Table of Contents

Introduction

Welcome, Coursera data researchers! This guide will help you get started with your research data exports by providing an introduction to the many tables. The guide will be updated to reflect major changes in the data export. Please refer to the Changelog to view the list of latest updates.

As a PDF file, this guide will be included in every data export. It will also be accessible online at the link: https://www.gitbook.com/book/coursera/data-exports/

Please request an account by emailing us at data-support@coursera.org and participate in commenting in this documentation.

Many of the links in this guide refer to related articles in our Coursera Partner Help Center for more details about our platform's features.

We aim to further your researching needs, whether there is a lack of clarity in how to use the data you have or there are other data you wish to see in the export. Please feel free to reach out to us if you have any questions or comments.

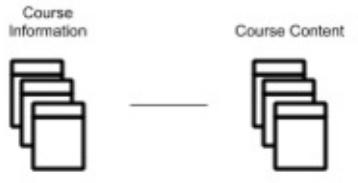
First-Time Readers

If you are a first-time reader, we strongly suggest you cover these sections to help you get started:

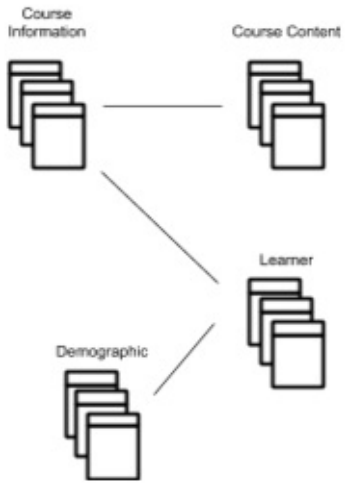
The Data Journey

Below is a journey of how actions from instructors and learners populate the data tables as you see today

- The instructor creates a course.
    - An instructor will create a new "draft" course, which populates Course Information tables, such as the **courses** table.
    - As the instructor creates modules, lessons, quizzes, etc, this populates Course Content tables, such as the **course_modules** table.
    - When ready, an instructor works with Coursera to set course pre-enrollment, launch, and sessions dates, which updates new information in the **courses** table and the **on_demand_sessions** table.



- Learners enroll in the course.
    - When Coursera users preview, pre-enroll, or enroll in a course, this populates the Learner tables, such as the **course_memberships** and **users** tables.
    - Some users voluntarily answer our demographic questions, which populates the Demographic tables.

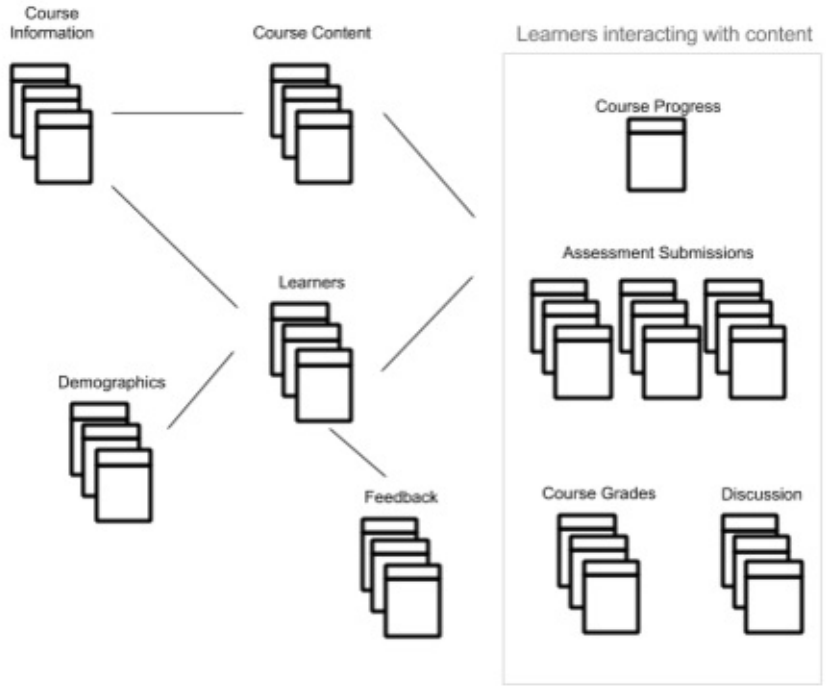


- Class starts and learners interact with course content.
    - As learners start and complete watching lecture videos or taking assessments, as

well as progressing on types of other course content, this populates the **course_progress** table. For detailed actions, like answering incorrectly on a quiz question, this populates Assessments tables, such as the **assessment_responses** table.

- As learners receive grades and complete the course, this populates the Course Grades tables, such as the **course_grades** table.
- As learners post and vote on the course forums, this populates the Discussions tables, such as the **discussion_answers** table.
- As learners provide feedback on course content and the entire course, this populates Feedback tables, such as the **feedback_course_ratings** table.

Below shows the major relationships between tables groups, with minor connections omitted.



Files Included

The data export .zip file includes the following:

- **guide.pdf**: This is the guide for Coursera Data Researchers, which also lives online at this Gitbook link.
- **readme.html:** This file contains documentation for all tables and table columns.
- **CSV** files (e.g. `course_grades.csv`): Comma-separated value files contain the data header and data values for each table.
- **HTML** files (e.g. `course_grades.html`): HTML files contain the documentation for each table and a `CREATE TABLE` script for PostgreSQL-compliant databases. In March 2016, we have made our first attempt at adding some `PRIMARY_KEY` and `FOREIGN_KEY` info in the tables.
- **SQL** files: Scripts for loading data export into a Postgres database. See Loading Data Exports for example usage. (Note: `PRIMARY_KEY` and `FOREIGN_KEY` constraints are currently not enforced.)
    - `setup.sql`: Creates tables in a Postgres database for data tables included in export.
    - `load.sql`: Loads data from CSV files into appropriate tables.

Loading Data Exports

Coursera collects and stores data for analytics, and we use a data warehouse product, Amazon Redshift, for most of our data processing. The data we provide as research exports are generated by processing and unloading subsets of this data. We want this data to be as useful as possible and recognize that the volume, variety, and formatting may pose challenges. To simplify this process, we have provided tools to create a portable database in a platform independent docker container.

Creating a database using `courseraresearchexports` tools

We provide tools for creating a Postgres database in a docker container via the courseraresearchexports project. To get started, follow the installation instructions on the project page and install docker for your platform. Please see the Command Line Access section for instructions on requesting research exports.

Create a containerized database

1. Request a research export programatically or through the web interface.
2. Set up a database using the `courseraresearchexports containers create` command in one of two ways:
    - **From an `$EXPORT_REQUEST_ID`**: Find the `$EXPORT_REQUEST_ID` using `courseraresearchexports jobs get_all`. The following command will download the completed export, set up a database in a container, and load the export data into the database:

      ```
      courseraresearchexports containers create --export_request_id $EXPORT_REQUEST
      _ID
      ```

      By default, the name of your database and container will be the course slug or partner short name associated with the export request. You may customize these names with the `--container_name` and `--database_name` flags.
    - **From a downloaded export request**: Unzip the export data archive onto your local machine and run the following command:

```
courseraresearchexports containers create --export_data_folder /path/to/expor
t/data/folder
```

After the container and database are initialized, the database is running and open to connections.

3. Check that your container is running:

   ```
   courseraresearchexports containers list
   ```

   This displays the `$CONTAINER_NAME`, `$DATABASE_NAME`, `$HOST_PORT` and `$HOST_IP` for each container. Use the `start`, `stop`, and `remove` commands to manage your containers.

4. Using docker, you can connect to the database via a Postgres shell with the following command:

   ```
   docker run -it --rm --link $CONTAINER_NAME postgres:9.5 psql -h $CONTAINER_NAME -d
    $DATABASE_NAME -U postgres
   ```

   Alternatively, if you have Postgres installed locally:

   ```
   psql -p $HOST_PORT -h $HOST_IP -d $DATABASE_NAME -U postgres
   ```

Alternative methods for loading data

Parsing the CSV

Each CSV file contains a header with column names followed by zero or more rows of data. If a table contains zero rows, this means that there is no available data. Rows are separated by a newline character (\n). Within each row, columns are enclosed by double-quotes (") and separated by comma characters (,). String column data may contain double-quotes and backslash characters (\). Both of these will be escaped using a backslash character.

As an example with high parsing complexity, here is a very short table with two columns:

```
number,bool,null,string,json
1,t,,"a string","{\"key1\":1,\"key2\":\"\\\"a nested quotation\\\"\"}"
```

The last column includes JSON-formatted data with escaped double-quote and backslash characters. You should take special care to interpret escaped characters correctly when importing data into other programs.

Using Excel

Excel is not an ideal tool for analysis for a few reasons. Firstly, some CSVs may be too large for Excel. Secondly, Excel's automated CSV parsing does not work well with the encoding described above. Finally, our data is highly normalized, so most research would require a significant number of VLOOKUPs.

Nevertheless, you may find success in loading some CSVs into Excel, which can be used to create very basic statistics using, say, the pivot table feature. However, this is likely going to give you similar insights as what you can find in our Course and Specialization Dashboards.

Using a Relational Database

A relational database management system (RDBMS) is ideal for storing and querying the data with SQL. Examples include:

RDBMS	Costs	Environment	Comments
Amazon Redshift	not free	runs in the cloud	what Coursera uses
PostgreSQL	free	runs locally	closest clone to Redshift
MySQL	free	runs locally	another popular RDBMS

If you are unfamiliar with the three examples, we recommend you try out PostgreSQL on your local machine.

PostgreSQL has a community of online resources on how to download, install, and use it. Here are some resources:

- http://postgresguide.com/
- http://www.postgresql.org/docs/9.5/static/tutorial.html
- http://www.postgresqltutorial.com/

The generic efforts of installing any RDBMS and importing data are:

1. Download a RDBMS installation file/package for your OS.
2. Install. The installation software will:
    i. install the database server
    ii. install a SQL client
    iii. prompt you to set up an admin account
3. Run the database server, which will run in background.

4. Run the SQL client program to connect to the database server with your admin account.
5. Write and run the SQL to create a new database and then use that database.
6. For a Postgres database, the included script `setup.sql` will setup all the tables included in the course data export.

   ```
   psql -e -U [USERNAME] -d [DATABASE_NAME] -f setup.sql
   ```

   where the `DATABASE_NAME` was specified during database creation and `USERNAME` is your account (`postgres` for the default admin account).
7. To import into PostgreSQL, run the `load.sql` script in the folder where the CSV files are stored:

   ```
   psql -e -U [USERNAME] -d [DATABASE_NAME] -f load.sql
   ```

8. Write SQL to analyze data on the loaded table, for example:

   ```
   SELECT *
   FROM courses
   ```

We encourage MOOC data researchers to share tools and techniques with us and to the community. Jasper Ginn, of Leiden University, maintains a thoroughly documented, open-source script to load Coursera export data into a PostgreSQL database.

Python, R, and Other Languages and Tools

Any other language or tool that can handle loading CSVs can be used to import and to "query" the data for research.

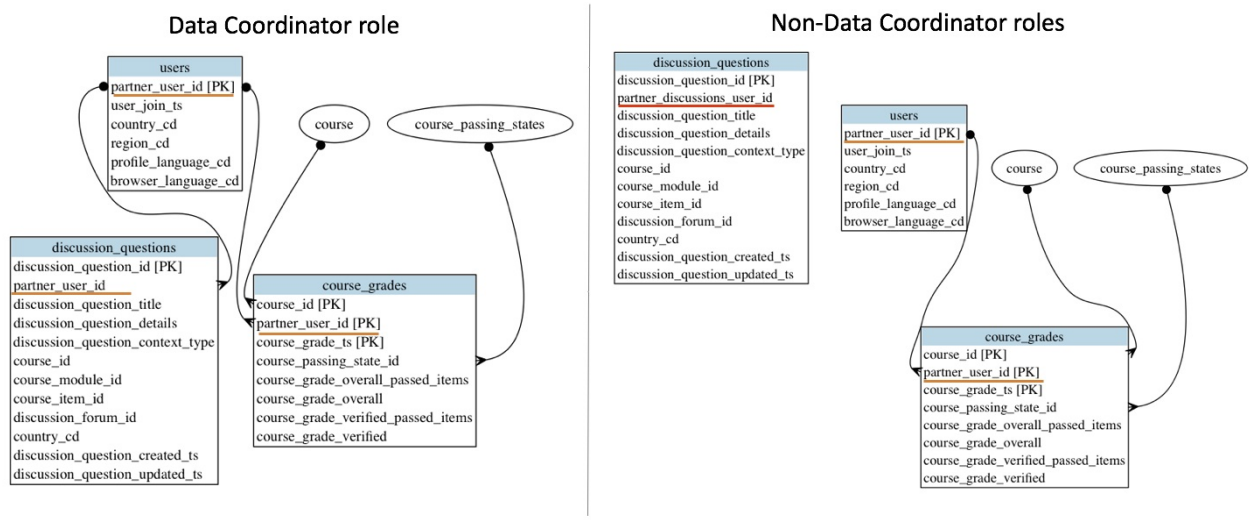
Below is how to configure python to do a successful load of the data export CSV:

```
import csv
csv.register_dialect(
    'coursera-postgres-format',
    delimiter=',',
    doublequote=False,
    escapechar='\\',
    lineterminator='\n',
    quotechar='"')
with open('path/to/courses.csv', 'r') as f:
    reader = csv.reader(f, dialect='coursera-postgres-format')
    header = next(reader)
    rows = list(reader)
```

Coursera welcomes our research partners to share best practices and code examples in these languages and tools outside of RDBMS and SQL.

ID Columns

Per Coursera's Data Sharing Policy and Research Policy, there are two different data export anonymity levels based on the exporter's role. Here is a quick illustration:



Exporters with the Data Coordinator role are allowed to receive data exports where tables with user information will contain the column named *[partner]_user_id*. Therefore, these exporters can identify the same user across the multiple table domains and must follow the PII guidelines in the agreed upon data policy. Data Coordinators can contact us to request this specific type of research data exports.

For exporters without the Data Coordinator role, each table domain uses a separate user ID to distinguish between the different types of learner-generated data. The learner data cannot be joined across domains. This is intended to reduce the scope and impact of accidental PII inclusion and to protect exporters and Coursera when PII data is not required to advance data research.

For example, the **course_grades** table will contain a *[partner]_user_id* column, which will purposely not connectable with the column *[partner]_discussions_user_id* in the **discussion_questions** table. Otherwise, if one provides PII in the discussion forums, a researcher might be able to associate that learner to a particular grade.

If you are not a Data Coordinator, your table domains and user ID columns will be as follows:

ID Column	Domain
[partner]_user_id	in most tables (**users**, **course_progress**, etc.)
[partner]_discussion_user_id	in the **Discussions** tables
[partner]_demographics_user_id	in the **Demographic** tables
[partner]_feedback_user_id	in the **Feedback** tables
[partner]_assessments_user_id	in the **Assessments** tables
[partner]_programming_assignments_id	in the **Assessments** tables
[partner]_peer_assignments_user_id	in the **Assessments** tables

For all exports, ID columns are consistent for a learner across all courses; this allows partners to connect learners' grades and progress across courses.

There is also an identifier column in the format of, *[course_slug]_user_id*, that is found in the table **[partner]_course_user_ids**. This table is very useful for instructors to administer surveys to learners, outlined in this Partner Help Center article on External Surveys. Survey responses can be traced to learners' Coursera accounts by including a special tag in the survey URL, such as `http://www.surveyvendor.com/mysurvey?user=%HASHED_USER_ID%`. This `%HASHED_USER_ID%` variable is equal to the *[course_slug]_user_id*.

Command Line Access

Background

Coursera's research exports are now available behind a command line interface, with a programmatic API available to users who wish to write scripts to periodically request and retrieve research export jobs.

Access is authorized using OAuth2. To simplify this process, we have created the courserareserachexports project. This project provide tools to manage, download, and load research export data into a Postgres database in a docker container.

If you wish to explore the API using another language, please refer to the Research Export API documentation.

Export Job Lifecycle

Coursera collects data for analytics and stores it in a data warehouse product, Amazon Redshift. The data in our warehouse is updated daily at around 7AM UTC. A newly created export job will pull data from our data warehouse and dump the results to Amazon S3, a web filesystem. When the export job is complete, a pre-signed download link is generated which can be used to download the exported data.

Setup

Please install the client following the directions on the project homepage. To get started, use the included authentication client, courseraoauth2client, to gain access to our APIs:

```
courseraoauth2client config authorize --app manage_research_exports
```

The tool will prompt you to authorize your Coursera account for the research export application by going to a Coursera URL. Alternatively, you can use your own OAuth2 client using the details here. To directly make requests to our APIs, format your requests according the API documentation.

Documentation

GET the status of export requests

After successful authorization, run the following in a command line shell:

```
courseraresearchexports jobs get_all
```

You should see a list of exports that you have previously generated. To examine an export request in more details, use the following command with one of the request ids:

```
courseraresearchexports jobs get $EXPORT_REQUEST_ID
```

DOWNLOAD a completed export

To download `SUCCESSFUL` table or clickstream export jobs, use the `download` command:

```
courseraresearchexports jobs download $EXPORT_REQUEST_ID \
  --dest /path/to/save/
```

REQUEST a research export

Coursera research data exports are separated into two categories: **tables** and **clickstream**.

Because creating exports is a computationally intensive process, we only allow **one** request per hour for any given scope. Please keep in mind that our data warehouse is updated once daily, so requests more frequent than that will generate the same data.

tables

Tables exports contain information stored while administering a course, such as assessment, learner progress and demographic data. To request this type of data for a given course, use the `request tables` command:

```
courseraresearchexports jobs request tables --course_slug $COURSE_SLUG \
  --purpose "purpose of request"
```

Your `$COURSE_SLUG` is the part after `/learn` in the course url. (e.g. `machine-learning` for `https://www.coursera.org/learn/machine-learning`). This will return a `$EXPORT_REQUEST_ID`, which you can use with the `get` command to retrieve the status of your request as shown above.

To limit your request to certain types of tables, specify the `schemas` desired:

```
courseraresearchexports jobs request tables --course_slug $COURSE_SLUG \
  --purpose "purpose of request" --schemas demographics
```

Which will restrict the export to tables related to `demographics`. Please view the help message, `courseraresearchexports jobs request tables -h`, and Data Tables for a list of available schemas. For partner-wide or group-wide exports, specify the `--partner_short_name` or `--group_id` instead of a `--course_slug`.

