birds.wingspan)) avg_wingspan
  2 from food, birds, birds_food
  3 where food.food_id = birds_food.food_id
  4   and birds.bird_id = birds_food.bird_id
  5 group by food.food_name
  6* order by 2 desc
SQL> /
```

The results of the query are displayed in a table:

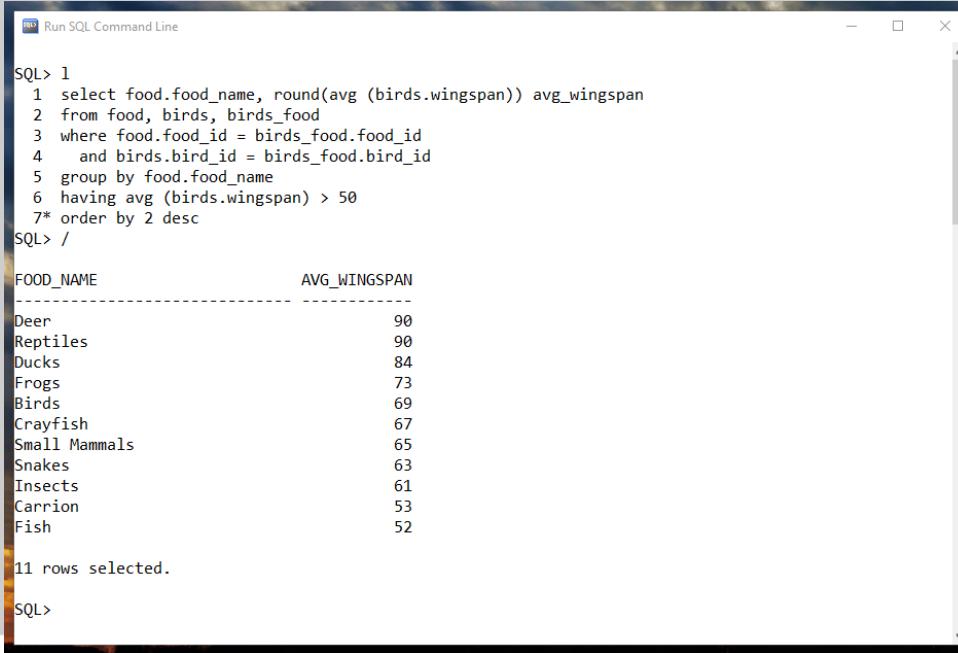
FOOD_NAME	AVG_WINGSPAN
Deer	90
Reptiles	90
Ducks	84
Frogs	73
Birds	69
Crayfish	67
Small Mammals	65
Snakes	63
Insects	61
Carrion	53
Fish	52
Plants	49
Rodents	48
Aquatic Plants	43
Crustaceans	41
Fruit	40
Aquatic Insects	40
Seeds	38
Corn	3

19 rows selected.

SQL>

# Grouping Data

Now let's refine the same report showing only foods that are eaten by birds with more than an average wingspan of 50.



```
Run SQL Command Line

SQL> 1
  1 select food.food_name, round(avg (birds.wingspan)) avg_wingspan
  2 from food, birds, birds_food
  3 where food.food_id = birds_food.food_id
  4   and birds.bird_id = birds_food.bird_id
  5 group by food.food_name
  6 having avg (birds.wingspan) > 50
  7* order by 2 desc
SQL> /
FOOD_NAME          AVG_WINGSPAN
-----
Deer                  90
Reptiles               90
Ducks                  84
Frogs                  73
Birds                  69
Crayfish                67
Small Mammals           65
Snakes                  63
Insects                  61
Carriion                 53
Fish                   52

11 rows selected.

