# Data Engineering Nanodegree Syllabus

## Contact Info

While going through the program, if you have questions about anything, you can reach us at support@udacity.com. For help from Udacity Mentors and your peers visit the Udacity Classroom.

## Nanodegree Program Info

**Version**: 2.0.0

**Length of Program**: 136 Days*

** This is a self-paced program and the length is an estimation of total hours the average student may take to complete all required coursework, including lecture and project time. Actual hours may vary.*

## Part 1: Welcome to the Nanodegree Program

## Part 2: Data Modeling

Learn to create relational and NoSQL data models to fit the diverse needs of data consumers. Use ETL to build databases in PostgreSQL and Apache Cassandra.

### Project: Data Modeling with Postgres

Students will model user activity data to create a database and ETL pipeline in Postgres for a music streaming app. They will define Fact and Dimension tables and insert data into new tables.

**Supporting Lessons**

| Lesson | Summary |
| --- | --- |
| **Introduction to Data Modeling** | In this lesson, students will learn the basic difference between relational and non-relational databases, and how each type of database fits the diverse needs of data consumers. |
| **Relational Data Models** | In this lesson, students understand the purpose of data modeling, the strengths and weaknesses of relational databases, and create schemas and tables in Postgres |

### Project: Data Modeling with Apache Cassandra

Students will model event data to create a non-relational database and ETL pipeline for a music streaming app. They will define queries and tables for a database built using Apache Cassandra.

#### Supporting Lessons

| Lesson | Summary |
| --- | --- |
| **NoSQL Data Models** | Students will understand when to use non-relational databases based on the data business needs, their strengths and weaknesses, and how to creates tables in Apache Cassandra. |

## Part 3: Cloud Data Warehouses

### Project: Data Warehouse

Students will build an ETL pipeline that extracts data from S3, stages them in Redshift, and transforms data into a set of dimensional tables for their analytics team.

## Part 4: Data Lakes with Spark

### Project: Data Lake

Students will build a data lake and an ETL pipeline in Spark that loads data from S3, processes the data into analytics tables, and loads them back into S3.

### Project: Optimize Your GitHub Profile

Other professionals are collaborating on GitHub and growing their network. Submit your profile to ensure your profile is on par with leaders in your field.

### Project: Data Pipelines

Students continue to work on the music streaming company's data infrastructure by creating and automating a set of data pipelines with Airflow, monitoring and debugging production pipelines

### Project: Improve Your LinkedIn Profile

Find your next job or connect with industry peers on LinkedIn. Ensure your profile attracts relevant leads that will grow your professional network.

## Part 6: Capstone Project

### Project: Data Engineering Capstone Project

In this Capstone project, students will define the scope of the project and the data they will be working with to demonstrate what they have learned in this Data Engineering Nanodegree.

Udacity

Generated Thu Apr 30 02:12:35 PDT 2020

UDACITY

Logout

Return to "Data Engineering Nanodegree" in the classroom

# Data Modeling with Postgres

REVIEW | CODE REVIEW | HISTORY

## Meets Specifications

Greetings Student,
This is awesome. You did a great job including everything which was required. This submission meets all of our expectations and you should be proud of yourself. Continue practicing on these types of projects and other projects of yours and you will be the best in all you do. Remember, practice makes everything.

### Table Creation

**The script, `create_tables.py`, runs in the terminal without errors. The script successfully connects to the Sparkify database, drops any tables if they exist, and creates the tables.**

Good implementation! The script, create_tables.py, perfectly runs in the terminal without errors. Furthermore, the script successfully connects to the Sparkify database, drops any tables if they exist, and creates the tables.