According to our data sharing policy, user_ids are anonymized depending on the exporter's role. Data coordinators can request that user_ids are linked such that users are identifiable between different domains and table schemas. For this type of export add the `--user_id_hashing` flag:

```
courseraresearchexports jobs request tables --course_slug $COURSE_SLUG \
  --purpose "purpose of request" --user_id_hashing linked
```

clickstream

Clickstream events are user interactions with your course such as accesses made to video and webpage resources. To protect user anonymity, only data coordinators have access to these exports. To request a clickstream data request:

```
courseraresearchexports jobs request clickstream --course_slug $COURSE_SLUG \
    --interval 2016-09-01 2016-09-03 --purpose "purpose of request"
```

Since clickstream exports contain large amounts of data and are time consuming to generate, we export the events for a given day into a single file. Use the `--interval` flag to request certain days (Sept 1 - Sept 3 in the above example).

Clickstream exports also have the `clickstream_download_links` command, which allows you to generate download links for clickstream exports in bulk.

```
courseraresearchexports jobs clickstream_download_links --course_slug $COURSE_SLUG --i
nterval 2016-01-01 2016-04-01
```

Similarly partner level exports can be requested with `--partner_short_name`

Research Export API

Below we document Coursera's research exports APIs, should you choose to not use the courseraresearchexports tool. Usage of these APIs require OAuth2 authorization, which we support via the courseraoauth2client library.

Research Export API Documentation

To access our research exports APIs, format your requests as documented in this section with the proper OAuth2 access token. Using the `courseraoauth2client` project, the access token part is handled as follows:

```
REQUEST_JSON = {...}
url = "https://coursera.org/api/onDemandExports.v2"
auth = oauth2.build_oauth2(app='manage_research_exports').build_authorizer()
resp = requests.post(url, auth=auth, headers = headers, json=json.dumps(REQUEST_JSON))
```

CREATE new export request (table)

HTTP POST to `https://coursera.org/api/onDemandExports.v2` with the following body:

```
{
   "scope":{
      "typeName":"courseContext",
      "definition":{
         "courseId":"kbmiwPT-EeSW1SIAC3oCCQ"
      }
   },
   "exportType":"RESEARCH_WITH_SCHEMAS",
   "schemaNames":[
      "course_grades"
   ],
   "anonymityLevel":"HASHED_IDS_NO_PII",
   "statementOfPurpose":"Test"
}
```

`scope` : An identifier that tells us what rows in our database we should export. We currently support `courseContext` , `partnerContext` , `groupContext` for the type names, which corresponds to learners within a course, and a partner, and a group, respectively.

`courseId` : The unique id assigned to your course by Coursera. You may find it by hitting `https://www.coursera.org/api/onDemandCourses.v1?q=slug&slug=YOUR_SLUG` where `YOUR_SLUG` is the part after `/learn` in the course url. (E.g. for `https://www.coursera.org/learn/machine-learning` , the slug is `machine-learning` .)

To get ids for `partnerContext` and `groupContext` please contact us at data-support@coursera.org.

`exportType` : Should always be `RESEARCH_WITH_SCHEMAS` . We are working on providing clickstream exports by the end of Q3 2016.

`schemaNames` is any combination of `demographics, users, course_membership, course_progress, feedback, assessments, course_grades, peer_assignments, discussions, programming_assignments, course_content` , corresponding to the types of data exported.

`anonymityLevel` : One of `HASHED_IDS_NO_PII` or `HASHED_IDS_WITH_ISOLATED_UGC_NO_PII` . If you are a data coordinator, you may request `HASHED_IDS_NO_PII` . Otherwise, you must request `HASHED_IDS_WITH_ISOLATED_UGC_NO_PII` . The difference between the two settings are explained here, where `HASHED_IDS_NO_PII` correspond to data coordinator exports, and `HASHED_IDS_WITH_ISOLATED_UGC_NO_PII` corresponds to different user id columns in every domain.

A successful `CREATE` request results in the following response

```
{
  "elements": [
    {
      "id": "cmqe3ihsuh",
      ...
    }
  ],
  "paging": null,
  "linked": null
}
```

The important part of this response is the `id` field, which can then be used to track whether the request finishes.

Rate Limiting

Because creating exports is a heavyweight process, we only allow 1 `CREATE` request per hour for any given scope. For our systems to perform, we also limit `CREATE` requests to up to 15 per hour over all scopes.

It's worth noting that the data is updated daily, and so there is no benefit to requesting the latest data more frequently than daily.

CREATE new export request (Clickstream)

HTTP POST to `https://coursera.org/api/onDemandExports.v2` .

Example body:

```
{
   "scope": {
     "typeName": "partnerContext",
     "definition": {
       "partnerId": {
         "maestroId": $YOUR_PARTNER_ID
       }
     }
   },
   "exportType":"RESEARCH_EVENTING",
   "anonymityLevel":"HASHED_IDS_NO_PII",
   "statementOfPurpose":"Test",
   "interval": {"start": "2016-08-01", "end": "2016-08-01"},
   "ignoreExisting": true
}
```

`scope` : Here we use `partnerContext` for demonstration purposes. It needs an extra layer of nesting with the `maestroId` field, as compared to the `courseContext` shown in the table export example. See the Finding Course/Partner Ids section to find out how to find your partner id.

`exportType` : `RESEARCH_EVENTING` for clickstream exports.

`anonymityLevel` : Must be `HASHED_IDS_NO_PII` , clickstream exports will only be allowed for data coordinators.

`interval` : The interval for which we export clickstream data, with `start` and `end` in `yyyy-MM-DD` format.

`ignoreExisting` : (Default false) If set to true, we will recompute clickstream data all dates in interval. Otherwise, if the selected interval consists of days that we have previously computed clickstream data for, we skip these days.

Make note of the returned `export_id` , and call the `GET export by id` API to check on the status of your export.

GET export by id

HTTP GET to `https://coursera.org/api/onDemandExports.v2/EXPORT_JOB_ID`

`EXPORT_JOB_ID` is an id returned by the `CREATE` request.

Example response:


```
{
  "paging": null,
  "elements": [
    {
      "status": "SUCCESSFUL",
      "exportType": "RESEARCH_WITH_SCHEMAS",
      "downloadLink": "https://coursera-course-exports.s3.amazonaws.com/export/results
/ijvdo41cmj/REDACTED",
      "statementOfPurpose": "Test end to end",
      "schemaNames": [
        "demographics",
        "users",
        "course_membership",
        "course_progress",
        "feedback",
        "assessments",
        "course_grades",
        "peer_assignments",
        "discussions",
        "programming_assignments",
        "course_content"
      ],
      "partnerId": {
        "maestroId": REDACTED
      },
      "scope": {
        "typeName": "courseContext",
        "definition": {
          "courseId": "REDACTED"
        }
      },
      "anonymityLevel": "HASHED_IDS_NO_PII",
      "id": "REDACTED",
      "metadata": {
        "snapshotAt": 1467763200000,
        "startedAt": 1467765249099,
        "createdAt": 1467765247826,
        "createdBy": 12206410,
        "completedAt": 1467766221218
      }
    }
  ],
  "linked": null
}
```


The fields returned contains all the fields passed to `CREATE`, as well as extra `metadata` field containing information about when the job was done. Also included as the `downloadLink` key is a pre-signed S3 link that allows you to download the result of this export. This pre-

signed link is generated on every request, and expires after 1 hour.

FIND all my past exports

HTTP GET to `https://coursera.org/api/onDemandExports.v2?q=my`

Response is of the same format as `GET`, except the `elements` array will have multiple export job bodies.

Generate Clickstream Exports Download Links

HTTP POST to `https://www.coursera.org/api/clickstreamExportsDownload.v1?action=generateLinks&scope=$SCOPE` Where `$SCOPE` is one of `courseContext~$COURSE_ID` and `partnerContext~$PARTNER_ID`.

This endpoint generates download links for previously computed exports.

Optional parameters: `startDate, endDate` in `yyyy-MM-DD` format. For example, `&startDate=2016-07-01&endDate=2016-07-01`. This generates download links within `startDate` and `endDate`, inclusive of both boundaries.

Example Python Request:

```
url = 'https://www.coursera.org/api/clickstreamExportsDownload.v1?action=generateLinks
&scope=courseContext~$YOUR_COURSE_ID'
auth = oauth2.build_oauth2(app=app).build_authorizer()
response = requests.post(url, auth=auth)
print json.dumps(response.json(), indent=2)
```

Response:

```
[
  "https://coursera-course-exports.s3.amazonaws.com/export/scopedUnloadJobs/courseCont
ext~$YOUR_COURSE_ID/events/2016-05-01/video-2016-05-01.csv.gz?x-amz-security-token=RED
ACTED&AWSAccessKeyId=REDACTED&Expires=1471447109&Signature=REDACTED",
  ...
]
```

The response is a JSON array, each element containing a pre-signed S3 download link for a file containing clickstream data, named according to the date and domain. For example, `video-2016-08-01.csv.gz`.

Finding Course/Partner Ids:

For course id, use the `https://www.coursera.org/api/onDemandCourses.v1?q=slug&slug=YOUR_SLUG` API where `YOUR_SLUG` is the part after `/learn` in the course url. (E.g. for `https://www.coursera.org/learn/machine-learning`, the slug is `machine-learning`.)

For partner id, use `https://www.coursera.org/api/partners.v1` and search for your institution's name. The integer `id` field corresond to the partner id.

OAuth2 Authentication Details

If you'd like to access our APIs in languages other than Python, please use the following OAuth2 details and go through the OAuth2 flow with a client in the language of your choosing.

```
client_id = sDHC8Nfp-b1XMbzZx8Wa4w
client_secret = pgD4adDd7lm-ksfG7UazUA
scopes = view_profile manage_research_exports
```

OAuth2 Details

To access our APIs programmatically, we need to ensure that the user is authenticated (they have the correct username/password combination), and authorized (they have access to research exports). We use OAuth2 to handle this need: when you use OAuth2's redirect flow, we enforce that you log into Coursera, and capture an access token. This access token can then be redeemed for programmatic access.

Clickstream Data

We record users' interactions on Coursera in what is called clickstream data. Each recorded interaction is called an **event**. Please refer to here for background information about events.

Because clickstream data contains information about a user's interactions, it is easy to deanonymize user behavior across the Coursera platform for a single course. As a result, clickstream data is only available to data coordinators presently.

Clickstream data is exposed in separate **domains**. A domain is a coherent group of events. The domains we currently export are:

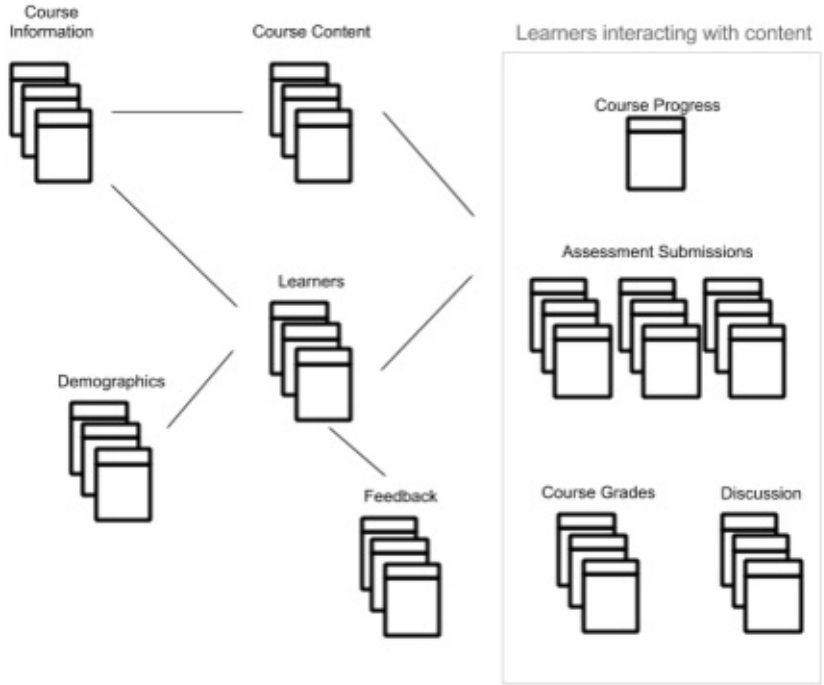
- Video: Interactions with lecture videos (e.g start, stop, pause, change subtitles, heartbeats)
- Access: Accessing course description page, course material, and anything else under the url `https://coursera.org/learn/YOUR_COURSE_SLUG`.

Data Tables

We will break out the 75+ tables included in your exports into a few key groups.

- **Course Information**: relates to basic information about the course, including a course's name, when its sessions ran, etc.
- **Course Content**: refers to the materials of the course, including modules, lessons, items, etc.
- **Course Progress**: describes learners' interaction with course content.
- **Assessments**: provides in-depth details of interactions with assessments.
- **Course Grades**: details the learners' grades and passing states within a course.
- **Discussions**: contains forums, forum posts, and vote information.
- **Feedback**: contains information regarding user ratings to course content and courses.
- **Learners**: describes learners' info, like when/where the user joined Coursera.
- **Demographics**: contains demographic data based on user surveys.

The major relationships between tables groups, with minor connections omitted, are as follows:

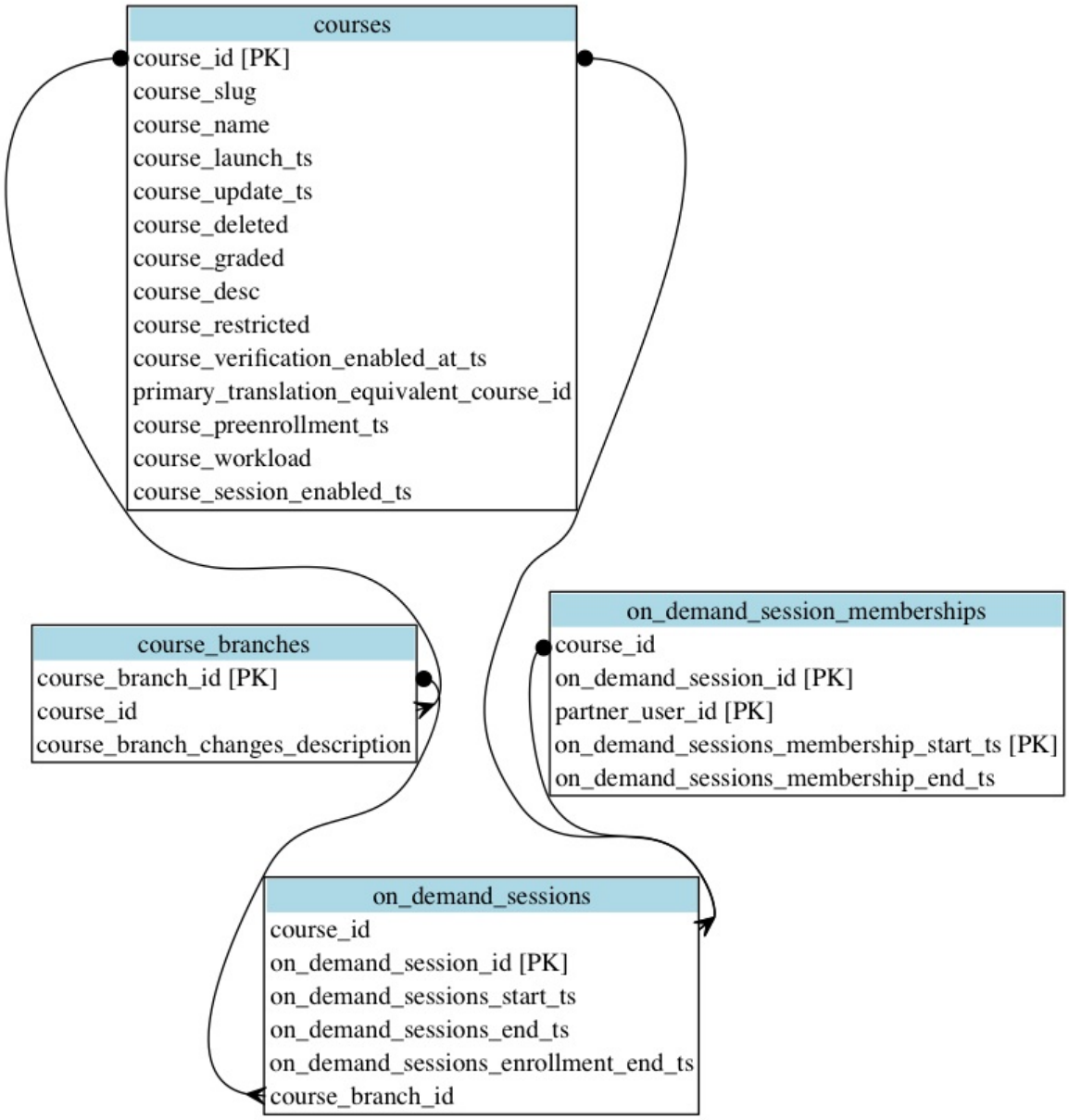


Course Information Tables

Coursera starts tracking data as soon as an instructor builds a draft course. Course data appears in the **courses** table. When ready, an instructor works with Coursera to set the course pre-enrollment and launch dates. This updates new information in the **courses** table.

Most courses on Coursera are offered in a session (cohort) format. Each session has a start and end date, and learners in the same session work through the course together according to a weekly schedule of suggested deadlines. Sessions are scheduled on a regular cadence (e.g., one start date per month). Each session runs on one course version of content. For more info, please refer to our Partner Help Center articles on Course Sessions and Course Versioning.

Details about a course's sessions are populated in the **on_demand_sessions** table which includes their start date, end date, and course version. The list of all authored course versions is populated in the **course_branches** table. When learners enroll in sessions, these details are populated in the **on_demand_session_memberships** table.



Are there more learners enrolled in my newer (versus older) sessions?