SQL>
```

The screenshot shows a terminal window titled "Run SQL Command Line". The SQL query selects the food name and the average wingspan of birds that eat that food, grouping by food name and filtering for an average wingspan greater than 50. The results are ordered by average wingspan in descending order. The output shows 11 rows of data, with Deer and Reptiles both having an average wingspan of 90, and Fish having the lowest average wingspan at 52.

# Subqueries: Defining Unknown Criteria

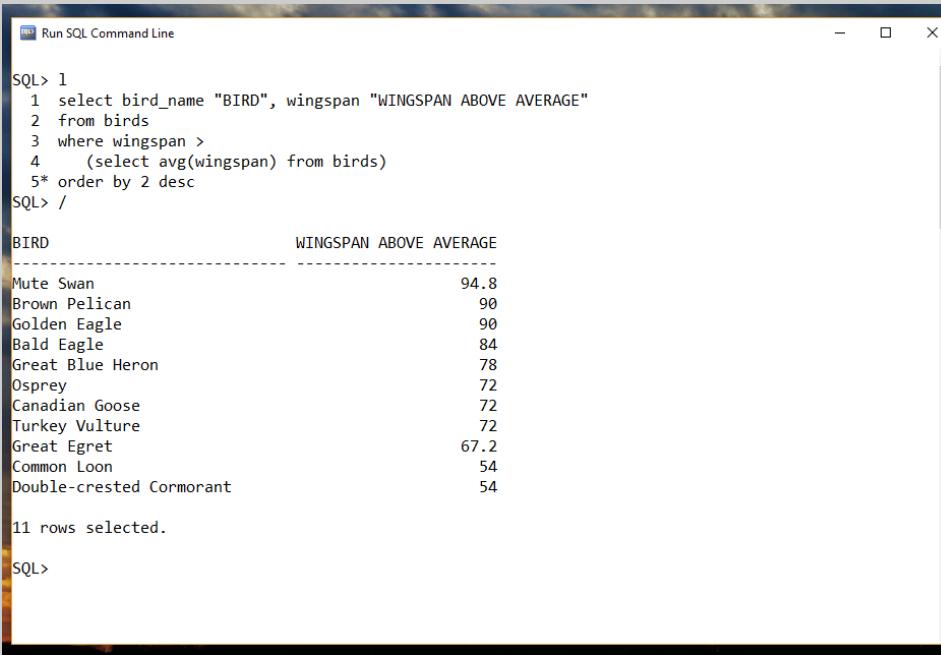
- A subquery is a query that is embedded within another query.
- Basic syntax:

```
SELECT columns  
FROM tables  
WHERE condition [ operator ]  
      SELECT columns  
      FROM tables  
      ...  
      ...
```

- Subqueries allow you to place criteria within a query even though you may not know the exact criteria values.

# Subquery Example

Here, we generate a report that lists all birds with a wingspan greater than the average wingspan. We don't know what the exact average wingspan is. Besides, that can change as data changes in the database.



The screenshot shows a Windows command-line interface window titled "Run SQL Command Line". The SQL query is as follows:

```
SQL> 1
  1 select bird_name "BIRD", wingspan "WINGSPAN ABOVE AVERAGE"
  2 from birds
  3 where wingspan >
  4   (select avg(wingspan) from birds)
  5* order by 2 desc
SQL> /
```

The results of the query are displayed in a table:

BIRD	WINGSPAN ABOVE AVERAGE
Mute Swan	94.8
Brown Pelican	90
Golden Eagle	90
Bald Eagle	84
Great Blue Heron	78
Osprey	72
Canadian Goose	72
Turkey Vulture	72
Great Egret	67.2
Common Loon	54
Double-crested Cormorant	54

Message at the bottom of the window: "11 rows selected."

# Views: Creating Subsets of Data

- Views can be created for subsets of data
- Views can be comprised of one or more tables
- A view looks and acts like a table, but is not a table.
- A view is a logical database object, a table is physical.
- Views can be used to simplify queries, group data, improve performance, and “remember” result sets.

Basic syntax to create a view:

```
CREATE VIEW view_name AS  
    SELECT columns  
    FROM tables  
    ...
```

# Examples of Views

In this example, we have created a view called “Fish\_Eaters.”

```
Run SQL Command Line
SQL> 1
 1  create or replace view fish_eaters as
 2  select birds.bird_id, birds.bird_name, food.food_name
 3  from birds, birds_food, food
 4  where birds.bird_id = birds_food.bird_id
 5  and birds_food.food_id = food.food_id
 6*   and food_name = 'Fish'
SQL> /
View created.

