# Introduction to MySQL

## SQL

* SQL is designed for working with relational databases
  + Most relational databases were created before SQL was standardized, so each database uses a slight variation on the standard for backwards compatibility
  + Relational databases are made up of relations (tables), which are themselves made up of records (rows) and fields (columns)
    - Each table will have a unique key, which identifies the record—when it appears as a field, it’s called the primary key
    - The unique key from another table is called a foreign key—these are what allow tables to be related
* SQL is whitespace insensitive—line breaks and indentation are often used to make complex statements more readable
* SQL is traditionally written with the keywords in all caps
* SQL parts
  + Statements: units of execution
    - Ends with ;
  + Clauses: part of a statement starting with a keyword???
  + Expression: a piece of SQL that evaluates to a value
    - Commonly used within clauses
* SQL has no undo function—once a change has been committed, it is permanent
  + Regular backups are important

## MySQL

H

# Basic SQL Syntax

## The Basic SELECT Statement

SELECT \* FROM table;

SELECT field1, field2 FROM table;

* Data is returned with the keyword SELECT
  + SELECT can return a literal string or the solution to a math problem with constants, but this functionality is rarely useful
* The fields to be returned are listed after SELECT
  + The asterisk (\*) is shorthand for everything—in a select statement, it means return all fields
  + If the field names contain spaces, other punctuation is needed, so including spaces in field names in highly discouraged
  + The order the fields are listed in is the order in which they appear in the results
* FROM table instructs the database to read from the table named table
* H

## Clauses

Add info about clauses used across CRUD tasks

### Basic Clauses

* Clauses must be appended to an expression in the following order
  + Filtering
  + Sorting
  + Display

#### Aliases

SELECT field AS alias FROM table;

SELECT field AS “Column Name” FROM table;

* AS is used to create aliases
  + Aliases assigned to fields
    - Appear in the column header for that field in the returned result
      * Normally, the column header text is field as it appears in table
* When creating an alias containing a space, it must be contained in double quotes

#### Sorting

SELECT \* FROM table ORDER BY field;

* ORDER BY is used to return the records in a specific order based on field
  + h
* h

#### Filtering

SELECT \* FROM table WHERE field = value;

* WHERE is the primary method for filtering records—it only returns those records where the expression in the clause is true
* When value is a literal string, it needs to be enclosed in single quotes

#### Display

SELECT \* FROM table LIMIT show;

SELECT \* FROM table LIMIT show OFFSET skip;

* LIMIT is used to limit the results displayed to only show number of records
* OFFSET is used to skip the first skip number of records returned before starting to display records
  + Since the order in which records are returned is random unless a sorting clause is used, using this only makes sense with a sorting clause
* LIMIT needs to be before OFFSET
* hh

H

H

H

h

## Language basics—to incorporate into other sections later

* Literal strings are in single quotes, but most other things are in double quotes
* Null is indicated with NULL
  + Null is not a value but the absence of a value
    - It’s distinct from an empty string or 0
    - Filtering based on null requires IS NULL or IS NOT NULL
    - hh
* RLIKE allows for matching with regexes
* hh

# Create

## Create schemas/databases

H

h

## Create Tables

CREATE TABLE table (

field1 type,

field2 type

);

* CREATE TABLE table creates a table named table
* After the table is named, a parenthetical contains the list of column declarations
  + Field declarations must contain the following elements in this order
    - field1—the name of the field
      * Field names should only contain letters, numbers, dashes, and underscores
    - type—the data type of the field
      * info on data types somewhere else
  + Field declarations are traditionally indented
  + Field declarations are separated by commas, but there’s no comma after the last declaration
* H

### Designating Keys

H

### Setting Default Values

J

* Unless otherwise specified, the default value for a field is NULL
* gg

### Setting Field Constraints

CREATE TABLE table (

field1 type NOT NULL,

field2 type

);

* Adding NOT NULL to the end of a field declaration prevents records where that field is null from being added to table
* hh

hj

j

### From Database Clinic

CREATE TABLE table name (

Column name variable type NOT NULL AUTO\_INCREMENT,

Column name variable type,

PRIMARY KEY (column name));