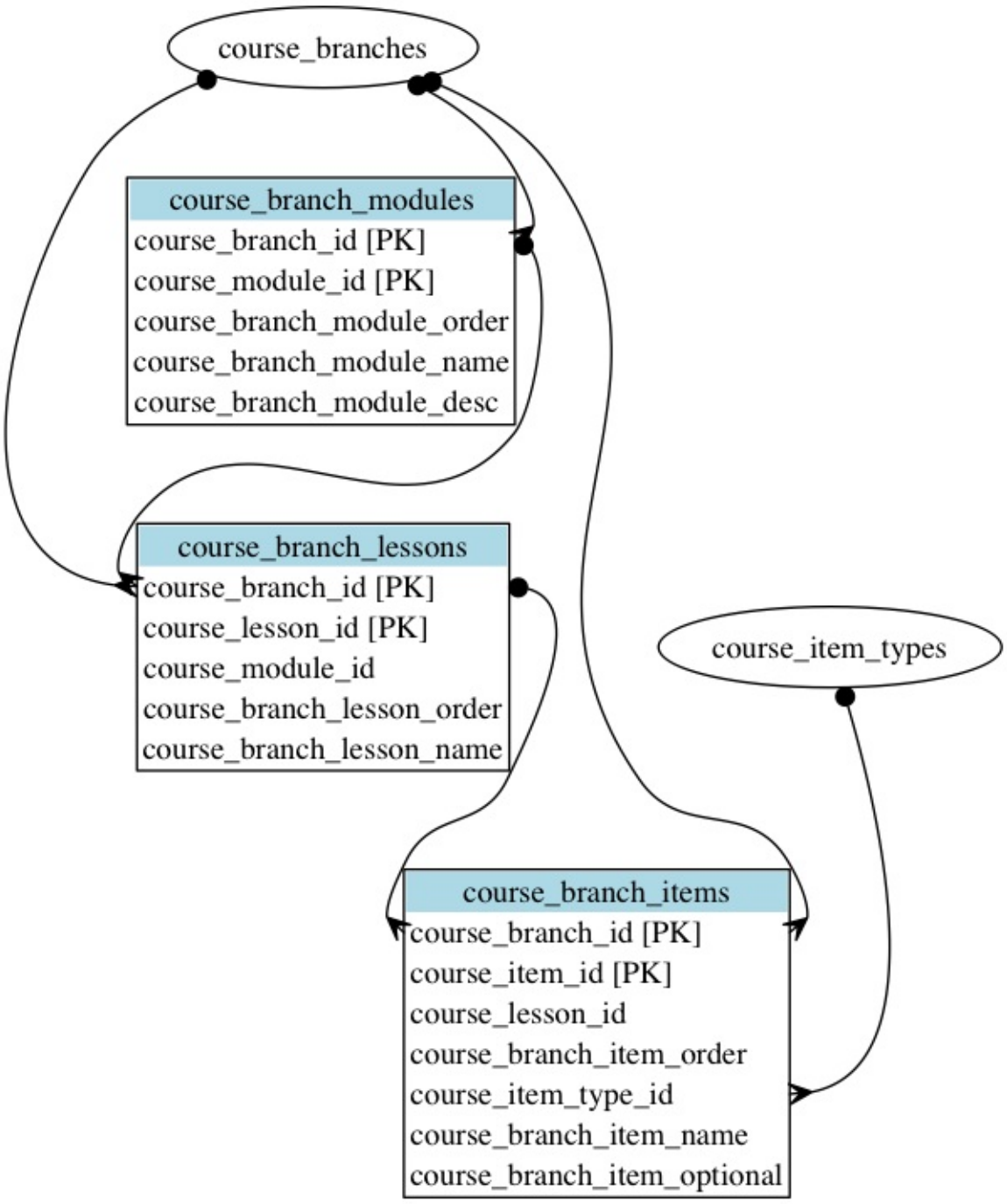
```
SELECT
    on_demand_session_id
    ,on_demand_sessions_start_ts::DATE AS session_start_date
    ,COUNT(DISTINCT [partner]_user_id) num_learners
FROM on_demand_session_memberships
JOIN on_demand_sessions
    USING (on_demand_session_id)
GROUP BY 1,2
ORDER BY 2;
```

Course Content Tables

In July 2016, this section was heavily modified due to the Course Versioning feature. For those who are familiar the previous version of this section, the changelog article on Course Versioning will provide more details.

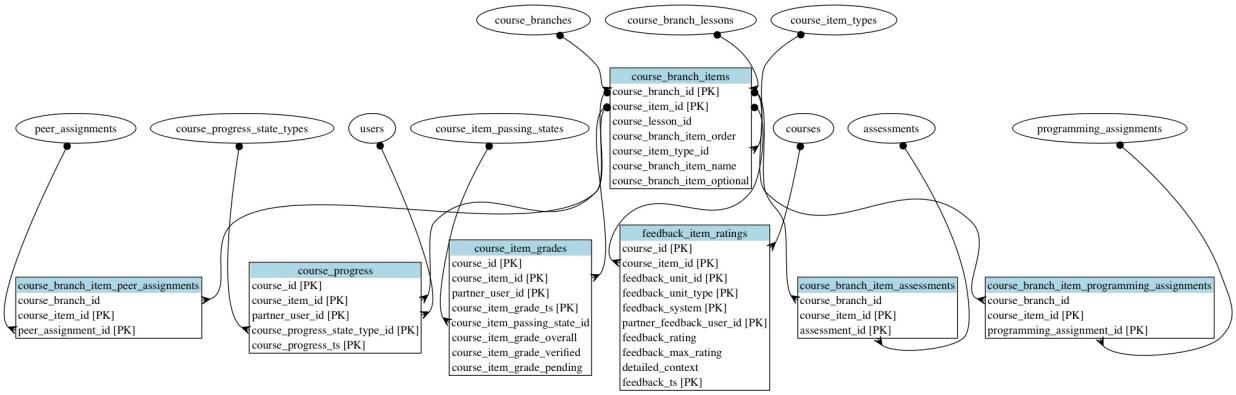
Course Content tables are populated when instructors create course content such as modules, lessons, lectures, quizzes, etc.

Every course begins with an initial course version which is labeled as the original version. At any time, the instructor can use the Course Versioning feature to create additional course versions, or “branches” as we call them in the data model. Each course version is structured as a hierarchy of modules, lessons, and items, which are recorded in the three tables: **course_branch_modules**, **course_branch_lessons**, and **course_branch_items**.



Many instructors will tweak, add, or delete content over the lifetime of a course, and these tables will be updated to reflect those changes. Data exports contain the current course offering at the time of export.

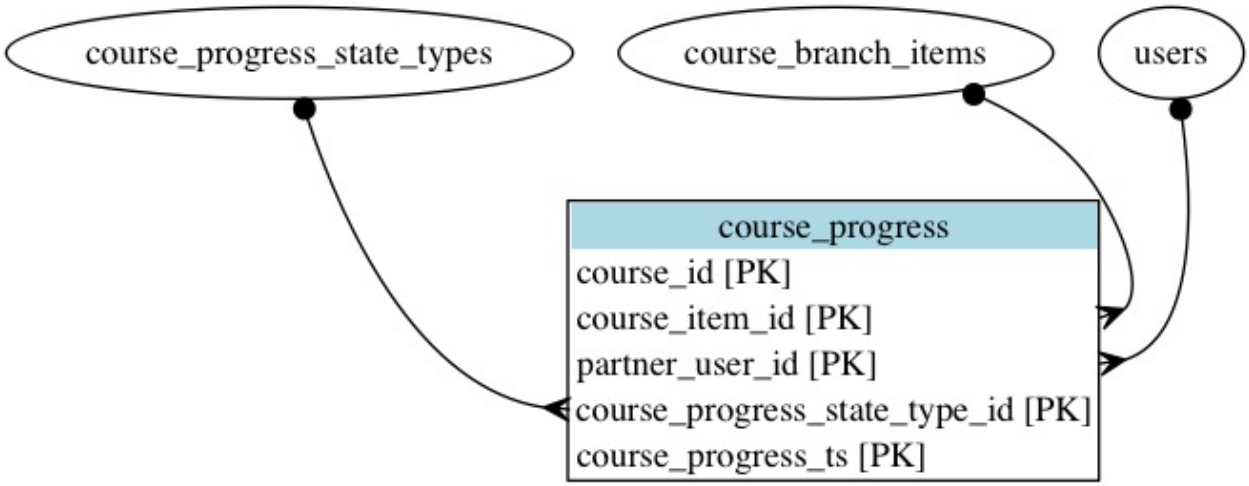
The ids of each course content, such as `course_module_id`, will exist in other related table groups. Of the three course content levels, course items have the most relationships with other tables, such as: Assessments, Course Progress, Course Grades, and Feedback.



> click image to enlarge

Course Progress Tables

When a learner interacts with a course item, a row is populated in the **course_progress** table. Each row shows a unique interaction with a user and course item, the time of that interaction, and a progress state of either "started" or "completed".



Each course item type has a different meaning for the progress states.

- For viewing lectures in a browser, playing the video for a few seconds "starts" the lecture, while playing the last few seconds of the video "completes" the lecture.
- For graded quizzes, starting or submitting a non-passing submission "starts" the quiz; submitting a passing submission "completes" the quiz. If the learner passed the completion on the first attempt, there would be no record of a "start", since the learner went straight to a "completed" state.

Related to the topic of instructor's tweaks/additions/deletions of course item contents in the Course Content Table chapter, the **course_progress** table will contain all activity to all `course_item_id` 's that existed in the course across all course versions.

The **course_progress** table does not include when a user downloads a video lecture. Please see the Frequently Asked Questions chapter for more exceptions.

SQL Example: How many learners started the first graded programming assignment in each original version of my courses?

```
WITH all_programming_assignments AS (
    SELECT
        course_id
```

```
        ,course_item_id
        -- window function returns true if it is the first grading programming for eac
h course
        ,1 = ROW_NUMBER() OVER (
            PARTITION BY course_id
            -- sort across all modules and lessons to determine first programming assi
gnment for each course
            ORDER BY course_branch_module_order, course_branch_lesson_order, course_br
anch_item_order
        ) AS is_first_graded_programming
    FROM course_branch_items
    -- join to identify the branch of original version
    JOIN course_branches
        USING (course_branch_id)
    -- join to allow the search for 'graded programming'
    JOIN course_item_types
        USING (course_item_type_id)
    -- join to get lessons-to-modules
    JOIN course_branch_lessons
        USING (course_lesson_id)
    -- join to get module's course_module_order
    JOIN course_branch_modules
        USING (course_module_id)
    WHERE
        course_item_type_desc = 'graded programming'
        -- the original version has the same value between these two columns
        AND course_branch_items.course_branch_id = course_id
)

,first_programming_assignment AS (
    SELECT
        course_id
        ,course_item_id
    FROM all_programming_assignments
    WHERE is_first_graded_programming
)

-- count number of learners that started first programming assignment
SELECT
    course_id
    ,COUNT(DISTINCT [partner]_user_id) AS num_learners
FROM course_progress
-- this inner join only keeps progress on desired course_item_id's
JOIN first_programming_assignment
    USING (course_id, course_item_id)
JOIN course_progress_state_types
    USING (course_progress_state_type_id)
WHERE course_progress_state_type_desc = 'started'
GROUP BY 1;
```

Assessment Tables

For assessment-based course items, a wealth of 30+ tables can provide in-depth details across:

- Quizzes: **assessment_*** tables
- Peer Assignments: **peer_*** tables
- Programming Assignments: **programming_*** tables

The three groups above are similar in structure. They each contain:

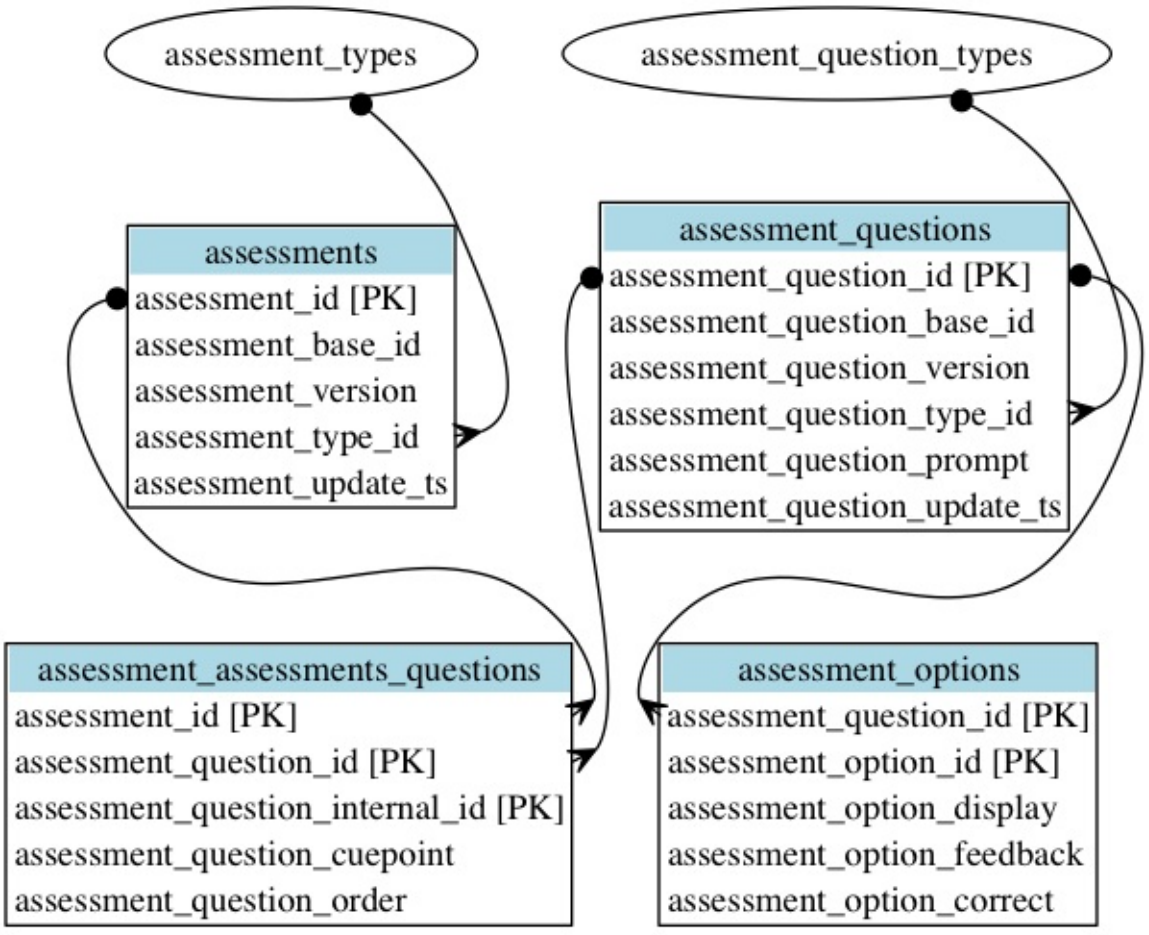
- A set of tables describe how the assessments are configured to test the learners. Each assessment has details such as when it was created, whether it is graded or ungraded, and what is the passing criteria. It also contains the questions and possible responses.
- A set of tables describe the learners' responses. For most assessments, these tables record the answers that learners submitted. Peer Assignments tables also includes the options chosen when learners was reviewing their peer's submissions.
- A combination of specific columns or additional tables has the populated values of assessment score.
- Each group has one table that allows the connection between these assessment tables to other data export tables, e.g. **course_branch_item_assessments**

Quiz Tables

Quizzes are automatically-graded assignments used to test learner knowledge in a course. A quiz, referred to as an "assessment" in the data export tables, could be a multiple-choice exam or an in-video quiz. The **assessment_types** and **assessment_question_types** tables describe the type of quizzes and the type of questions, respectively.

The **assessments** table contains information for each quiz. The `assessment_base_id` field uniquely identifies the quiz within a course, and the `assessment_id` field is a combination of the `assessment_base_id` field and the `assessment_version`.

The **assessment_questions** table contains information for each question. Here, the `assessment_question_base_id` field uniquely identifies a question within a course, and the `assessment_question_id field` is a combination of the `assessment_question_base_id` field and the `assessment_question_version`.

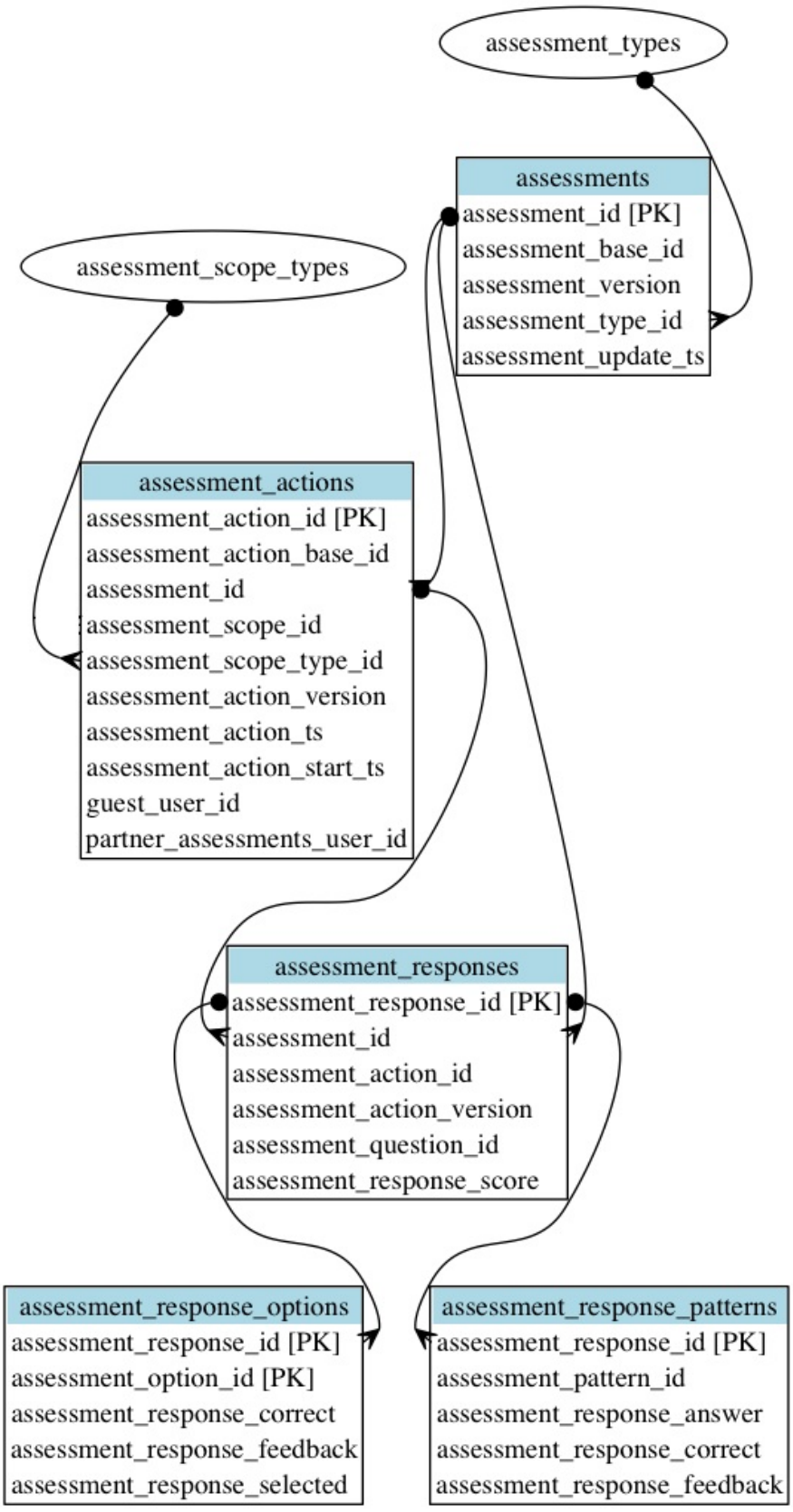


There are seven tables that correspond to different types of quiz questions, as summarized in the following table.

