Database Admin

Class 6 - Optimization

By Ian Robert Blair, M.Sc.

Agenda

- Discuss Presentation (Week 15)
- Optimization
- Review for Final Exam (Week 17)

Presentation

- Make a Presentation of your application
- Discuss the following:
    - Description and features
    - Data Model (ERD)
    - Database Schema
    - Page map
    - Show the user interface and functionality
    - Discuss Issues and Problems
    - Discuss features you would like to add if you had more time
- Conclude with your thoughts and impressions on the project

Tuning and Optimization

Optimizing Datatypes, pt. 1

- Smaller is usually better - uses less disk space, memory, and require fewer CPU cycles to process
- Simple is good - use built in types and integers are better than strings
- Avoid NULL if possible - they make indexes, index statistics, and value comparisons more complicated
- Use tinyint(8 bit), smallint(16), mediumint(24), int(32) and bigint(64) appropriately
- VARCHAR string type saves space, using only whats required to store the string and 1 or 2 bytes for the length
- It’s usually worth using VARCHAR when the maximum column length is much larger than the average length

Optimizing Datatypes, pt. 2

- CHAR is useful if you want to store very short strings, or if all the values are nearly the same length
- For example, CHAR is a good choice for MD5 or other hashed values for user passwords, which are always the same length
- Use an ENUM column instead of conventional string types, MySQL stores them packed into one or two bytes
- Datetime uses 8 bytes, and stores the time into an sortable integer in YYYYMMDDHHMMSS format
- Its better to use TIMESTAMP uses unix, time since jan. 1 1970, and uses 4 bytes of memory
- Integers are fast and work with auto increment so they are ideal for identifiers (keys)

Normalized vs. De-normalized Data

- Normalized data benefits:
  - Updates are faster
  - Less duplication, less in memory, less on storage, and less to change
- De-normalized data benefits:
  - No or fewer joins, so query performance is better
  - Indexing can increase performance greatly

Alter Table Performance

- MySQL performs most alterations by making an empty table with the desired new structure, inserting all the data from the old table into the new one, and deleting the old table

- This can take a long time on big tables with a lot of indexes

- Use the **ALTER COLUMN** command instead of the **MODIFY COLUMN** command, because, depending on the change, MySQL will skip creating a temporary table and change the tables .frm file directly

B-Tree Index, pt. 1

- A B-Tree index speeds up data access because the storage engine doesn't have to scan the whole table to find the desired data
- Starts at the root node
- The slots in the root node hold pointers to child nodes, and the storage engine follows these pointers
- It finds the right pointer by looking at the values in the node pages, which define the upper and lower bounds of the values in the child nodes
- Eventually, the storage engine either determines that the desired value doesn't exist or successfully reaches a leaf page
- Because B-Trees store the indexed columns in order, they're useful for searching for ranges of data

B-Tree Index, pt. 2

- B-Tree Indexes are good for:
    - Matching the full value
    - Matching a leftmost prefix
    - Matching a column prefix
    - Matching a range of values
    - Matching one part exactly and matching a range on the other part
    - Index-only queries

B-Tree Indexes, pt. 3

- Are not useful if they don’t match the leftmost column in the index, skip a column, and must access a range of values to the right of the index

Clustered Index

- InnoDB's clustered indexes actually store a B-Tree index and the rows together in the same structure
- Only one clustered index per table, because you can't store the rows in two places at once
- InnoDB clusters the data by the primary key
- Data access is fast, because both the index and the data together in one B-Tree
- Updating the clustered index columns is expensive, because it forces InnoDB to move each updated row to a new location

Covering Index

- An index that contains (or “covers”) all the data needed to satisfy a query is called a covering index
- Covering indexes can be a very powerful tool and can dramatically improve performance

Index Best Practices

- Indexes should not be part of an expression or be inside a function in the query
- You can often save space and get good performance by indexing the first few characters instead of the whole value (prefix index)

  ALTER TABLE sakila.city_demo ADD KEY (city(7));
- Place the most selective columns first in the index

Hardware Performance

- More Memory
  - MySQL allocates buffers and caches to improve performance of database operations
  - The InnoDB buffer pool is a memory area that holds cached InnoDB data for tables, indexes, and other auxiliary buffers (50-75% of Memory)
- SSD
  - Good hard drives will do 200 I/O's per second
  - SSD can do 20,000 I/O's per second

Partitioning Tables

- Partitioning may reduce the size of indexes, effectively reducing the table into many smaller tables
- This technique involves splitting a table into several physical files or tablespaces
- Foreign keys are not allowed for partitioned tables

```
CREATE TABLE article (
  id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  date DATE NOT NULL,
  author VARCHAR(100),
  language TINYINT UNSIGNED,
  text TEXT,
  PRIMARY KEY (id, date)
)
  ENGINE = InnoDB
PARTITION BY RANGE (YEAR(date)) (
  PARTITION p0 VALUES LESS THAN (1990),
  PARTITION p1 VALUES LESS THAN (2000),
  PARTITION p3 VALUES LESS THAN (2010),
  PARTITION current VALUES LESS THAN (2020)
);
```

Checking Performance

- **Benchmarking** is the execution of a standard suite of programs in order to assess the performance of the system or systems being tested in a methodical manner relative to performance of other systems
- **Profiling** is a methodical way of collecting metrics about systems you are testing

Benchmark Function

- There is a BENCHMARK function in mysqld that allows you to see the execution time of an expression
- BENCHMARK() is for testing scalar expressions
- The BENCHMARK function takes two arguments — a number for how many times to evaluate the expression, and an expression
- For example:
  - SELECT BENCHMARK(10000000, 1+1);
  - SELECT BENCHMARK(1000000, 1/1);
- We can compare how long different functions that return the same result take:
  - SELECT BENCHMARK(10000000, LEFT('2009-05-01 00:00:00',10));
  - SELECT BENCHMARK(10000000, DATE('2009-05-01 00:00:00'));

mysqlslap

- The **mysqlslap** program will emulate client load on mysqld
- It can executes an arbitrary number of SQL statements contained in a text file
- It is included with MySQL Server running on Unix-based systems
- Some of the options:
  - —auto-generate-sql, —auto-generate-sql-add-auto-increment, —auto-generate-sql-write-number=num— auto-generate-sql-unique-write-number=num
  - —csv=file_name (output file), —iterations=num, —concurrency
  - —query (query or query file)
- For example:
  - mysqlslap --user=root --password --concurrency=5 --auto-generate-sql --auto-generate-sql-execute-number=1000 --auto-generate-sql-unique-query-number=1000

Mysqlslap Example

- For example, the following query run against a table of 300,000 rows:

  sudo mysqlslap --user=sysadmin --password --host=localhost --concurrency=50 --iterations=10 --create-schema=employees --query="SELECT * FROM dept_emp;" —verbose

- Returns the following output:

Benchmark

Average number of seconds to run all queries: 18.486 seconds

Minimum number of seconds to run all queries: 15.590 seconds

Maximum number of seconds to run all queries: 28.381 seconds

Number of clients running queries: 50

Average number of queries per client: 1

Sysbench Intro

- The SysBench program was written to provide a more general system-level view of a server
- Has tests for CPU performance, I/O performance, mutex contention, memory speed, thread performance, and database performance
- If you make some changes to the MySQL server settings, Sysbench is a good tool to test the effectiveness of those changes
- It can be installed from a repository (ie. yum install sysbench) or from source
- There are six possible test modes: CPU, I/O File, Mutex, Memory, thread, OLTP

Sysbench File I/O Example

- First run:

  cd /tmp && sysbench --num-threads=16 --test=fileio --file-total-size=1G --file-test-mode=rndrw prepare

- Then:

  sysbench --num-threads=16 --test=fileio --file-total-size=1G --file-test-mode=rndrw run

```
Operations performed:  6006 Read, 3994 Write, 12800 Other = 22800 TotalRead 93.844Mb
Written 62.406Mb  Total transferred 156.25Mb  (1.2801Mb/sec)   81.92 Requests/sec
executedTest execution summary:    total time:                          122.0649s    total
number of events:              10000    total time taken by event execution: 300.1585
per-request statistics:         min:                                  0.01ms
avg:                                 30.02ms         max:
1491.05ms         approx.  95 percentile:             191.59msThreads fairness:    events
(avg/stddev):           625.0000/49.50    execution time (avg/stddev):   18.7599/1.56
```

Sysbench OLTP Example

- Example OLTP test
- First run, to prepare the sample table:

  sysbench --test=oltp --oltp-table-size=1000000 --mysql-db=dbtest --mysql-user=[USER] --mysql-password=[PASSWORD] prepare

- Then run, to test:

sysbench --test=oltp --oltp-table-size=1000000 --oltp-test-mode=complex --oltp-read-only=off --num-threads=6 --max-time=60 --max-requests=0 --mysql-db=dbtest --mysql-user=[USER] --mysql-password=[PASSWORD] run

Sysbench OLTP Output

```
sysbench 0.4.12:  multi-threaded system evaluation benchmark

No DB drivers specified, using mysql
Running the test with following options:
Number of threads: 6

Doing OLTP test.
Running mixed OLTP test
Using Special distribution (12 iterations,  1 pct of values are
returned in 75 pct cases)
Using "BEGIN" for starting transactions
Using auto_inc on the id column
Threads started!
Time limit exceeded, exiting...
(last message repeated 5 times)
Done.
```

Sysbench OLTP Output, pt. 2

```
OLTP test statistics:
    queries performed:
        read:                            456680
        write:                           163100
        other:                           65240
        total:                           685020
    transactions:                        32620  (543.63 per sec.)
    deadlocks:                           0      (0.00 per sec.)
    read/write requests:                 619780 (10329.05 per sec.)
    other operations:                    65240  (1087.27 per sec.)

Test execution summary:
    total time:                          60.0036s
    total number of events:              32620
    total time taken by event execution: 359.8823
    per-request statistics:
         min:                                  1.66ms
         avg:                                 11.03ms
         max:                                981.94ms
         approx.  95 percentile:              15.13ms

Threads fairness:
    events (avg/stddev):           5436.6667/31.44
    execution time (avg/stddev):   59.9804/0.00
```

Explain Example: Without Index

```
mysql> EXPLAIN SELECT title FROM employees WHERE lastname LIKE
'T%' \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: employees
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1420
        Extra: Using where
1 row in set (0.00 sec)
```

Explain Example: With Index

```
mysql> EXPLAIN SELECT title FROM employees WHERE lastname LIKE
'T%' \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: employees
         type: range
possible_keys: index_name
          key: index_name
      key_len: 22
          ref: NULL
         rows: 70
        Extra: Using where; Using index
1 row in set (0.00 sec)
```

Profiling, pt. 1

- Profiling gathers information that can be later compared to determine if there were any changes in behavior
- Extra fields
  - ALL, BLOC IO, CONTEXT SWITCHES, CPU, IPC, MEMORY, PAGE FAULTS, SOURCE, SWAPS
- Example:

```
SET profiling=1;
SELECT COUNT(*) FROM sakila.film;
SHOW PROFILE;
SHOW PROFILE CPU, SOURCE FOR QUERY 1 LIMIT 2 OFFSET 0\G
```

```
MariaDB [(none)]> show profile cpu;
+----------------------+----------+----------+------------+
| Status               | Duration | CPU_user | CPU_system |
+----------------------+----------+----------+------------+
| starting             | 0.000083 | 0.000061 | 0.000021   |
| checking permissions | 0.000009 | 0.000006 | 0.000002   |
| Opening tables       | 0.148794 | 0.001415 | 0.002478   |
| After opening tables | 0.000015 | 0.000008 | 0.000006   |
| System lock          | 0.000007 | 0.000005 | 0.000002   |
| Table lock           | 0.000010 | 0.000010 | 0.000001   |
| init                 | 0.027554 | 0.000031 | 0.000233   |
| optimizing           | 0.003397 | 0.000017 | 0.000067   |
| statistics           | 0.000024 | 0.000018 | 0.000004   |
| preparing            | 0.015086 | 0.000048 | 0.000133   |
| executing            | 0.000012 | 0.000006 | 0.000005   |
| Sending data         | 0.043148 | 0.000481 | 0.000544   |
| end                  | 0.000010 | 0.000006 | 0.000003   |
| query end            | 0.000008 | 0.000007 | 0.000001   |
| closing tables       | 0.000005 | 0.000004 | 0.000002   |
| Unlocking tables     | 0.000012 | 0.000019 | 0.000012   |
| freeing items        | 0.000008 | 0.000009 | 0.000002   |
| updating status      | 0.000026 | 0.000019 | 0.000047   |
| cleaning up          | 0.000008 | 0.000010 | 0.000017   |
+----------------------+----------+----------+------------+
```

Explain, pt. 1

- The explain tool allows to see:
    - How many tables are involved, joined, and looked up
    - If there are subqueries or unions
    - If DISTINCT or WHERE clause is used
    - If a temporary table is used
    - Possible, Actual, and Length of actual indexes used
    - Approximate number of records returned
    - If sorting requires an extra pass through the data

