MySQL & DB Design

Notes open for creative commons use by Kyle Miskell @ developer blog: https://kylemiskell.com, *github:* SmilingStallman, *email:* klabweb@tuta.io

Learning Resources
Primary: *"Learning PHP, MYSQL, and JavaScript – With jQuery, CSS, and HTML5"* (2018)
http://lpmj.net/5thedition/

Intro to MySQL

-Free, highly scale-able DBMS commonly used in web servers (Mariadb on

-Database – collection of records stored in a way where info can be retrieved rapidly
-DB contains tables, which hold records or rows, which are composed of fields of specific data, set in specific columns, which determine the names for fields

MySQL CLI 101
-Can start on Mac, if installed via homebrew, via *brew services start mysql.* Install and purge instructions at: *https://stackoverflow.com/questions/4359131/brew-install-mysql-on-mac-os*

-Start via: *mysql.server start*
-Connect via: *mysql -u root*

CLI Command Prompts:
mysql> - Ready and waiting for a command
-> - Waiting for the next line of a command
'> - Waiting for the next line of a string started with a single quote
"> - Waiting for the next line of a string started with a double quote
`> - Waiting for the next line of a string started with a backtick
/*> - Waiting for the next line of a comment started with /*

-ctrl+c will exit mysql. Use *\c* (with no semi-colon) at end of line instead.

-Also many front end tools for running queries and seeing results

MySQL Casing
-Commands are case in-sensitive, though uppercase usually preferred for clarity
-Table names are case sensitive on Linux and Mac, but not on Win

DB Basics
-*CREATE DATABASE dbnameX;*

-*SHOW databases;* - view list of all DB

-*USE dbnameX;* - Connects to DB

Creating Users
-*CREATE USER 'username'@'localhost' IDENTIFIED BY 'password';* *//or* other hostname

Granting Access
-*GRANT ALL ON db1.* TO 'jeffrey'@'localhost';*

-Also **.** for all db and obj and *db1.objectX* for only *db1* and its object called *objectX*

-Many more privileged types, as well as *REVOKE* to remove privileges

Logging Into DB
-*mysql -u root* //as root before pwd set
-*mysql -u username -p dbNameX* //as created user, dbNameX optional

Changing Used Pwd
-*ALTER USER 'username'@'localhost' IDENTIFIED BY 'password'*

-Should do for root user immediately after installing MySQL for security access, as well as for webserver DB users

Table Creation
-*CREATE TABLE tablename (name TYPE, name TYPE, name TYPE);*

-*DESCRIBE tablename;* - will show all field names, types, if null, if key, default field val, etc.
-*SHOW tables;* - list all tables in DB

Data Types
-*CHAR(n)* – Creates type of *n* bytes and if smaller than *n*, pads with strings. String type, n <= 255 bytes.

-*VARCHAR(n)* – Creates type of up to *n* bytes, only as large as needed for each field. String type, n <= 65,535.

-*VARCHAR* requires a bit of overhead to track sizes of each entry, so if all records of same size, use *CHAR*. *VARCHAR* useful if entries vary widely in size.

-*BINARY(n)* – string of bytes that do not have an associate char set (ex. gif image). n <= 255.
-*VARBINARY(n)* – n <= 65,535

-*TINYTEXT(n)* – n <= 255. Text filed. Hold char data. TEXT fields cannot have default val. When set index on TEXT, specify first *n* char to use for index. VARCHAR faster for if using entire field contents. TEXT better for faster searching of partial (initial *n*) data.
-*TEXT(n)* – n <= 65,535.
-*MEDIUMTEXT(n)* – n <= 1.67e+7
-*LONGTEXT(n)* – n <= 4.29e+9

-*TINYBLOB(n)* – n <= 255. Binary data (no char set). Binary Large Object. Similar to BINARY, but

cannot hold default values.
-*BLOB(n)* – n <= 65,535
-*MEDIUMBLOB(n)* – n <= 1.67e+7
-*LONGBLOB(n)* – n <= 4.29e+9