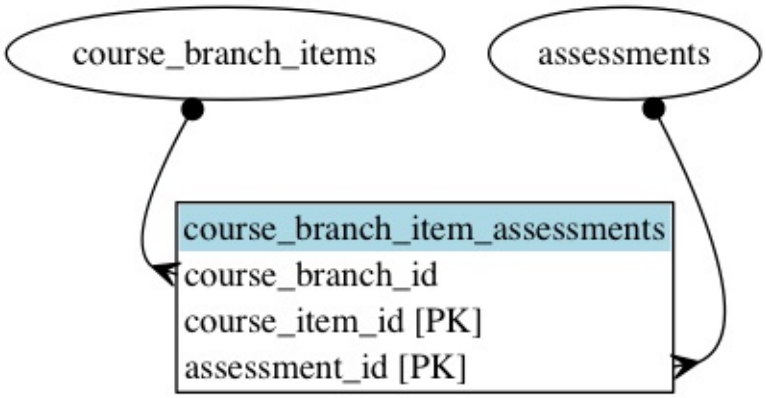
Question Type	Table Name
graded checkbox	**assessment_checkbox_questions**
ungraded checkbox	**assessment_checkbox_reflect_questions**
math expression	**assessment_math_expression_questions**
single numeric entry	**assessment_single_numeric_questions**
graded multiple-choice	**assessment_mcq_questions**
ungraded multiple-choice	**assessment_mcq_reflect_questions**
text matching	**assessment_text_exact_match_questions**

The **assessment_options** table describes the response options of each quiz question, how they are shown to learners, and whether they are correct.

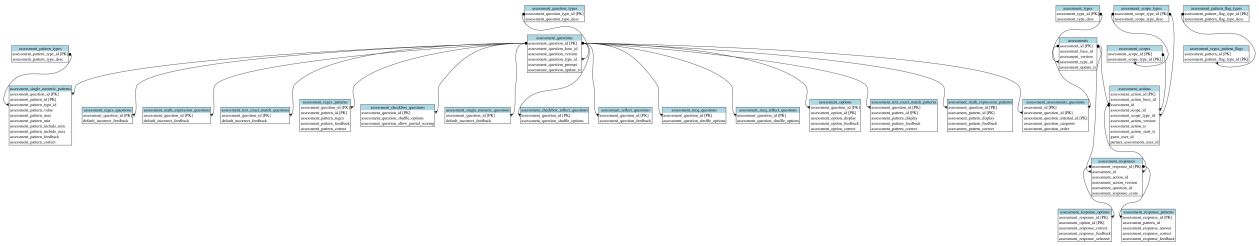
When learners attempt a quiz, their actions will be recorded in the **assessment_actions** table, and after they submit a response to the quiz, their responses will be recorded in the **assessment_responses** table. Learners can attempt a quiz as many times as they like. However, some may require them to wait a certain amount of time before resubmitting, and the highest-scoring attempt will be saved and will count as their final score for the quiz. The **assessment_response_options** table shows the option chosen by the learner, whether it is correct, and the feedback provided to the learner.



The `assessment_id` column can be joined with tables outside of the assessment tables by using the **course_branch_item_assessments** table.



Below is our best attempt to build one ER diagram for all tables with the **assessments_*** prefix.



> click image to enlarge

SQL Example: Which choices did learners select in the in-video quizzes?

The SQL below illustrates how the multiple assessment tables relate across assessment questions, actions, and responses. It returns the list of each in-video quiz, each quiz question, the lecture it is associated with, what/when the user responded, and the response score.

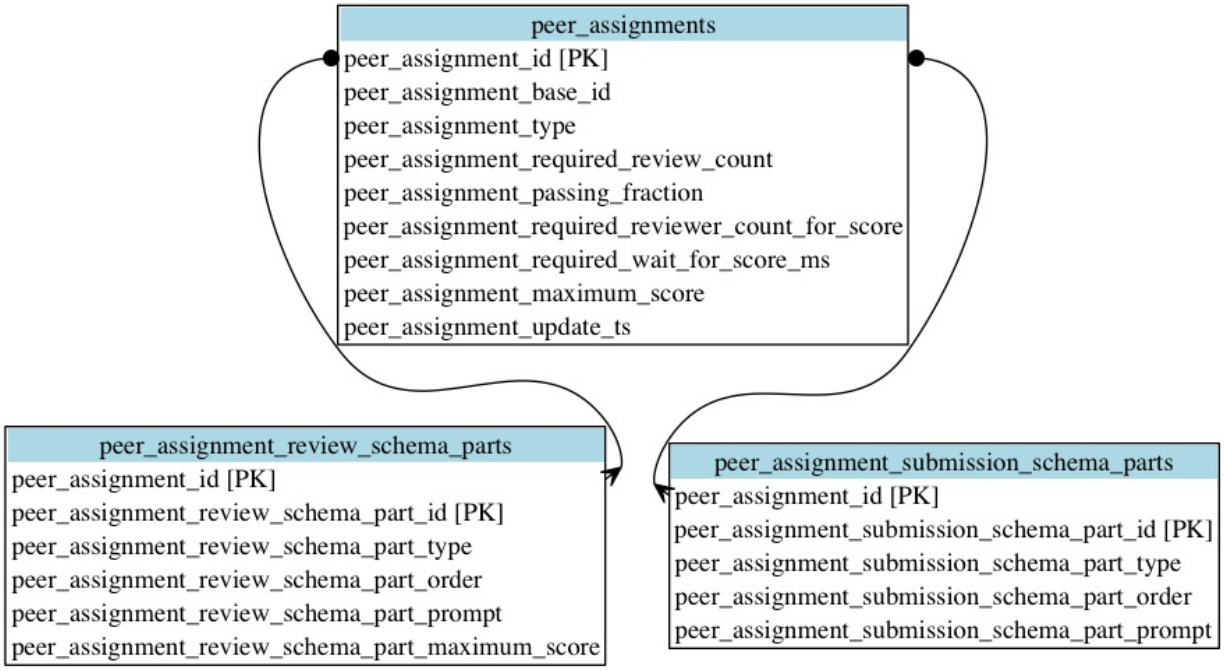
```sql
SELECT
    ci.course_branch_item_name AS lecture_title
   ,aq.assessment_question_prompt
   ,aa.[partner]_assessments_user_id
   ,ao.assessment_option_display
   ,aa.assessment_action_ts
   ,ar.assessment_action_version
   ,ar.assessment_response_score
FROM course_branch_item_assessments cia
-- add assessments table to get the assessment_type_id column
JOIN assessments
    USING (assessment_id)
-- to allow filter for only in-video quizzes
JOIN assessment_types
    USING (assessment_type_id)
-- to get lecture title
JOIN course_branch_items ci
    USING (course_branch_id, course_item_id)
-- to get assessment_action_id and assessment action timestamp
JOIN assessment_actions aa
    USING (assessment_id)
-- to get assessment_action_version, response_options, and assessment_response_score
JOIN assessment_responses ar
    USING (assessment_id, assessment_action_id)
-- next two joins are to get assessment_option_display
JOIN assessment_response_options aro
    USING (assessment_response_id)
JOIN assessment_options ao
    USING (assessment_question_id, assessment_option_id)
-- to get assessment_question_prompt
JOIN assessment_questions aq
    USING (assessment_question_id)
WHERE assessment_type_desc = 'in video'
-- only return what the user selected
    AND assessment_response_selected = TRUE;
```

Peer Review Assignments Tables

Peer review assignments allow learners to grade assignments of other learners. It is composed of instructions, prompts, and rubric parts, each of which is described in Partner Help Center article on Peer Review Assignments.

Peer Review Assignments

The **peer_assignments** table contains details on how the instructor has chosen the type (e.g. graded or ungraded), scoring criterion (e.g. minimum number of reviews), and other configurations about the assignment.



We can use the `peer_assignment_id` column to get more information about the assignment from the **peer_assignments** table. The query and table below describe the different configurations for this specific peer review assignment.

```
SELECT *
FROM peer_assignments
WHERE peer_assignment_id = 'H8ED3xtoEeWshyIAC7OtLQ@19'
```

peer_assignment_id	peer_assignment_base_id	peer_assignment_type	peer_assignment_required_review_count	peer_assignment_passing_fraction	peer_assignment_required_reviewer_count_for_score	peer_assignment_required_wait_for_score_ms	peer_assignment_maximum_score	peer_assignment_update_ts
H8ED3xtoEeWshyIAC7OtLQ@19	H8ED3xtoEeWshyIAC7OtLQ	phased	3	0.75	3	86400000	19	4/6/2016 4:34:23

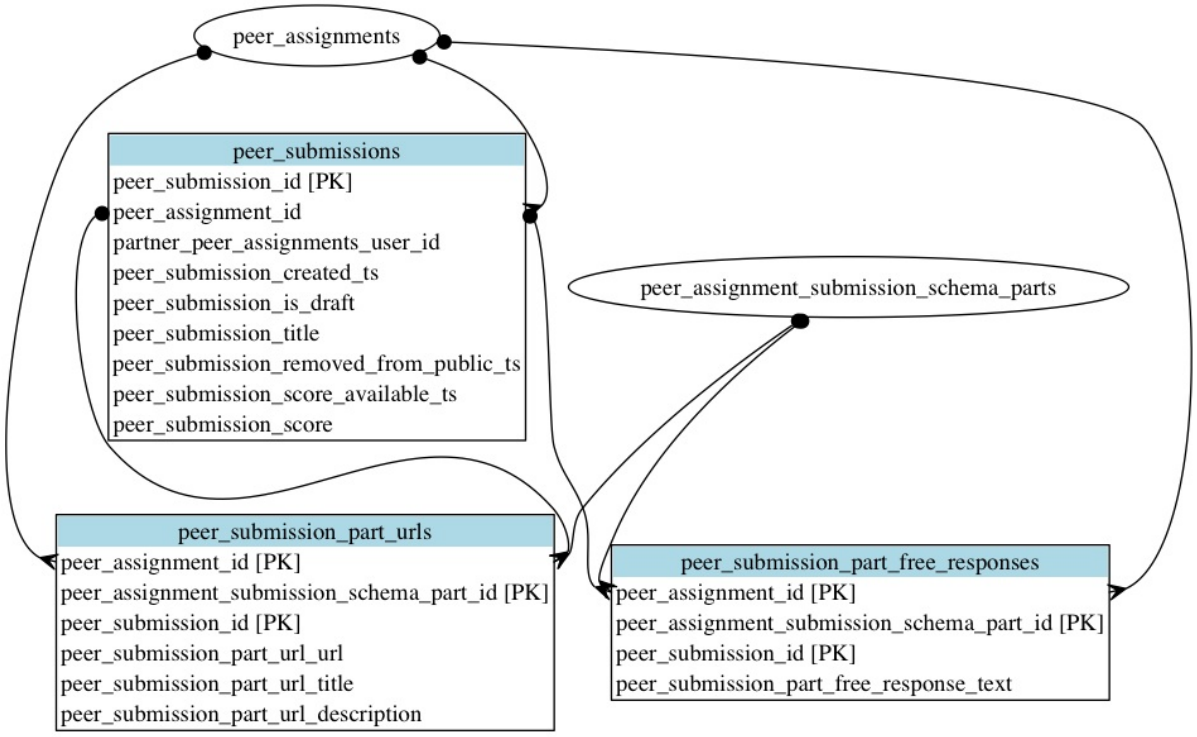
The columns in this table are:

- `peer_assignment_type` : This column is usually either "phased" or "standard". In phased assignments, learners have to submit a response to the assignment before they can evaluate the work of other learners or view the full rubric. Phased peer review assignments also impose deadlines for submission and evaluation. In standard assignments, peer reviews are not required for the assignment, and learners can evaluate their peers even before submitting their own work. This is the default type for ungraded peer review assignments.
- `peer_assignment_required_review_count` : This column states the number of assignments a learner has to review to receive a score for their assignment. If the assignment is optional, this value will be set, but is not really meaningful as learners are not obligated to complete the assignment to pass the course.
- `peer_assignment_passing_fraction` : This column is only set for phased peer review assignments. It is a value between 0 and 1 that states the percentage of overall points learners must achieve to pass the assignment.
- `peer_assignment_required_reviewer_count_for_score` : This column states the number of reviews an assignment must receive in order to get a score.
- `peer_assignment_required_wait_for_score_ms` : This column states the amount of time in milliseconds that must elapse since a peer review assignment has been submitted before it can have a score. This only exists for graded peer review assignments.
- `peer_assignment_maximum_score` : This column records the maximum score that can be awarded from all parts of the rubric.
- `peer_assignment_update_ts` : This column indicates the timestamp when the assignment version was created.

In rare cases, you may also find the `peer_assignment_type` column to be populated with the value "closed" or "graded". These are similar to the phased and standard types respectively, but do not impose deadlines. These were unique to the on-demand course mode, in which learners could begin at any time. The on-demand mode was deprecated in the summer of 2016.

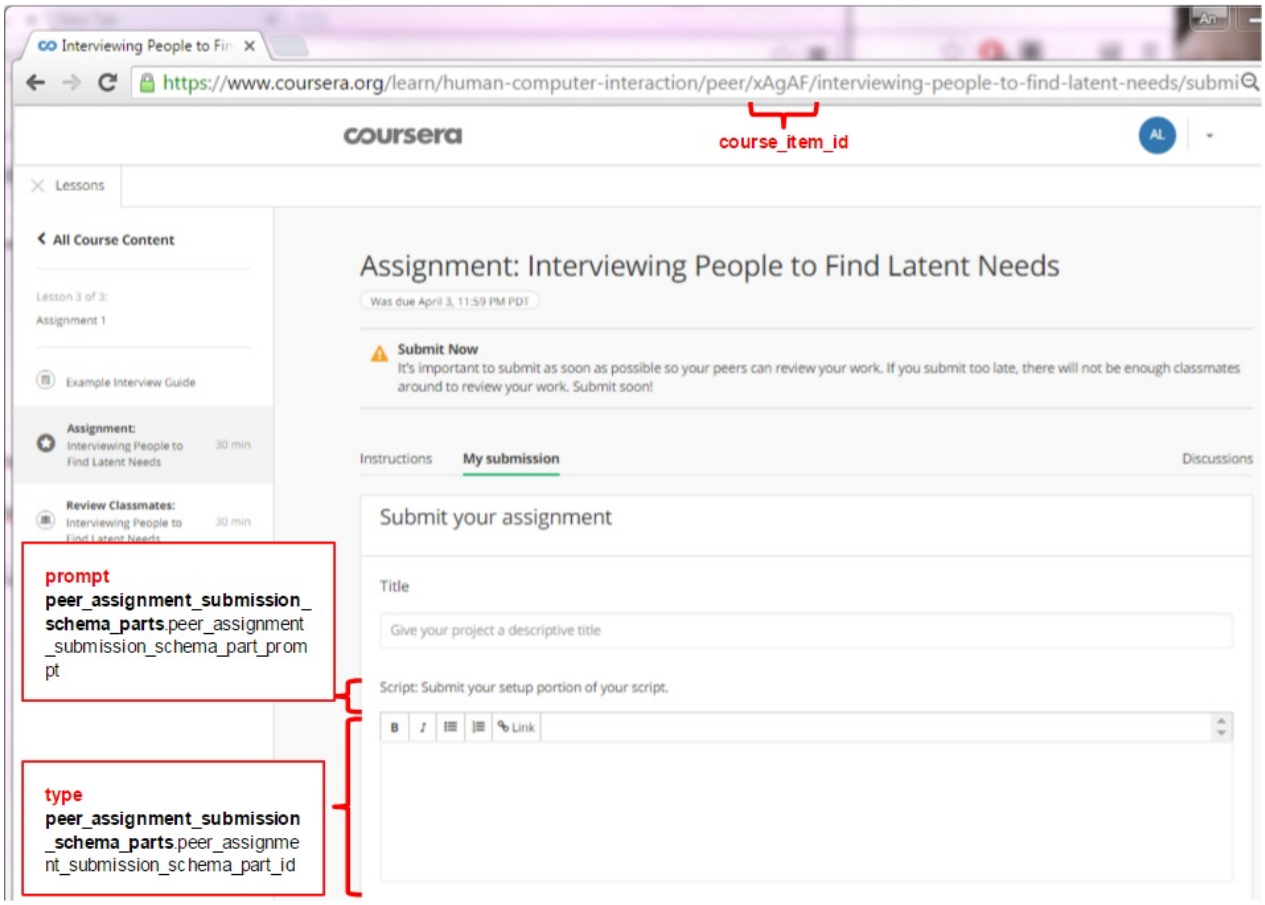
Peer Submission Parts

The **peer_submissions** table is populated when learners take peer review assignments. They create draft submissions when they click the "save" button. Each draft that a user saves is stored as a separate submission. The answers that learners submit are recorded in the **peer_submission_part_free_responses** table for Rich Text and plain text type submissions and **peer_submission_part_urls** table for URL type and file upload type submissions.



When a peer review assignment is created, the instructor configures one or more questions to create submission parts, found in the **peer_assignment_submission_schema_parts** table. Each submission part consists of:

- A prompt: This refers to the text of the question.
- A type: This indicates the format in which learners will submit their answer for this submission part. The four types are "richText", "url", "fileUpload" and "plainText".



The screenshot above shows a sample submission part with the text “Script: Submit your setup portion of your script.” and uses Rich Text type. To find this in the data exports, we first have to find the `course_item_id` for this peer review assignment, which can be located from the URL, either viewing as a learner or in the authoring environment as shown in the screenshot above. Then use the **course_branch_item_peer_assignments** table to show the `peer_assignment_id`.

```
SELECT *
FROM course_branch_item_peer_assignments
WHERE course_item_id = 'xAgAF'
```

The results are:

course_id	course_item_id	peer_assignment_id
DDBg7AIXEeWT	xAgAF	H8ED3xtoEeWshylAC7OtLQ@1
DDBg7AIXEeWT	xAgAF	H8ED3xtoEeWshylAC7OtLQ@10
DDBg7AIXEeWT	xAgAF	H8ED3xtoEeWshylAC7OtLQ@11
DDBg7AIXEeWT	xAgAF	H8ED3xtoEeWshylAC7OtLQ@12
DDBg7AIXEeWT	xAgAF	H8ED3xtoEeWshylAC7OtLQ@13
DDBg7AIXEeWT	xAgAF	H8ED3xtoEeWshylAC7OtLQ@14
DDBg7AIXEeWT	xAgAF	H8ED3xtoEeWshylAC7OtLQ@15
DDBg7AIXEeWT	xAgAF	H8ED3xtoEeWshylAC7OtLQ@16
DDBg7AIXEeWT	xAgAF	H8ED3xtoEeWshylAC7OtLQ@17
DDBg7AIXEeWT	xAgAF	H8ED3xtoEeWshylAC7OtLQ@18
DDBg7AIXEeWT	xAgAF	H8ED3xtoEeWshylAC7OtLQ@19
DDBg7AIXEeWT	xAgAF	H8ED3xtoEeWshylAC7OtLQ@2
DDBg7AIXEeWT	xAgAF	H8ED3xtoEeWshylAC7OtLQ@3
DDBg7AIXEeWT	xAgAF	H8ED3xtoEeWshylAC7OtLQ@4
DDBg7AIXEeWT	xAgAF	H8ED3xtoEeWshylAC7OtLQ@5
DDBg7AIXEeWT	xAgAF	H8ED3xtoEeWshylAC7OtLQ@6
DDBg7AIXEeWT	xAgAF	H8ED3xtoEeWshylAC7OtLQ@7
DDBg7AIXEeWT	xAgAF	H8ED3xtoEeWshylAC7OtLQ@8
DDBg7AIXEeWT	xAgAF	H8ED3xtoEeWshylAC7OtLQ@9

If an instructor has modified and saved an assignment multiple times, this query will return multiple rows. In this example, the instructor has saved this assignment 19 times and each revision has a unique ID. The latest value is usually the one that was ready for course launch and this is what learners will ultimately take. This may not be true if the instructor modifies the peer review assignment after course launch.