Explain, pt. 2

- The first field, id, is a sequential identifier, which is different for each row.
- Each row represents a physical table, subquery, temporary table, or derived table
- The second field, select_type, is the type of SELECT represented by the row
- The third field, table, shows the table alias that the row refers to
- The data access strategy is in the type field of the EXPLAIN plan
- The last field in the EXPLAIN plan is Extra. This is a catch-all field that shows good, neutral, and bad information about a query plan

Explain, pt. 3

Type	Access Stategy
ALL	Full scan of the entire table structure
index	Full scan of the entire index
range	partial scan of the index structure
index_subquery	Subquery using a non unique index
unique_subquery	More than one the indexes are used to perform multiple scans
fulltext	MySQL full text search
ref	More than one record may be looked up for each record joined, or for each set fo results from a pervious records
eq_ref	Fewer than two records are looked up for each record being joined from previous records
Const	Fewer than two rerecords are looked up from a non-system table
System	Fewer than two records are look up from a system table
null	Data is not lookup up using a table

HA Metrics

- These two dimensions of high availability can be measured by two corresponding metrics:
  - Mean time between failures (MTBF)
    - The predicted elapsed time between inherent failures of a system during operation
    - It be calculated as the arithmetic mean (average) time between failures of a system
  - Mean time to recovery (MTTR)
    - The average time that a device will take to recover from any failure

Improving MTBF, pt. 1

- Steps to improving MTBF:
  - Test your recovery tools and procedures, including restores from backups
  - Follow the principle of least privilege
  - Keep your systems clean and neat
  - Use good naming and organization conventions to avoid confusion, such as whether servers are for development or production use
  - Upgrade your database server on a prudent schedule to keep it current
  - Test carefully before upgrading
  - Use InnoDB, configure it properly, and ensure that it is set as the default storage engine and the server cannot start if it is disabled
  - Make sure the basic server settings are configured properly
  - Disable DNS with skip_name_resolve

Improving MTBF, pt. 2

- Steps to improving MTBF:
  - Disable the query cache unless it has proven beneficial
  - Avoid complexity, such as replication filters and triggers, unless absolutely needed
  - Monitor disk space and RAID volume status
  - Record as many historical metrics as possible about server status and performance, and keep them forever if you can
  - Test replication integrity on a regular basis
  - Make replicas read-only, and don't let replication start automatically
  - Perform regular query reviews
  - Archive and purge unneeded data
  - Reserve some space in filesystems

Improving MTTR

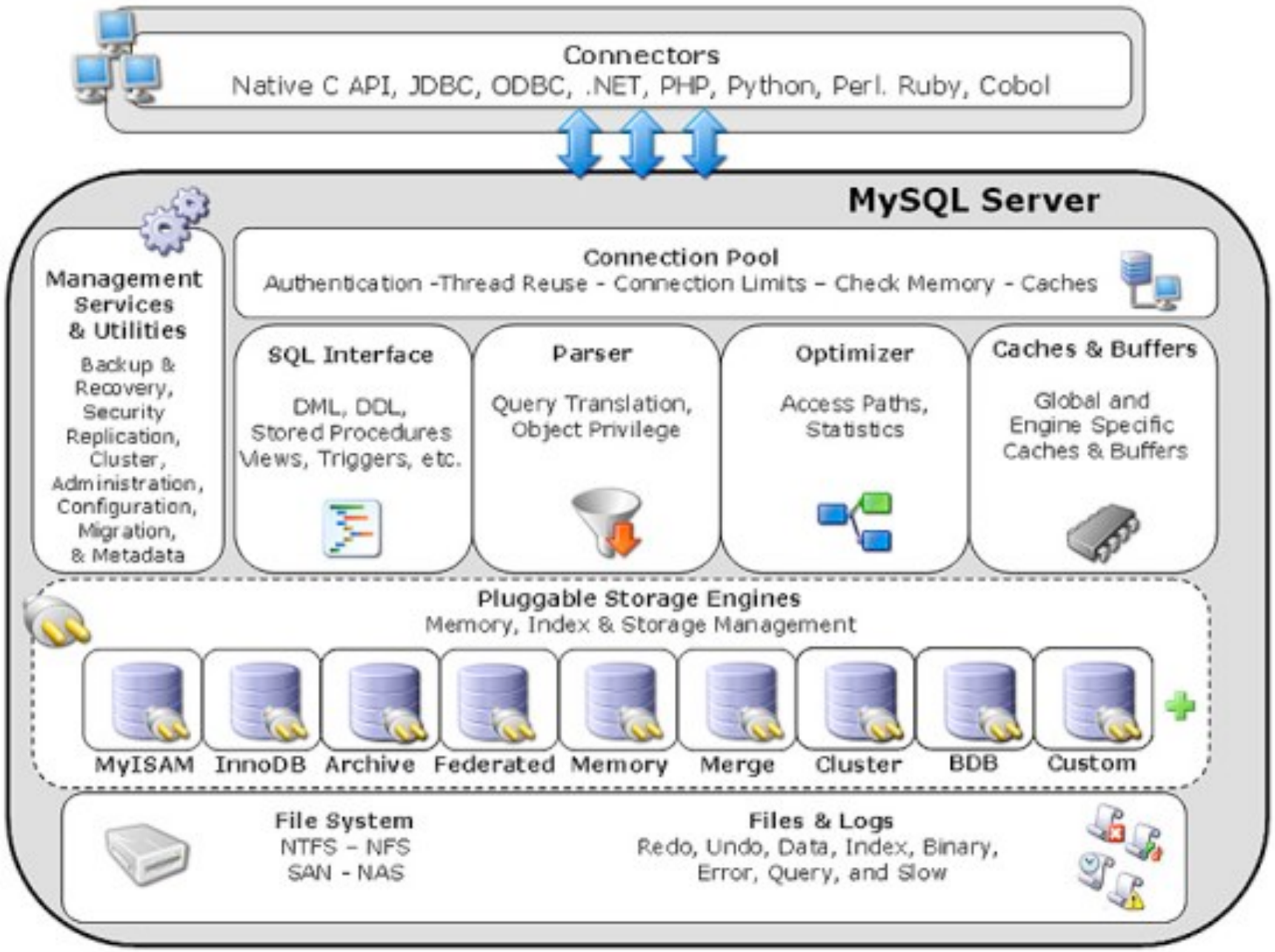
- Finding and eliminating single points of failure in your system is one way of improving availability by reducing recovery time (MTTR)
- Having redundancy in the following sub-systems will improve recovery time:
    - Storage (ie. Raid systems)
    - Power supplies and UPS
    - Network connections (including internet connections)
    - Network services (DNS, LDAP, etc.)
    - Clusters or Replicas
    - Disaster Recovery Site

HW

- Read Chapter 5, 6, and Appendix D in “High Performance MySQL”
- Prepare your presentations

Exam Review

MySQL Architecture



RAID

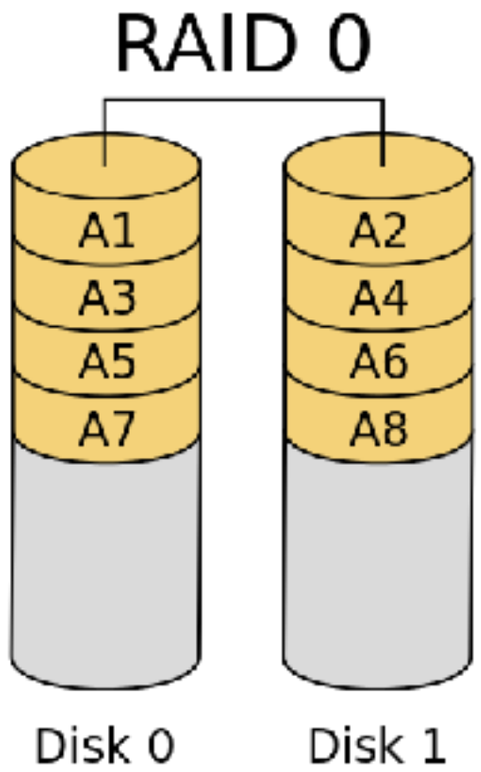
- Statistics about server component failures indicate that 50% of server down-time can be attributed to disk drive failures
- RAID systems can simultaneously protect data and provide immediate online access to it, even when disks fail

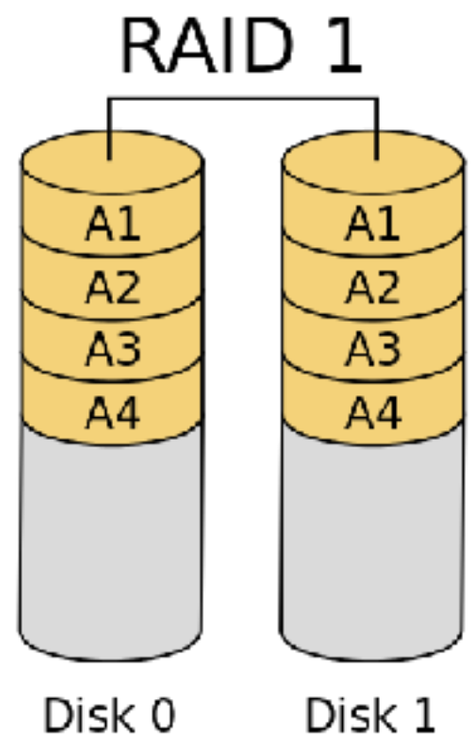
RAID, pt. 1

- **RAID level 0** (striping)—Improves performance but offers no redundancy

- **RAID level 1** (mirroring)—Provides simple redundancy, improving data reliability

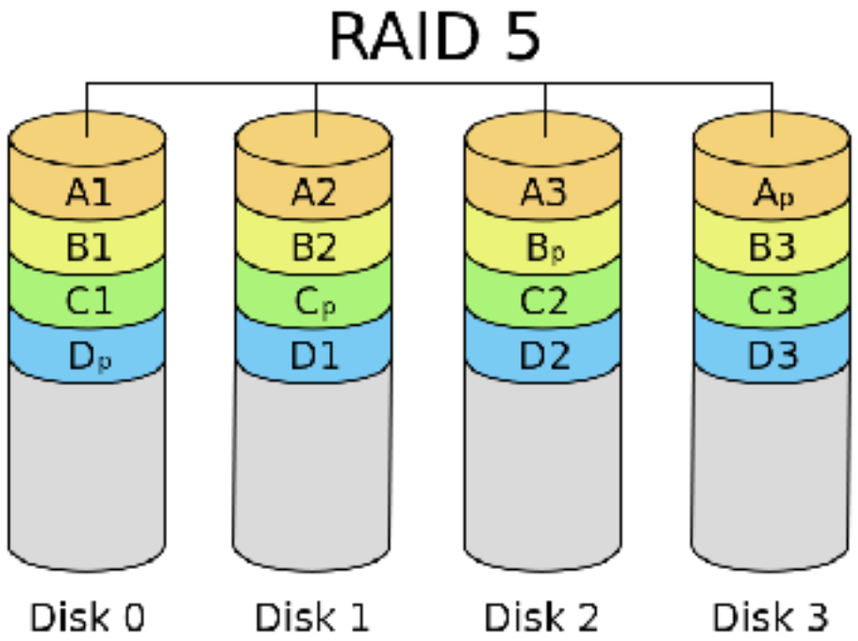
- **RAID level 10** (mirroring and striping)—A combination of RAID 1 and RAID 0 (also called RAID 1+0)



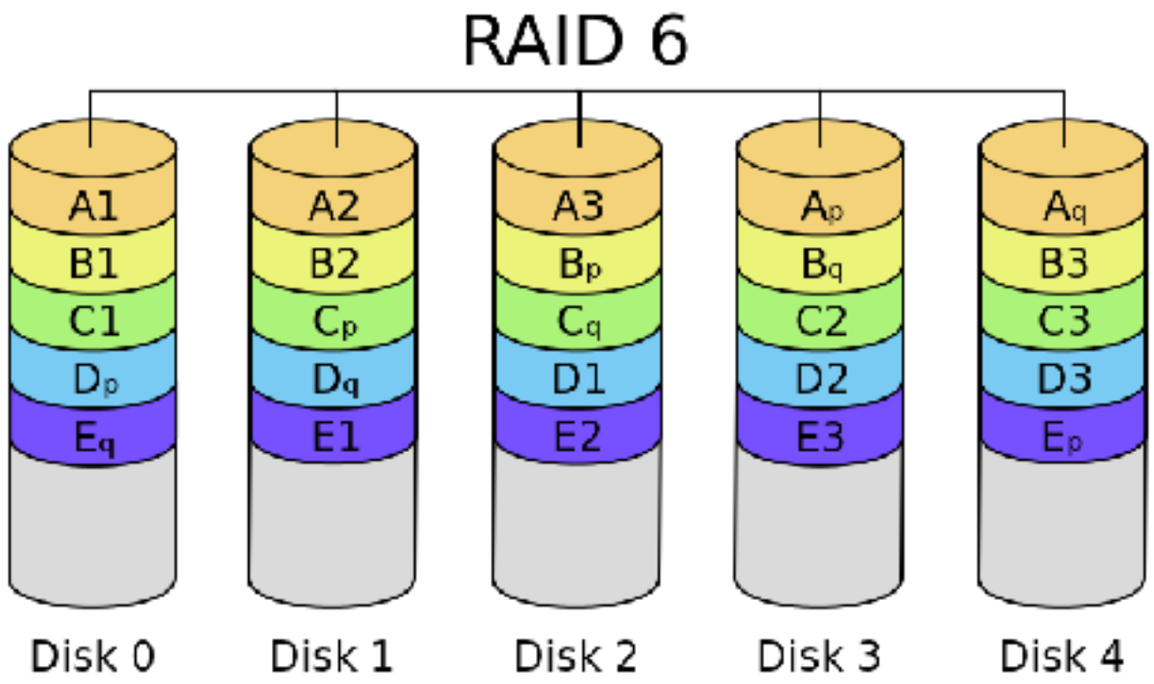


RAID, pt 2

- **RAID level 5** (disk striping with parity)—Provides redundancy and improves performance (most notably, read performance)



