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

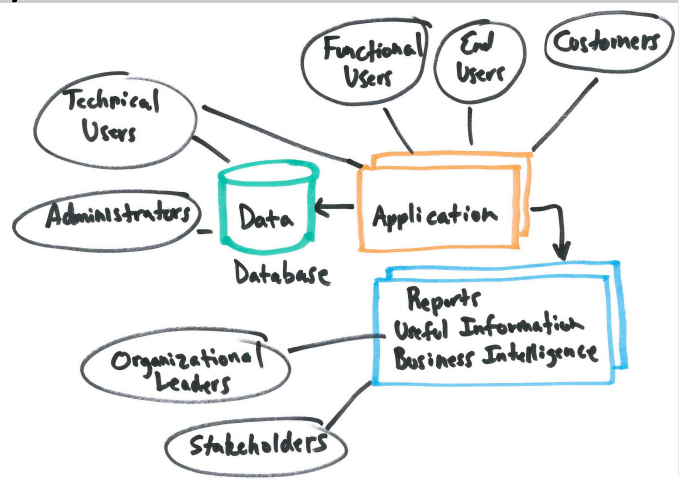
Polling Question (radio buttons)

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

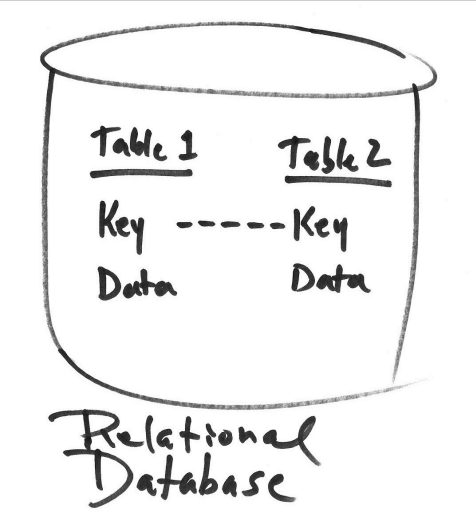
- Structured Query Language (SQL)
  - the standard language used to communicate with any relational database.
- SQL is English-like.
- National and international standards exist that make SQL universal, easy to use, and adaptable to most modern databases and applications.
- Vendor implementations (flavors)
  - 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

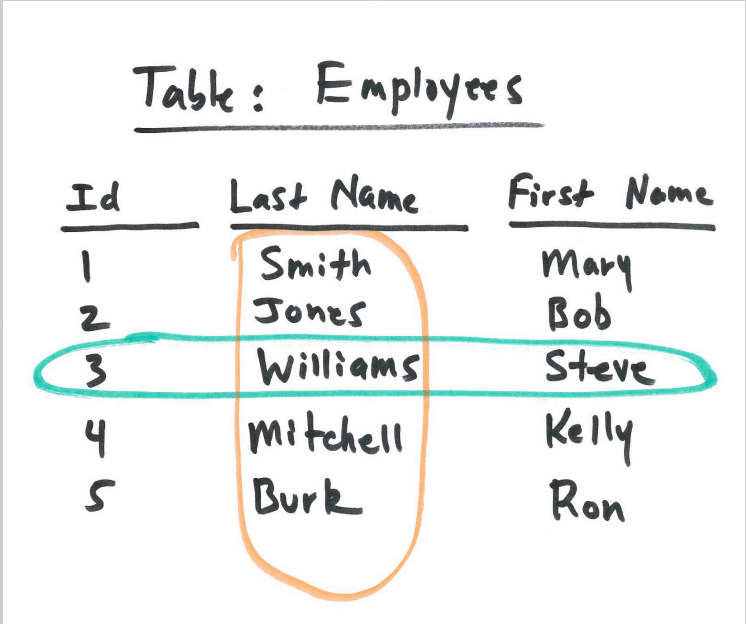
The Relational Database

- 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.



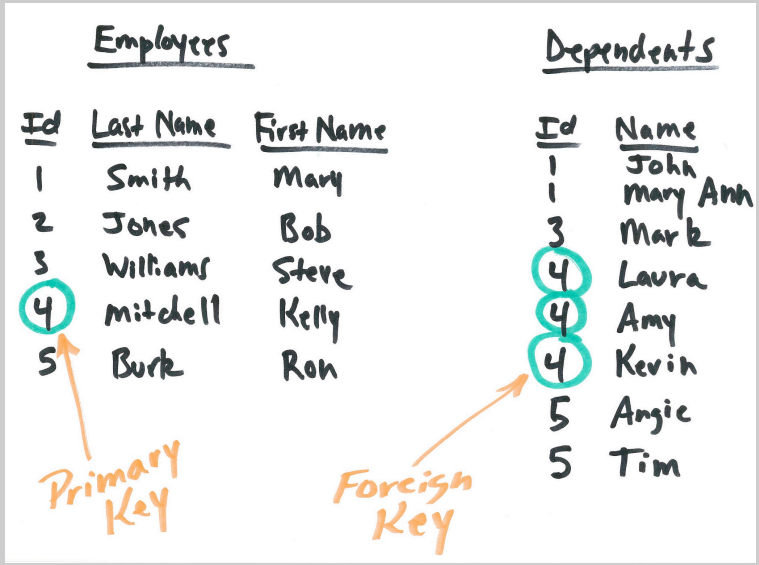
Basic Relational Database Design

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



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.



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

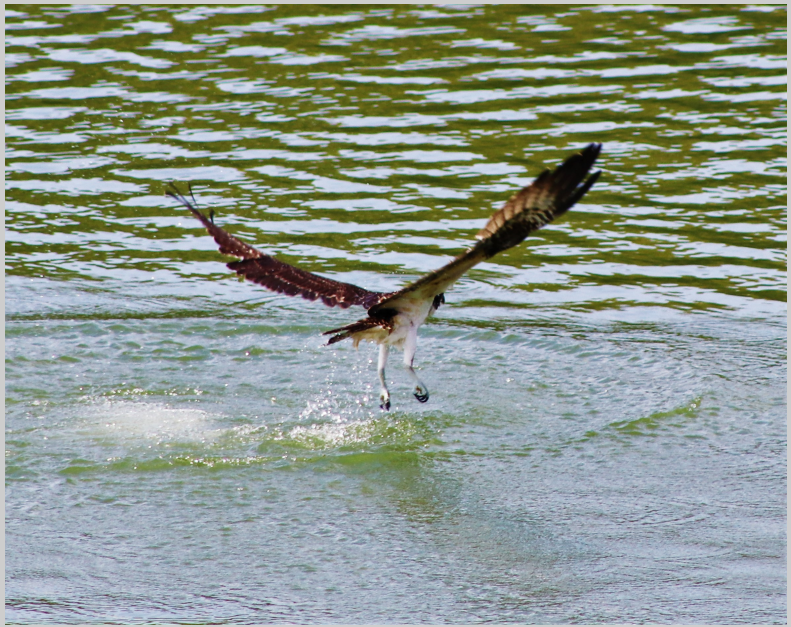
RDBMS Advantages over Flat-file and Non-relational models:

- 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.”

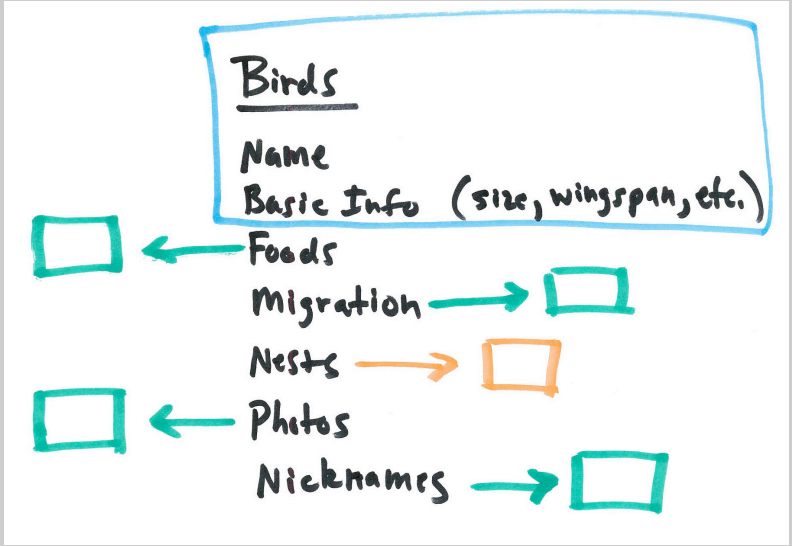
- Here is some basic information about the data:
  - Each bird will have some standard information, or data.
  - Each bird may have photographs, taken from different locations.
  - Information about each bird’s diet will be stored.
  - Information about each bird’s migration habits will be stored.
  - 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.

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:



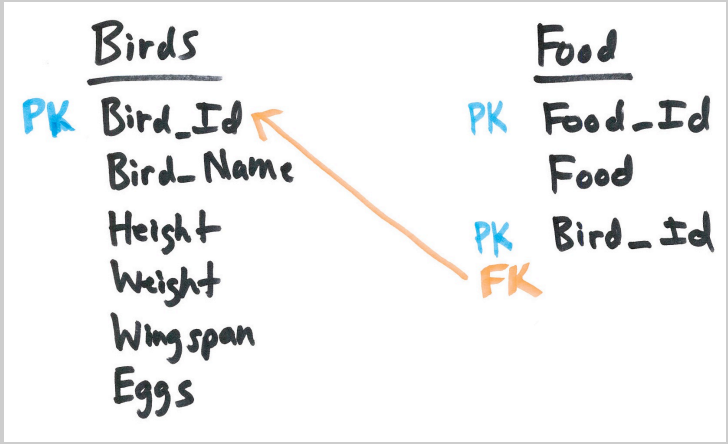
Thinking Through a Simple DB Design

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.