Querying the **peer_assignment_submission_schema_parts** table can now show us the list of this assignment's submission part prompts and submission part types:

```
SELECT *
FROM peer_assignment_submission_schema_parts
WHERE peer_assignment_id = 'H8ED3xtoEeWshyIAC7OtLQ@19'
ORDER BY peer_assignment_submission_schema_part_order
```

peer_assignment_id	peer_assignment_submission_schema_part_id	peer_assignment_submission_schema_part_type	peer_assignment_submission_schema_part_order	peer_assignment_submission_schema_part_prompt
H8ED3xtoEeWsl	2332432432	richText	0	<co-content><text>Script: Submit your setup portion of your script.</text></co-content>
H8ED3xtoEeWsl	2000	richText	1	<co-content><text>Script: Submit your introduction & participant background portion of your script.</
H8ED3xtoEeWsl	2010	richText	2	<co-content><text>Script: Submit the main interview questions portion of your script.</text></co-content>
H8ED3xtoEeWsl	3000	richText	3	<co-content><text>Script: Submit the "In Closing" portion of your script.</text></co-content>
H8ED3xtoEeWsl	3050	richText	4	<co-content><text>English: If your interview is in a different language, please submit a transcript of the in
H8ED3xtoEeWsl	4000	url	5	<co-content><text>Submit the Soundcloud URL of your interview recording here. Remember to make sur
H8ED3xtoEeWsl	5000	richText	6	<co-content><text>Short Reflection: Submit a short reflection on your experience.</text></co-content>

The second column, `peer_assignment_submission_schema_part_id`, contains the unique identifier for each submission part. Depending on when your assignment was created, the values in the `peer_assignment_submission_schema_part_id` column (and similarly, the

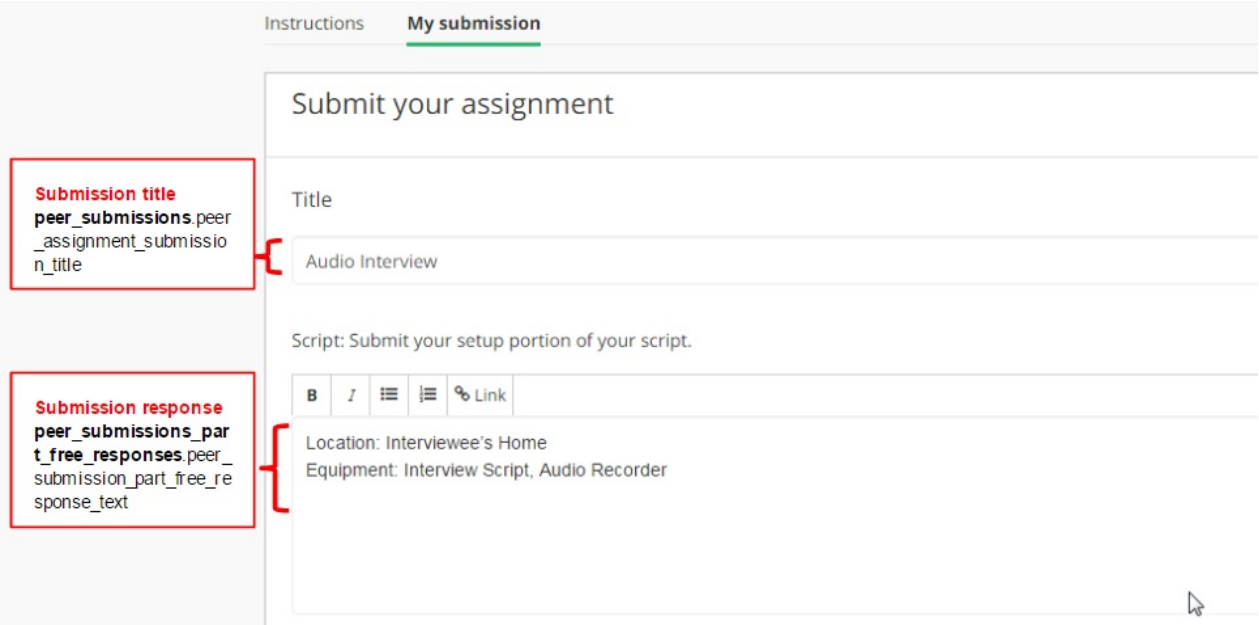
`peer_assignment_review_schema_part_id` column in the **peer_assignment_review_schema_parts** table) may have a different pattern. Historically, these IDs were hand-generated as a sequence of numbers by a member of Coursera's partner support team. Those assignments created by on-platform authoring tools generated these IDs with hashed strings of letters and numbers."2332432432" is one example, another might be "Gs2leZhgSq2JbKdLmO_Tnw" - not shown in the image). You should treat the * `_part_id` columns as VARCHAR type, regardless of when your assignment was created.

The third column, `peer_assignment_submission_schema_part_type`, shows that this assignment uses two of the four types, "richText" and "url".

In the last column, `peer_assignment_submission_schema_part_prompt`, the first row contains the same text that is shown in the user interface, surrounded by Coursera tags.

Peer Submissions from Learners

Here is an example of a learner and his submitted assignment.



When a learner clicks the "submit" button to submit their work for review, this creates a completely new entry in addition to the previous drafts. The `peer_submission_is_draft` column distinguishes saved drafts from the final submission.

The query below on the **peer_submissions** table will find all non-draft submissions with a title filter. For the title "Audio Interview", there was only one result.

```
SELECT *
FROM peer_submissions
WHERE peer_assignment_id = 'H8ED3xtoEeWshyIAC7OtLQ@19'
    AND NOT peer_submission_is_draft
    AND peer_submission_title = 'Audio Interview'
```

peer_submission_id	peer_assignment_id	user_id	peer_submission_created_ts	peer_submission_is_draft	peer_submission_title	peer_submission_removed_from_public_ts	peer_submission_score_available_ts	peer_submission_score
9eFL8A0BEeaAVhLVpQtKaw	H8ED3xtoEeWshyIAC7OtLQ@19	########	4/28/2016 5:27:55	FALSE	Audio Interview	[NULL]	5/4/2016 21:55:21	19

If this learner had multiple non-draft submissions, then the seventh column, `peer_submission_removed_from_public_ts`, provides evidence of which ones were replaced with a new submission. When this value is NULL, it can be interpreted as the latest non-draft submission.

Knowing the `peer_submission_id` in the first column and the column `peer_assignment_submission_schema_part_id` from the section before, we can query to find the learner's final answer to this submission prompt part in the **peer_submission_part_free_responses** table, since the response type was "richText".

```
SELECT *
FROM peer_submission_part_free_responses
WHERE peer_assignment_id = 'H8ED3xtoEeWshyIAC7OtLQ@19'
    AND peer_submission_id = '9eFL8A0BEeaAVhLVpQtKaw'
    AND peer_assignment_submission_schema_part_id = '2332432432'
```

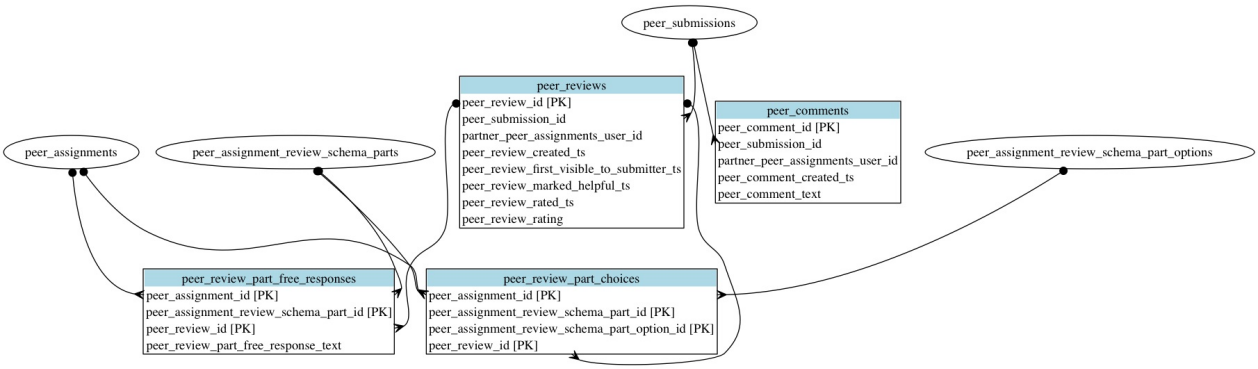
peer_assignment_id	peer_assignment_submission_schema_part_id	peer_submission_id	peer_submission_part_free_response_text
Search	Search	Search	Search
H8ED3xtoEeWshyIAC7OtLQ@19	2332432432	9eFL8A0BEeaAVhLVpQtKaw	Location: Interviewee's Home Equipment: Interview Script, Audio Recorder

The last column, `peer_submission_part_free_response_text`, shows the same text that the learner submitted in the screenshot above. If the question is of type "url" or "fileUpload", then the submission data can be found in the **peer_submission_part_urls** table.

Peer Review Rubric Parts

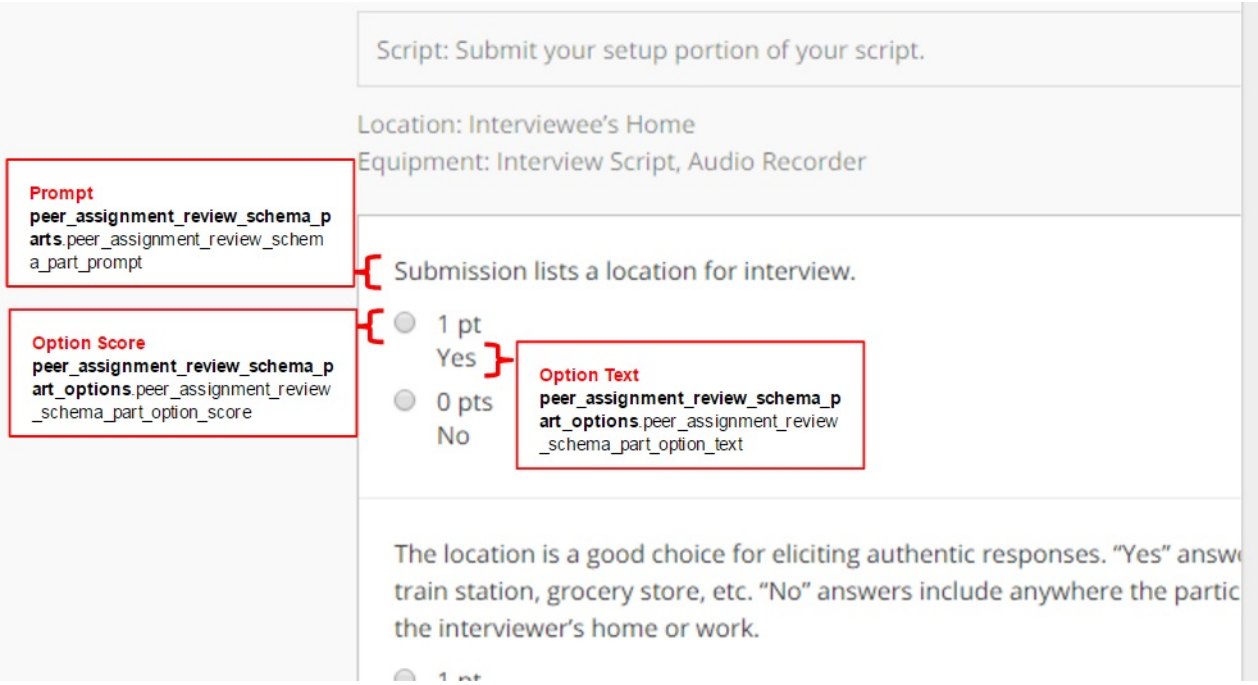
The **peer_reviews** table is populated when learners review their classmates' assignments. Unlike submissions, there are no draft reviews, and reviews cannot be deleted or modified. Each reviewer may submit at most one review per submission. The rubric choices and responses that reviewers submit are recorded in the **peer_review_part_choices** and **peer_review_part_free_responses** tables. A reviewer can also provide comments to

submissions, which are stored in the **peer_comments** table. The ER diagram below depicts the different peer review tables:



Reviewers may also decline to review a submission by providing a reason or by selecting one of the preset options such as "inappropriateContent", "incompleteSubmission", or "plagiarism". These are recorded in the **peer_skips** table (omitted from the diagram above).

Here is an example of a reviewer viewing a learner's submission with the rubric created by the instructor.



Querying the **peer_assignment_review_schema_parts** table will list all the rubric parts associated with all the submission prompts of a peer review assignment.

```
SELECT *
FROM peer_assignment_review_schema_parts
WHERE peer_assignment_id = 'H8ED3xtoEeWshyIAC7OtLQ@19'
```

peer_assignment_id	peer_assignment_review_schema_part_id	peer_assignment_review_schema_part_type	peer_assignment_review_schema_part_order	peer_assignment_review_schema_part_prompt	peer_assignment_review_schema_part_maximum_score
H8ED3xtoEeWshyIAC7OtLQ@19	1	yesNo	0	<co-content><text>Submission lists a location for interview. </text></c	1
H8ED3xtoEeWshyIAC7OtLQ@19	11	yesNo	1	<co-content><text>The location is a good choice for eliciting authentic	1
H8ED3xtoEeWshyIAC7OtLQ@19	18	yesNo	3	<co-content><text>Questions are focused on the topic of the participar	1
H8ED3xtoEeWshyIAC7OtLQ@19	19	yesNo	6	<co-content><text>There is a question asking about what the participar	1
H8ED3xtoEeWshyIAC7OtLQ@19	65	multiLineInput	0	<co-content><text>The best part of the submission was...</text></co-c	[NULL]
H8ED3xtoEeWshyIAC7OtLQ@19	111	yesNo	2	<co-content><text> Submission includes a list of equipment, such as r	1
H8ED3xtoEeWshyIAC7OtLQ@19	653	multiLineInput	1	<co-content><text>One thing the submission could do to improve is...<	[NULL]

The value seen in the first row of the `peer_assignment_review_schema_part_prompt` is the same as the rubric prompt seen in the screenshot above. The second column, `peer_assignment_review_schema_part_id`, provides the ID for this rubric part.

The rubric part in the screenshot above uses the yes/no options. The query below lists the possible options for this rubric part.

```
SELECT *
FROM peer_assignment_review_schema_part_options
WHERE peer_assignment_id = 'H8ED3xtoEeWshyIAC7OtLQ@19'
    AND peer_assignment_review_schema_part_id = '1'
```

peer_assignment_id	peer_assignment_review_schema_part_id	peer_assignment_review_schema_part_option_id	peer_assignment_review_schema_part_option_text	peer_assignment_review_schema_part_option_score
H8ED3xtoEeWshyIAC7OtLQ@19	1	Yes	Yes	1
H8ED3xtoEeWshyIAC7OtLQ@19	1	No	No	0

It is possible to have rubric parts associated with the entire submission and not one specific prompt.

Peer Reviews from Learners

Normally, a submission is expected to receive the same number of reviews that learners are required to complete. See the `peer_assignment_required_review_count` column on the **peer_assignments** table. Occasionally, there may not be any submissions in need of reviews when a learner goes to complete the review requirement. In that situation, old submissions are recycled, resulting in excess reviews. Any review that exceeds the number of required reviews is not factored into the submission's grade and is never displayed to the learner who submitted the work.

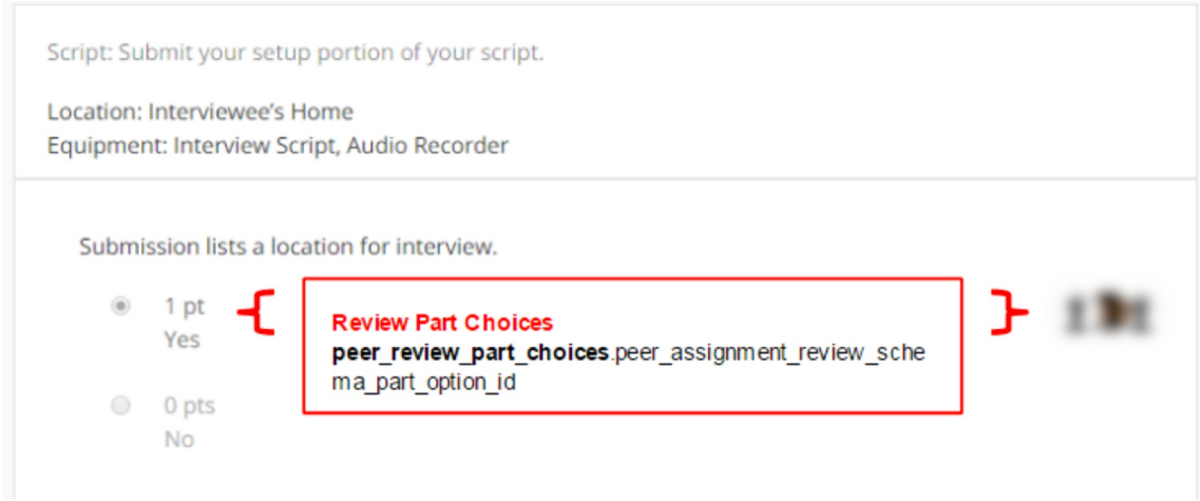
The **peer_reviews** table can be queries for the reviews on the submission titled "Audio Interview"

```
SELECT *
FROM peer_reviews
WHERE peer_submission_id = '9eFL8A0BEeaAVhLVpQtKaw'
ORDER BY peer_review_created_ts
```

peer_review_id	peer_submission_id	user_id	peer_review_created_ts	peer_review_first_visible_to_submitter_ts	peer_review_marked_helpful_ts	peer_review_rated_ts	peer_review_rating
rwkb6w21EeaxsBIihWq87Q	9eFL8A0BEeaAVhLVpQtKaw	######	4/29/2016 2:54:26	5/4/2016 21:55:21	[NULL]	[NULL]	[NULL]
_Jp_Gw35Eea63hJ3Ni_BAw	9eFL8A0BEeaAVhLVpQtKaw	######	4/29/2016 11:03:22	5/4/2016 21:55:21	[NULL]	[NULL]	[NULL]
wbFqixJCEeaO1w7d1s7iLw	9eFL8A0BEeaAVhLVpQtKaw	######	5/4/2016 21:54:21	5/4/2016 21:55:21	[NULL]	[NULL]	[NULL]
I8DeoRJDEeaqig5uXwUeGw	9eFL8A0BEeaAVhLVpQtKaw	######	5/4/2016 21:57:05	[NULL]	[NULL]	[NULL]	[NULL]

Although four reviews were completed, this peer review assignment will score upon only the first three reviews per the assignment configuration. The reviews with a populated value in the fifth column, `peer_review_first_visible_to_submitter_ts`, will be used. The last row contains a NULL value, and thus will neither be used in scoring, nor made visible to the learner who submitted the assignment. In the example above, the last review was completed after three other learners had already completed their reviews.