- **RAID level 6** (disk striping with double parity)—Improves upon level 5 RAID by protecting data when two disks fail at once



Scale Up/Out

- Two common methods are used to upgrade servers
- They are referred to as either scale up or scale out:
  - In a **scale-up** operation, components are added to the server computer, making it more powerful (CPU, Memory, Storage, etc.)
  - In **scale-out** operations, additional servers are installed, load-balancing techniques are implemented, and increased out-of-chassis redundancy is also employed
- Three primary factors that server system administrators take into account when deciding between scale-up or scale-out scenarios are capacity, reliability, and cost

Terms

- Datacenter
- Storage Area Network
- Virtualization
- DAAS

MySQL Client

- The most common way to access mysqld is through the command-line client tool simply called mysql
- u user_name or --user=user_name
- -p pass or --password=pass
- -h mysqld_host or --host=mysqld_host
- --protocol=protocol_type (most likely TCP)
- --compress
- --version or -V (Displays version info)

MySQL Client 2

- One common use of mysql is to execute a non-interactive batch file of SQL queries.
- mysql < sakila-schema.sql
- mysql < sakila-data.sql
- f or --force
- Another way to run a batch file of SQL queries is to use the **source** command (source or \.) within the mysql interactive shell

MySQL Admin

- The mysqladmin program is used to perform administrative tasks

Options	Description
create	create database
debug	sends debugging information to the error log
drop	drop database
flush-hosts, logs, privileges, tables, threads, and status	flushes cached data and writes to disk
kill	Kills client threads
password	Changes password
ping	Pings instance
start-slave, stop-slave	Starts and stops replication
processlist	Displays active threads
variables	Shows global variables and values

Loading Data from CSV

- On occasion data from comma delimited format (.csv) files may need to be imported or exported
- CSV is supported by most spread sheet applications like MS Excel, Numbers, or Open Calc
- LOAD DATA INFILE 'filename.csv' INTO TABLE details FIELDS TERMINATED BY ',';
- SELECT artist_name, album_name FROM artist, album WHERE artist.artist_id=album.artist_id INTO OUTFILE '/tmp/outputfile.csv' FIELDS TERMINATED BY ',';

Options File

- Contains the configuration options for the mysql server, client, and other mysql programs
- The configuration for the server is in the [mysqld] section, client [mysql], etc…
- The system option file is /etc/my.cnf (Linux/Mac), <Windows_Directory>\my.ini (Win)
- The user option file is .my.cnf in your home directory (~/.my.cnf)
- To secure this file run chmod u=rw,g=,o= ~/.my.cnf
- So for example, to configure client to auto login for a user account
- [mysql]
  user=root password=the_mysql_root_password

MySQL Show Commands

- SHOW DATABASES;
- SHOW CREATE DATABASE *database_name*;
- SHOW COLUMNS FROM .... (table);

MyISAM

- Non-transactional
- Global table-level locking
- Writes may end up blocking the reads
- There are three files on disk that represent a MyISAM table
    - Table format file(.frm), Data file(.myd), and index file(.myi)
- They can be copied without any problem from the server to a backup location for a raw backup or even directly to another server for use on a new server
- Very fast read activity, suitable for data warehouses
- Compressed data (with myisampack)

InnoDB

- The most widely used transactional storage engine is the InnoDB storage engine
- InnoDB brought support for foreign keys to mysqld
  - Transactional support provided by MVCC (Multi Version Concurrency Control)
  - Row-level locking
  - Foreign key support
  - Indexing using clustered B-tree indexes
  - Configurable buffer caching of both indexes and data
  - Online non-blocking backup through separate commercial backup program

Entity Relationship Model (ERD)

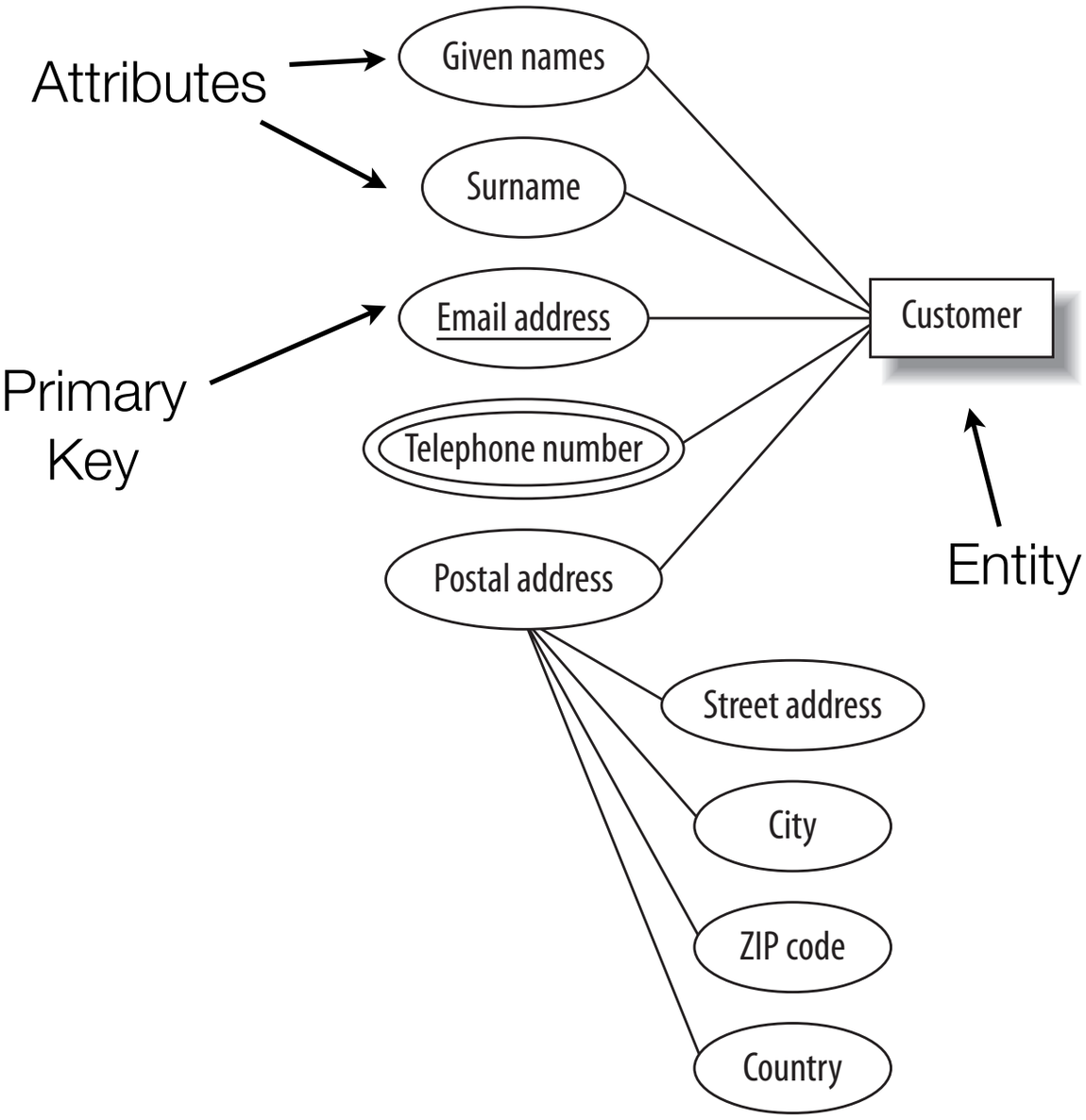
- To help visualize the design, the Entity Relationship Modeling approach involves drawing an Entity Relationship (ER) diagram

- The entity is a person, object, place or event for which data is collected

- A attribute is a characteristic of the entity (can be multivalued)

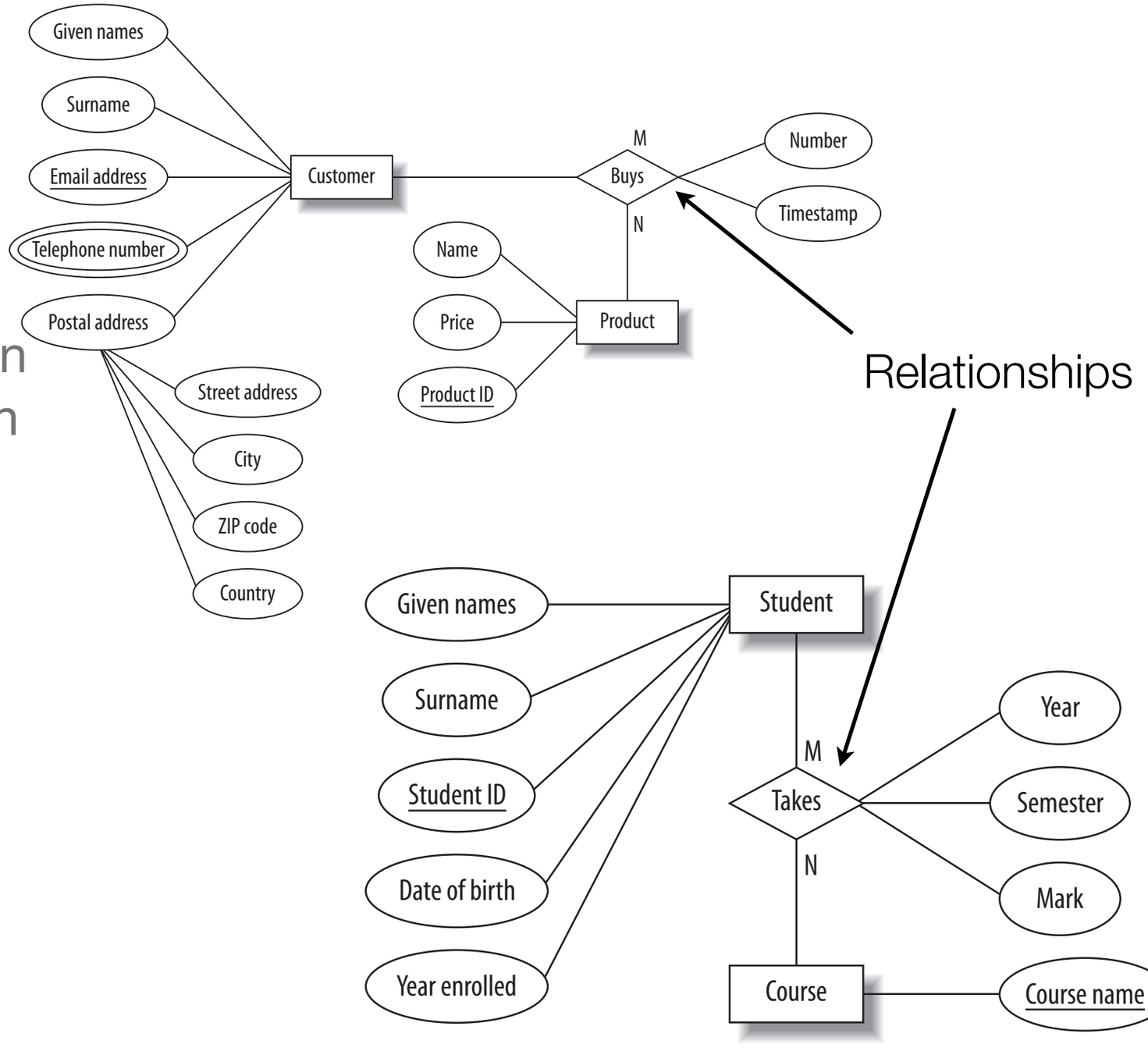
- A primary key uniquely identifies an entity

- Try to keep the design simple



Relationships

- Entities can participate in relationships with other entities
- For example, a customer can buy a product, a student can take a course, an artist can record an album, and so on
- We often use the shorthand terms 1:1, 1:N, and M:N for one-to-one, one-to-many, and many-to-many relationships, respectively

