# 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
* Schema: the structure of a database

## 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
  + Grouping for Aggregate Functions
  + 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;

SELECT \* FROM table ORDER BY field1 DESC, field2;

* ORDER BY is used to return the records in a specific order based on field
  + Sorting can be made descending or ascending by adding DESC or ASC respectively right after the field
    - Default order is ascending
  + Collation???
* To sort based on multiple fields, list all the fields in the order in which they should be used for sorting

#### Filtering

SELECT \* FROM table WHERE field = value;

SELECT \* FROM table WHERE field = value1 OR field = value2;

SELECT \* FROM table WHERE field LIKE ‘substring’;

SELECT \* FROM table WHERE field IN (value1, value2);

* WHERE is the primary method for filtering records—it only returns those records where the expression in the clause is true
  + The expression can use the equal (=) or not equal to (!=) check this one operators
  + If field is a number or date (or other value types??), the expression can also use the greater than (>), greater than or equal to (>=), less than (<), and less than or equal to (<=) operators
  + If field is a Boolean, use IS NULL or IS NOT NULL instead of the operators
* When value is a literal string, it needs to be enclosed in single quotes
* When combining clauses, the field must be given for each clause
  + Combining with OR returns records where either expression is true
  + Combining with AND returns records where both expressions are true
* LIKE is used to search within strings
  + It returns all records where substring is found in the value of field
  + The multicharacter wildcard is the percent sign (%)
  + The single character wildcard is the underscore (\_)
  + Search is case sensitive???
  + Should the wildcards be moved elsewhere to indicate they work in other literal string situations?
* IN is used to compare the value of field to a list of values
  + It returns all records where the values in the list in parentheses match the value in field
  + If the list is a list of strings, the individual items need to be in single quotes
* h

#### 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

* Hh

### Conditional Expressions

SELECT

CASE WHEN field THEN true value ELSE false value END

FROM table;

SELECT

CASE field WHEN value THEN true value ELSE false value END

FROM table;

* Conditional expressions return one of two values based on value in another field
  + Two types
    - WHEN field evaluates field as a Boolean
      * Rules for converting other types to Boolean values are at
        + 0 is false, all other numbers are true
    - field WHEN value evaluates depending on if field is or isn’t equal to value
  + Multiple CASE keywords can be part of a single conditional expression as long as all the fields are from the same table???
  + h
* The location where true value or false value should go is placed after END
* H
* H
* h
* hh

h

h

h

# Create

## Create Databases

H

h

## Create Tables

CREATE TABLE table (

field1 type,

field2 type

);

* CREATE TABLE table creates a table named table
  + Table names should only contain letters, numbers, dashes, and underscores
* 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

CREATE TABLE table (

field1 type NOT NULL AUTO\_INCREMENT,

field2 type

PRIMARY KEY (field1));

* Unique to each RDBMS—double check this syntax—taken from database clinic
* All tables have a primary key—the PK constraint just binds this value to a field in table for use in queries
* h

### Setting Field Constraints

* Field constraints are used to control what data can go into a given field

#### NOT NULL

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

#### Default Values

CREATE TABLE table (

field1 type DEFAULT value,

field2 type

);

* Adding DEFAULT value to the end of a field declaration makes value the default value for that field
  + Default values are used in a field when the create record statement doesn’t specify the value for that field
  + If value is a string, it needs to be enclosed with single quotes
* Unless otherwise specified, the default value for a field is NULL
* h

#### Unique Values

CREATE TABLE table (

field1 type UNIQUE,

field2 type

);

* Adding UNIQUE to the end of a field declaration requires all values in that field to be unique
  + An attempt to add a record that violate the unique constraint will cause an error
  + NULL as exempt from this???—depends of the RDBMS

h

h

h

h

h

h

h

## 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 with a NOT NULL constraint has no value, 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

h

### 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

## Querying for Aggregate Data

### Aggregate Functions

#### Counting

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
* h

#### Mathematical Operations

SELECT AVG(field) FROM table;

SELECT MIN(field) FROM table;

SELECT MAX(field) FROM table;

SELECT SUM(field) FROM table;

* AVG returns the mean of the values in field
* MIN returns the smallest value in field
* MAX returns the largest value in field
* SUM returns the sum of all the values in field
* h

#### Unique Values

SELECT DISTINCT field FROM table;

SELECT DISTINCT field1, field2 FROM table;

* DISTINCT returns a list of all the values in field
* Multiple fields can be provided—in this case, the query returns a list of all the unique combinations of those fields’ values
* The DISTINCT keyword can also be used inside other aggregate functions before the field name to perform the aggregate function based only on the unique values in that field

H

H

H

H

### Grouping for Aggregate Functions

SELECT field, COUNT(\*)

FROM table

GROUP BY field

;

* GROUP BY allows an aggregate function to be applied to groups of records instead of all the records in the table
  + The records are grouped by their value in field—all records with the same value in field are in a group
  + For clarity, the field in the group by clause should also be one of the fields in the query
* H
* h

#### Filtering Aggregate Groups

SELECT field, COUNT(\*) AS c

FROM table

GROUP BY field

HAVING c = value

;

* The HAVING clause allows for filtering of aggregated values
  + The clause’s expression follow all the same rules as the general Filtering clause; see that section for details
* h

H

H

H

H

h

## Joins

SELECT \*

FROM table1

JOIN table2 ON table1.field = table2.field

;

