My SQL Databases

Listing Databases:

```
SHOW Databases;
```

Selecting Databases

```
USE Databases;
```

Showing Current Databases

```
SELECT database
```

Inspecting Database

```
SHOW CREATE DATABASE database_name;
```

TABLES

- Data in databases is orgainized in tables
  - Break data down into columns

Data Types:

*Statically typed (when tables are created, the data type of each column must be specified)

Numeric Types:

INT: Any number w/o a decimal point
FLOAT: A number w/ decimal points
Decimal(length/precision): A precice decimal #
ex) DECIMAL(4, 2)
99.99, 4.50

Unsigned:

- Allows potential storage of large numbers in colum
- Only positive values

Boolean: TINYINT: -128 to 127

String Types: * Use single quotes (') to indicate string values

CHAR(length): String w/ fixed # of characters
- length can be 1-255

VARCHAR(length): - Lengths can go up to 65,535

TEXT
- Can be any length
- Only use for very large blocks of text.
  ex) Full text of an article, or the pages of a book

Date Types:
- DATE - Date value w/o any time. (YYYY-MM-DD)
- TIME - Time down to the seconds. 24 hr
- DATETIME - Combined time + date
  - No timezone information
  - (YYYY-MM-DD HH:MM:SS

NULL: Abscense of a value
- Behaves like ∅

* Data Integrity *

Creating Tables

```
CREATE TABLE table_name (
    column1_name data_type,
    column2_name data_type,
    ...
);
```

Primary Keys

- Uniquely identify a single (seperately) row in a table.
- Special type of column:
  1) Each value must be unique
  2) They cannot be NULL.
  3) There can only be one primary key in a table

```
#Primary key table
CREATE TABLE quotes (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    author_first_name VARCHAR(50),
    author_last_name VARCHAR(100) NOT NULL,
    content TEXT NOT NULL,
    PRIMARY KEY(id)
);
```

- AUTOINCREMENT - Instructs MySQL to generate new values for the colums

Showing Tables

```
SHOW tables
```

- Get Information on Existing Tables

```
DESCRIBE quotes;
```

- Display Original Command used to create table

```
SHOW CREATE TABLE quotes;
```

Basic Statements

- SQL commands let us explore the structure of databases
- SQL queries are often called CRUD ("Create, Read, Update, Delete")
  - Basic building block for working w/ data in any system
- SQL Quotes (') - All string values enclosed in single quotes
- SELECT - Find + return rows from a table
  - basic syntax: `SELECT column1[, column2[,...]] FROM table_name;`
    - * [ ] - Square brackets indicate optional parts of the command

ex) # Select the fruits and their quantity

IN:
```
USE fruits_db;
SELECT name, quantity FROM fruits;
```

ex) # Select all of the available colums

```
SELECT * FROM fruits;
```

- Where Clause
  - Change what data is being returned
  - Allows you to specify a condition that must be true for a given row to be displayed

ex) # WHERE CLAUSE

```
SELECT column1, column2, ...
FROM table_name
WHERE column_name = 'value'
```

* fastest/most precice way to find a single record is to use the tables primary key:

```
SELECT * FROM fruits WHERE id=5;
```

MySQL Operators:

* MySQL represents true/false with 1 and 0

`=` - equal

`!=` or `<>` - Not equal

`<` - Less than

`>` - Greater than

`<=` - Less than or equal to

`>=` - Greater than or equal to

`BETWEEN value1 AND value2` - Greater than or equal to value1 and less than or equal to value2

Aliases: Temporarily rename column, table, or misc. pieces of query.
(AS)

ex) # ALIAS

```
SELECT 1+1 AS two;
```

Miscellaneous output:
outputs arbitrary data from SQL scripts

ex) # Misc Output

```
SELECT 'I am output!' AS 'Info';
```

Command Line SQL Scripts

* mysql -u USERNAME -p -h DB_HOST -t < filename.sql

- u : Username
- p : Users password
- h : host you wish to connect to

* MySQL Clauses *

- Further Stipulations added to our queries that determine which rows are targeted

WHERE

- Filters results to do what we want using specific criteria
- Evaluates to be true for each row to be selected

ex) # Where clause example

```
SELECT * FROM employees WHERE
hire_date = '1985-01-01';
```

Chain WHERE Clauses

- Chain IN clause w/ LIKE
- Chain ALL other clauses using AND and OR

* Force evaluation grouping by using '( )' *

ORDER BY - Allows us to specify the order we wish to view our data

ex) # ORDER BY

```
SELECT column FROM table ORDER BY
column_name [ASC | DESC]
```

ex) # Chaining ORDER BY clauses #

```
SELECT first_name, last_name
FROM employees
ORDER BY last_name DESC, first_name ASC;
```

LIMIT

- Limits the number of results returned to a number or range you specify.

ex) LIMIT Clause

```
SELECT columns FROM table LIMIT count
[OFFSET count]
```

ex) # LIMIT CLAUSE Example

```
SELECT emp_no, first_name, last_name
FROM employees
WHERE first_name LIKE 'M%'
LIMIT 10;
```

* Adding OFFSET tells MySQL which row to start with

(ex) `LIMIT 25 OFFSET 50;`

Functions

- CONCAT() - Concatenates (links) strings together

  ex) # CONCAT()

  SELECT CONCAT('Hello', 'CodeUp', '!')