-*TINYINT:* -128 to 127 (signed) or 0 to 255 (unsigned). *n* represents display width (ex. *TINYINT(4)* is a width of 0000)
-*SMALLINT*: -32,768 to 32,767 (signed) or 0 to 65,535 (unsigned)
-*MEDIUMINT*: -8.38+6 to 8.38+6 (signed) or 0 to 1.67e+7 (unsigned)
-*INT / INTEGER*: -2.15e+9 to 2.15e+9 (signed) or 0 to 4.29e+9 (unsigned)
-*BIGINT*: -9.22e+18 to 9.22e+18 (signed) or 0 to 1.84e+19 (unsigned)
-FLOAT: -3.40e+38 to 3.40e+38 (signed)
-DOUBLE / REAL: -1.80e+308 to 1.80e+308 (signed)

-To set as unsigned, specify *UNSIGNED* anfter declare type (ex. *INT(4) UNSIGNED)*

-Can declare *ZEROFILL* after declaring type to pad zeros to entry if not of specified size to fill *n* used when declaring field (ex. *TINYINT(4)* with value 29 pads to 2900

-*DATETIME* - '0000-00-0000:00:00'
-*DATE* - '0000-00-00'
-*TIMESTAMP* - '0000-00-00 00:00:00' (1970 through 2037 only). Current time as default val.
-*TIME* - '00:00:00'
-*YEAR* – 0000 (Only years 0000 and 1901–2155)

AUTO_INCREMENT
-If place after declaration of datatype, will set val to contents of previous entry in table +1, starting count at 1. Useful for keys.

ADDITIONAL FIELD CONDITIONS
-*NOT NULL* – if specify after data type will not allow to be null
-*KEY* – specify as key for table

Altering Tables
-*ALTER TABLE tablename ADD colname KEY NOT NULL (etc.);*

-*ALTER TABLE tablename DROP colname;*

Adding Table Data
-*INSERT INTO tablename(colNameA, colNameB, …)*
    VALUES(‘value a”, “value b”, “1234”, …);

-One INSERT per row. Col names must be listed in order same as table or else data will insert into wrong columns. If skip colum in list, field for that row will hold NULL.

More Table Alteration

-*ALTER TABLE tablename RENAME newname;* - rename table

-*ALTER TABLE tablename ADD DATATYPEX …;* - add column

-*ALTER TABLE tablename MODIFY colname NEWTYPEX;* - change type of col

-*ALTER TABLE tablename CHANGE oldcolname newcolname DATATYPEX …;*

-*ALTER TABLE tablename DROP colname;* - use cautiously. Will delete all data for dropped col.

-*DROP TABLE tablename* - " "

Indexes
-Standard query searches through all rows. Very slow after table > couple hundred rows. Indexes speed up searches.

-Creates a subset of data that is searched through to increase speed at slight overhead cost.

Creating and Index
-Indexes created on different columns or multiple columns. Should be created on columns or column groups that will be queried.

-*ALTER TABLE tablename ADD INDEX(colname(optional-n);* – n is num of first n chars in col to search for variable length cols *(VARCHAR,* etc). Increases query speed.

-After add index to table, col will show up as *MUL* under the *key* in *DESCRIBE*

-*CREATE INDEX indexname ON tablename (colname(optional-n));* - alternate to *ALTER TABLE…* call, except cannot be used to create a pkey.

-Can create indexes during table creation (much faster than creating in table full of data) with *INDEX(colname(optional-n)),* statements.

```
        -ex.    CREATE TABLE classics (
                        author VARCHAR(128),
                        //…
                        INDEX(author(20));
```

-If creating index for groups of columns: *INDEX(colA(n), colB, …)*

Primary Keys
-Must be unique and non-null

-When creating table, specify col by having *PRIMARY KEY(colname)* at end of comma separate list of col, index, etc. definitions.