* Joined queries pull related data from multiple tables
  + Usually this is done through primary and foreign keys
    - See Designating Keys for info about creating a primary key field
    - See h about creating relationships
* When joining tables, records are matched based on the specified fields and returned depending on if there’s a matching record in all tables and the type of join
  + INNER JOIN queries return data from only those records which match to a record in the other table
    - This is the default join type used with only the JOIN keyword
  + LEFT OUTER JOIN queries return data from all records in the table mentioned immediately after the join keywords as well as all matching data from other table(s)
    - RIGHT OUTER JOIN is the same, but returns all records from the other table—does MySQL support?
  + FULL OUTER JOIN queries return data from all records in both tables
    - Does MySQL support?
  + Check what keywords MySQL supports
* In the join clause, the field names need to be written as table.field to clarify which table the field is from

### Using Aliases in Joins

SELECT \*

FROM table1 AS one

JOIN table2 AS two ON one.field1 = two.field2

;

SELECT one.field1, two.field2

FROM table1 AS one

JOIN table2 AS two ON one.field1=two.field2

;

* For long table names and long queries, using aliases on the table names can be helpful
* Aliases can be used before they’re declared in the query

### Using JOIN with More than Two Tables

SELECT \*

FROM table1

JOIN table2 ON table1.fieldA = table2.field2

JOIN table3 ON table1.fieldB = table3.field3

;

* It’s possible to have multiple tables on the “right” of a join
* When a many-to-many relationship relies on a junction table, this is the type of join required to query the full relationship
  + If all of the records on either side of the many-to-many relationship need to be returned by the query, the table those records are in needs to be table1 and the joins needs to be LEFT JOIN
    - Check words on joins
  + g

### Using CASE to Join Tables with No Matching Fields

Have a better example based on below

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;

* g

h

h

h

h

h

# Update

## Update Database Schemas

H

h

## Update Table Schemas

### Add New Field

ALTER TABLE table ADD field type;

* To add field to an existing table, use ADD field type
  + List of type options somewhere else
  + All the values in field will be of the default value
    - For more info on default values, see Default Values
  + Constraints can be added to added fields, see Setting Field Constraints for more info

H

H

H

H

H

h

## 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

H

## Transactions

START TRANSACTION;

INSERT INTO table1 VALUES (value1, value2, value3);

UPDATE table2 SET record1 = value1 WHERE record2 = value2;

END TRANSACTION;

* A transaction is a group of statements executed as a single unit of work
  + If any of the operations fail, the entire transaction fails, rolling the database state back to before any of the statements were executed
* Transactions begin with START TRANSACTION; and end with END TRNASACTION;
* To stop a transaction in process, use the ROLLBACK; statement
  + Confirm keywords
* Transactions are often used to improve performance because the data is committed only at the end of the transaction

## Triggers

H

* Triggers are operations that occur when a specific database even occurs
* Triggers are highly idiosyncratic

H

## Views and Subselects

H

### Views

H

### Subselects

SELECT table1.field1, subselect.fieldA

FROM (SELECT fieldA FROM tableA) AS subselect

JOIN table1 ON table1.field1 = subselect.fieldA

;

* Subselects are select queries used as a data source in another select query
* Subselects are enclosed in parentheses and don’t have a semicolon at the end
* Subselects are usually given Aliases for convenience
* g

H

H

H

H

h

# Delete

## Delete Databases

H

## Delete 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

## Delete 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 Data

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

# Data Types and Working with Data Elements

* The TYPEOF(data) returns the data type of data
  + Check accuracy

## Numbers

* Numbers have both precision and scale
  + Precision is how many decimal places are reported
  + Scale is the magnitude (absolute value) of the number

### Numeric Data Types

* Integers
  + Integer
  + Decimal
  + Money
* Real—sacrifice accuracy/percision for scale
  + Real
  + Float
* Determine how mathematical operations with numbers of different types have their answers typed
* h

#### Mathematical Operations

* Mathematical operators
  + + Addition
  + - Subtraction
  + \* Multiplication
  + / Division
  + % Modulo division (remainder)
  + Check accuracy
* ROUND(number) rounds number to an integer
  + ROUND(number, precision) rounds number to precision decimal places
  + Check accuracy
* h

H

H

H

H

h

## Dates and Times

* Dates and times can be used for both chronological and durational values
* The standard SQL format is yyyy-mm-dd hh:mm:ss
  + Check accuracy
* Dates and times are usually saved in UTC

H

H

H

H

H

H

h

## Strings

* Literal strings are in single quotes
  + But most other things are in double quotes—clarify what these things are
  + MySQL will also accept double quotes
    - Double check accuracy
  + To use a single quote in a string, escape it with another single quote
* RLIKE allows for matching with regexes
* String comparisons are case sensitive
  + Check this is true with MySQL
* Wildcard characters for filtering based on string matching found in Filtering and might be better here
* h

### String Manipulation

* String manipulation functionality is somewhat idiosyncratic
  + Check functions below—if MySQL uses different function for described function, change function
* String manipulation functions
  + SUBSTR(string, start, length);
    - Returns a string from the start character of string
    - Has an optional third parameter to specify the length of the string to be returned
  + LENGTH(string);
    - Returns the length of string
  + TRIM(string);
    - Returns string without leading or trailing spaces
    - Check if MySQL has trim from left only, trim from right only, trim character other than spaces
    - h
  + UPPER(string);
    - Makes all the letters in string uppercase
    - Check collation used
  + LOWER(string);
    - Makes all the letters in string lowercase
    - Check collation used
  + CONCAT(string1, string2);
    - Returns string1 and string2 combined into a single string

H

H

H

h

## Boolean Values and Null

H

### Booleans

H

### Null

* 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

h