- LIKE / NOT LIKE
  - Use 'LIKE' to find similarities
  - '%' - Wildcards
- SUBST.
  - Extracts a part of a string
- Case Conversion
  - Upper and Lower (converts to upper or lower case
- REPLACE
  - Replace substrings

  ex) #REPLACE

IN: SELECT REPLACE('abcdefg', 'abc', '123');

OUT: 123defg

Date and Time Functions

- NOW() : current time (YYYY-MM-DD HH:MM:SS)
- CURDATE : current time (no time information)
- CURTIME : current time (HH:MM:SS)
- UNIX_TIMESTAMP : Represents time as an integer

Numerical Functions

- AVG : The mean
- MIN : Minimum
- MAX : Maximum

Casting

- Convert one type to another
- Treat a number as a string, vice versa

* Unsigned : no signs / no negatives *
* Be sure to store phone #s in strings, not number data types

* GROUP BY *

- Grouping results allows us to remove duplicates
- Can be combined w/ aggregate functions.

ex) GROUP BY vs. DISTINCT

DISTINCT

```
SELECT DISTINCT first_name
FROM employees;
```

GROUP BY

```
SELECT first_name
FROM employees
GROUP BY first_name
```

Aggregate Functions

- Works w/ data across all the rows in our result
- COUNT() : Returns the number of non-null expression values in a result

ex) # How many rows in employees table

IN:
```
SELECT COUNT(*) FROM employees;
```

ex) #Values in a certain column

```
SELECT COUNT(first_name)
FROM employees
WHERE first_name NOT LIKE 'a%';
```

GROUP BY WITH AGGREGATE FUNCTIONS

- Combining use of aggregate functions w/ the GROUP BY clause produces more meaningful results.

ex) # Find how many unique first names that do not contain an 'a'

```
SELECT first_name, COUNT(first_name)
FROM employees
WHERE first_name NOT LIKE '%a%'
GROUP BY first_name;
```

ex) # Find 10 most common hire dates for employees

```
SELECT
FROM
GROUP BY
ORDER BY COUNT(*) DESC
LIMIT 10;
```

Relationships

- Relate data from different tables

- Indexes (Indices)
  - Optimize queries + ensure integrity of data.
  - Primary Key: Unique identifier for each row.

    ex) # Primary Key

```
CREATE TABLE quotes (
  id INT NOT NULL AUTO_INCREMENT,
  author VARCHAR(50) NOT NULL,
  content VARCHAR(240) NOT NULL,
  PRIMARY KEY (id)
);
```

- Joins:
  - Allows us to obtain query results from more than one table in a single query.
  - Three Types:
    1) JOIN (aka inner join. Finds records from Table A that correspond to Table B)
    2) Left Join (Records from table A w/ nulls to Table B)
    3) Right Join (Any records from table B that have nulls on table A)

Relationship Types:

- One to many: Many rows on one table associated w/ a single row on another table

Many to many: Many different rows on one table are related to many different roles on another.

Sub-queries (nested queries)

- More than one query expression in a query.

ex) # Nested Query Syntax

```
SELECT column_a, column_b, column_c
FROM table_a
WHERE column_a IN(
    SELECT column_a
    FROM table_b
    WHERE column_b = true
);
```

ex) # Find all the department managers names and birth names.

```
SELECT first_name, last_name, birth_date
FROM employees
WHERE emp_no IN(
    SELECT emp_no
    FROM dept_manager
)
LIMIT 10;
```

* Column on the 'WHERE' needs to be same as table your calling from

temporary tables

- Data manipulation w/o modifying original data.

ex) # Create table that stores a single number in each row.

1)
```
CREATE TEMPORARY TABLE my_numbers(
    n INT UNSIGNED NOT NULL);
```

2) Insert some data into it

```
INSERT INTO my_numbers(n) VALUES (1), (2), (3), (4),
(5);
```

ex) Update (modify info in table)

```
UPDATE my_numbers SET n=n+1;
```

ex) Delete (Remove records)

```
DELETE FROM my_numbers WHERE n % 2 = 0;
```

ex) Create Table From Query Results

```
CREATE TEMPORARY TABLE employees_with_departments AS
SELECT emp_no, first_name, last_name, dept_no, dept_name
FROM employees
JOIN dept_emp USING(emp_no)
JOIN departments USING(dept_no)
LIMIT 100;
```

ex) ALTER (drop 'dept_no' column

```
ALTER TABLE employees_with_departments DROP COLUMN
dept_no;
```

ex) ALTER(add)

```
ALTER TABLE employees_with_departments ADD email VARCHAR(100)
```

1) Use own database as a scratch pad
2) write the query you need to, using database_name.table_name syntax
3) wrap the query in a `CREATE TEMPORARY TABLE table_name AS();`

Case Statements

- Returns a value when a condition is true.

* USE CASE Statements when:

1) You have more than 2 optional values.
2) You need more flexibility in your conditional tests.

CASE Statement Concise (can only test for equality, CANT test for NULL)

```
SELECT CASE column_a
WHEN condition_a THEN value_1
WHEN condition_b THEN value_2
 ELSE value_3
 END AS new_column_name
FROM table_a;
```

(Value tested can only come from a single column)

CASE STATEMENT (flexible option)

```
Select CASE
        When column_a > condition_1 then value_1
        When column_b <= condition_2 then value_2
        Else value_3
        END AS new_column_name
from table_a;
```

IF() - Evaluates to a true or false.

```
Select IF(column_a = condition_1, value_1, value_2)
AS new_column from table_a;
```

* IF (1, 0)
  (true, false)

Ideas:
Join to_date w/ emp_no using dept_no