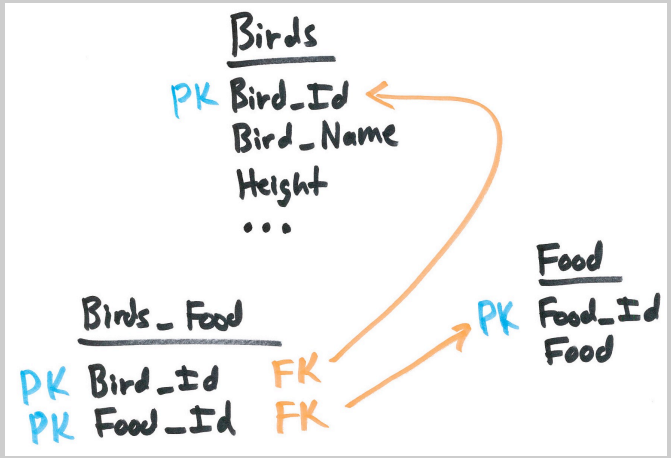
A Simple Entity Relationship Diagram

- 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

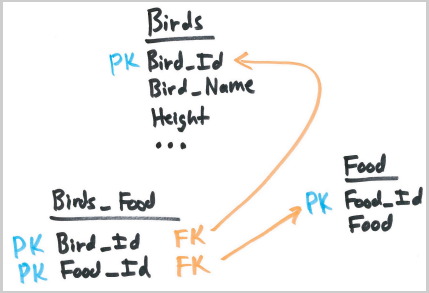
- The problem with the previous design is that we will end up with duplicate values for “Food” in the “Food” table.
- The following figure shows how we can resolve this problem by designing a table specifically to join the “Birds” and “Food” tables, ensuring not duplicate records exist in either “Birds”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



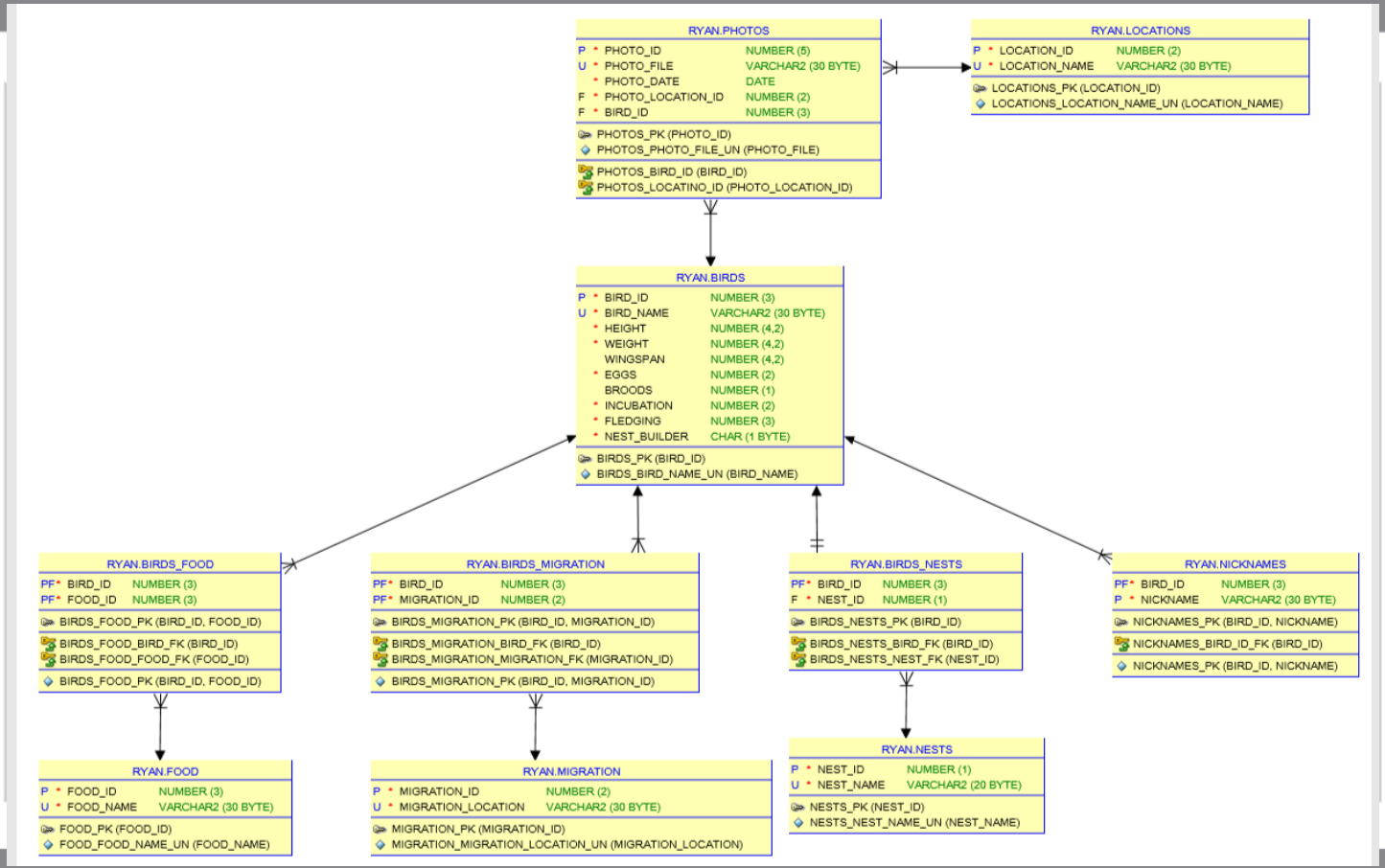
Normalizing Your Database Design

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];
```

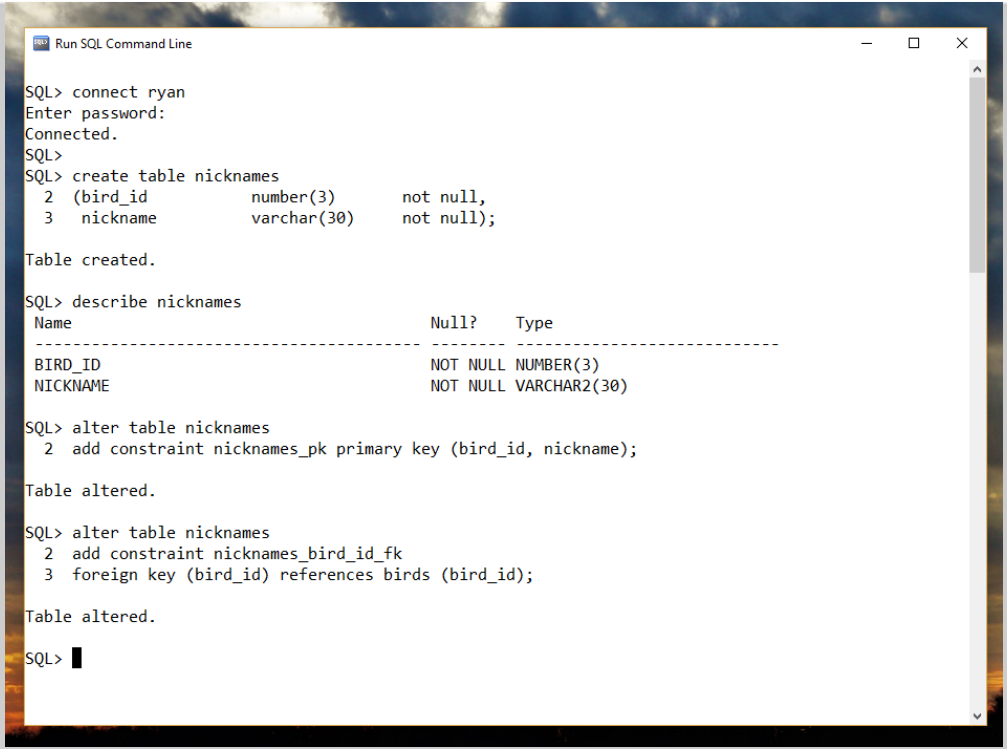
Modifying Tables