**CREATE statements in `sql_queries.py` specify all columns for each of the five tables with the right data types and conditions.**

Good work with the sql_queries.py. You specified the correct primary key for each table in this script and Not Null where appropriate:+1:

## ETL

**The script, `etl.py`, runs in the terminal without errors. The script connects to the Sparkify database, extracts and processes the `log_data` and `song_data`, and loads data into the five tables.**

**Since this is a subset of the much larger dataset, the solution dataset will only have 1 row with values for value containing ID for both `songid` and `artistid` in the fact table. Those are the only 2 values that the query in the `sql_queries.py` will return that are not-NONE. The rest of the rows will have NONE values for those two variables.**

Awesome work with the ETL script. Your ETL script connects with Sparkify database, extracts both the log and song data and loads them into the five tables as required in the rubric.:+1:

**INSERT statements are correctly written for each table, and handle existing records where appropriate. `songs` and `artists` tables are used to retrieve the correct information for the `songplays` INSERT.**

Excellent work utilizing JOIN clause correctly to retrieve the correct information for thesong_select query and utilizing ON CONFLICT clause to handle existing records where appropriate in your INSERT statements. 👏

## Code Quality

**The README file includes a summary of the project, how to run the Python scripts, and an explanation of the files in the repository. Comments are used effectively and each function has a docstring.**

Impressive README file! It contains all the details and docstrings were effectively used to explain each function. 👏🏼

**Scripts have an intuitive, easy-to-follow structure with code separated into logical functions. Naming for variables and functions follows the PEP8 style guidelines.**

Most of the code is well optimized with an intuitive and easy-to-follow structure that follows PEP8 style guidelines.

DOWNLOAD PROJECT

RETURN TO PATH

UDACITY

Logout

Return to "Data Engineering Nanodegree" in the classroom

# Data Modeling with Cassandra

REVIEW | HISTORY

## Meets Specifications

You've done a **fantastic job** in this project!

**Keep up the good work!**

### ETL Pipeline Processing

| |
|---|
| **Student creates `event_data_new.csv` file.** |
| Nicely done correctly creating the `event_data_new.csv` file |

**Student uses the appropriate datatype within the `CREATE` statement.**

Great job using appropriate datatypes within the `CREATE` statement:

- The `artist_name` and `song_title` do have the `text` / `varchar` datatype
- The `length` does use a `float` / `double` datatype

## Data Modeling

**Student creates the correct Apache Cassandra tables for each of the three queries. The `CREATE TABLE` statement should include the appropriate table.**

Good job creating correct Apache Cassandra tables designed to be used to answer the questions in the prompt by adhering to the one table per query rule of Apache Cassandra

**Student demonstrates good understanding of data modeling by generating correct SELECT statements to generate the result being asked for in the question.**

**The SELECT statement should NOT use `ALLOW FILTERING` to generate the results.**

Well done selecting specific columns from the table without having to call `SELECT *`, which creates unnecessary performance overhead

**Student should use table names that reflect the query and the result it will generate. Table names should include alphanumeric characters and underscores, and table names must start with a letter.**

Perfect! You have provided meaningful table names

**The sequence in which columns appear should reflect how the data is partitioned and the order of the data within the partitions.**

Nicely done following a the order of the COMPOSITE PRIMARY KEY for the `CREATE` and `INSERT` statements. The data is being inserted and retrieved in the same order as how the `COMPOSITE PRIMARY KEY` is set up.

## PRIMARY KEYS

**The combination of the PARTITION KEY alone or with the addition of CLUSTERING COLUMNS should be used appropriately to uniquely identify each row.**

Great job properly setting the combination of the `PARTITION KEY` to uniquely identify each row.

The usage of `song` and `userId` columns for Query 3 does uniquely identify each row.

## Presentation

**The notebooks should include a description of the query the data is modeled after.**

Nicely done adding explanation headers for each query!

**Code should be organized well into the different queries. Any in-line comments that were clearly part of the project instructions should be removed so the notebook provides a professional look.**

Great job removing the in-line comments with TO-DO statements!

Your code is well organized and has a professional look.