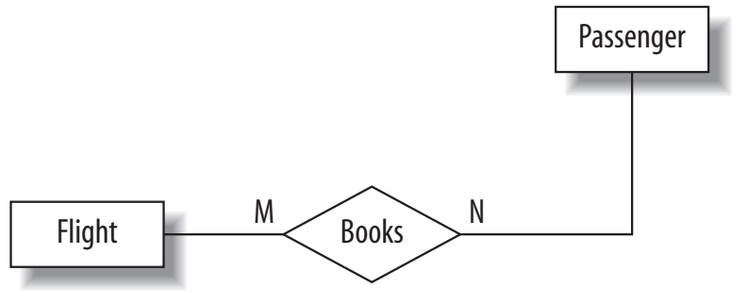


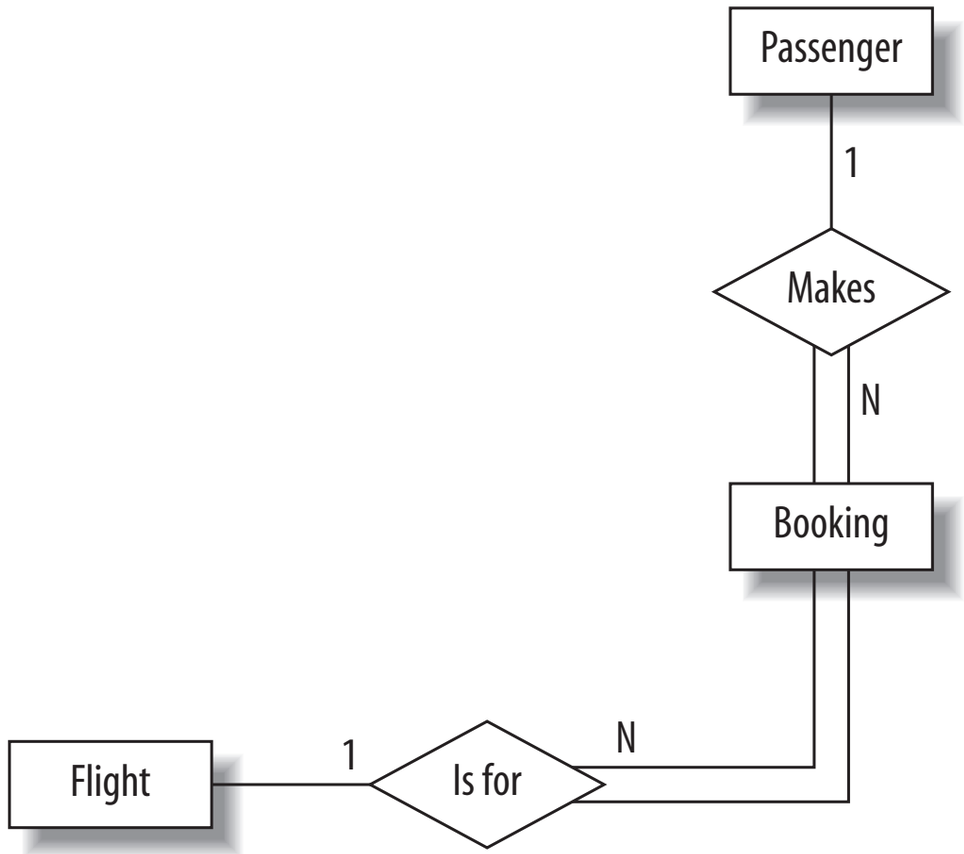
Intermediate Entities

- It is often possible to simplify many-to-many relationships by replacing the many-to-many relationship with a new intermediate entity and connecting the original entities through a many-to-one and a one- to-many relationship

- For example: “Passengers can book a seat on a flight.”

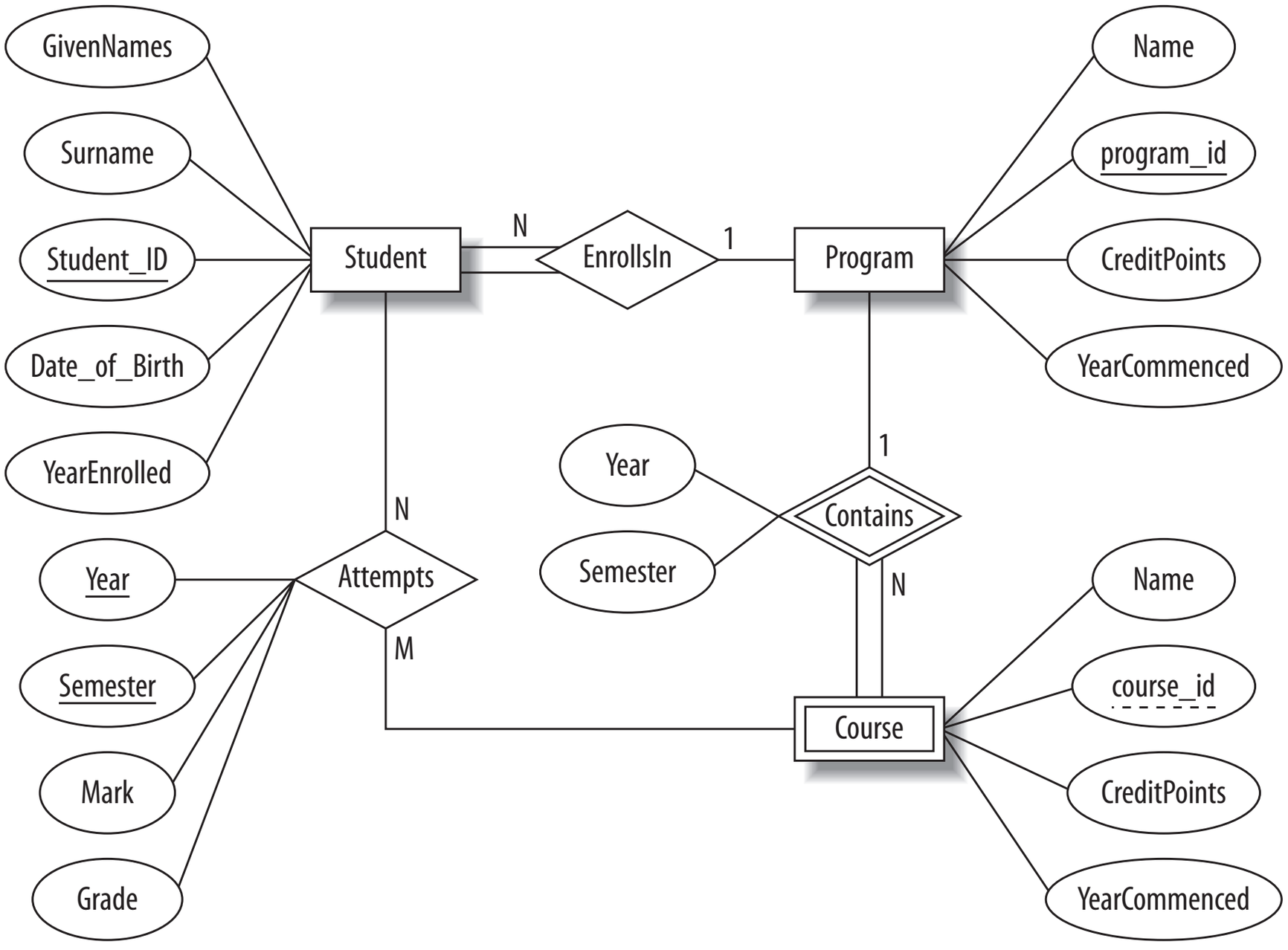
- Better: “A passenger can make a booking for a seat on a flight.”



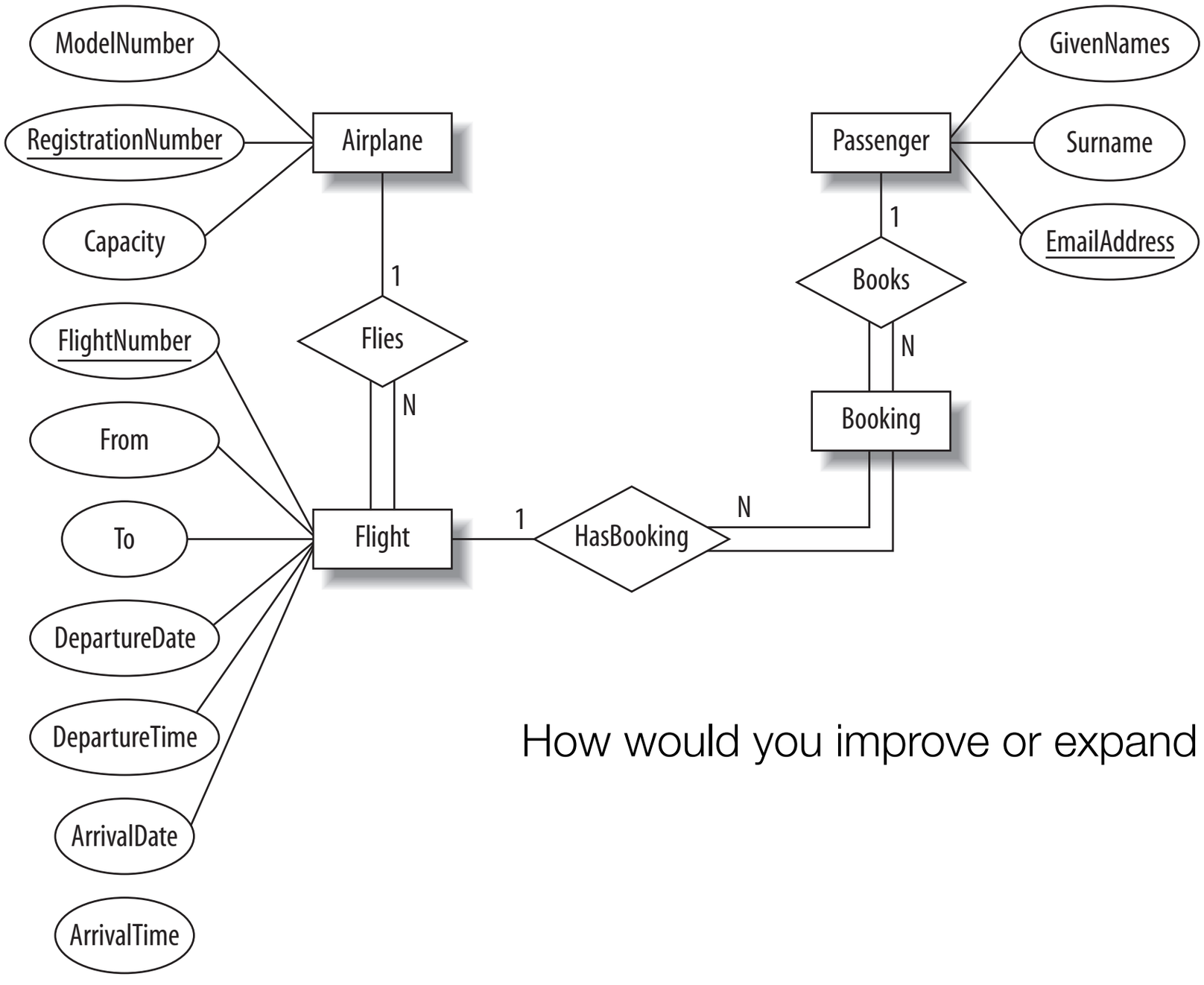


Examples: University Database

How would you improve or expand on this design?



Examples: Flight Database



How would you improve or expand on this design?