Learners are able to see how their reviewers scored each submission part, as seen in the screenshot below.



This data can be queried from the **peer_review_part_choices** table. The query below will only show the review responses that were used to calculate the grade, not any extra scores that might have been generated by reviewers above the number of reviews required for a grade.

```
SELECT *
FROM peer_review_part_choices
WHERE
    peer_assignment_review_schema_part_id = '1'
    AND peer_review_id IN (
        SELECT peer_review_id
        FROM peer_reviews
        WHERE
            peer_submission_id = '9eFL8A0BEeaAVhLVpQtKaw'
            AND peer_review_first_visible_to_submitter_ts IS NOT NULL
    )
```

peer_assignment_id	peer_assignment_review_schema_part_id	peer_assignment_review_schema_part_option_id	peer_review_id
H8ED3xtoEeWshyIAC7OtLQ@19	1	Yes	rwkb6w21EeaxsBIihWq87Q
H8ED3xtoEeWshyIAC7OtLQ@19	1	Yes	_Jp_Gw35Eea63hJ3Ni_BAw
H8ED3xtoEeWshyIAC7OtLQ@19	1	Yes	wbFqixJCEeaO1w7d1s7iLw

This result shows that all three reviewers chose the “Yes” option in the rubric for this submission part.

For these "Yes" part options, they were assigned to one point which we saw earlier in the **peer_assignment_review_schema_parts** table. In this simple case of the learner getting three ones, the median of those value will give the rubric part score of one.

Peer Submission Scoring

The score for each submission part for a specific submission can be found by querying the **peer_submission_part_scores** table. The part score is the median of the points received from each qualified review. A submission's score is the sum of all its part scores. In this example:

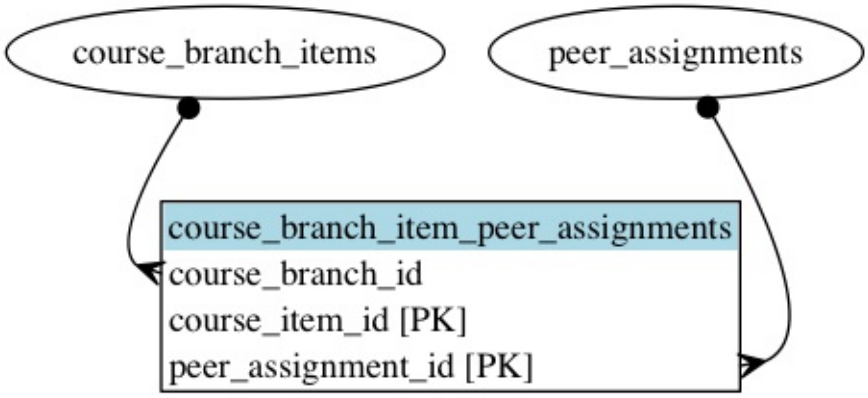
```
SELECT *
FROM peer_submission_part_scores
WHERE peer_submission_id = '9eFL8A0BEeaAVhLVpQtKaw'
```

peer_assignment_id	peer_assignment_review_schema_part_id	peer_submission_id	peer_submission_part_score
H8ED3xtoEeWshyIAC7OtLQ@19	1	9eFL8A0BEeaAVhLVpQtKaw	1
H8ED3xtoEeWshyIAC7OtLQ@19	11	9eFL8A0BEeaAVhLVpQtKaw	1
H8ED3xtoEeWshyIAC7OtLQ@19	18	9eFL8A0BEeaAVhLVpQtKaw	1
H8ED3xtoEeWshyIAC7OtLQ@19	19	9eFL8A0BEeaAVhLVpQtKaw	1
H8ED3xtoEeWshyIAC7OtLQ@19	111	9eFL8A0BEeaAVhLVpQtKaw	1
H8ED3xtoEeWshyIAC7OtLQ@19	21111	9eFL8A0BEeaAVhLVpQtKaw	1
H8ED3xtoEeWshyIAC7OtLQ@19	211112	9eFL8A0BEeaAVhLVpQtKaw	1
H8ED3xtoEeWshyIAC7OtLQ@19	2111122	9eFL8A0BEeaAVhLVpQtKaw	1
H8ED3xtoEeWshyIAC7OtLQ@19	21111223	9eFL8A0BEeaAVhLVpQtKaw	1
H8ED3xtoEeWshyIAC7OtLQ@19	211112234	9eFL8A0BEeaAVhLVpQtKaw	1
H8ED3xtoEeWshyIAC7OtLQ@19	2111122344	9eFL8A0BEeaAVhLVpQtKaw	1
H8ED3xtoEeWshyIAC7OtLQ@19	21111223444	9eFL8A0BEeaAVhLVpQtKaw	1
H8ED3xtoEeWshyIAC7OtLQ@19	211112234444	9eFL8A0BEeaAVhLVpQtKaw	1
H8ED3xtoEeWshyIAC7OtLQ@19	2111122344442	9eFL8A0BEeaAVhLVpQtKaw	1
H8ED3xtoEeWshyIAC7OtLQ@19	2342423423432	9eFL8A0BEeaAVhLVpQtKaw	2
H8ED3xtoEeWshyIAC7OtLQ@19	2342423423432444	9eFL8A0BEeaAVhLVpQtKaw	1
H8ED3xtoEeWshyIAC7OtLQ@19	23424234234324442	9eFL8A0BEeaAVhLVpQtKaw	2

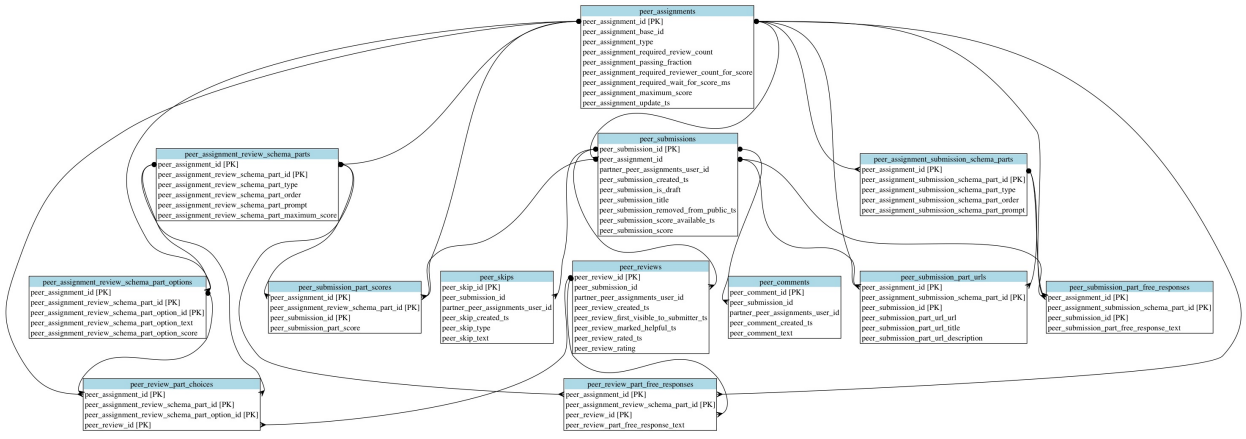
The results on the first row matches our earlier median calculation that the learner received a `peer_submission_part_score` of one for the first rubric part for the peer_submission_id "9eFL8A0BEeaAVhLVpQtKaw". The above table contains all seventeen rubric parts and the median score for each part. The sum of all rubric parts gives this learner's submission a final score of 19, which is consistent with the result in the last column, `peer_submission_score` , of the **peer_submissions** table.

peer_submission_id	peer_assignment_id	user_id	peer_submission_created_ts	peer_submission_is_draft	peer_submission_title	peer_submission_removed_from_public_ts	peer_submission_score_available_ts	peer_submission_score
9eFL8A0BEeaAVhLVpQtKaw	H8ED3xtoEeWshylAC7OtLQ@19	########	4/28/2016 5:27:55	FALSE	Audio Interview	[NULL]	5/4/2016 21:55:21	19

The `peer_assignment_id` column can be joined with tables outside of the peer review assignments tables by using the **course_branch_item_peer_assignments** table.



Below is our best attempt to build an ER diagram for all tables with the **peer_*** prefix.



> Click image to enlarge

SQL Example: What were the peer review responses in peer submissions?

The SQL below illustrates how multiple peer review assignments tables relate across peer review assignment questions, submissions and peer reviews. It returns the list of each submission to a peer review assignment question, submission time, submitted content, and each review received for each part of the submission.

```
SELECT
    ci.course_branch_id
    ,ci.course_branch_item_name
    ,ps.[partner]_peer_assignments_user_id
    ,ps.peer_submission_created_ts
    ,ps.peer_submission_title
    ,ps.peer_submission_score
    ,pspr.peer_submission_part_free_response_text
    ,psp.peer_submission_part_url_url
    ,psp.peer_submission_part_url_title
    ,psp.peer_submission_part_url_description
    ,pr.[partner]_user_id AS reviewer_id
    ,par.peer_assignment_review_schema_part_prompt
    ,parsp.peer_assignment_review_schema_part_option_text
    ,parsp.peer_assignment_review_schema_part_option_score
    ,prfr.peer_review_part_free_response_text
    ,prpc.peer_assignment_review_schema_part_option_id
FROM course_branch_item_peer_assignments cip
-- to get the item description
JOIN course_branch_items ci
    ON ci.course_item_id = cip.course_item_id
    AND ci.course_branch_id = cip.course_branch_id
-- to get the submitter's user_id, submission timestamp, and score
JOIN course_item_types cit
    ON cit.course_item_type_id = ci.course_item_type_id
JOIN peer_submissions ps
    ON ps.peer_assignment_id = cip.peer_assignment_id
-- to get the free text submission information of the submitted work if applicable
LEFT JOIN peer_submission_part_free_responses pspr
    ON ps.peer_submission_id = pspr.peer_submission_id
    AND ps.peer_assignment_id = pspr.peer_assignment_id
-- to get the url information of the submitted work if applicable
LEFT JOIN peer_submission_part_urls psp
    ON ps.peer_submission_id = psp.peer_submission_id
    AND ps.peer_assignment_id = psp.peer_assignment_id
-- to get information of the received reviews for each submission
LEFT JOIN peer_reviews pr
    ON ps.peer_submission_id = pr.peer_submission_id
-- the following two joins get the reviewer's assessment rubric
JOIN peer_assignment_review_schema_parts par
    ON par.peer_assignment_id = ps.peer_assignment_id
LEFT JOIN peer_review_part_free_responses prfr
    ON prfr.peer_assignment_id = par.peer_assignment_id
    AND prfr.peer_review_id = pr.peer_review_id
    AND par.peer_assignment_review_schema_part_id = prfr.peer_assignment_review_schema
_part_id
```

```
-- the following two joins get the review scores given to each assessment rubric
LEFT JOIN peer_review_part_choices prpc
    ON prpc.peer_assignment_id = par.peer_assignment_id
    AND prpc.peer_review_id = pr.peer_review_id
    AND par.peer_assignment_review_schema_part_id = prpc.peer_assignment_review_schema
_part_id
LEFT JOIN peer_assignment_review_schema_part_options parsp
    ON parsp.peer_assignment_id = prpc.peer_assignment_id
    AND parsp.peer_assignment_review_schema_part_id = prpc.peer_assignment_review_sche
ma_part_id
    AND parsp.peer_assignment_review_schema_part_option_id = prpc.peer_assignment_revi
ew_schema_part_option_id
WHERE course_item_type_desc ILIKE '%peer%';
```

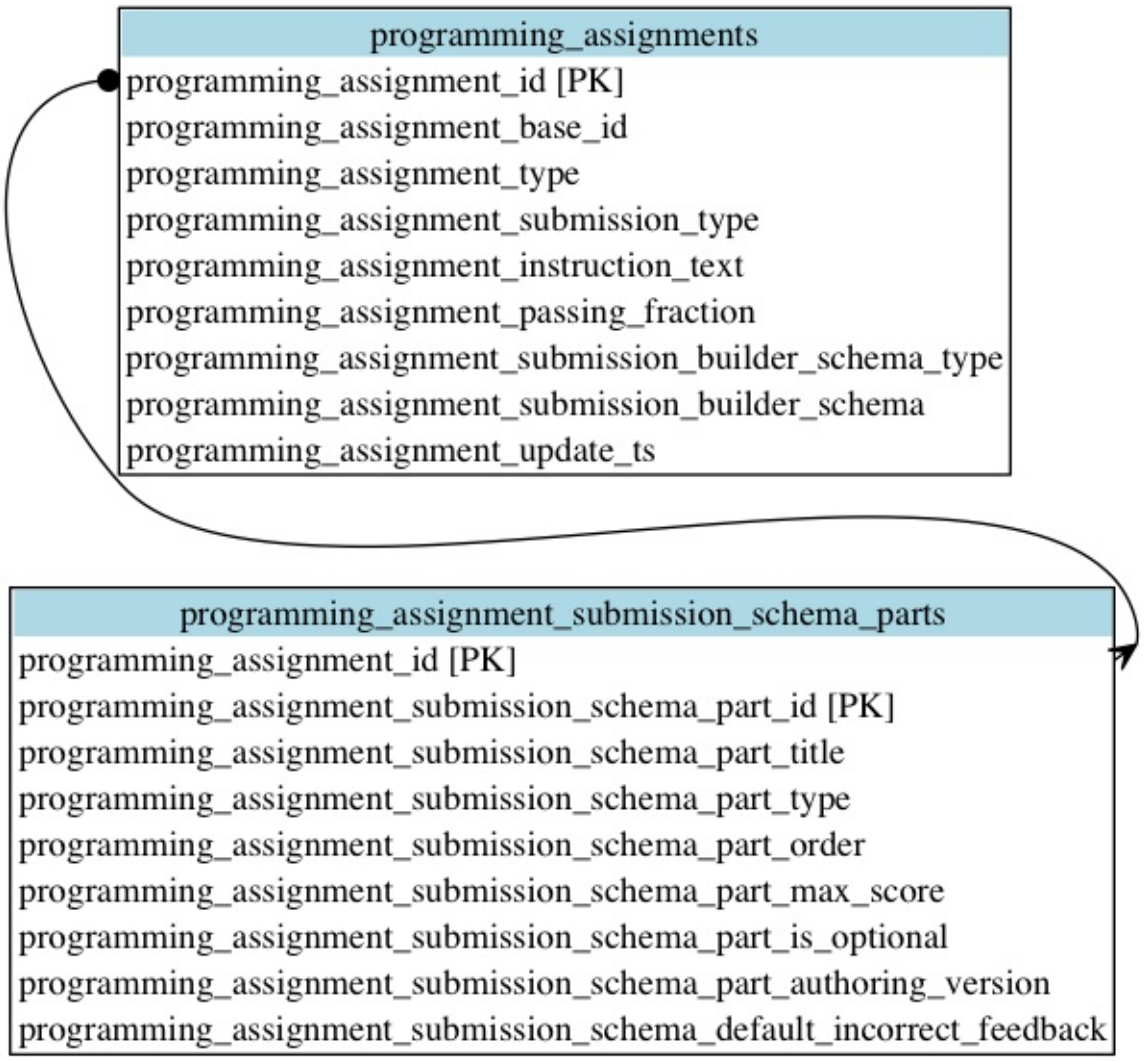
We appreciate the efforts put by Beth Simon in collaboration with UCSD to produce this in-depth chapter on Peer Review Assignments. We also thank Scott Klemmer for allowing us to use this course and its screenshots for documentation.

Programming Assignments

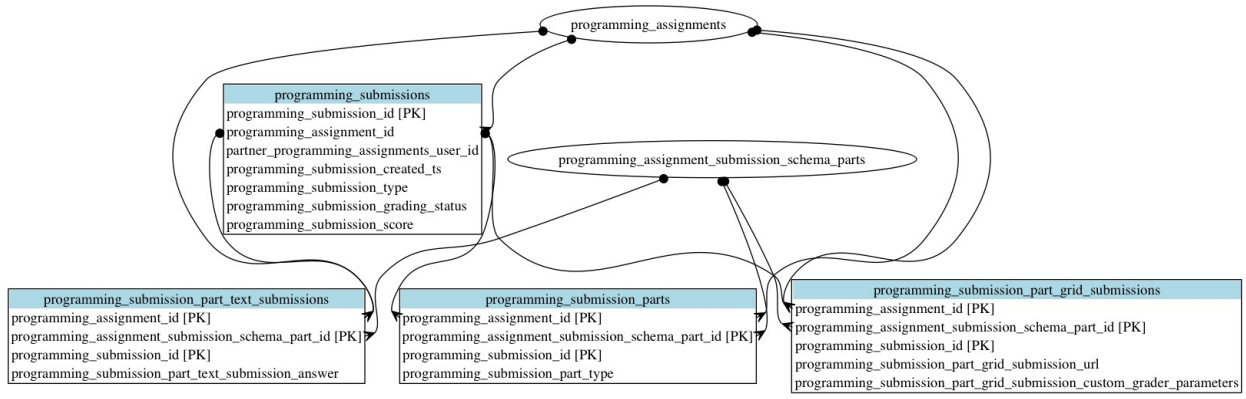
Programming Assignments are assignments that require learners to write and run a computer program. There are two types of assignments based on when they are graded after submission: 1) synchronously graded questions that are described in tables containing **_text_** in their names, and 2) asynchronously graded questions that are described in tables containing **_grid_** in their names.

The **programming_assignments** table gives you information on each programming assignment, assignment type (graded vs. ungraded), instructions, passing criterion, and creation time. The `programming_assignment_base_id` field uniquely identifies a programming assignment within a course, and the `programming_assignment_id` field combines the `programming_assignment_base_id` field and the version of the assignment.