DOWNLOAD PROJECT

RETURN TO PATH

UDACITY

Logout

Return to "Data Engineering Nanodegree" in the classroom

# Data Warehouse

REVIEW | CODE REVIEW | HISTORY

## Meets Specifications

Dear student

Congratulations on passing this project with flying colors! You have done a great job at it. All the very best for the next one! 😊 👏🏻

### Table Creation

**The script, create_tables.py, runs in the terminal without errors. The script successfully connects to the Sparkify database, drops any tables if they exist, and creates the tables.**

CREATE statements in sql_queries.py specify all columns for both the songs and logs staging tables with the right data types and conditions.

Great job with the staging tables definition!

CREATE statements in sql_queries.py specify all columns for each of the five tables with the right data types and conditions.

Great job with the final tables definition!

## ETL

The script, `etl.py`, runs in the terminal without errors. The script connects to the Sparkify redshift database, loads `log_data` and `song_data` into staging tables, and transforms them into the five tables.

INSERT statements are correctly written for each table and handles duplicate records where appropriate. Both staging tables are used to insert data into the songplays table.

## Code Quality

The README file includes a summary of the project, how to run the Python scripts, and an explanation of the files in the repository. Comments are used effectively and each function has a docstring.

Scripts have an intuitive, easy-to-follow structure with code separated into logical functions. Naming for variables and functions follows the PEP8 style guidelines.

DOWNLOAD PROJECT

RETURN TO PATH

UDACITY

Logout

Return to "Data Engineering Nanodegree" in the classroom

# Data Lake

REVIEW | CODE REVIEW | HISTORY

## Meets Specifications

Congratulations.

The project is complete and meets all the expectation. Your work on this project meets all the required rubric.
The code structure is clean and organised with proper comments.

All the best for the next project.
Keep learning.

### ETL

**The script, etl.py, runs in the terminal without errors. The script reads song_data and load_data from S3, transforms them to create five different tables, and writes them to partitioned parquet files in table directories on S3.**

Great start.

The script runs in the terminal just fine, There are no errors reported.
At the end of successful execution

- Database is created
- Staging files are rightly loaded from s3.
- All the 5 facts and dimension tables are also correctly updated.

**Each of the five tables are written to parquet files in a separate analytics directory on S3. Each table has its own folder within the directory. Songs table files are partitioned by year and then artist. Time table files are partitioned by year and month. Songplays table files are partitioned by year and month.**

Nicely done.

All the parquet files are correctly setup, Songs and songplays tables are partitioned right.
The files are getting written on the s3 location, I was able to run the code and update the s3 location of my choice.
Great work

**Each table includes the right columns and data types. Duplicates are addressed where appropriate.**

All the tables have the right columns and data types.
You have used `drop_duplicates` to select on distinct values.Dimension tables should not store duplicate values

## Code Quality

**The README file includes a summary of the project, how to run the Python scripts, and an explanation of the files in the repository. Comments are used effectively and each function has a docstring.**

The readme files is very detailed and rightly start with introduction the problem at hand before moving on to Database Schema and how tables are related to each other.
The readme also talks about the files used in the process, purpose they serve and how to run them from terminals.

A good thing about any project is the ability of how much of it can be reproduced by the other users and the detailed readme like yours clearly helps,

Impressive work

**Scripts have an intuitive, easy-to-follow structure with code separated into logical functions. Naming for variables and functions follows the PEP8 style guidelines.**

The code is logically organized and clean. There are no leftover commented code and code is very clean.
There is optimum level of comments and multi line docstring is used for functions description

DOWNLOAD PROJECT

RETURN TO PATH

UDACITY

Logout

Return to "Data Engineering Nanodegree" in the classroom

# Data Pipelines with Airflow

REVIEW | CODE REVIEW | HISTORY

## Meets Specifications

Congratulations on completing the project! You should be very proud of your accomplishments in building an awesome ETL pipeline with Airflow! You have now the knowledge of how to build dynamic and reusable ETL pipelines and how to make sure the data quality meets specifications! 😊👏

## General

| DAG can be browsed without issues in the Airflow UI |
| --- |
| Awesome! DAG can be browsed without issues in the Airflow UI. 👏 |

### Suggestions

To make the DAG even more compact, you could try to use the SubDag operator with the dimension loads and hide the repetitive parts behind that. Depending on your set up, using a subdag operator could make your DAG cleaner.