Model -> Tables, pt. 1

- For Strong Entities:
    - Create a table comprising its attributes and designate the primary key
- For each multi-valued attribute of an entity:
    - Create a table comprising the entity's primary key and the attribute

Model -> Tables, pt. 2

- For each weak entity:
    - Create a table comprising its attributes and including the primary key of its owning entity
    - The primary key of the owning entity is known as a foreign key here, because it’s a key not of this table, but of another table
    - The primary key of the table for the weak entity is the combination of the foreign key and the partial key of the weak entity
    - If the relationship with the owning entity has any attributes, add them to this table

Model -> Tables, pt. 3

- For one-to-one relationships
    - For each one-to-one relationship between two entities, include the primary key of one entity as a foreign key in the table belonging to the other
    - If one entity participates totally in the relationship, place the foreign key in its table
    - If both participate totally in the relationship, consider merging them into a single table

Model -> Tables, pt. 4

- For each non-identifying one-to-many relationship between two entities:
    - Include the primary key of the entity on the “1” side as a foreign key in the table for the entity on the “M” side.
    - Add any attributes of the relationship in the table alongside the foreign key
- For each many-to-many relationship between two entities:
    - Create a new table containing the primary key of each entity as the primary key, and add any attributes of the relationship

Model -> Tables, pt. 5

- For each relationship involving more than two entities:
    - Create a table with the primary keys of all the participating entities, and add any attributes of the relationship

SELECT Statement, pt. 1

- Basic SELECT all rows from a table
  - SELECT * FROM album;
- SELECT all from from one or more columns
  - SELECT artist_name,artist_id FROM music.artist;
- WHERE clause allows you to filter rows
  - SELECT * FROM artist WHERE artist_name = "New Order";

```
+-----------+----------+----------------------------------------+
| artist_id | album_id | album_name                             |
+-----------+----------+----------------------------------------+
|         2 |        1 | Let Love In                            |
|         1 |        1 | Retro - John McCready FAN              |
|         1 |        2 | Substance (Disc 2)                     |
|         1 |        3 | Retro - Miranda Sawyer POP             |
|         1 |        4 | Retro - New Order / Bobby Gillespie LIVE |
|         3 |        1 | Live Around The World                  |
|         3 |        2 | In A Silent Way                        |
|         1 |        5 | Power, Corruption & Lies               |
|         4 |        1 | Exile On Main Street                   |
|         1 |        6 | Substance 1987 (Disc 1)                |
|         5 |        1 | Second Coming                          |
|         6 |        1 | Light Years                            |
|         1 |        7 | Brotherhood                            |
+-----------+----------+----------------------------------------+
```

```
+---------------------------+-----------+
| artist_name               | artist_id |
+---------------------------+-----------+
| New Order                 |         1 |
| Nick Cave & The Bad Seeds |         2 |
| Miles Davis               |         3 |
| The Rolling Stones        |         4 |
| The Stone Roses           |         5 |
| Kylie Minogue             |         6 |
+---------------------------+-----------+
```

```
+-----------+-------------+
| artist_id | artist_name |
+-----------+-------------+
|         1 | New Order   |
+-----------+-------------+
```

SELECT Statement, pt. 2

- The frequently used operators are equals (=), greater than (>), less than (<), less than or equal (<=), greater than or equal (>=), and not equal (<> or !=)
    - SELECT artist_name FROM artist WHERE artist_id < 5
    - SELECT artist_name FROM artist WHERE artist_name < 'M';
- The LIKE clause is used only with strings and means that a match must meet the pattern
- The percentage character (%) as a wildcard character that matches all possible strings
- The underscore character (_) matches exactly one wildcard character
- SELECT * FROM track WHERE track_name LIKE "R__ %";

```
+---------------------------+
| artist_name               |
+---------------------------+
| New Order                 |
| Nick Cave & The Bad Seeds |
| Miles Davis               |
| The Rolling Stones        |
+---------------------------+
```

```
+---------------+
| artist_name   |
+---------------+
| Kylie Minogue |
+---------------+
```

```
+----------+----------------+-----------+----------+----------+
| track_id | track_name     | artist_id | album_id | time     |
+----------+----------------+-----------+----------+----------+
|        4 | Red Right Hand |         2 |        1 | 00:06:11 |
|       14 | Run Wild       |         1 |        1 | 00:03:57 |
|        1 | Rip This Joint |         4 |        1 | 00:02:23 |
+----------+----------------+-----------+----------+----------+
```

SELECT with REGEX

```
SELECT common_name AS 'Birds Great and Small'
    FROM birds
    WHERE common_name REGEXP 'Great|Least'
    ORDER BY family_id LIMIT 10;
```

```
+-----------------------------+
| Birds Great and Small       |
+-----------------------------+
| Great Northern Loon         |
| Greater Scaup               |
| Greater White-fronted Goose |
| Greater Sand-Plover         |
| Great Crested Tern          |
| Least Tern                  |
| Great Black-backed Gull     |
| Least Nighthawk             |
| Least Pauraque              |
| Great Slaty Woodpecker      |
+-----------------------------+
```

Regular Expressions 1

Expression	Matches	Examples
/ring/	Any string	ring, spring, ringing, stinging
/.ing/	“.” Matches any single character	sing, ping, inglenook
/[bB]ill/	Matches any single character	Bill, bill
/#[6-9]/	“-” matches any range of characters	#60, #8, #9
/[^a-zA-Z]/	“^” any character that is not a within the brackets	1, 7, @
/t*c/	“*” matches any number of preceding characters	tttc, tc, ttc
/t.*ing/	“.*” matches any number of any character	thing, ting

Regular Expressions 2

Expression	Matches	Examples
/^T/	A regular expression that begins with a caret (^) can match a string only at the beginning of a line	This line..., That Time...,
/:$/	In a similar manner, a dollar sign ($) at the end of a regular expression matches the end of a line	...below:
/end \ ./	You can quote any special character by preceding it with a backslash	The end., send., pretend.mail

Boolean Operations, pt. 1

- You can combine two or more conditions using the Boolean operators AND, OR, NOT, and XOR

  - SELECT album_name FROM album WHERE album_name > "C" AND album_name < "M";

  - SELECT album_name FROM album WHERE album_name LIKE "L%" OR album_name LIKE "S%" OR album_name LIKE "P%";

  - SELECT * FROM album WHERE NOT (album_id = 1 OR album_id = 3);

```
+-----------------------+
| album_name            |
+-----------------------+
| Let Love In           |
| Live Around The World |
| In A Silent Way       |
| Exile On Main Street  |
| Light Years           |
+-----------------------+
```

```
+---------------------------+
| album_name                |
+---------------------------+
| Let Love In               |
| Substance (Disc 2)        |
| Live Around The World     |
| Power, Corruption & Lies  |
| Substance 1987 (Disc 1)   |
| Second Coming             |
| Light Years               |
+---------------------------+
```

```
+-----------+----------+-----------------------------------------+
| artist_id | album_id | album_name                              |
+-----------+----------+-----------------------------------------+
|         1 |        2 | Substance (Disc 2)                      |
|         1 |        4 | Retro - New Order / Bobby Gillespie LIVE |
|         3 |        2 | In A Silent Way                         |
|         1 |        5 | Power, Corruption & Lies                |
|         1 |        6 | Substance 1987 (Disc 1)                 |
|         1 |        7 | Brotherhood                             |
+-----------+----------+-----------------------------------------+
```

Boolean Operations, pt. 2

- An exclusive OR evaluates as true if only one—but not both—of the expressions is true
    - SELECT artist_name FROM artist WHERE artist_name LIKE "The%" XOR artist_name LIKE "%es";

ORDER BY

- The ORDER BY clause indicates that sorting is required, followed by the column that should be used as the sort key
- SELECT artist_name FROM artist ORDER BY artist_name DESC;

```
+---------------------------+
| artist_name               |
+---------------------------+
| The Stone Roses           |
| The Rolling Stones        |
| Nick Cave & The Bad Seeds |
| New Order                 |
| Miles Davis               |
| Kylie Minogue             |
+---------------------------+
```

LIMIT

- The LIMIT clause is a useful, non-standard SQL tool that allows you to control which rows are output

- SELECT track_name FROM track LIMIT 9;

```
+----------------------------+
| track_name                 |
+----------------------------+
| Do You Love Me?            |
| Nobody's Baby Now          |
| Loverman                   |
| Jangling Jack              |
| Red Right Hand             |
| I Let Love In              |
| Thirsty Dog                |
| Ain't Gonna Rain Anymore   |
| Lay Me Low                 |
```

- SELECT track_name FROM track LIMIT 5,5;

```
+----------------------------+
| track_name                 |
+----------------------------+
| I Let Love In              |
| Thirsty Dog                |
| Ain't Gonna Rain Anymore   |
| Lay Me Low                 |
| Do You Love Me? (Part Two) |
+----------------------------+
```

JOIN

- JOIN allows you select data from more than one table:
    - SELECT artist_name, album_name FROM artist INNER JOIN album USING (artist_id);

```
+---------------------------+-----------------------------------------+
| artist_name               | album_name                              |
+---------------------------+-----------------------------------------+
| New Order                 | Retro - John McCready FAN               |
| New Order                 | Substance (Disc 2)                      |
| New Order                 | Retro - Miranda Sawyer POP              |
| New Order                 | Retro - New Order / Bobby Gillespie LIVE |
| New Order                 | Power, Corruption & Lies                |
| New Order                 | Substance 1987 (Disc 1)                 |
| New Order                 | Brotherhood                             |
| Nick Cave & The Bad Seeds | Let Love In                             |
| Miles Davis               | Live Around The World                   |
| Miles Davis               | In A Silent Way                         |
| The Rolling Stones        | Exile On Main Street                    |
| The Stone Roses           | Second Coming                           |
| Kylie Minogue             | Light Years                             |
+---------------------------+-----------------------------------------+
```

String Functions

CONCAT – combine two or more strings into one string

SELECT CONCAT('My', 'S', 'QL');

-> ‘MySQL'

LENGTH & CHAR_LENGTH – get the length of strings in bytes and in characters

SELECT LENGTH('text');

-> 4

LEFT – get the left part of a string with a specified length

SELECT LEFT('foobarbar', 5);

-> 'fooba'

String Functions

SUBSTRING – extract a substring starting from a position with a specific length

SELECT SUBSTRING('Quadratically',5);

-> 'ratically'

TRIM – remove unwanted characters from a string

SELECT TRIM(' bar   ');

-> 'bar'

FIND_IN_SET – find a string within a comma-separated list of strings

SELECT FIND_IN_SET('b','a,b,c,d');

-> 2

FORMAT – format a number with a specific locale, rounded to the number of decimals
FORMAT(X,D[,locale])

SELECT FORMAT(12332.123456, 4);

-> ’12,332.1235'

String Function Examples

```
SELECT CONCAT(formal_title, '. ', name_first, SPACE(1),
name_last) AS Birder,
CONCAT(common_name, ' - ', birds.scientific_name) AS Bird,
time_seen AS 'When Spotted’ FROM birdwatchers.bird_sightings
JOIN birdwatchers.humans USING(human_id)
JOIN rookery.birds
USING(bird_id) GROUP BY human_id DESC LIMIT 4;
```

```
+----------------------+-----------------------------------+--------------------+
| Birder               | Bird                              | When Spotted       |
+----------------------+-----------------------------------+--------------------+
| Ms. Marie Dyer       | Red-billed Curassow - Crax blu...| 2013-10-02 07:39:44|
| Ms. Anahit Vanetsyan | Bar-tailed Godwit - Limosa lap...| 2013-10-01 05:40:00|
| Ms. Katerina Smirnova| Eurasian Curlew - Numenius arq...| 2013-10-01 07:06:46|
| Ms. Elena Bokova     | Eskimo Curlew - Numenius borea...| 2013-10-01 05:09:27|
+----------------------+-----------------------------------+--------------------+
```

Additional Functions

- LAST_INSERT_ID – obtain the last generated sequence number of the last inserted record using auto_increment

INSERT

- The INSERT statement is used to add new data to tables
    - INSERT INTO artist VALUES (7, "Barry Adamson");
    - INSERT INTO track VALUES (1, "Diamonds", 7, 1, 4.10), (2, "Boppin Out / Eternal Morning", 7, 1, 3.22), (3, "Splat Goes the Cat", 7, 1, 1.39), (4, "From Rusholme With Love", 7, 1, 3.59);
    - INSERT INTO album (artist_id, album_id, album_name) VALUES (7, 2, "Oedipus Schmoedipus");
    - INSERT INTO played SET artist_id = 7, album_id = 1, track_id = 1;

DELETE

- The DELETE statement is used to remove one or more rows from a database.
  - DELETE FROM played;
  - DELETE FROM played WHERE played < "2006-08-15";
  - DELETE FROM played ORDER BY played LIMIT 528;

TRUNCATE

- If you want to remove all rows in a table use truncate
    - TRUNCATE TABLE played;

UPDATE

- The UPDATE statement is used to change data.
    - UPDATE artist SET artist_name = UPPER(artist_name);

CREATE/DROP Databases

- Use the following to create a database:
    - CREATE DATABASE lucy;
    - CREATE DATABASE IF NOT EXISTS lucy;
- A new directory under the data directory is created for the new database and stores the text file db.opt that lists the database options;
- To delete a database:
    - DROP DATABASE music;

CREATE Tables

- Syntax:

```
CREATE TABLE IF NOT EXISTS artist (
    artist_id SMALLINT(5) NOT NULL DEFAULT 0,
    artist_name CHAR(128) DEFAULT NULL,
    PRIMARY KEY (artist_id)
);
```

COLLATION and CHARACTER SETS

- Character sets define what characters can be stored

- A collation defines how strings are ordered, and there are different collations for different languages

- SHOW CHARACTER SET;

- SHOW COLLATION;

Column Data Types, pt. 1

- https://www.tutorialspoint.com/mysql/mysql-data-types.htm
- INT[(width)]
  - range –2,147,483,648 to 2,147,483,647
  - my_number INT(4) ZEROFILL
  - TINYINT(1),SMALLINT(2),MEDIUMINT(3), INT(4), BIGINT(8)
- DECIMAL
  - unit_cost DECIMAL(4,2) default 0.00
- DATE
  - format stored as YYYY-MM-DD
  - alternate insertion formats allowed
- TIME
  - Stores a time in the format HHH:MM:SS
  - Input formats: [DD] HH:MM:SS, [DD] HH:MM, [DD] HH, or SS
  - Days can’t exceed 34

Column Data Types, pt. 2

- DATETIME
  - Stored as YYYY-MM-DD HH:MM:SS
- TIMESTAMP
  - TIMESTAMP values are stored as the number of seconds since the epoch ('1970-01-01 00:00:00' UTC).
  - DEFAULT CURRENT_TIMESTAMP will add current time on insert
- YEAR[(digits)]
  - 4 (2 Digit year deprecated and will be removed in future)
- CHAR[(width)]
  - Stores a fixed-length string
  - maximum value of width is 255, default is 1
- VARCHAR(width)
  - Stores variable-length strings
  - maximum value of width is 65,535 characters
- BOOLEAN
  - false (zero) or true (nonzero)