SQL> select * from fish_eaters;

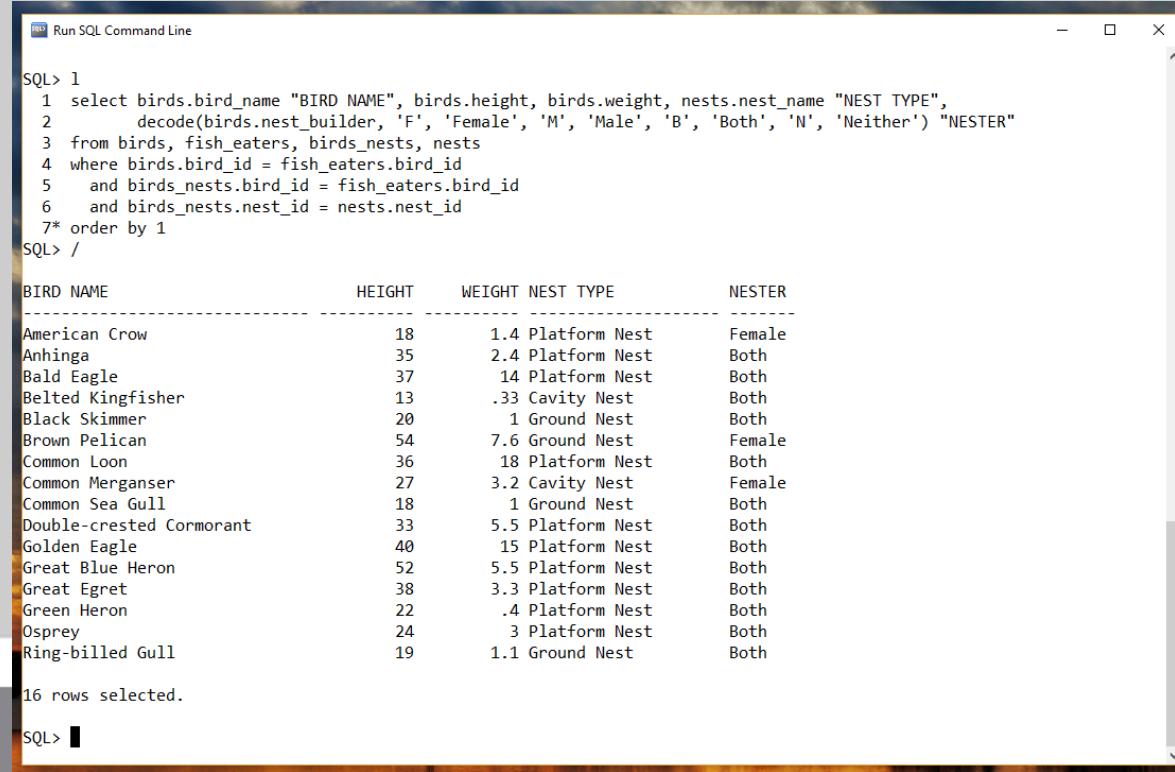
  BIRD_ID BIRD_NAME      FOOD_NAME
-----  -----
1 Great Blue Heron    Fish
3 Common Loon          Fish
4 Bald Eagle           Fish
5 Golden Eagle          Fish
7 Osprey                Fish
8 Belted Kingfisher    Fish
12 Common Sea Gull     Fish
13 Ring-billed Gull    Fish
14 Double-crested Cormorant Fish
15 Common Merganser    Fish
17 American Crow        Fish
18 Green Heron           Fish
20 Brown Pelican        Fish
21 Great Egret           Fish
22 Anhinga               Fish
23 Black Skimmer         Fish

16 rows selected.

SQL>
```

# Examples of Views

Now that we have a view for fish-eating birds, that will change appropriately as data in the tables are modified, we can easily use data in "Fish\_Eaters" as criteria by joining this view with other database tables as desired. For example, let's generate a report showing some basic information for fish eaters.



The screenshot shows a Windows command-line interface window titled "Run SQL Command Line". Inside, an SQL query is run against a database:

```
SQL> 1
  1 select birds.bird_name "BIRD NAME", birds.height, birds.weight, nests.nest_name "NEST TYPE",
  2      decode(birds.nest_builder, 'F', 'Female', 'M', 'Male', 'B', 'Both', 'N', 'Neither') "NESTER"
  3 from birds, fish_eaters, birds_nests, nests
  4 where birds.bird_id = fish_eaters.bird_id
  5   and birds_nests.bird_id = fish_eaters.bird_id
  6   and birds_nests.nest_id = nests.nest_id
  7* order by 1
SQL> /
```

The results of the query are displayed in a tabular format:

BIRD NAME	HEIGHT	WEIGHT	NEST TYPE	NESTER
American Crow	18	1.4	Platform Nest	Female
Anhinga	35	2.4	Platform Nest	Both
Bald Eagle	37	14	Platform Nest	Both
Belted Kingfisher	13	.33	Cavity Nest	Both
Black Skimmer	20	1	Ground Nest	Both
Brown Pelican	54	7.6	Ground Nest	Female
Common Loon	36	18	Platform Nest	Both
Common Merganser	27	3.2	Cavity Nest	Female
Common Sea Gull	18	1	Ground Nest	Both
Double-crested Cormorant	33	5.5	Platform Nest	Both
Golden Eagle	40	15	Platform Nest	Both
Great Blue Heron	52	5.5	Platform Nest	Both
Great Egret	38	3.3	Platform Nest	Both
Green Heron	22	.4	Platform Nest	Both
Osprey	24	3	Platform Nest	Both
Ring-billed Gull	19	1.1	Ground Nest	Both

16 rows selected.

```
SQL> ■
```

# Live Examples of Advanced Queries

1. What is the average wingspan of birds?
2. What is the average wingspan of birds that eat fish?
3. How many different types of food does the “Common Loon” eat?
4. What is the average number of eggs per type of nest?
5. What is the lightest bird?
6. Generate a list of birds that are above average in all of the following areas: height, weight, and wingspan.
7. Create a view for fish-eating birds.

# SQL For Any IT Professional

- Final Tips



# Asking the Right Questions

- In order to get the right answers, you have to ask the right questions.
- To ask the right questions, you must know your data.
- With a good, normalized database design, it will be easier to ask the right questions.
- The key is to understand why the data is needed and how it will be used by a person, organization, or customer.
- Good design and knowledge of data simplifies:
  - Data management
  - Data retrieval
  - Customer service

# Accurate, Clean, and Complete Data

One of the worst things you can do is give someone “bad” data. The following tips will help ensure you are retrieving good data.

- Accurate and Clean Data
  - Data should be entered into the database accurately
  - Constraints, manual attention to detail, quality assurance
  - Specify ALL of the right criteria when retrieving data
- Complete Data
  - Retrieve data from all of the right tables, and join tables properly
  - Otherwise you might have missing information, or too much information
- Consistency, data standards, naming conventions
- Avoid Data Redundancy!
  - Redundancy in the database makes it more difficult to effectively manage and query data
  - Redundancy makes it difficult to ensure consistency
  - There is no substitute for a solid database design!

# Managing a Growing Database

- Will more tables need to be created for additional data sets?
- Will indexes need to be created on tables to improve query performance?
- Will the database need to be further normalized?
- Will growing data require more hardware and other resources?
  - Database Administrators
  - System Administrators
- How will the data be backed up, and how often?
- How often will recoveries of data be tested?
- How will you ensure performance stays at an acceptable level?
- How will and who will manage a growing user base?
- What will your “Change Management” process be, as changes and enhancements are made to the database to adapt to ongoing business needs?

# Summary and Q&A

- In this class, you have learned some of the basic concepts of SQL and the Relational Database, as well as some of the basic standard SQL syntax to do the following:
  - Create and manage database tables
  - Insert data into tables, and manage data
  - Write SQL queries to get useful information out of the database
  - Explore some of the more advanced capabilities of SQL to creatively render reports
- We used Oracle's implementation as a base for examples in this class, but remember there are many vendor implementations of SQL, and although most adhere to national and international standards, each may have slight variations in syntax.
- With a good understanding of SQL and Relational Database concepts, you will be able to apply your knowledge to any vendor implementation of SQL, and effectively retrieve useful data from any well-designed database.

# Thank You!

## Polling Question (check boxes)

Which of the following subjects interest you for future courses?

- Advanced SQL Queries Workshop
- Oracle SQL\*Plus
- Oracle SQL Developer
- Intro to Oracle PL/SQL Programming
- Database Design
- Introduction to Unix
- Writing Unix Shell Scripts
- Oracle Database Administration
- Project Management
- Organizational Leadership