-To change existing col to pkey:

ALTER TABLE tablename ADD PRIMARY KEY(colname);

-Must change modify col to NOT NULL if not already in order to be pkey via *MODIFY*

-To drop pkey: *ALTER TABLE tablename DROP PRIMARY KEY;*

***FULLTEXT* Indexes**
-MySQL holds a dictionary of most words (minus 500 very common words like "a", "the", etc.) that is can use for super fast searches using *FULLTEXT* indexes

-Create same was as *INDEX*, except FULLTEXT(colA). Can create across multi cols (seperated by commas)

-Can only create on *CHAR*, *VARCHAR*, and *TEXT* cols
-For large data sets, much faster to load data into tables, then create FULLTEXT indexes after

SELECT
-Used to query a DB

-*SELECT something FROM tablename;* - where something is list of cols, COUNT, etc.
-To select all cols, use * for *something*

SELECT COUNT
-*SELECT COUNT(something)...*- Returns count of all qaulifying rows. Cannot use with other cols in *SELECT*, unless also using with *GROUP BY*

SELECT DISTINCT
-*SELECT DISTINCT colname…* - If use with multiple col-names, will search for distinct pairs of cols (ex. *charles, 32* would only appear once but *charles* could appear multi)

DELETE
-*DELETE FROM tablename WHERE colname='value'…;*

WHERE
-Narrows down query by enabling limiting qualification(s) for query

-Can search for similiar/substrings in fields with *LIKE* clause, where *%* in value acts as "any chars," including empty string
    -ex. *SELECT author FROM classics WHERE title LIKE "%and%";*

LIMIT
-Tells to limit num of records returned.

-Attach at end of query *LIMIT 3* (return 3 records only) or *LIMIT 10, 20 (*return 20 rows starting at row 10). First row in table is 0, not 1.

-For two args does not search from row 10 to 20 only, etc., but rather returns up to 20 results, starting at row 10.

MATCH...AGAINST
-Can use only on columns (or groups of columns) that have been given a *FULLTEXT* index.

-*SELECT col(s) FROM table WHERE MATCH(col(s)) AGAINST('value');*

-Searches each col specified in *MATCH* list and if any of columns hold *'value'* (even as substring, etc.), returns those records

-Unlike other search functions, MATCH lets the you search multiple words (separated by spaces inside of the ' ' for *MATCH*) and searches case insensitive

-ex. *AGAINST('Charles Dickens')* will return *Charles Darwin* and *Mary Dickens* rows

***MATCH...AGAINST* boolean**

-To search in boolean mode use AGAINST('search terms' IN BOOLEAN MODE).

-Lets you puts more restraints on search list. Preface search term with + to make term mandatory or – for only return if term not present.

-ex. *AGAINST('+charles -dickens' IN BOOLEAN MODE)*

//returns all records that contain *charles,* but not if they also contain *dickens*

-Can also search for a group of words, via above syntax, except including search terms list in " " (inside of ' ')

-ex. *AGAINST('"Charles Dickens"' IN BOOLEAN MODE)*

//returns only if contains phrases "Charles Dickens"

Update...Set

-Used to update records (ie, change existing values)

-UPDATE tablename SET col='val' WHERE…;

ORDER BY

-Sorts results in ascending or descending order for specified column(s). If multi columns, sorts by first listed col first, then by second col within initial sort

-Append onto end of query: *ORDER BY* col(s) //ascending by default

-If wish to sort descending: *ORDER BY col(s) DESC*

GROUP BY

-Append *GROUP BY col(s)* onto end of query

-Records grouped together by specified columns.