Column Data Types, pt. 3

- FLOAT[(width, decimals)] or FLOAT[(precision)]
- DOUBLE[(width, decimals)]
- TEXT
  - Stores a variable amount of data (such as a document or other text file) up to 65,535 bytes in length
- ENUM('value1'[,'value2'[, ...]]
  - A list, or enumeration of string values
  - CREATE TABLE fruits_enum (fruit_name ENUM('Apple', 'Orange', 'Pear'));
- SET('value1'[,'value2'[, ...]])
  - A column of type SET can be set to zero or more values from the list value1, value2, and so on, up to a maximum of 64 different values
  - CREATE TABLE fruits_set (fruit_name SET('Apple', 'Orange', 'Pear'));

Time Date Examples

```
SELECT common_name AS 'Endangered Bird',
CONCAT(name_first, SPACE(1), name_last) AS 'Birdwatcher',
DATE_FORMAT(time_seen, '%W, %M %e, %Y') AS 'Date Spotted',
TIME_FORMAT(time_seen, '%l:%i %p') AS 'Time Spotted'
FROM bird_sightings
JOIN humans USING(human_id)
JOIN rookery.birds USING(bird_id)
JOIN rookery.conservation_status USING(conservation_status_id)
WHERE conservation_category = 'Threatened' LIMIT 3;
```

```
+--------------------+-------------+----------------------------+-----------+
| Endangered Bird    | Birdwatcher | Date Spotted               | Time      |
+--------------------+-------------+----------------------------+-----------+
| Eskimo Curlew      | Elena Bokova | Tuesday, October 1, 2013   | 5:09 AM   |
| Red-billed Curassow | Marie Dyer  | Wednesday, October 2, 2013 | 7:39 AM   |
| Red-billed Curassow | Elena Bokova | Wednesday, October 2, 2013 | 8:41 AM   |
+--------------------+-------------+----------------------------+-----------+
```

Primary Key Index

- A primary key uniquely identifies each row in a table
- When you declare one to MySQL, it creates a new file on disk that stores information about where the data from each row in the table is stored
- This information is called an index, and its purpose is to speed up searches that use the primary key
- You can display the indexes available on a table using the SHOW INDEX command
- ALTER TABLE staff ADD PRIMARY KEY (name);

More Indexes

- More Index Examples:

- ALTER TABLE artist ADD INDEX by_name (artist_name);

- CREATE TABLE artist (
  artist_id SMALLINT(5) NOT NULL DEFAULT 0,
  artist_name CHAR(128) DEFAULT NULL,
  PRIMARY KEY (artist_id),
  KEY artist_name (artist_name)
  );

AUTO_INCREMENT

- AUTO_INCREMENT feature allows you to create a unique identifier for a row without running a SELECT query
- SERIAL (Alias for AUTO_INCREMENT, PRIMARY, UNIQUE)
- NOT NULL is required for AUTO_INCREMENT columns
- The column it is used on must be indexed
- The column that is it used on cannot have a DEFAULT value
- There can be only one AUTO_INCREMENT column per table

Altering Tables

- You can use the ALTER TABLE statement to add new columns to a table, remove existing columns, and change column names, types, and lengths

  - ALTER TABLE played CHANGE played last_played TIMESTAMP;

  - ALTER TABLE artist MODIFY artist_name CHAR(64) DEFAULT "Unknown";

  - ALTER TABLE artist ADD formed YEAR;

  - ALTER TABLE artist DROP formed;

  - ALTER TABLE played RENAME TO playlist

Deleting Structures

- DROP DATABASE IF EXISTS music;
- DROP TABLE IF EXISTS temp;
- ALTER TABLE artist DROP INDEX by_name;
- ALTER TABLE artist DROP PRIMARY KEY;

Aliases, pt. 1

- Allow columns to be renamed
- SELECT artist_name AS artists FROM artist;
- SELECT CONCAT(artist_name, " recorded ", album_name) AS recording FROM artist INNER JOIN album USING (artist_id) ORDER BY recording;

```
+---------------------------+
| artists                   |
+---------------------------+
| New Order                 |
| Nick Cave & The Bad Seeds |
| Miles Davis               |
| The Rolling Stones        |
| The Stone Roses           |
| Kylie Minogue             |
+---------------------------+
```

```
+-----------------------------------------------------------+
| recording                                                 |
+-----------------------------------------------------------+
| Kylie Minogue recorded Light Years                        |
| Miles Davis recorded In A Silent Way                      |
| Miles Davis recorded Live Around The World                |
| New Order recorded Brotherhood                            |
| New Order recorded Power, Corruption & Lies               |
| New Order recorded Retro - John McCready FAN              |
| New Order recorded Retro - Miranda Sawyer POP             |
| New Order recorded Retro - New Order / Bobby Gillespie LIVE |
| New Order recorded Substance (Disc 2)                     |
| New Order recorded Substance 1987 (Disc 1)                |
| Nick Cave & The Bad Seeds recorded Let Love In            |
| The Rolling Stones recorded Exile On Main Street          |
| The Stone Roses recorded Second Coming                    |
```

Aliases, pt. 2

- Aliases can also be used for tables

- SELECT ar.artist_id, al.album_name, ar.artist_name FROM album AS al INNER JOIN artist AS ar USING (artist_id) WHERE al.album_name = "Brotherhood";

```
+-----------+-------------+-------------+
| artist_id | album_name  | artist_name |
+-----------+-------------+-------------+
|         1 | Brotherhood | New Order   |
+-----------+-------------+-------------+
```

Distinct Clause

- Removes duplicates

- Query 1: SELECT DISTINCT artist_name FROM artist INNER JOIN album USING (artist_id);

- Query 2: SELECT artist_name FROM artist INNER JOIN album USING (artist_id);

```
+----------------------------+
| artist_name                |
+----------------------------+
| New Order                  |
| Nick Cave & The Bad Seeds  |
| Miles Davis                |
| The Rolling Stones         |
| The Stone Roses            |
| Kylie Minogue              |
+----------------------------+
```

```
+----------------------------+
| artist_name                |
+----------------------------+
| New Order                  |
| New Order                  |
| New Order                  |
| New Order                  |
| New Order                  |
| New Order                  |
| New Order                  |
| Nick Cave & The Bad Seeds  |
| Miles Davis                |
| Miles Davis                |
| The Rolling Stones         |
| The Stone Roses            |
| Kylie Minogue              |
```

Group By

- The GROUP BY clause sorts data into groups for the purpose of aggregation
- SELECT artist_name, COUNT(artist_name) FROM artist INNER JOIN album USING (artist_id) GROUP BY artist_name;
- SELECT artist_name, album_name, COUNT(*) FROM artist INNER JOIN album USING (artist_id) INNER JOIN track USING (artist_id, album_id) GROUP BY artist.artist_id, album.album_id;

```
+--------------------------+--------------------+
| artist_name              | COUNT(artist_name) |
+--------------------------+--------------------+
| Kylie Minogue            |                  1 |
| Miles Davis              |                  2 |
| New Order                |                  7 |
| Nick Cave & The Bad Seeds |                 1 |
| The Rolling Stones       |                  1 |
| The Stone Roses          |                  1 |
+--------------------------+--------------------+
```

```
+--------------------------+-------------------------------------+----------+
| artist_name              | album_name                          | COUNT(*) |
+--------------------------+-------------------------------------+----------+
| New Order                | Retro - John McCready FAN           |       15 |
| New Order                | Substance (Disc 2)                  |       12 |
| New Order                | Retro - Miranda Sawyer POP          |       14 |
| New Order                | Retro - New Order / Bobby Gillespie |       15 |
| New Order                | Power, Corruption & Lies            |        8 |
| New Order                | Substance 1987 (Disc 1)             |       12 |
| New Order                | Brotherhood                         |       10 |
| Nick Cave & The Bad Seeds | Let Love In                        |       10 |
| Miles Davis              | Live Around The World               |       11 |
| Miles Davis              | In A Silent Way                     |        2 |
| The Rolling Stones       | Exile On Main Street                |       18 |
| The Stone Roses          | Second Coming                       |       13 |
| Kylie Minogue            | Light Years                         |       13 |
```

Aggregation Functions

- AVG( )
- MAX( )
- MIN( )
- STD( ) or STDDEV( )
- SUM( )

Having Clause

- Adds additional control to the aggregation of rows in a GROUP BY operation
- Must contain an expression or column that's listed in the SELECT clause
- SELECT artist_name, album_name, COUNT(*) FROM artist INNER JOIN album USING (artist_id) INNER JOIN track USING (artist_id, album_id) GROUP BY artist.artist_id, album.album_id HAVING COUNT(*) > 10;

```
+--------------------+-----------------------------------------+----------+
| artist_name        | album_name                              | COUNT(*) |
+--------------------+-----------------------------------------+----------+
| New Order          | Retro - John McCready FAN               |       15 |
| New Order          | Substance (Disc 2)                      |       12 |
| New Order          | Retro - Miranda Sawyer POP              |       14 |
| New Order          | Retro - New Order / Bobby Gillespie LIVE |       15 |
| New Order          | Substance 1987 (Disc 1)                 |       12 |
| Miles Davis        | Live Around The World                   |       11 |
| The Rolling Stones | Exile On Main Street                    |       18 |
| The Stone Roses    | Second Coming                           |       13 |
| Kylie Minogue      | Light Years                             |       13 |
+--------------------+-----------------------------------------+----------+
```

Inner Join

- Joins two tables, and only rows that match are displayed
  - SELECT artist_name, album_name FROM artist INNER JOIN album USING (artist_id);
  - SELECT artist_name, album_name FROM artist, album WHERE artist.artist_id = album.artist_id;
  - SELECT artist_name, album_name FROM artist INNER JOIN album ON artist.artist_id = album.artist_id;

```
+--------------------------+-----------------------------------------+
| artist_name              | album_name                              |
+--------------------------+-----------------------------------------+
| New Order                | Retro - John McCready FAN               |
| New Order                | Substance (Disc 2)                      |
| New Order                | Retro - Miranda Sawyer POP              |
| New Order                | Retro - New Order / Bobby Gillespie LIVE |
| New Order                | Power, Corruption & Lies                |
| New Order                | Substance 1987 (Disc 1)                 |
| New Order                | Brotherhood                             |
| Nick Cave & The Bad Seeds | Let Love In                             |
| Miles Davis              | Live Around The World                   |
| Miles Davis              | In A Silent Way                         |
| The Rolling Stones       | Exile On Main Street                    |
| The Stone Roses          | Second Coming                           |
| Kylie Minogue            | Light Years                             |
+--------------------------+-----------------------------------------+
```

Union

- Allows you to combine the output of more than one SELECT statement to give a consolidated result set
- The queries should output the same number of columns and the same type
- Results are distinct, unless you use union all
- (SELECT track_name FROM track INNER JOIN played USING (artist_id, album_id, track_id) ORDER BY played ASC LIMIT 5) UNION ALL (SELECT track_name FROM track INNER JOIN played USING (artist_id, album_id, track_id) ORDER BY played DESC LIMIT 5);

```
+----------------------+
| track_name           |
+----------------------+
| Fine Time            |
| Temptation           |
| Fine Time            |
| True Faith           |
| The Perfect Kiss     |
| New Blues            |
| Intruder             |
| In A Silent Way      |
| Bizarre Love Triangle |
| Crystal              |
+----------------------+
```

Left/Right Join

- Includes all the data from either the left or right table in the join

- SELECT track_name, played FROM track LEFT JOIN played USING (artist_id, album_id, track_id) ORDER BY played DESC;

```
| Crystal                | 2006-08-14 10:47:21 |
| Regret                 | 2006-08-14 10:43:37 |
| Ceremony               | 2006-08-14 10:41:43 |
| The Perfect Kiss       | 2006-08-14 10:36:54 |
| True Faith             | 2006-08-14 10:30:25 |
| Temptation             | 2006-08-14 10:25:22 |
| Fine Time              | 2006-08-14 10:21:03 |
| Do You Love Me?        |                NULL |
| Nobody's Baby Now      |                NULL |
| Loverman               |                NULL |
| Jangling Jack          |                NULL |
| Red Right Hand         |                NULL |
| I Let Love In          |                NULL |
```

Nested Query

- Query using join: SELECT artist_name FROM artist INNER JOIN album USING (artist_id) WHERE album_name = "In A Silent Way";

- Query using nested query: SELECT artist_name FROM artist WHERE artist_id = (SELECT artist_id FROM album WHERE album_name = "In A Silent Way");

```
+-------------+
| artist_name |
+-------------+
| Miles Davis |
+-------------+
```

ANY/IN Subqueries

- SELECT producer_name FROM producer WHERE producer_name = ANY (SELECT engineer_name FROM engineer);

- SELECT producer_name FROM producer WHERE producer_name IN (SELECT engineer_name FROM engineer);

```
+---------------+
| producer_name |
+---------------+
| George Martin |
+---------------+
```