## Create Records

### Manually Add Records

INSERT INTO table VALUES (value1, value2, value3);

INSERT INTO table (field2, field3) VALUES (value2, value3);

* INSERT INTO adds a record to a table
* Matching values to their fields
  + If a value is being added for all fields, and the values for those field are in the same order as the fields in table, the fields don’t need to be listed
  + If the field names are listed, the field names and the values for those fields for the new record need to be in the same position in their respective lists
  + Not all fields in table need to be in the insert statement, but if a field indicated as not null is missing, the statement will fail
* When the values are literal strings, they need to be enclosed in single quotes
* Using DEFAULT VALUES instead of a values clause listing specific values will fill all fields in the row with the default value specified for that field
* H

#### Add Records with Values from a SELECT Statement

INSERT INTO table1 SELECT field1, field2 FROM table2;

* Instead of a values clause, a select statement can take data from table2 to be used in a record in table1
  + A record will be inserted into table1 for all matches to the select statement found in table2
* hhh

### Upload Records From…

H

## From Database Clinic

LOAD DATA LOCAL INFILE ‘absolute path to CSV file, escaping all backslashes’

INTO TABLE table name

FIELDS TERMINATED BY ‘field delimeter’

ENCLOSED BY ‘field enclosing marks, usually quotations’

LINES TERMINATED BY ‘record delimeter’

IGNORE 1 LINES # if file contains headers

(@field1, @field2, @field3, @field4) # only needed if not loading all fields or fields not in same order in source data and table

SET column name=@field1, column name=@field3; # Field2 and field4 values won’t be imported

* Note: @ signals variable name

H

h

# Read

## Aggregate Functions

### Count

SELECT COUNT(\*) FROM table;

SELECT COUNT(field) FROM table;

* COUNT is for counting records and/or values
  + COUNT(\*) counts the number of records
  + COUNT(field) counts the number of records with a value in field
* hh

h

h

h

## From Database Clinic 1

SELECT

Table1.field,

AVG(table.field)

FROM table1

JOIN table2 ON table1.field=table2.field # can be repeated for multiple joins in a single query

WHERE table.field LIKE ‘string to match where % is a multi-character wildcard’

GROUP BY table.field #required if an aggregate function like AVG is used

ORDER BY table.field

* inner join

## From Database Clinic 2

SELECT

subquery.field1 AS ‘display name’,

table2.field2 AS ‘display name’,

SUM(an aggregated column)

FROM (SELECT field1, field2, SUM(field3) AS sum FROM table1 GROUP BY field1, field2) AS subquery

JOIN table2

ON table2.field1 = CASE # works as if-else statement

WHEN subquery.field = x THEN a value in table2.field1

WHEN subquery.field = y THEN a value in table2.field1

ELSE a value in table2.field1 END

GROUP BY non-aggregated columns;

H

h

# Update

## Update Records

UPDATE table

SET changing record1 = new value1, changing record2 = new value2

WHERE condition record = condition value;

* UPDATE is used to change the data in existing records
* The WHERE condition at the end is used to limit what records are changed—without this clause, the changes described in the SET clause will be applied to all records in table
* When the new values are literal strings, they need to be enclosed in single quotes

# Delete

## Deleting Data

Removing individual data elements is updating them so the new value is null

## Deleting Records

DELETE FROM table WHERE condition record = condition value;

* DELETE is used to remove records from a table
* The WHERE clause at the end is used to limit what records are deleted—if not included, all records in the table are deleted
  + Because the FROM table and WHERE clauses are exactly like those used in read statements, using SELECT \* in place of DELETE returns the records that the delete statement will remove
* hh

## Deleting Tables

DROP TABLE table;

DROP TABLE IF EXISTS table;

* DROP TABLE table will delete the table named table
  + If there’s no table named table, this will return an error—DROP TABLE IF EXISTS will do nothing if table doesn’t exist
* h