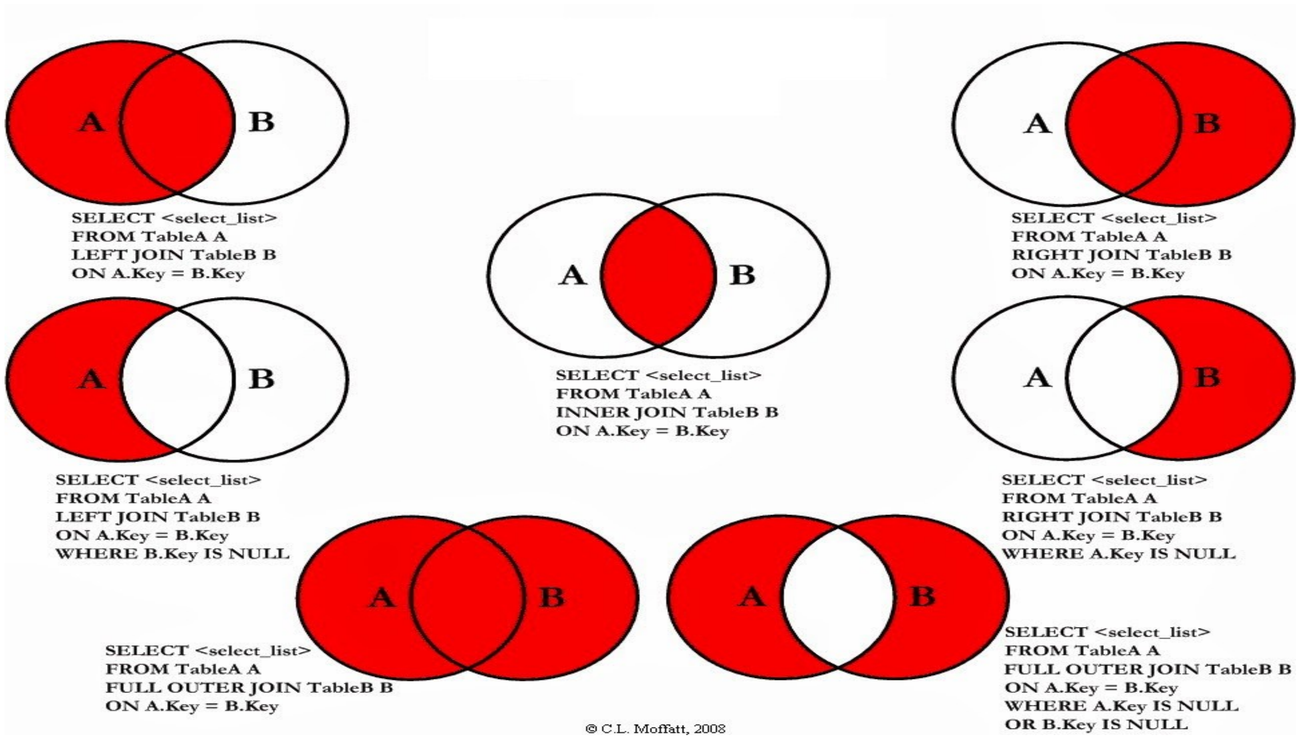
-Often used with *COUNT(something)* to tell the number of unique somethings per *GROUP BY* specifier(s) or with a single col selection to tell the number of unique values in the *GROUP BY* specifier(s)

Joining Tables

-Natural Join – *SELECT...FROM tableA NATURAL JOIN TableB* – Returns results from both tables. Column names must match on tables.

-*JOIN...ON – JOIN tableA ON tableB.col = tableA.col* – joins col on specific col. Can use without specifying a specific type of join (ex left join) and simply put after *SELECT* statement

-Types of joins:



-*AS* – Creates alias for col or table, which can then use as ref name in future. Specify *colname AS shortname* during first reference to column (in *SELECT*) or table (in *FROM)*

-Logical operators – *AND, OR, NOT*. Can group together in parentheses for order of opp.

-MySQL Functions – Variety of SQL functions exist that can prevent needless backend lang work post selection

-*AVG(col), COUNT(col), SUM(colA +/*- colB), MAX(col), MIN(col), CONCAT(colA, colB), CURDATE( ), DATEDIFF(colA, colB), NOW( ), TIMEDIFF(colA, colB), CEIL(col), ABS(col), FLOOR(col), ROUND(col), TRUNCATE(col, numDecimals)*

-Transactions – Prevent statements from being executed in improper order. To execute transaction: *BEGIN; [statement-here]; COMMIT;* No changes made to DB until commited.

-*ROLLBACK;* - Reverts previous transaction

-*EXPLAIN [statement]* – returns details on query, such as private key of table, key length, num rows, etc.

Mastering MySQL (DB, Relationships, Normalization, etc.)

-Primary Key – Column that is unique for each entry in DB

-Normalization – the process of separating table into tables, using primary uses, to ensure proper relations, and avoidance of repeating data needlessly across multiple tables, which typically results in inconsistencies when data is changed

First Normal Form
1) No repeating cols containing same purpose data (ex. *author1, author2*)
2) All cols contain a single value
3) Must be PKEY to uniquely ID each row

Second Normal Form
-Focused largely on duplicate data across tables

1) Must be in 1NF
2) Full functional dependencies on PKEY only. No partial dependencies (where non-key attribute is dependent on another non-key attribute).