Exists and NOT Exists

- SELECT * FROM artist WHERE EXISTS (SELECT * FROM played);

```
+-----------+--------------------------+
| artist_id | artist_name              |
+-----------+--------------------------+
|         1 | New Order                |
|         2 | Nick Cave & The Bad Seeds |
|         3 | Miles Davis              |
|         4 | The Rolling Stones       |
|         5 | The Stone Roses          |
|         6 | Kylie Minogue            |
+-----------+--------------------------+
```

Variables

- Save values that are returned from queries
- SELECT @artist:=artist_name FROM artist WHERE artist_id = 1;
- SELECT @artist;

```
+-----------------------+
| @artist:=artist_name  |
+-----------------------+
| New Order             |
+-----------------------+
```

INSERT using SELECT

- Tracks inserted into shuffle table using select statement, which contains random tracts from albums and artists

    - INSERT INTO shuffle (artist_id, album_id, track_id) SELECT artist_id, album_id, track_id FROM track ORDER BY RAND() LIMIT 10;

```
+-----------+----------+----------+-------------+
| artist_id | album_id | track_id | sequence_id |
+-----------+----------+----------+-------------+
|         1 |        7 |        0 |           1 |
|         3 |        1 |        3 |           2 |
|         1 |        3 |       10 |           3 |
|         6 |        1 |        1 |           4 |
|         4 |        1 |        8 |           5 |
|         1 |        7 |        1 |           6 |
|         1 |        1 |        4 |           7 |
|         2 |        1 |        6 |           8 |
|         1 |        6 |        0 |           9 |
|         4 |        1 |        1 |          10 |
+-----------+----------+----------+-------------+
```

INSERT using SELECT

- Data inserted from another database, note the use of “.” to specify database
- INSERT INTO artist.people (people_id, name) SELECT artist_id, artist_name FROM music.artist;
- SELECT * FROM ARTIST;

artist_id	artist_name
1	New Order
2	Nick Cave & The Bad Seeds
3	Miles Davis
4	The Rolling Stones
5	The Stone Roses
6	Kylie Minogue
60	Kylie Minogue
50	The Stone Roses
40	The Rolling Stones
30	Miles Davis
20	Nick Cave & The Bad Seeds
10	New Order

Create Tables with Existing Data

- New tables can be created to replicate data to for backup or testing
- Doesn’t replicate indexes, they can be created after using an alter table statement
- CREATE TABLE artist_2 SELECT * from artist;

```
+-----------+-----------------------------+
| artist_id | artist_name                 |
+-----------+-----------------------------+
|         1 | New Order                   |
|         2 | Nick Cave and The Bad Seeds |
|         3 | Miles Dewey Davis           |
|         4 | The Rolling Stones          |
|         5 | The Stone Roses             |
|         6 | Kylie Minogue               |
|        10 | Jane's Addiction            |
+-----------+-----------------------------+
```

Replace Data

- The replace command will delete an existing row and insert a new row with data

- The same could be accomplished by using an UPDATE commands, which changes existing data

- Bulk replace example:

    - REPLACE artist (artist_id, artist_name) VALUES (2, "Nick Cave and The Bad Seeds"), (3, "Miles Dewey Davis");

Views

- A view is a virtual table that doesn’t store any data itself
- Instead, the data “in” the table is derived from a SQL query that MySQL runs when you access the view
- Create View statement:
    - CREATE VIEW Oceania AS
      SELECT * FROM Country WHERE Continent = 'Oceania' WITH CHECK OPTION;
- Views can be removed with
    - DROP VIEW Oceania

Updating Views

- A view is not updatable if it contains GROUP BY, UNION, an aggregate function, or any of a few other exceptions

- A query that changes data might contain a join, but the columns to be changed must all be in a single table

- Any view that uses the TEMPTABLE algorithm is not updatable

- MySQL doesn’t support materialized views

- A materialized view generally stores its results in an invisible table behind the scenes, with periodic updates to refresh the invisible table from the source data

Foreign Key Relationships

- Foreign key constraints are only supported by the InnoDB storage engine
- A foreign key constraint can be defined within a CREATE TABLE or ALTER TABLE statement:
  - FOREIGN KEY (field1) REFERENCES other_table (fieldA)
- Only compound foreign key constraints that reference one table are allowed, such as:
  - FOREIGN KEY (field1, field2) REFERENCES other_table (fieldA, fieldB)
- If you want to specify the name for the foreign key, you can optionally prepend CONSTRAINT name to the preceding syntax:
  - CONSTRAINT fk_name FOREIGN KEY (field1, field2) REFERENCES other_table (fieldA, fieldB)

Foreign Key Relationships

- The foreign key name is used when removing a foreign key constraint:
  - ALTER TABLE tblname DROP FOREIGN KEY fk_name
- A table can have a foreign key that references itself — that is:
  - ALTER TABLE this_table ADD FOREIGN KEY (field1) REFERENCES this_table (fieldA);
- If an UPDATE or DELETE statement tries to change existing data on the parent table, the server can cascade the changes (ON DELETE and ON UPDATE):
  - RESTRICT, CASCADE, SET NULL, NO ACTION

Transactions

- Used to assure atomicity
- Increase performance

START TRANSACTION;

INSERT INTO sales (inventory_id, quantity, price) VALUES (1,5,500);

UPDATE inventory SET onhand = (onhand - 5) WHERE inventory_id = 1;

COMMIT

Stored Procedures

- Advantages:
  - Save bandwidth and reduce latency running code from the server
  - Centralize business rules
  - Ease maintenance, backup, and recovery
  - More secure (prevents external access to underlying tables)
  - Performs better due to caching of code and execution plans
  - Enables division of labor between database programmers and application developers

Stored Procedures

- Disadvantages:
  - MySQL development and debugging tools are not as good as other development tools
  - SQL is slower and more primitive than others (ie. complex logic and string manipulation)
  - Add complexity to the application
  - Adds extra load on the database server (hard to scale out)

Stored Procedure Example 1

```
DELIMITER //
CREATE PROCEDURE GetOfficeByCountry(IN countryName VARCHAR(255))
    BEGIN
      SELECT *
      FROM offices
      WHERE country = countryName;
    END //
DELIMITER ;

CALL GetOfficeByCountry('USA');
```

	officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory
▶	1	San Francisco	+1 650 219 4782	100 Market Street	Suite 300	CA	USA	94080	NA
	2	Boston	+1 215 837 0825	1550 Court Place	Suite 102	MA	USA	02107	NA
	3	NYC	+1 212 555 3000	523 East 53rd Street	apt. 5A	NY	USA	10022	NA

Stored Procedure Example 2

```
DELIMITER $$
CREATE PROCEDURE CountOrderByStatus(
    IN orderStatus VARCHAR(25),
    OUT total INT)
BEGIN
    SELECT count(orderNumber)
    INTO total
    FROM orders
    WHERE status = orderStatus;
END$$
DELIMITER ;

CALL CountOrderByStatus('Shipped',@total);
SELECT @total;
```

	@total
▶	303

Stored Function Example

```
DELIMITER $$
CREATE FUNCTION CustomerLevel(p_creditLimit double) RETURNS VARCHAR(10)
    DETERMINISTIC
BEGIN
    DECLARE lvl varchar(10);
    IF p_creditLimit > 50000 THEN
 SET lvl = 'PLATINUM';
    ELSEIF (p_creditLimit <= 50000 AND p_creditLimit >= 10000) THEN
        SET lvl = 'GOLD';
    ELSEIF p_creditLimit < 10000 THEN
        SET lvl = 'SILVER';
    END IF;
 RETURN (lvl);
END
```

Stored Function Example, part 2

```
SELECT
    customerName, CustomerLevel(creditLimit)
FROM
    customers
ORDER BY customerName;
```

	customerName	CustomerLevel(creditLimit)
▶	Alpha Cognac	PLATINUM
	American Souvenirs Inc	SILVER
	Amica Models & Co.	PLATINUM
	ANG Resellers	SILVER
	Anna's Decorations, Ltd	PLATINUM
	Anton Designs, Ltd.	SILVER
	Asian Shopping Network, Co	SILVER
	Asian Treasures, Inc.	SILVER
	Atelier graphique	GOLD
	Australian Collectables, Ltd	PLATINUM
	Australian Collectors, Co.	PLATINUM

Triggers

- Triggers let you execute code when there's an INSERT, UPDATE, or DELETE statement
- You can direct MySQL to activate triggers before and/or after the triggering statement executes
- They cannot return values, but they can read and/or change the data that the triggering statement changes
- Thus, you can use triggers to enforce constraints or business logic that you'd otherwise need to write in client code
- The can be hard to debug and have the same drawbacks as stored procedures

Trigger Example 1

```
CREATE TABLE customer ( customer_id SERIAL, name VARCHAR(255,
last_order_int INT );
CREATE TABLE sale ( sale_id SERIAL, item_id INT, customer_id INT, quantity
INT, price DECIMAL (9,2) );
```

```
CREATE TRIGGER newSale AFTER INSERT ON sale
    FOR EACH ROW
    UPDATE customer SET last_order_id = NEW.sale_id WHERE customer_id
    = NEW.customer_id
```

Trigger Example 2

```
CREATE TABLE customer ( customer_id SERIAL, name VARCHAR(255,
last_order_int INT );

CREATE TABLE sale ( sale_id SERIAL, item_id INT, customer_id INT, quantity
INT, price DECIMAL (9,2) );

CREATE TABLE log ( log_id, SERIAL, stamp TIMESTAMP event
VARCHAR(255), username VARCHAR(255), tablename VARCHAR(255),
table_id BIGINT );

CREATE TRIGGER logSale AFTER INSERT ON sale
    FOR EACH ROW
    INSERT INTO log (event, username, tablename, table_id)
        VALUES ( 'INSERT', 'TRIGGER', 'sale', NEW.sale_id);
```

Comments

- - - Single line comment
- /* C Style multi-line

comment */

- # - deprecated

Full Text Searches, pt. 1

- Allow text searching in multiple columns
- Natural language query matches words based on relevance
- Boolean searches, the query itself specifies the relative relevance of each word in a match
- Match() function specifies the columns
- Against() function specifies the text to search against

Full Text Searches, pt. 2

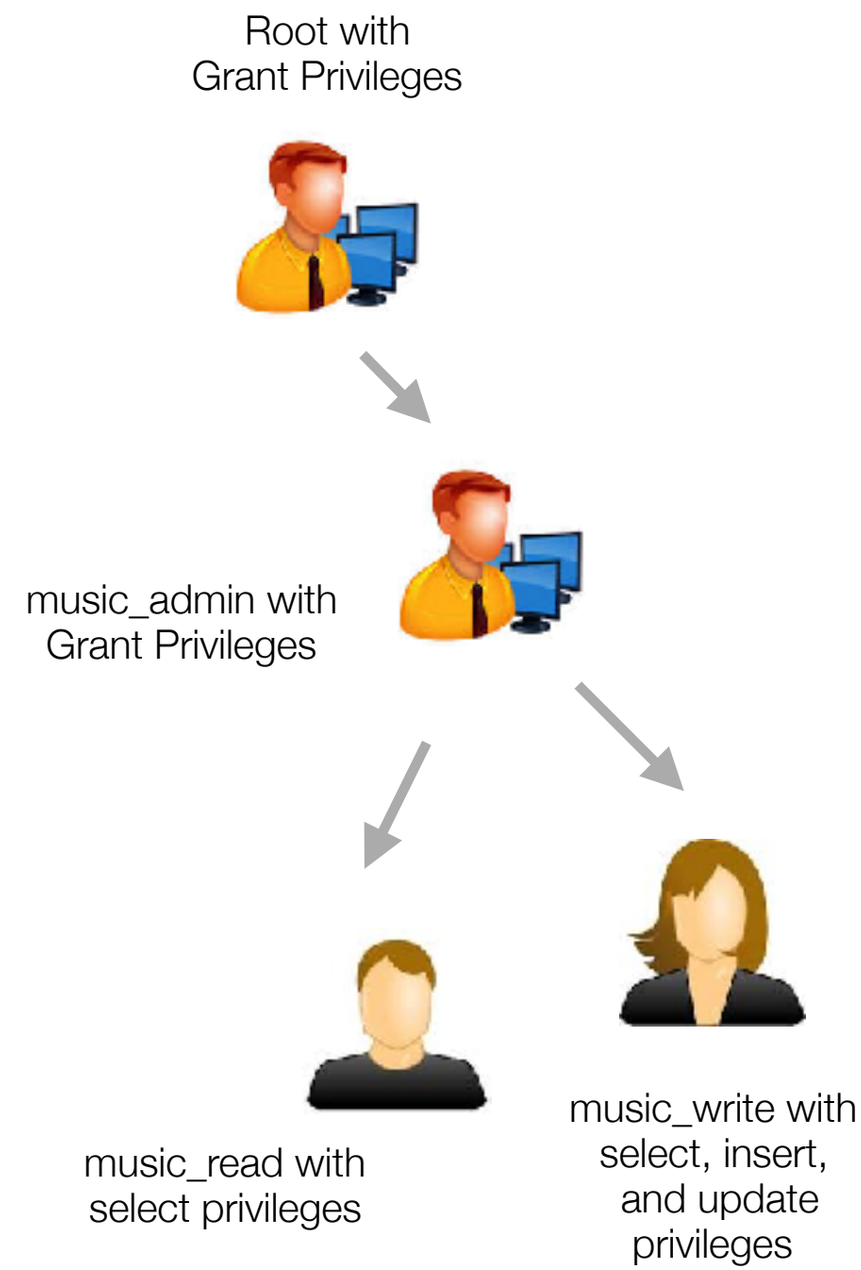
- Natural Language Query
- Tutorial: http://www.mysqltutorial.org/mysql-full-text-search.aspx

```
SELECT film_id, title, RIGHT(description, 25),
MATCH(title, description) AGAINST('factory casualties') AS relevance
FROM sakila.film_text
WHERE MATCH(title, description) AGAINST('factory casualties');
```

```
+---------+----------------------+---------------------------+-----------------+
| film_id | title                | RIGHT(description, 25)    | relevance       |
+---------+----------------------+---------------------------+-----------------+
|     831 | SPIRITED CASUALTIES  | a Car in A Baloon Factory | 8.4692449569702 |
|     126 | CASUALTIES ENCINO    | Face a Boy in A Monastery | 5.2615661621094 |
|     193 | CROSSROADS CASUALTIES | a Composer in The Outback | 5.2072987556458 |
|     369 | GOODFELLAS SALUTE    | d Cow in A Baloon Factory | 3.1522686481476 |
|     451 | IGBY MAKER           | a Dog in A Baloon Factory | 3.1522686481476 |
```