```
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

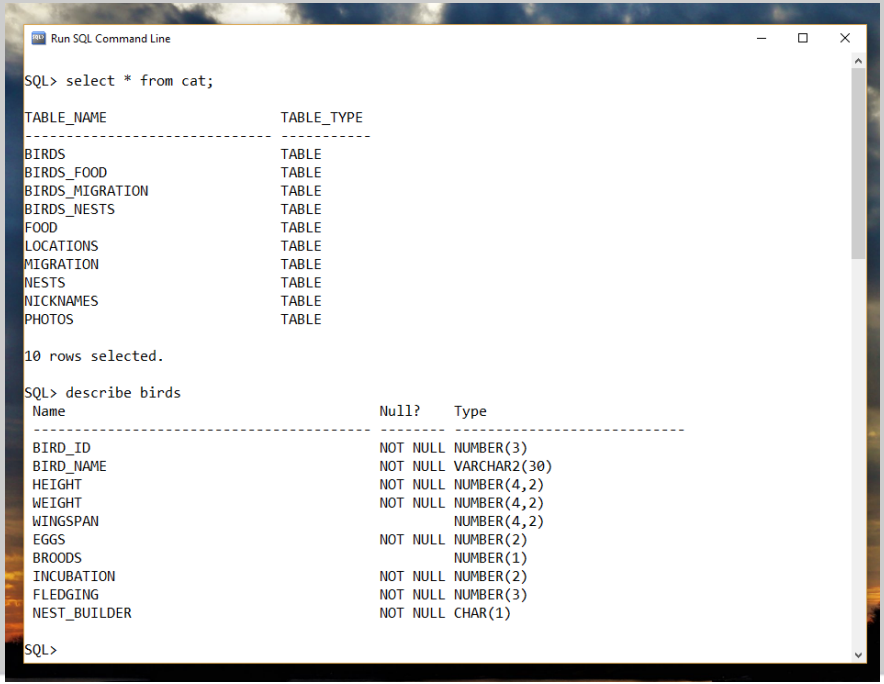
```
TRUCATE TABLE table_name;
```

Example Using Oracle SQL*Plus



Finding Information About Your Tables

Getting a list of your tables from the Oracle catalog:



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

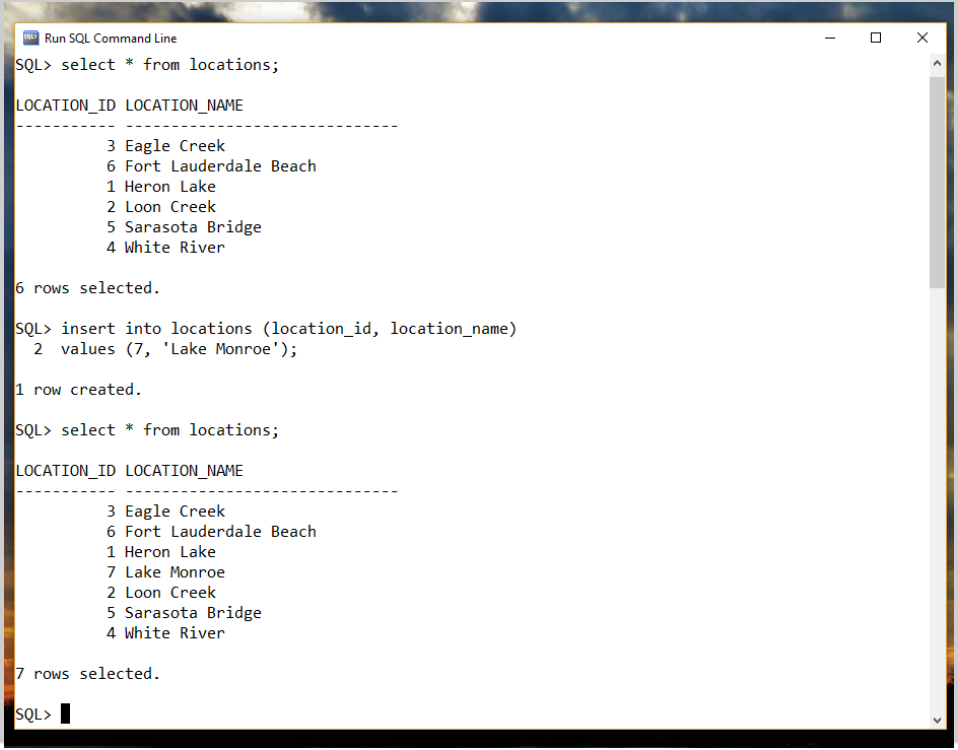
INSERT Statement

- 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



Updating/Modifying Data in Tables

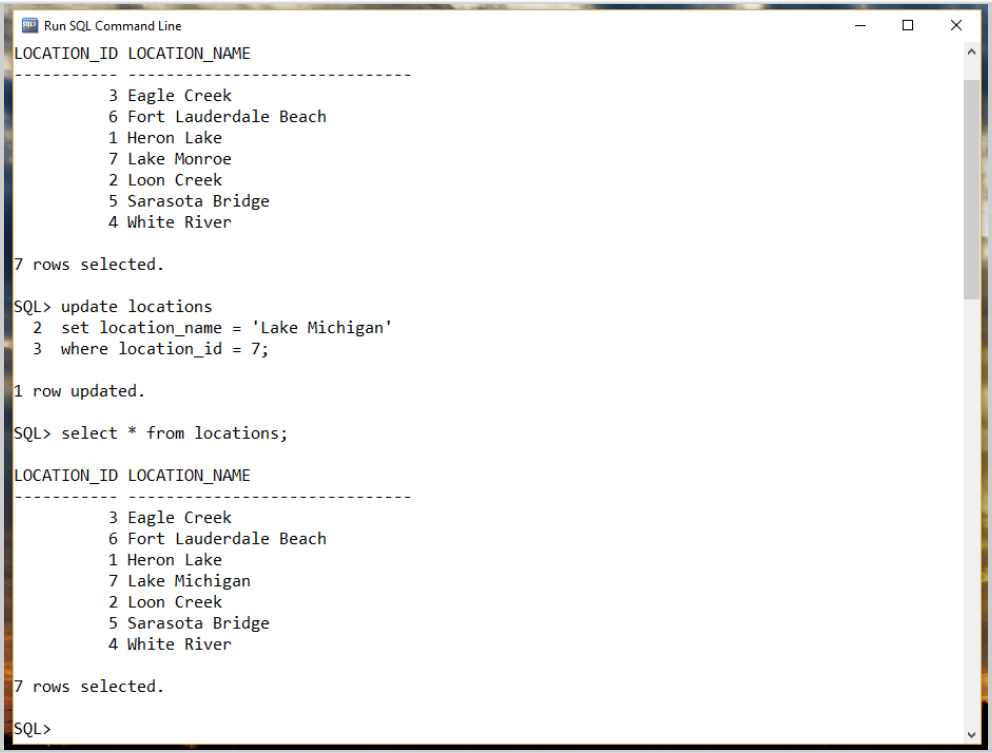
UPDATE Statement

- 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



Deleting Data from Tables

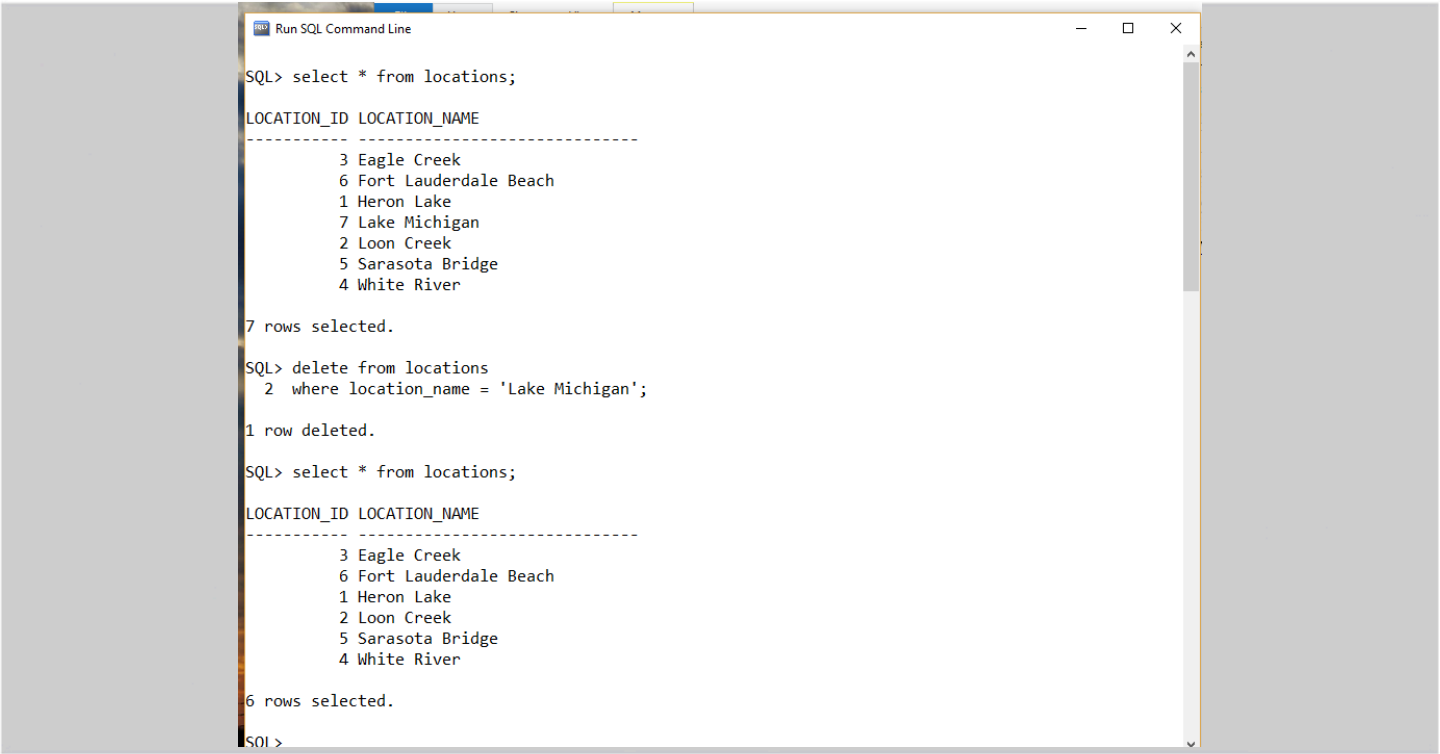
DELETE Statement

- 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:

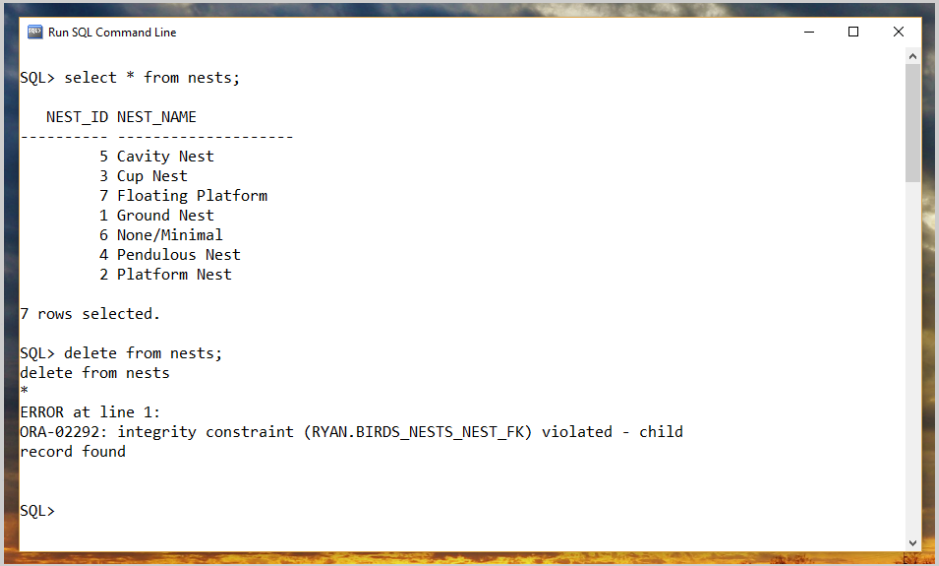
```
DELETE FROM table_name
[ WHERE condition ];
```

Deleting Data from Tables



Deleting Data from Tables

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



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?

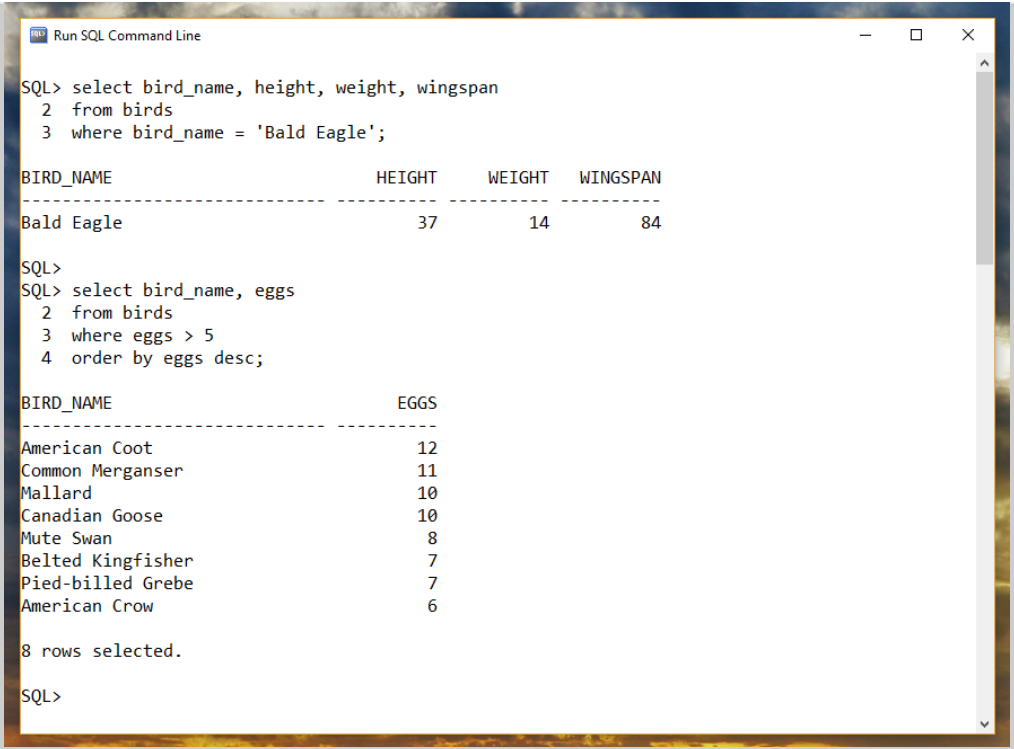
SELECT: Writing a Simple Query

- 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



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

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.