-Ex. Have table *customers and purchases* that stores *customer_name, customer_address, purchase_date*, and *ISBN* (pkey). Table lacks clear purpose, and thus has partial dependencies. To move to 2NF, create tables *books, customers, purchases,* where *purchases* has an auto-incrementing PKEY *purchaseID*, in addition to *ISBN*, and *customer_id*, which are PKEYs in *customer* and *books* tables, where more details (ex. *title, phone num*) would be queried via joins, etc..

Third Normal Form
1) Must be in 1NF, 2NF
2) All data must be dependent on PKEY. If not, move to new table.

ex. *customer* table has *address, zip* fields, which could be said to be not dependent on customer. To move to 3NF, create *addresses* table with and ref via joins, etc..

-2NF is often very sufficient for normalization and 3NF can lead to a glut of tables and relationships, so normalize to 3NF wisely.

When to Normalize Less
-Many users with many sequential access results in many DB calls, which can slow down server. If this is the case, consider less normalization, with more data duplication, to increase query times.

-If doing this, make sure all redundant data is properly updated when changes made to it.

Relationships

<u>One-to-One</u>
-each X only has on Y, and each Y only has one X
    -Ex. each author only has one set of fingerprints, each set of prints has only one author

-Typically in same table, though can separate if table getting too large or in case it might stop being one-to-one

<u>One-to-Many</u>
-each X has many Y, but each Y only has one X

-ex. each customer has many order IDs, but order ID has only one customer

-Two tables: one for “many,” one for “one”

<u>Many-to-Many</u>
-each X has multiple Y, and each Y has multiple addresses
-ex. each city has multiple companies, and each company has multiple cities

-Three tables: one for each many, then in between with a combined Pkey of nothing but combo of two PKEYs from “many” tables and nothing else (ex. contains *customerID*, *ISBN*), then get more info from “many” tables via joins, etc.

Transactions
-MySQL auto-queues queries, in order, executed sequentially, but still allowing concurrent access from multi users. Enabled by InnoDB Engine, which is standard and default since 2010 in v5.5

-Transactions start after *BEGIN;* statement, and are committed via *COMMIT;*, after which transactions are done and new *BEGIN;* is needed for more.

-ex.

```
BEGIN;
UPDATE accounts SET balance=balance+25.11 WHERE number=12345;
UPDATE accounts SET balance=balance * 2 WHERE number=12345;
COMMIT;
```

-Can undo transactions (since last *commit*) by calling *ROLLBACK;*

EXPLAIN
-If run *EXPLAIN some-query-here;* will get details include *SELECT* type, tables called, etc.
-Useful for DB optimization.

Mysqldump & DB Backups
-Can be used to backup tables, export CSVs of them, etc.. Typical use outputs a series of queries which will recreate the DB if run.

-DB Syntax: *mysqldump -u user -p password database*
-Prints by default. Useful to concat with >

-Downside: tables must not be written to during. Easiest way == shut down mysql server during. Alternatively run: *LOCK TABLES tablename1 READ, tablename2 READ, …;* Then, after mysqldump, run *UNLOCK TABLES;*

-Must run command from dir it is contained in, often in *bin*

-Example of use:
1) Create bash script to run daily for DB backup, calling:

```
mysqldump -u lainon –p pass123 publications > publications.sql
```

2) New hire accidentally wipes all tables
2) Run *publications.sql* to delete existing empty tables and populate to backup state

Individual Table Backups

LOCK TABLES publications.classics READ;
mysqldump -u lainon –p pass123 publications classics > classics.sql
UNLOCK TABLES;

All DB/Table Backups

LOCK TABLES all-tables-needing-lock READ;
mysqldump -u lainon –p pass123 --all-databases > all_databases.sql;
UNLOCK TABLES;

Backup Restoration
-All DB: *mysql -u user -p password < all-db-backup.sql*
-Single DB: *mysql -u user -p password -D publications < some-DB-backup.sql*
-Single table DB: *mysql -u user -p password -D publications < some-table-backup.sql*

CSV Table Export
-Syntax: *mysqldump -u user -p password --no-create-info --tab='output-path-here' --fields-terminated-by=',' tablename*

Backup Timeline
-If DB updated daily, should be backed up daily. If updated many times daily, should be backed up multi times daily.