Each part of a programming assignment can be found in the **programming_assignment_submission_schema_parts** table. Possible responses to a part and their orders can be found in the **programming_assignment_submission_schema_part_possible_responses** table.

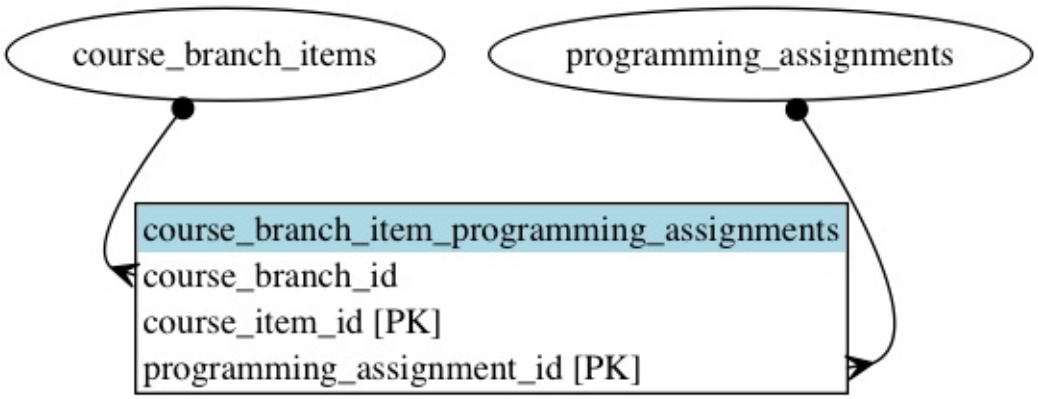


After learners submit their answers to a part of a programming assignment, their answers and scores are recorded in the **programming_submission_parts** table. If you are interested in learner submissions to asynchronously graded parts, refer to the **programming_submission_part_grid_submissions** table. If you are interested in learner submissions to synchronously graded parts, refer to the **programming_submission_part_text_submissions** table. The **programming_submissions** table tells you the aggregated submission information of the programming assignment by the learner.



> click image to enlarge

The `programming_assignment_id` column can be joined with tables outside of the programming assignments tables by using the **course_branch_item_programming_assignments** table.



SQL Example: What were learners' scores across programming submissions?

The SQL below illustrates how the multiple programming assignment tables relate across assignment questions, submissions, and received scores. It returns the list of each programming question, learner submission, the status of grading the submission, and the received score.

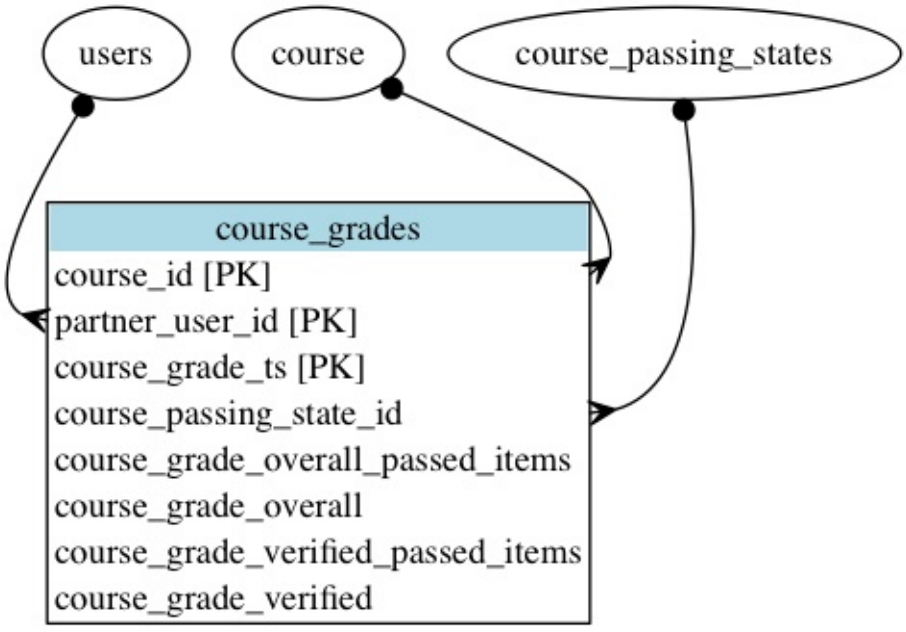
```
SELECT
    course_branch_id
    ,course_branch_item_name
    ,[partner]_programming_assignments_user_id
    ,programming_submission_created_ts
    ,programming_submission_grading_status
    ,programming_submission_score
FROM course_branch_item_programming_assignments cip
JOIN course_branch_items
    USING (course_branch_id, course_item_id)
JOIN programming_assignments
    USING (programming_assignment_id)
JOIN programming_submissions
    USING (programming_assignment_id)
```

Course Grades Tables

The four **Course Grades** tables are:

1. **course_grades:** records the highest grade reached for each learner in a given course and whether the learner has passed the course.
2. **course_branch_grades:** records the highest grade reach for each learner in a given course version and whether the learner has passed the course version.
3. **course_item_grades:** records the most recent grade reached for each course item that are mandatory for course completion.
4. **course_formative_quiz_grades:** includes grades for items that not mandatory for course completion; this table is mutually exclusive to the **course_item_grades**.

The **course_grades** table describes whether learners have passed a course by passing any course version. Each row records the grading event when the learner reached their highest grade in the course in addition to the date and time of that event, how many items were passed at that time, and whether the learner had reached the passing state. The **course_passing_states** table provides the descriptions of the passing states.



The **course_branch_grades** table describes whether learners have passed a course version. Each row records the grading event when the learner reached their highest grade in course versions.

The **course_item_grades** table contains a row for each learner's most recent grade for each course item attempted and whether the item was passed or not. For example, if a learner passed a course that consists of four graded items, then this table would contain four

rows for this learner, each the datetime when the item was passed.

The **course_formative_quiz_grades** table has similar logic to the previous table, with the exception that there is no column to denote a passing state, since formative quizzes are not mandatory for course completion.

Discussion Tables

Discussion forums provide a space for learners to interact, share resources, and help one another with questions about the course materials and assignments.

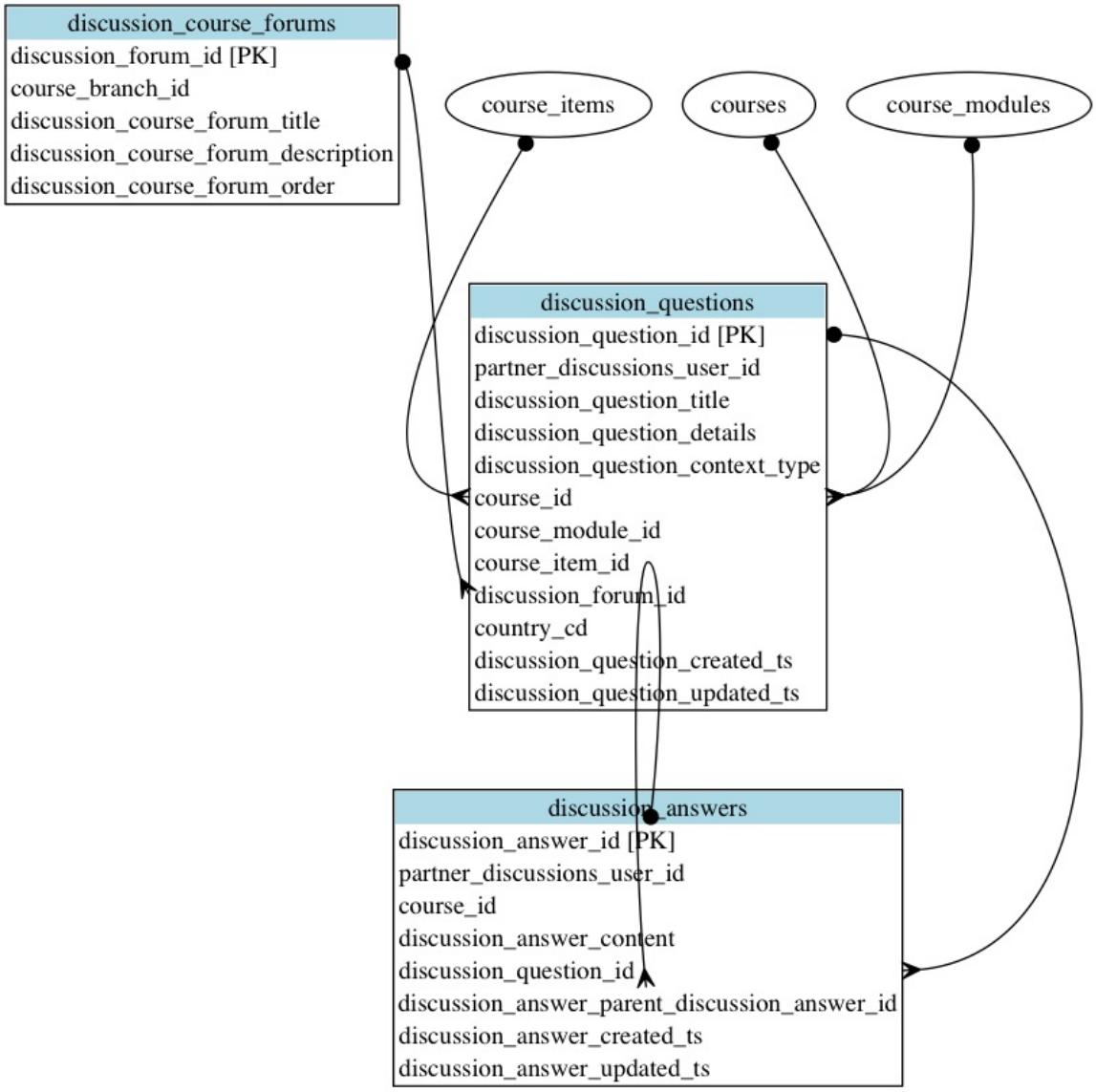
In November 2016, Coursera enhanced the existing Course Forums and released new types of discussion forums: Mentor Forums, Custom Forums, and Subforums. More infomation can be found in our Partner Help Center article on Discussion Forums.

Exports currently include Course Forums data via three main tables:

- **discussion_course_forums**: lists the forums (e.g. "General Discussions", "Meet and Greet", etc.) with the course it is present in
- **discussion_questions:** includes the original posts in each thread, the author, and the thread's placement within the course
- **discussion_answers**: lists the replies to the threads in the previous table

All other discussion tables focus on followings, votes, flags (not shown in ER diagram below):

- **discussion_answer_flags**: records when a learner flags an answer for the Coursera community operations team to review, because the answer does not follow community guidelines (offensive or spam)
- **discussion_questions_flags**: records when a learner flags a question for the Coursera community operations team to review, because the question does not follow community guidelines (offensive or spam)
- **discussion_answer_votes**: records when an answer is upvoted (and when a user revokes their upvote)
- **discussion_questions_votes**: records when a discussion question is upvoted
- **discussion_questions_followings**: records when a discussion question is followed or unfollowed


discussion_course_forums
discussion_forum_id [PK]
course_branch_id
discussion_course_forum_title
discussion_course_forum_description
discussion_course_forum_order
course_items
courses
course_modules
discussion_questions
discussion_question_id [PK]
partner_discussions_user_id
discussion_question_title
discussion_question_details
discussion_question_context_type
course_id
course_module_id
course_item_id
discussion_forum_id
country_cd
discussion_question_created_ts
discussion_question_updated_ts
discussion_answers
discussion_answer_id [PK]
partner_discussions_user_id
course_id
discussion_answer_content
discussion_question_id
discussion_answer_parent_discussion_answer_id
discussion_answer_created_ts
discussion_answer_updated_ts

Feedback Tables

The Feedback tables are populated as learners flag course content, like/dislike course content, or rate courses. The main two tables are **feedback_course_ratings** and **feedback_item_ratings**.

The **feedback_course_ratings** contains two different sets of course ratings, which can be distinguished based on the `feedback_system` column. The value of "STAR" is for the system of 5-star ratings for the entire course.

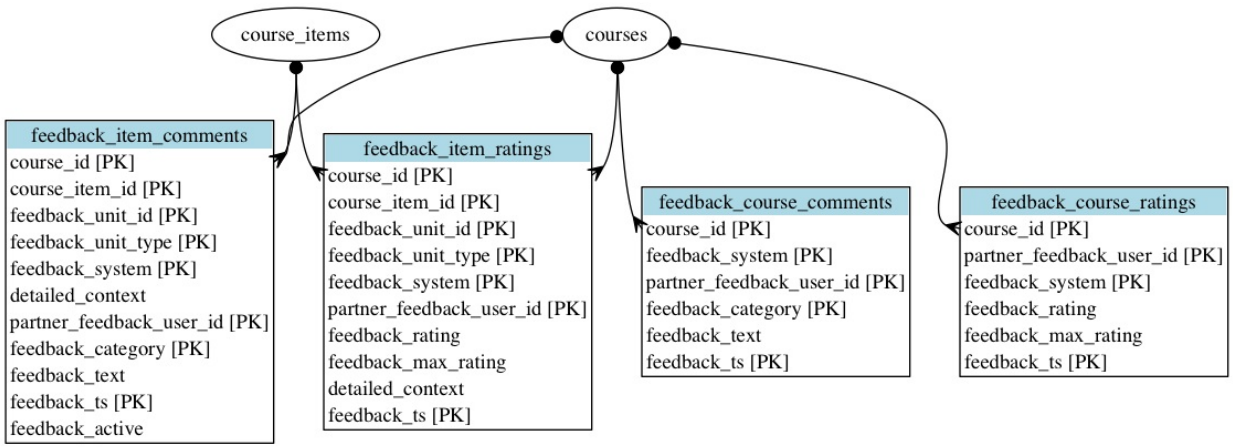
As of the spring of 2016, Coursera is surveying a measurement called Net Promoter Score (NPS) to better understand how satisfied learners are with courses overall. We ask learners, "How likely are you to recommend this course to a friend or colleague?", provide a rating scale between 1 to 10. Net Promoter Score is calculated as the percentage of Promoters (those selecting a rating of 9 or 10) minus the percentage of Detractors (those selecting a rating of 6 or below).

In the `feedback_system` column, the new values for NPS results are "NPS_FIRST_WEEK" and "NPS_END_OF_COURSE". For any data researcher who does not filter on one `feedback_system`, such as "STAR", the results will ***erroneously aggregate scores from multiple systems***.

The **feedback_item_ratings** table contains the "like" or "dislike" ratings for a course item.

Although a learner can rate the same course (item) multiple times, the data provided in exports is filtered to only provide the most recent rating per learner-course (learner-item) pairing.

There are also two feedback tables that log the free-text responses. The **feedback_item_comments** contains comments from learners who flag a course content. The **feedback_course_comments** contains learners' comments about the course in both star and NPS feedback systems. Both of these tables contain the entire history of comments to your course, not just the most recent.



SQL Example: How does my overall course 5-stars rating vary across months?

```sql
SELECT
    course_id
    ,TO_CHAR(feedback_ts, 'YYYY-MM')
    ,COUNT(*) AS num_feedback
    ,AVG(feedback_rating::FLOAT) AS avg_feedback_rating
FROM feedback_course_ratings
-- Ensure that only one feedback system is used.
WHERE feedback_system = 'STAR'
GROUP BY 1,2
ORDER BY 1,2;
```

Learner Tables

The Learner tables contain information about Coursera's registered accounts.

The **users** table lists all learners and instructors that are related to your export. Per Coursera's Data Sharing Policy and Research Policy, this table does not include personally identifiable information (PII), such as user names and emails. Below are details for some of the columns in the **users** tables.

Geographic columns are populated by inferring the user's browser IP via the Maxmind geolocation service.

Language columns are populated by the user's profile setting or by the value from the user's browser.

The `reported_or_inferred_gender` column contains the values of "male", "female", or NULL. This column is populated by a curation across multiple user self-reported sources and by inferred gender via this python library.

The `employment_status` column contains the values below that correspond to the learner's selection to the Employment Status question in their Coursera profile. A "NULL" value means the learner did not choose one of these selections.

employment_status	Profile Selection
EMPLOYED_FULL_TIME	Employed full time (35 or more hours per week)
EMPLOYED_PART_TIME	Employed part time (less than 35 hours per week)
HOMEMAKER	Homemaker, taking care of a family member, or on maternity/paternity leave
RETIRED	Retired
SELF_EMPLOYED_FULL_TIME	Self-employed full time (35 or more hours per week)
SELF_EMPLOYED_PART_TIME	Self-employed part time (less than 35 hours per week)
UNABLE_TO_WORK	Unable to work
UNEMPLOYED_LOOKING_FOR_WORK	Unemployed and looking for work
UNEMPLOYED_NOT_LOOKING_FOR_WORK	Unemployed and not looking for work

The `educational_attainment` column contains the values below that correspond to the learner's selection to the Educational Attainment question in their Coursera profile. A "NULL" value means the learner did not choose one of these selections.

educational_attainment	Profile Selection
ASSOCIATE_DEGREE	Associate Degree (e.g., AA, AS)
BACHELOR_DEGREE	Bachelor's degree (e.g., BA, AB, BS)
COLLEGE_NO_DEGREE	Some college but no degree
DOCTORATE_DEGREE	Doctorate degree (e.g., PhD, EdD)
HIGH_SCHOOL_DIPLOMA	High school diploma (or equivalent)
LESS_THAN_HIGH_SCHOOL_DIPLOMA	Less than high school diploma (or equivalent)
MASTERS_DEGREE	Master's degree (e.g., MA, MS, MEng, MEd, MSW, MBA)
PROFESSIONAL_DEGREE	Professional school degree (e.g., MD, DDS, DVM, LLB, JD)

The `student_status` column contains the values below that correspond to the learner's selection to the Student Status question in their Coursera profile. A "NULL" value means the learner did not choose one of these selections.

student_status	Profile Selection
FULL_TIME_DEGREE_STUDENT	Yes, I am currently a full-time student in a degree-granting educational program
NOT_DEGREE_STUDENT	No, I am not currently a student in a degree-granting educational program
PART_TIME_DEGREE_STUDENT	Yes, I am currently a part-time student in a degree-granting educational program

---

The **course_membership** table logs when a Coursera user gets assigned to a membership role in the course. The most common values are:

- "LEARNER": users that clicked the Enroll button on an already-launched course
- "PRE_ENROLLED_LEARNER": users that pre-enrolled in a course or session
- "NOT_ENROLLED": users that unenrolled in the course for any reason
- "MENTOR": users that are invited and then accepted to moderate course discussions
- "BROWSER": users that previewed content on the course description page, but has not clicked on the Enroll button
- "INSTRUCTOR": users that were ever set as an instructor in the course

The other possible values are "UNIVERSITY_ADMIN" and "TEACHING_STAFF".