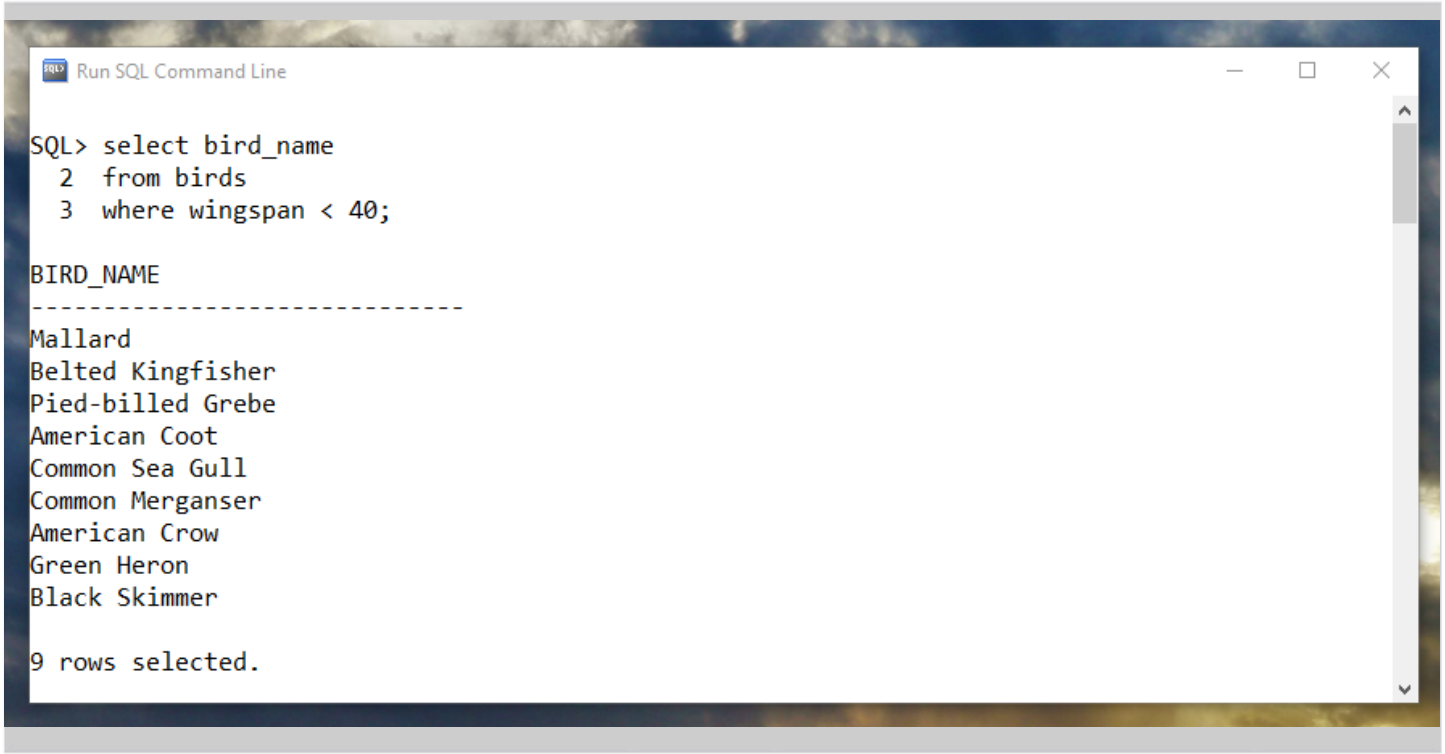
SQL For Any IT Professional

- Getting More Out of Your SQL Queries

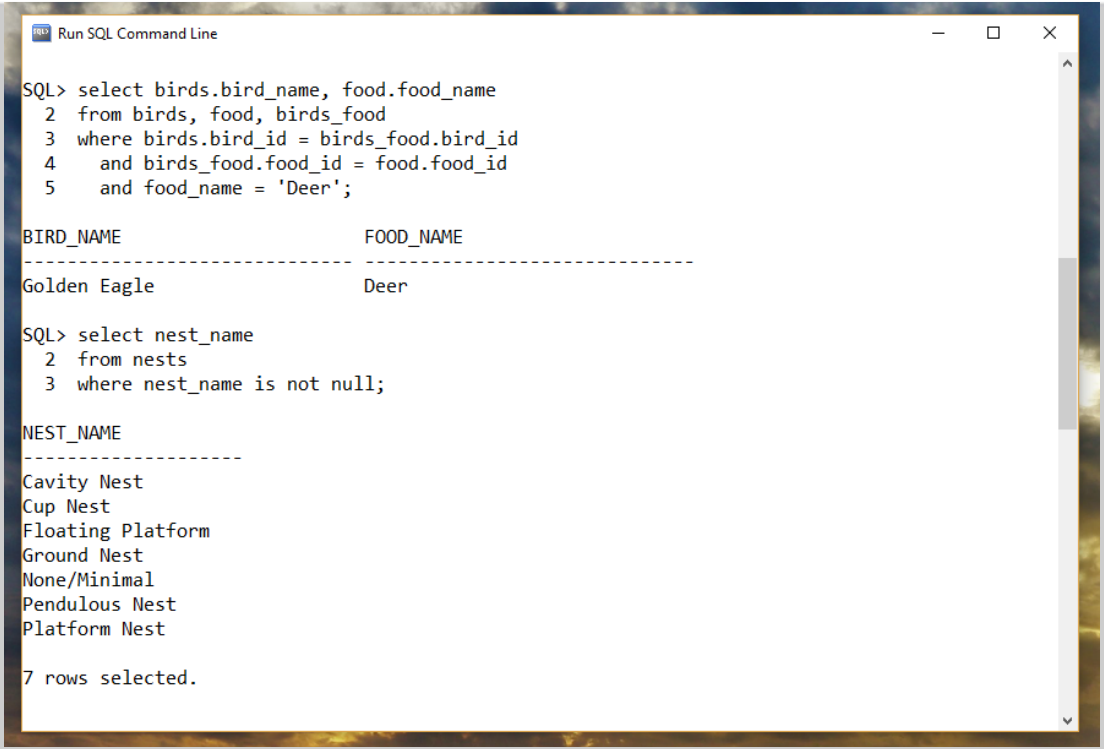
Comparison Operators

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

Comparison Operators



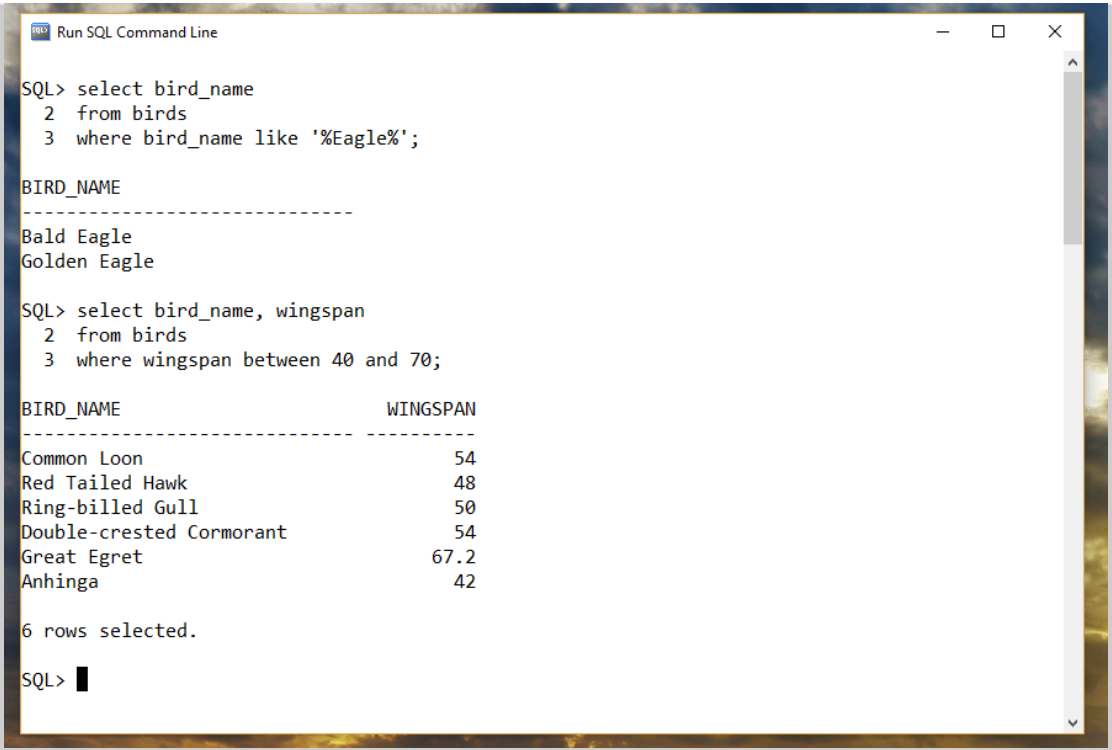
Comparison Operators



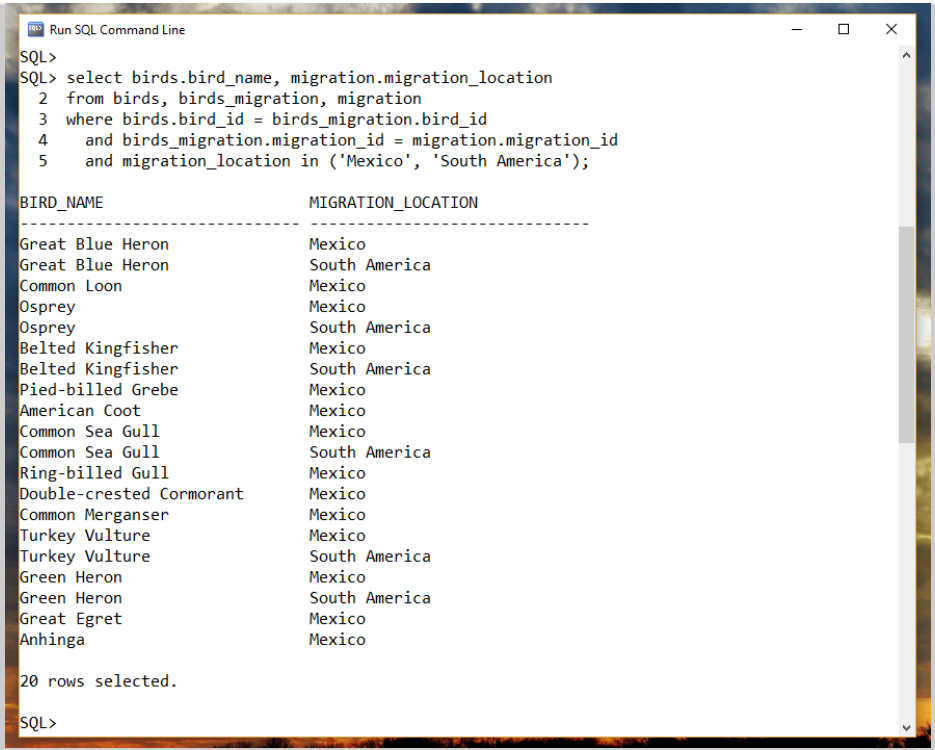
Logical Operators

- 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



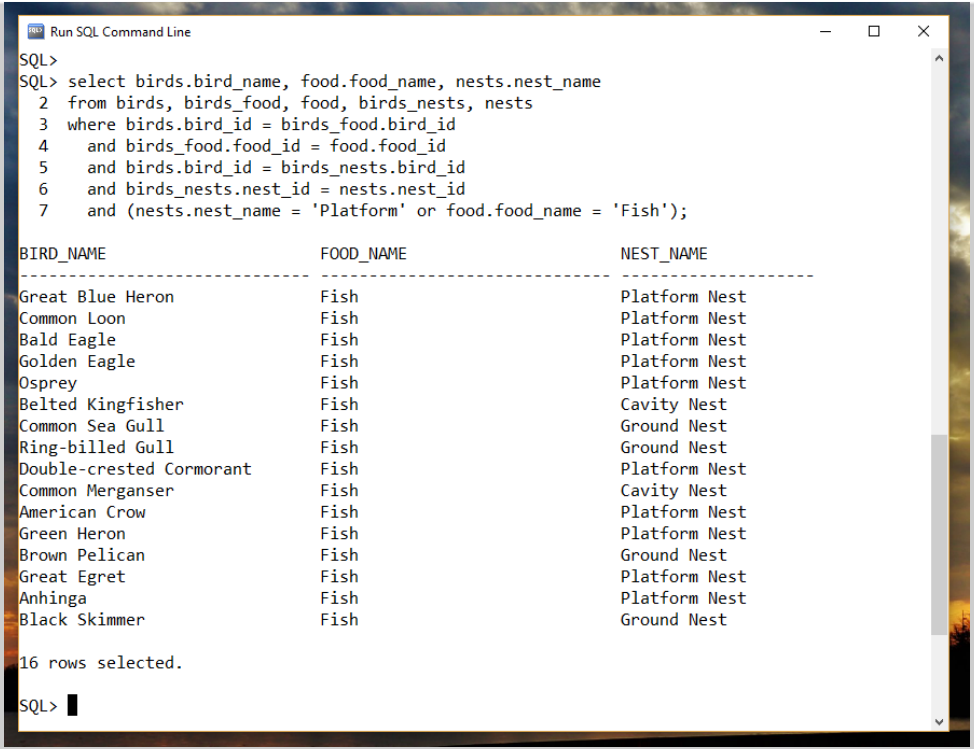
Logical Operators



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



Negative Operators

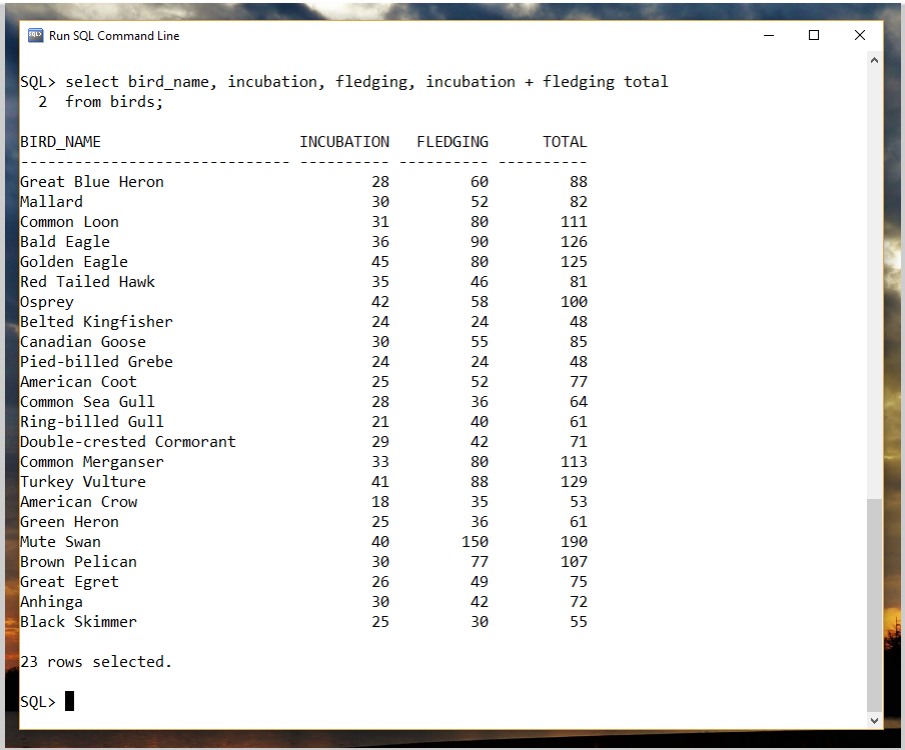
- NOT EQUAL
- NOT BETWEEN
- NOT IN
- NOT LIKE
- IS NOT NULL
- NOT EXISTS
- NOT UNIQUE