Least Privilege

- The user root, who can do everything on the server, including creating and deleting users, databases, tables, indexes, and data
- Most applications don’t need superuser privileges for day-to-day activities
- Define less powerful users who have only the privileges they need to get their jobs done
  - music_admin - can perform any database operation on the music database
  - music_read - can read data from the music database but can’t change anything



Remote Access

- If you want to allow a user to connect to the server from another computer, you must specify the host from which they can do so (the remote client)
  - GRANT ALL ON *.* TO 'hugh'@'192.168.1.2' IDENTIFIED BY 'the_password';
- The wildcard character % matches any string
- For example, this command allows jill to connect from any machine in the domain
  - GRANT ALL ON *.* TO 'jill'@'%.invyhome.com' IDENTIFIED BY ‘the_password';
- bind-address = * in my.cnf

Error Logs

- Error log - contains the error occurred during the server execution
  - Located in data diretory (.err)
  - Enable with log_error variable (or start with —log-error)

```
federico@this:~$ sudo tail -20 /usr/local/mysql/data/this.err | grep
ERROR

140101 18:11:21 [ERROR] /usr/local/mysql/bin/mysqld: unknown variable
'base_dir=/usr/local/mysql'

140101 18:11:21 [ERROR] Aborting
```

- SQL_ERROR_LOG (MariaDB) - errors generated by the SQL statements, stored routine, or trigger into a file

General Query Log

- All statements can be sent to the general query log
- Suitable for finding problems that are caused by the application's bugs
- By default, it is disabled, but it can also be enabled or disabled at runtime using the general_log dynamic system variable (—general-log start-up option)
  - SET GLOBAL general_log = 1
- The general query log can be written in the form of a table called general_log in the MySQL database

Slow Query Logs

- Slow query logs - stores the queries that take more than a given amount of time or do not use any index, and is useful for finding out why an application or database is slow
- To enable it, the slow_query_log variable or the —slow-query-log startup option can be set to 1

```
# Time: 140116 11:19:05
# User@Host: root[root] @ localhost []
# Thread_id: 4  Schema: test  QC_hit: No
# Query_time: 0.059419  Lock_time: 0.000340  Rows_sent: 1  Rows_examined:
66620
SET timestamp=1389867545;
SELECT COUNT(*) FROM t
 WHERE a > b;
```

Variables

- You can use the SHOW STATUS command to see the values of over 250+ status variables:
  - SHOW STATUS
  - SHOW GLOBAL STATUS (All users)
  - SHOW GLOBAL STATUS LIKE ‘%tmp%’;
  - SHOW TABLE STATUS \G
- The SHOW VARIABLES can be used to determine the values of system variables:
  - SHOW GLOBAL VARIABLES LIKE ’%tmp_%’;

MariaDB Statistics

- Stores statistics on users, tables, and indexes in a hash table in memory
- To enable
  - SET GLOBAL userstat = 1;
- INFORMATON_SCHEMA tables
- SHOW USER_STATISTICS, CLIENT_STATISTICS, TABLE_STATISTICS, INDEX_STATISTICS

Connect Remotely

- To connect remotely
  - mysql -u username -p -h hostname
- To see status on connection
  - status

```
--------------
mysql  Ver 14.14 Distrib 5.7.19, for macos10.12 (x86_64) using  EditLine
wrapper

Connection id:      2548
Current database:   cars
Current user:    root@localhost
SSL:          Not in use
Current pager:      stdout
Using outfile:      ''
Using delimiter: ;
Server version:     5.7.19 MySQL Community Server (GPL)
Protocol version:   10
Connection:     Localhost via UNIX socket
Server characterset:    latin1
Db     characterset:    latin1
Client characterset:    utf8
Conn.  characterset:    utf8
UNIX socket:    /tmp/mysql.sock
Uptime:         5 days 13 hours 28 min 11 sec

Threads: 1  Questions: 29  Slow queries: 0  Opens: 107  Flush tables: 1  Open
tables: 100  Queries per second avg: 0.000
--------------
```

Revoking Rights and Dropping Users

- You can selectively revoke privileges with the REVOKE statement, which essentially has the same syntax as GRANT
  - REVOKE SELECT (track_id) ON music.track FROM 'partmusic'@'localhost';
  - REVOKE ALL PRIVILEGES ON music.artist FROM 'partmusic'@'localhost';
  - REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'partmusic'@'localhost';
- You can remove access to the MySQL server by removing a user
  - DROP USER 'partmusic'@'localhost';

Changing Passwords

- Passwords can be changed by re-issuing grant command
  - GRANT USAGE ON *.* TO 'selina'@'localhost' IDENTIFIED BY '*another_password*';
- Also issue a set password statement
  - SET PASSWORD=PASSWORD('the_password');
  - SET PASSWORD FOR 'selina'@'localhost' = PASSWORD('another_password');

View Users

- View all users
    - SELECT user,host FROM mysql.user;

Resource Limits

- MySQL allows you to set
  - MAX_QUERIES_PER_HOUR
  - MAX_UPDATES_PER_HOUR
  - MAX_CONNECTIONS_PER_HOUR
- For example:

```
GRANT USAGE ON *.* to 'partmusic'@'localhost' WITH
MAX_QUERIES_PER_HOUR 100
MAX_UPDATES_PER_HOUR 10
MAX_CONNECTIONS_PER_HOUR 5;
```

MySQL Database Tables

- The user table holds user accounts
- The db, tables_priv, column_priv, hold privileges for all databases, tables, and columns

Reset Root Password

- mysqld_safe --skip-grant-tables
- FLUSH PRIVILEGES;
- SET PASSWORD for ‘root'@'localhost'= PASSWORD('the_new_mysql_root_password');
- FLUSH PRIVILEGES;

User Accounts, pt. 1

- To create a new user, you need to have permission to do so (ie. root)
  - CREATE USER 'paola'@'localhost’ IDENTIFIED BY 'her_password',
- The GRANT statement can create and give privileges to users
- To grant with all rights to the music database
  - GRANT ALL ON music.* TO 'allmusic'@'localhost' IDENTIFIED BY 'the_password';
- ‘localhost’ specifies that the user can connect only from the localhost
- The “GRANT OPTION” privilege allows a user to pass on any privileges she has to other users
  - GRANT GRANT OPTION ON music.* TO 'hugh'@'localhost';

User Accounts, pt. 2

- Privileges can be granted at the global, database, table, or column level
- To create a less privileged user:
  - GRANT ALL ON music.artist TO 'partmusic'@'localhost' IDENTIFIED BY 'the_password';
  - GRANT ALL ON music.album TO 'partmusic'@'localhost';
  - GRANT SELECT (track_id, time) ON music.track TO 'partmusic'@'localhost';
- To see a full list of all available privileges run:
  - SHOW GRANTS [for user@host]
- To see current user:
  - SELECT CURRENT_USER();

User Accounts

- MySQL comes with 2 default users:
  - root - super user who can do anything
  - anonymous - limited privilege user (no username specified)

Best Practices

- Set a strong password for root
- Remove anonymous access
  - DROP USER ''@'localhost';
- Drop the test database
- Remove entirely or strongly restrict remote access
  - DELETE FROM mysql.user WHERE Host <> 'localhost';
- Implement a default deny security policy on databases, tables, and privileges
- Use the OS firewall to secure your server (Windows Firewall or Linux/Unix IP Tables)

Types of Backup

- Raw - backup the database files (requires a shutdown of the database)
- Logical - run dumps of data and scheme
- Snapshots - snapshots of the logical volume
- Binary logs - record changes to the database at different points in time
- Third party backup software - Enterprise backup software with agents for mysql (often allows online backups and recovery)

Incremental/Differential Backups

- A **differential backup** is a backup of everything that has changed since the last full backup
- An **incremental backup** contains everything that has changed since the last backup of any type
- Incremental backups add complexity, risk, and time to recovery
- A **Full backup** is an entire backup of the database
- Very common run a full back on tape and diff/inc backups to disk
- Require the use of a Enterprise Backup solution

Things to Consider

- A production server may take a significant amount of time to flush data and complete the backup
- Running online backup may put load on the server CPU and memory (usually done in the off hours)
- Recovery of the data, in case of failure, may take a significant amount of time

Offline Backup

- Shut down the server
- On Linux/Unix system run
  - tar zcf /tmp/`date +"%Y.%m.%d.%H.%M"`.MySQL_Backup.tgz mysql_data_directory

MySQL Dump

- Utility allows you to backup your database:
  - mysqldump --user=root --password=the_mysql_root_password --result-file=music.sql music
- Useful mysqldump options:
  - —add-drop-table
  - —all-databases, —databases
  - —create-options (Add Engine and Character set to create table)
  - —lock-tables
  - —opt (add-drop-table, locks, disable index updates, create-options, quick)

MySQL Dump Examples

- Full backup of all databases:
  - mysqldump --user=root --password=the_mysql_root_password --result-file=outputfile.sql --all-databases
- Specific database:
  - mysqldump --user=root --password=the_mysql_root_password --result-file=outputfile.sql --databases database_name
- Specific data:
  - mysqldump --user=root --password=the_mysql_root_password --result-file=outputfile.sql database_name table_name where=where_clause
  - mysqldump --user=root --password=the_mysql_root_password --result-file=outputfile.sql --where="artist_name like 'N%'" music artist

Restore Data from Dumps

- To restore data:
  - SOURCE dumpfile.sql
- Or from a command prompt:
  - mysql mysql_options < dumpfile.sql

Binary Log

- An update log contains all the information needed to re-create any changes to the database since the server was started or the logs were flushed
- This feature allows you to always have an up-to-date backup of your database
- You can keep a list of every SQL query that changes data on the server by passing the log-bin option to the MySQL server (mysqld_safe, mysqld-nt.exe, or mysqld).
- To view log contents:
  - mysqlbinlog /usr/lib/mysql/data/eden-bin.000002

Recover Using Binary Log

- To recover from binary logs
  - mysqlbinlog hostname-bin.* | mysql
- To stop and start from certain positions in log
  - mysqlbinlog --database=sakila /var/log/mysql/mysql-bin.000215 --stop-position=352 | mysql -uroot -p
  - mysqlbinlog --database=sakila /var/log/mysql/mysql-bin.000215 --start-position=429 | mysql -uroot -p

Repairing Tables

- To check the integrity of tables:
  - CHECK TABLE music.artist;
- If it fails the check, run a repair:
  - REPAIR TABLE music.artist;

```
+--------------+-------+----------+----------------------------+
| Table        | Op    | Msg_type | Msg_text                   |
+--------------+-------+----------+----------------------------+
| music.artist | check | error    | Checksum for key:  1 doesn't |
|              |       |          |   match checksum for records |
| music.artist | check | error    | Corrupt                    |
+--------------+-------+----------+----------------------------+
2 rows in set (0.00 sec)
 +--------------+--------+----------+----------+
 | Table        | Op     | Msg_type | Msg_text |
 +--------------+--------+----------+----------+
 | music.artist | repair | status   | OK       |
 +--------------+--------+----------+----------+
 1 row in set (0.00 sec)
```

- mysqlcheck allows you to check and repair tables from the command line.
  - mysqlcheck --user=root --password=the_mysql_root_password --repair music

Replication Process

- The master records changes to its data in its **binary log** (binary log events)
- The replica copies the master's binary log events to its **relay log**
- The replica replays the events in the relay log, applying the changes to its own data