- Using SubDAGs in Airflow
- Subdag Operator examples

**The dag follows the data flow provided in the instructions, all the tasks have a dependency and DAG begins with a start_execution task and ends with a end_execution task.**

Excellent work! The DAG's graph view all the task have a dependency and DAG begins with a `begin_execution` task and ends with a `stop_execution` task. 👍

## Dag configuration

**DAG contains default_args dict, with the following keys:**

- **Owner**
- **Depends_on_past**
- **Start_date**
- **Retries**
- **Retry_delay**
- **Catchup**

Good job defining the default_args dictionary as required.

Notes
If a dictionary of default_args is passed to a DAG, it will apply them to any of its operators. This makes it easy to apply a common parameter to many operators without having to type it many times.

External Resources

- Backfill and Catchup
- How to prevent airflow from backfilling dag runs?

**The DAG object has default args set**

Nice work! The DAG object contains a binding to the default args. 👏🏼

**The DAG should be scheduled to run once an hour**

Good work scheduling the DAG to run once an hour as required.

## Staging the data

**There is a task that to stages data from S3 to Redshift. (Runs a Redshift copy statement)**

Nice work the stage operator. I suggest using a template field that would allow to load timestamped files from S3 based on the execution time and run backfills.
but your current implementation is also great

```
template_fields = ("s3_key",)
```

Check the detail here https://airflow.readthedocs.io/en/latest/howto/custom-operator.html

**Instead of running a static SQL statement to stage the data, the task uses params to generate the copy statement dynamically**

Good job dynamically generating the copy statement using params as opposed to static SQL statements.

**The operator contains logging in different steps of the execution**

`logging.info` shows the status of staging execution. Nice work! 👍

**The SQL statements are executed by using a Airflow hook**

Good job connecting to the database via an Airflow hook. 👏

## Loading dimensions and facts

**Dimensions are loaded with on the LoadDimension operator**

There is a separate functional operator for dimensions LoadDimensionOperator.

**Facts are loaded with on the LoadFact operator**

There is a separate functional operator for facts LoadFactOperator as well. 👏

**Instead of running a static SQL statement to stage the data, the task uses params to generate the copy statement dynamically**

The parameters were used to add some dynamic functionality to the operators and allow to run various SQL statements instead of hardcoded SQL statement. Well done! ⭐

**The DAG allows to switch between append-only and delete-load functionality**

Great job that you have added a truncate param to allow the switch.
However, instead of deleting the table, it would be better to use the truncate.

**The TRUNCATE statement can provide the following advantages over a DELETE statement:**

- The TRUNCATE statement can ignore delete triggers
- The TRUNCATE statement can perform an immediate commit
- The TRUNCATE statement can keep storage allocated for the table

## Data Quality Checks

**Data quality check is done with correct operator**

Great work! The operator that runs a check on the fact or dimension table(s) after the data has been loaded is DataQualityOperator. The data quality operator looks awesome. It is simple but still allows you to do import checks on the data and catch the possible data quality issues as soon as possible.

**The DAG either fails or retries n times**

The dag fails if the check is not passed, by raising ValueError. Well done!

**Operator uses params to get the tests and the results, tests are not hard coded to the operator**

DOWNLOAD PROJECT

RETURN TO PATH

VERIFIED CERTIFICATE OF COMPLETION

March 26, 2020

UDACITY

# Ahmed Mohamed

Has successfully completed the

## Data Engineering Nanodegree

NANODEGREE PROGRAM

Sebastian Thrun
Founder, Udacity

Udacity has confirmed the participation of this individual in this program.
Confirm program completion at confirm.udacity.com/777D74U