Arithmetic Operators

- Addition +
- Subtraction -
- Multiplication *
- Division /
- Combinations of arithmetic operators

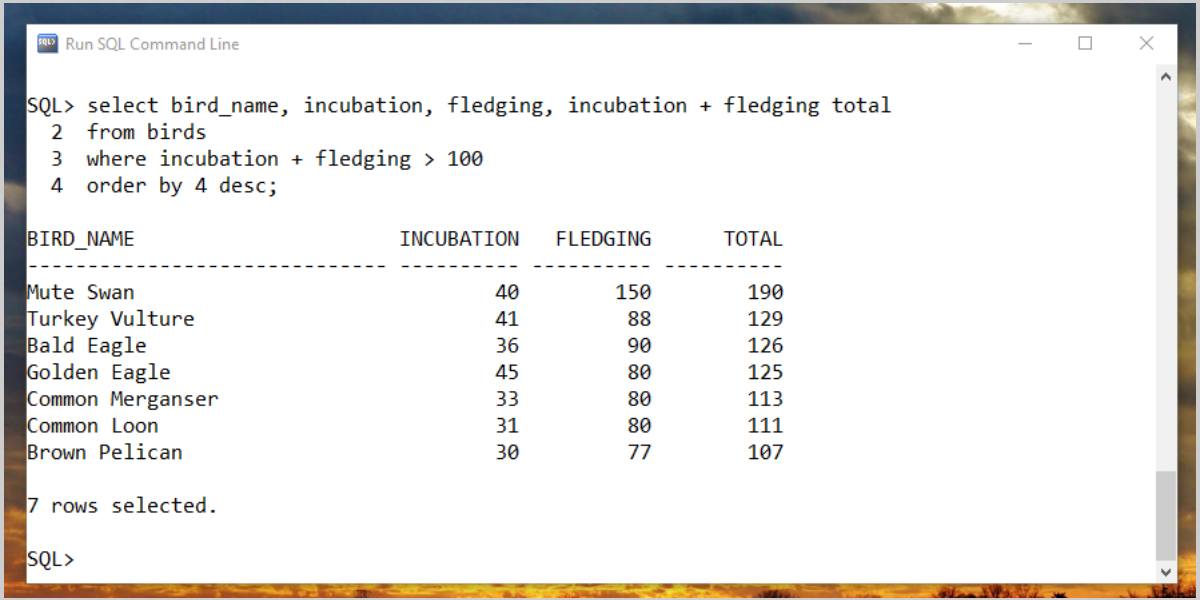
- *Ensure your math is correct! If not, incorrect or incomplete data results will be returned.*

Arithmetic Operators



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.



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:

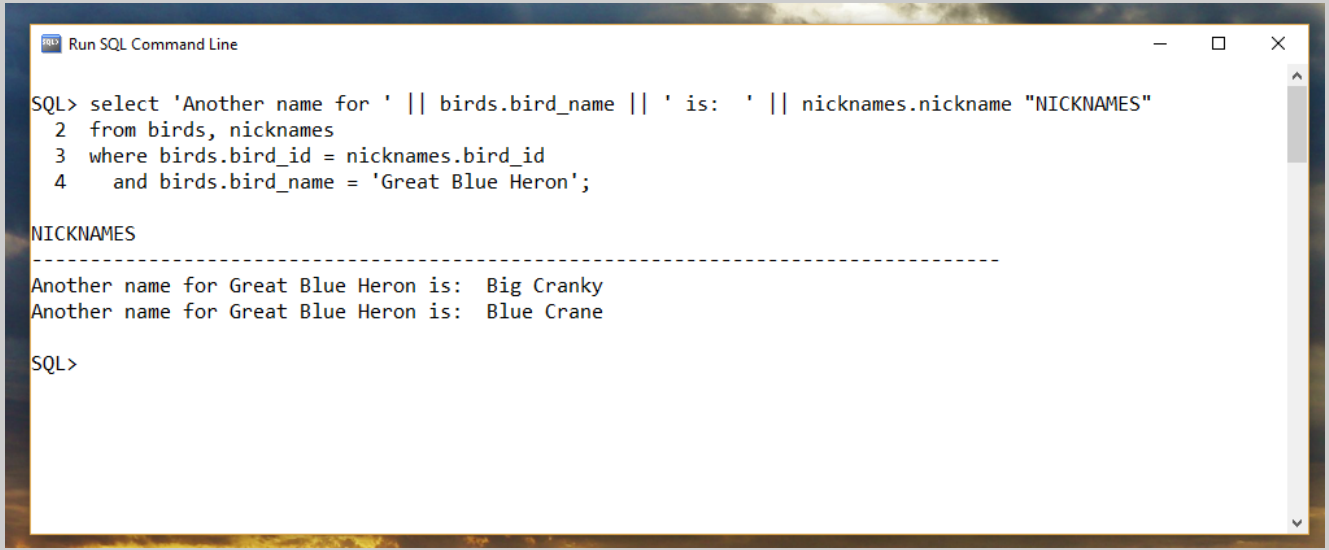
- Character Functions
- Mathematical Functions
- Date Functions
- Aggregate (allow you to group data retrieved)

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)`

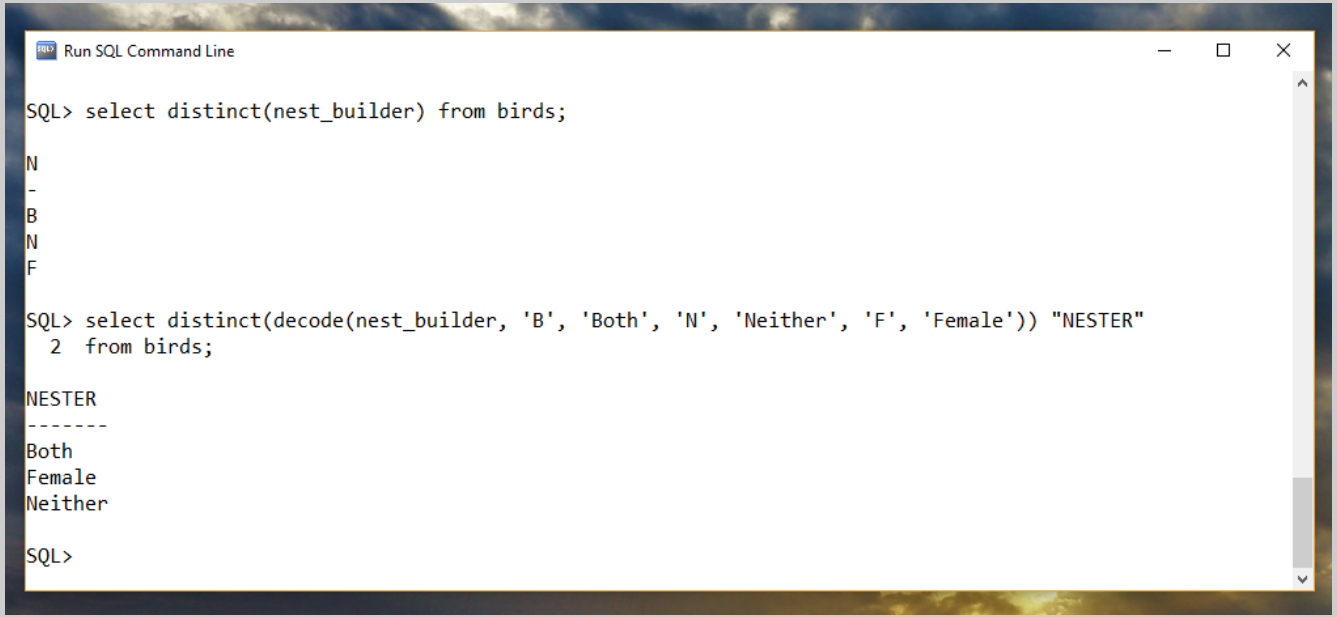
Character Functions

Concatenation example:



Character Functions

DISTINCT (UNIQUE) and DECODE example:



Mathematical Functions

• ABS	(absolute value)	`ABS(-11.3) = 11.3`
• ROUND	(round a value)	`ROUND(11.3) = 11`
• SQRT	(square root)	`SQRT(16) = 4`
• SIGN	(sign values)	`SIGN (-11.3) = -1`
• POWER	(power)	`POWER(2,2) = 4`
• CEIL	(ceiling value)	`CEIL(11.3) = 12`
• FLOOR	(floor value)	`FLOOR(11.3) = 11`
• EXP	(exponential value)	`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, '*date picture*')

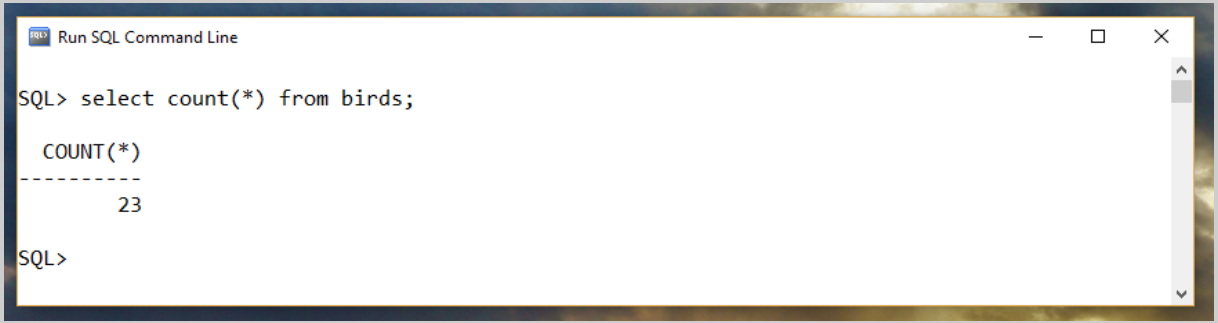
Example Date Pictures

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

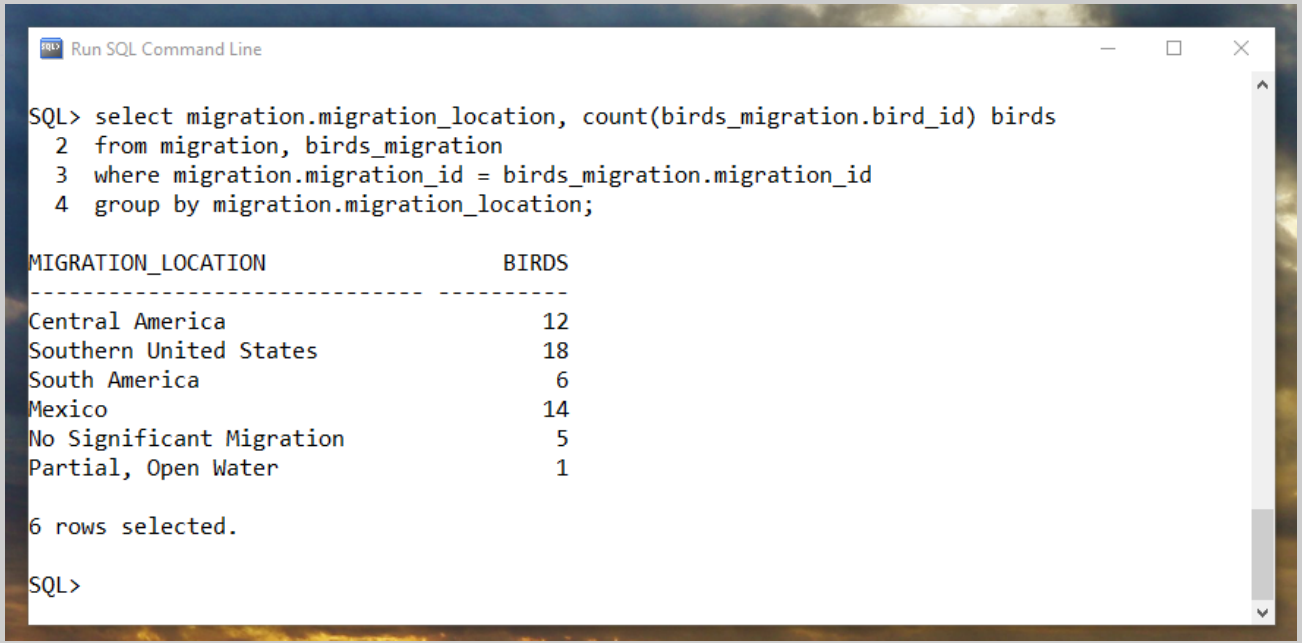
Aggregate functions are useful in summarizing and analyzing data.

- 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)



Aggregate Functions

Counting birds for each “Migration Location”:



Grouping Data

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

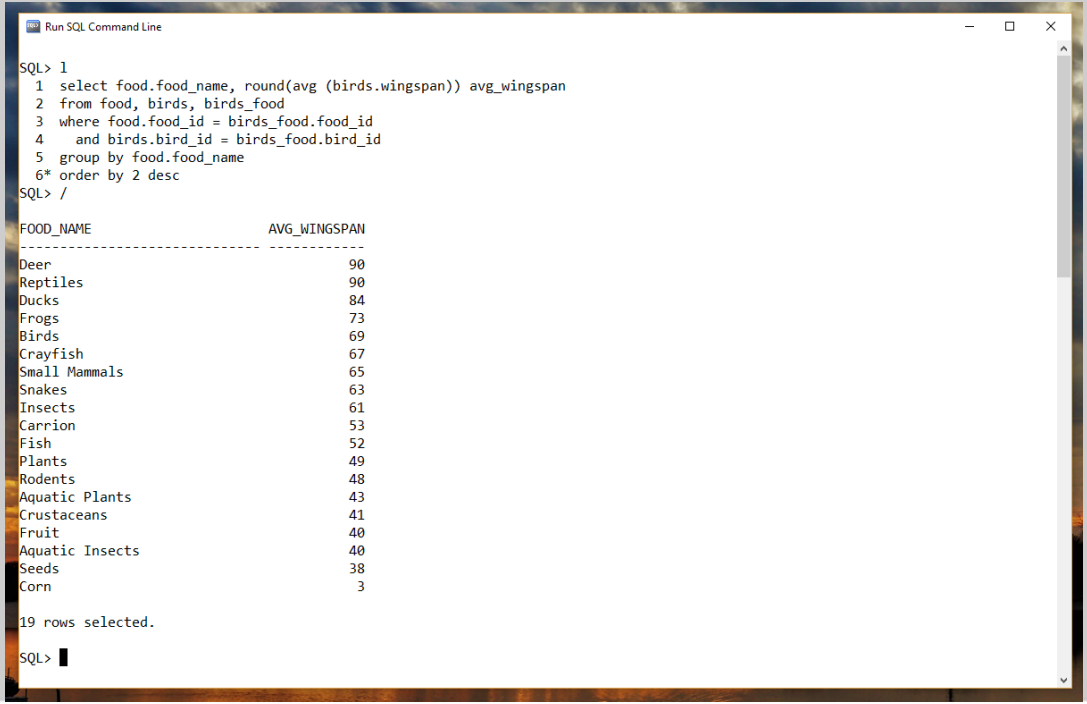
- GROUP BY — allows summarized data to be grouped
- HAVING — allows conditions to be placed on grouped data before final query results are returned

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, …
```

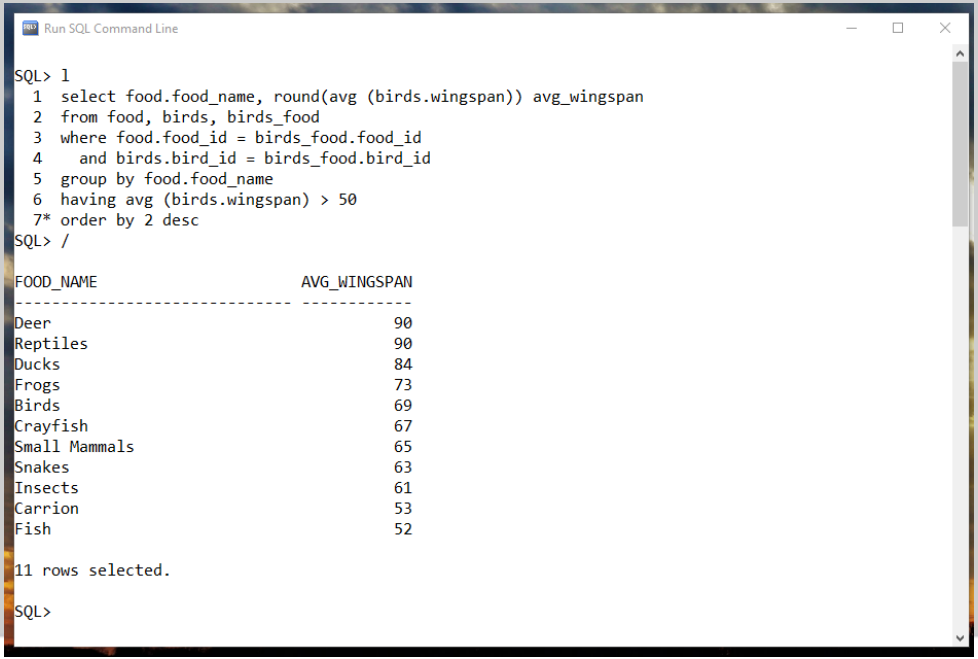
Grouping Data

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



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.



Subqueries: Defining Unknown Criteria

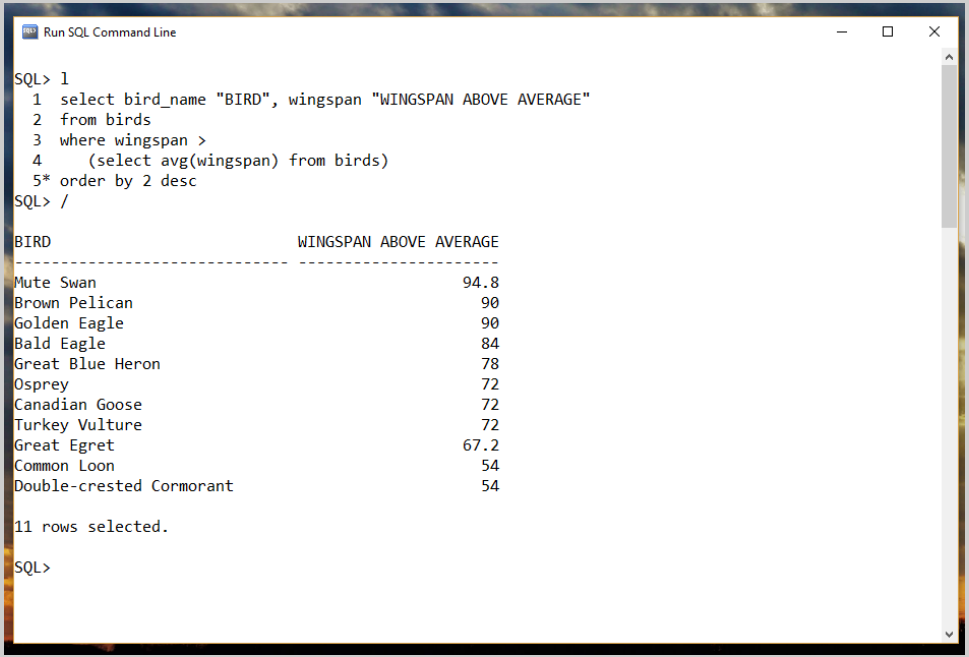
- A subquery is a query that is embedded within another query.
- Basic syntax:

```
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.



Views: Creating Subsets of Data

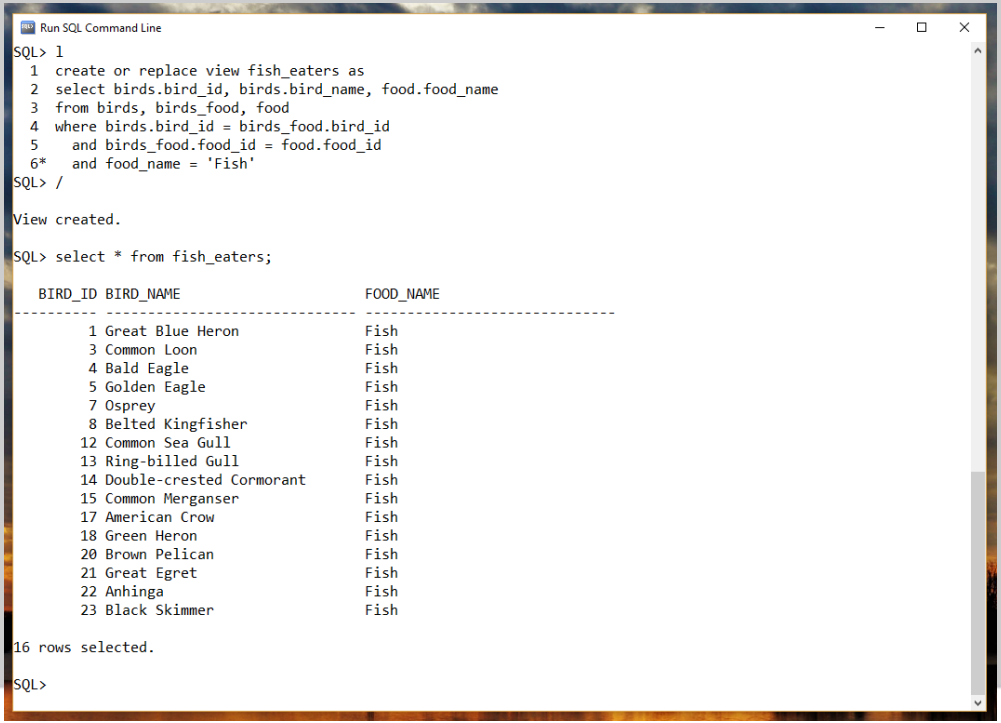
- Views can be created for subsets of data
- Views can be comprised of one or more tables
- A view looks and acts like a table, but is not a table.
- A view is a logical database object, a table is physical.
- Views can be used to simply queries, group data, improve performance, and "remember" result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

Basic syntax to create a view:

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

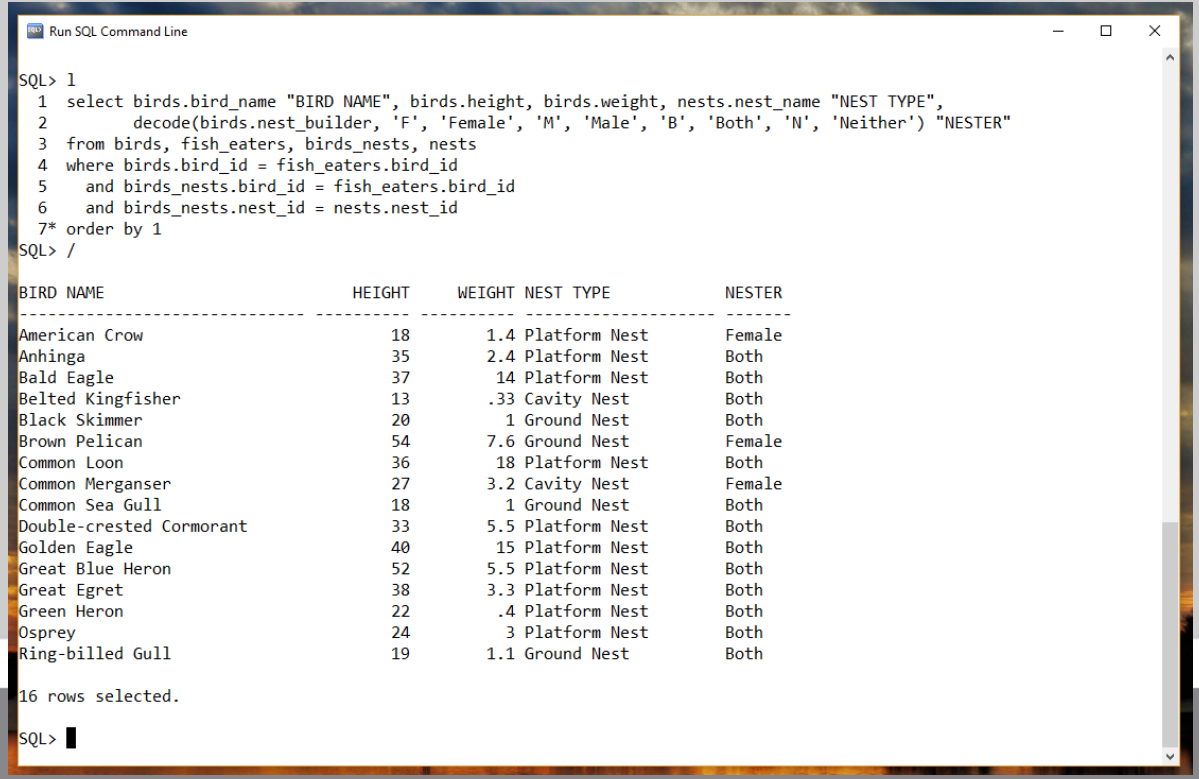
Examples of Views

In this example, we have created a view called “Fish_Eaters.”



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.



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 effective 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