A single learner may have had multiple roles in a course over time; this learner will have multiple records in the table. For example, a learner could have watched the preview content (“BROWSER”), then hit the enroll button to join (“LEARNER”), and then unenrolled a few days later (“NOT_ENROLLED”).

This table only contains membership information for users that ever reached the “LEARNER”, "MENTOR", or "INSTRUCTOR" role. For example, it does not contain any only-"BROWSER" users that previewed course material but never completed the enrollment process.

There is an export table with the name **[partner]_course_user_ids**. It has two id columns:

- `[partner]_user_id` : consistent with other user ID(s) in the data export
- `[course_slug]_user_id` : used in marketing or surveying efforts

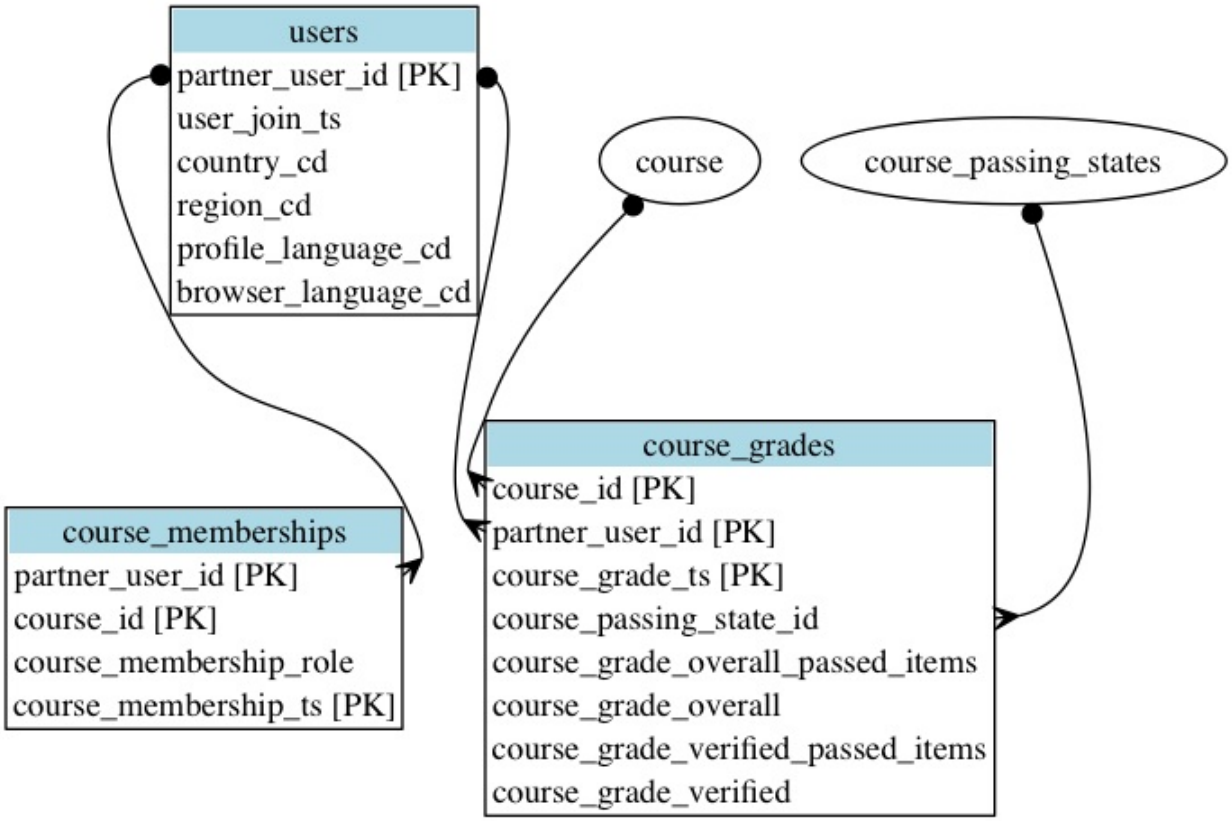
This table is very useful for instructors to administer surveys to learners, outlined in this Partner Help Center article on External Surveys. See the ID Columns chapter for more details.

SQL Example: How many learners, pre-enrollers, and completers do I have from each country?

```
-- learners who pre-enrolled
WITH course_pre_enrollers AS (
    SELECT DISTINCT [partner]_user_id
    FROM course_memberships
    WHERE course_membership_role = 'PRE_ENROLLED_LEARNER'
)

-- learners who passed the course
, course_completers AS (
    SELECT DISTINCT [partner]_user_id
    FROM course_grades
    WHERE
        -- 1 = passed, 2 = verified passed
        course_passing_state_id IN (1,2)
)

-- by country_cd (e.g. 'US'), provide counts of users
SELECT
    country_cd
    ,COUNT(*) AS num_course_learners
    ,COUNT(course_pre_enrollers.[partner]_user_id) AS num_pre_enrollers
    ,COUNT(course_completers.[partner]_user_id) AS num_completers
FROM users
LEFT JOIN course_pre_enrollers
    USING ([partner]_user_id)
LEFT JOIN course_completers
    USING ([partner]_user_id)
GROUP BY 1
ORDER BY 2 DESC;
```


users
partner_user_id [PK]
user_join_ts
country_cd
region_cd
profile_language_cd
browser_language_cd
course
course_passing_states
course_grades
course_id [PK]
partner_user_id [PK]
course_grade_ts [PK]
course_passing_state_id
course_grade_overall_passed_items
course_grade_overall
course_grade_verified_passed_items
course_grade_verified
course_memberships
partner_user_id [PK]
course_id [PK]
course_membership_role
course_membership_ts [PK]

Demographic Tables

At any time, learners can take a voluntary Coursera survey in which they volunteer answers to questions such as:

- In what country do you currently live?
- What is your gender?
- What is the highest level of school you have completed or the highest degree you have received?
- Which of the following best describes your current employment status?
- Besides English, what other languages do you speak?

The complete list of questions is contained in the **demographics_questions** table. The **demographics_answer** table contains the answers from Coursera learners.

This survey contains responses from about 300k Coursera users from 2013 to March 2015. The data export only contain those users who enrolled in your course and volunteered to answer the survey.

Summary Tables

The tables described in Data Tables chapter act as our base sources of facts and dimensions in our data warehouse. Coursera also designs and builds additional summary tables (also known as aggregate tables). By joining multiple base tables and applying business logic, we create a single table with summarized results for one analytic purpose.

This chapter will list the summary tables that have been added to our data exports. The first example is **users_courses__certificate_payments**. These tables will be distinguishable by the format of their table names as:

`grain__purpose`

The **grain** of the table will help indicate the primary key of the table. In this example, the prefix **users_courses** is used because the summary table's primary key consists of two columns: `[partners]_user_id` and `course_id`.

The **purpose** of the table will help indicate the meaning of summary table. In this example, the suffix **certificate_payments** implies that the other columns in this table are related to payment condition of course certificates.

The prefix and suffix are delimited by two underscores '__'. Each can contain compound words, separated by an underscore '_'.

Enrollments Summary Tables

These set of tables center around a Coursera enrollment, which is simply the pair of a *[partner]_user_id* to a *course_id*.

Course Certificate Payments (users_courses__certificate_payments)

The **users_courses__certificate_payments** table provides details on whether an enrollment has met the payment condition for the eligibility of earning a Course Certificate. Below are the three paths to meet the payment condition:

- The boolean column, `was_payment`, states whether the learner has ever paid for the eligibility of a course certificate, enrolled in the course, and has not made a refund. This purchase could be a "single payment" for the course or a "bulk payment" for a specialization that contains the course. For bulk payments, this table will only contain rows for those specalization's courses (or capstone) that user has already enrolled in. For example, a learner who has just purchased a four-course specialization will likely be enrolled in the first course, which adds a row to this table. Only after the learner enrolls in the specialization's second course will a second row appear.
- The boolean column, `was_finaid_grant`, states whether the learner has ever been granted financial aid for the course. A learner can only apply and be granted financial aid to courses and not to an entire specialization at once. Of note, there are a few cases where the learner applied and was granted financial aid and then later paid for the course.
- The boolean column, `was_group_sponsored`, states whether the learner has ever been sponsored to meet the payment criteria by being a member of a Coursera group which has the premium experience enabled. Past and future examples are:
    - Partners assigning on-campus learners in Coursera groups for the blended learning experience
    - Employees of Coursera's enterprise partners assigned to Coursera groups
    - Soon, the "Coursera for Refugees" program to provide assistance to refugees

The last boolean column, `met_payment_condition`, states whether the user has ever met the payment condition to be eligible for the course certificate. This is calculated via the logical 'OR' of the three other columns above.

If a learner has enrolled in a course and that user-course pairing does not appear in this table, then that enrollment is considered as a free or audit enrollment. The learner cannot receive a Course Certificate by not meeting the payment condition.

There are other conditions for a user to achieve a Course Certificate. For example, the completion condition is met when a learner passes and verifies all graded items in the course. The **users_courses__certificate_payments** summary table only contains the payment condition.

Clickstream Data Guide

Coursera's clickstream data is a log of the user's interactions with Coursera. We call each row of this log an **event**. Every event captures some user interaction at some specific point in time. This allows for fine-grained analysis of user behavior, engagement, and trends.

Event Structure

Every event consists of metadata, a key, and a value. The metadata contains information about the client sending the event. The key identifies the type of user action being recorded, and the value is a JSON object consisting of data specific to that event.

Each row is in Postgresql CSV format, with fields in the following order:

- [partner]_user_id
- hashed_session_cookie_id
- server_timestamp
- hashed_ip
- user_agent
- url
- initial_referrer_url
- browser_language
- course_id
- country_cd
- region_cd
- timezone
- os
- browser
- key
- value

An example row looks as follows:

```
"REDACTED_HASHED_USER_ID","REDACTED_HASHED_SESSION_COOKIE_ID","2016-08-01 02:06:44.008
","REDACTED_HASHED_IP","Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML
, like Gecko) Chrome/51.0.2704.103 Safari/537.36","https://www.coursera.org/learn/REDA
CTED_COURSE_SLUG","REDACTED_REFERER_URL","en-US,en;q=0.8","REDACTED_COURSE_ID","KE","1
10","Africa/Nairobi","Windows 10","Chrome","page_preloaded","{}"
```

This row is for a pageview event for a course, taking place on August 1, 2016, for a user using Chrome on Windows 10 in Kenya.

The following SQL can create a table in postgres that is then suitable to have clickstream data copied into:

```
CREATE TABLE clickstream_events (
    hashed_user_id varchar,
    hashed_session_cookie_id varchar,
    server_timestamp timestamp,
    hashed_ip varchar,
    user_agent varchar,
    url varchar,
    initial_referrer_url varchar,
    browser_language varchar,
    course_id varchar,
    country_cd varchar,
    region_cd varchar,
    timezone varchar,
    os varchar,
    browser varchar,
    key varchar,
    value varchar
);
```

To copy data into this table, create a Postgres database, and run the following command:

```
psql -d DATABASE_NAME -c "\COPY clickstream_events FROM '/PATH/TO/EVENTS.csv' CSV NULL
 '' QUOTE '\"' ESCAPE '\\'"
```

For more tips on how to make use of clickstream data, see How To Analyze.

Data Flow

As users interact with Coursera via a browser, or a mobile application, events are sent to servers with information on user actions. This information is then persisted by the server by uploading to Amazon S3, and then made available for exports.

Reliability

Clickstream data is persisted on a best-effort basis. If users have intermittent internet connections, then it is possible for events generated on the client to not make its way to the server. Our clients on web and mobile do implement retry logic for when the initial request was unsuccessful.

Coursera tracks the reliability of our clients, and we do not notice systematic event losses. We believe events are lost in an unbiased fashion, but please contact data-support@coursera.org if you see abnormalities and we are happy to investigate them with you.

Batching

Events are batched on the client before sending to the server. The batching means the server_timestamp for each event may be seconds away from when the event was generated. Coursera does record the client generated timestamp for each event, but it is not used for internal analysis due to the variability of clocks between different users.

Tracking Lossiness

Coursera attaches a monotonically increasing sequence number to each event constructed on the client. Once events make its way to the server, we look at the gaps in sequence numbers to understand lossiness.

Overall, platform wide loss is approximately 6%, which is a meaingful but not significant amount of all events generated. However, this 6% ranges significantly by country. We notice lossiness as high as up to 15% in countries with poor internet infrastructure, such as India and China.

Metadata

The following fields are available for each event. A field may either be raw (recorded directly from the client sending the event), or derived (a transformation of raw fields to a more useful format). Fields also have varying degrees of reliability, which we note when appropriate.

Raw Fields

[partner]_user_id

A hashed id that may identify the same user across multiple table domains. For example, if this learner received a course grade, his/her hashed id may be found in the course grades tables.

Example: `eeac5ed1e11cda66032c30ad77b25071b5487f55`

hashed_session_cookie_id

A hashed identifier of the user's cookie, which is used to track individual client machines. A user may have multiple hashed_session_cookie_ids if they clear their cookies after browsing and a session_cookie_id may have multiple users. Multiple users may have the same `hashed_session_user_id` if the computer is shared, or if a single learner has multiple accounts.

Example: `6c85c4e268f8762f70d58d2a77aee65c8851fc11`

server_timestamp

A timestamp recorded on the server in UTC when the server received the event, with in the format `yyyy-MM-dd hh:mm:ss.SSS`.

Example: `2015-04-01 03:58:57.864`

hashed_ip

A hashed identifier of the IP address from which the client sent the event.

Example: `d34de38afb81930d474d546169606f94831db591`

user_agent

The user agent sending the event to Coursera servers.

Example: `Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.101 Safari/537.36`

url

The url the user was on when sending the event to Coursera servers, including all parts of the url. This means any query string (e.g `?foo=bar`) and hash fragment (e.g `#foobar`) will be attached to the url. While in some cases the query string is significant, it is just as often appended for no particular reason when a user takes an unusual route to the site (an external link from a blog, for instance, might include special referrer data in the query string on the off-chance that the linked site will handle it). Handling this field with a url parser, such as Pyton's urlparse is recommended.

Example: `https://www.coursera.org/learn/negotiation-skills/outline`

initial_referrer_url

The url whence the user first navigated to your course. The url must be from a non-Coursera domain, and the initial referrer is refreshed after 30 minutes of inactivity.

Example: `https://www.google.com/`

browser_language

The `Accept-Language` headers sent by the client. Specifications of these headers are available here

The language values may have a associated quality value. It is recommended to parse this field using a parser. Here is a very simple example for Python.

Example: `en-US,en;q=0.8`

Derived Fields

course_id

The course id for the event. This is derived from the course slug in the url. The course slug is the part of the url after `/learn/` (For example, the course slug for `https://coursera.org/learn/machine-learning` is `machine-learning`). We map the course slug to a course id using a historical mapping table.

Example: `Gtv4Xb1-EeS-ViIACwYKVQ`

country_cd

A two-letter (ISO 3166-1 alpha-2) country code derived from the ip using the Maxmind GeoIP2 database. Maxmind claims

> 99.8% accurate on a country level

For more details, see https:\/\/www.maxmind.com\/en\/geoip2-city-database-accuracy

Example: `US`

region_cd

An alphanumeric string up to three letters (second part of ISO 3166-2) denoting a country subdivision dervied from the ip using the Maxmind GeoIP2 database. This code is usually obtained from coding systems already in use in the country concerned.

Maxmind claims

> 90% accurate on a state level

For more details, see https:\/\/www.maxmind.com\/en\/geoip2-city-database-accuracy

Example: `CA`

If the `country_cd` was `US`, then this `region_cd` represents the state.

timezone

The timezone of the client sending the event, derived from the ip using the Maxmind GeoIP2 database. Maxmind does not publish the accuracy of the timezone inferrence, but we believe its accuracy to be on par with `region_cd`.

Example: `America/New_York`

os

The Operating System the client sending the event operates on, derived from the user agent using the ua-parser library. This library maintains a large regex mapping the user agent to OS.

Example: `Windows 7`

browser

The Operating System the client sending the event operates on, derived from the user agent using the ua-parser library. This library maintains a large regex mapping the user agent to browser.

Example: `Chrome`

Video

Video events are recorded as users interact with your lecture videos through the browser, or through our mobile applications. We only export video events from the browser currently, meaning videos watched on our iOS and Android app will not be reflected in the exports. Please contact us at data-support@coursera.org if this is a need of yours.

As a reminder, each event has common metadata, is identified by a key field, and contains a value field with a JSON object. In the video domain, the value JSON object contains common fields, and fields specific to individual keys. We first document the common fields and then document the available keys, and available fields for each of them.

Common Value Fields

item_id

See course content tables

Example: `0K5HA`

Keys

Each of these keys may have additional fields in the value JSON object. The possible fields are explained in the section additional fields

heartbeat

An event that is fired every 5 seconds of the video playing.

The value JSON object contains the following additional fields: `playback_rate, subtitle_language, tech, volume, module_id, timecode, video_name`

wait

An event fired when the user requires to wait to buffer additional video. (For example, due to slow internet)

The value JSON object contains the following additional fields: `module_id, timecode, video_name`

seek

An event fired when the user seeks to another part of the video. The `timecode` field in this event refers to the time the user is seeking to.

The value JSON object contains the following additional fields: `module_id, timecode, video_name`

start

An event fired when the video first starts. The `timecode` field should be 0.

The value JSON object contains the following additional fields: `module_id, timecode, video_name`

end

An event fired when the video ends. The `timecode` field should be close to the length of the video.

The value JSON object contains the following additional fields: `module_id, timecode, video_name`

pause

An event fired when the user pauses the video.

The value JSON object contains the following additional fields: `module_id, timecode, video_name`

play

An event fired when the user restarts a paused video.

The value JSON object contains the following additional fields: `module_id, timecode, video_name`

volume_change

An event fired when the user changes the volume.

The value JSON object contains the following additional fields: `playback_rate, subtitle_language, tech, volume, volume_delta, module_id, timecode, video_name`

subtitle_change

An event fired when the user changes the subtitle language.

The value JSON object contains the following additional fields: `playback_rate, subtitle_language, tech, volume, module_id, timecode, video_name`

playback_rate_change

An event fired when the user changes the playback rate.

The value JSON object contains the following additional fields: `playback_rate, playback_rate_delta, subtitle_language, tech, volume, module_id, timecode, video_name`

download_subtitle

An event fired when the user clicks the download subtitle button.

The value JSON object contains the following additional fields: `language_code`

download_video

An event fired when the user clicks the download video button.

The value JSON object does not contain any additional fields.

Additional Fields

module_id

See course content tables

Example: `iXIFE`

playback_rate

A number representing the playback rate. A playback rate of 1 represents playing the video at 1x speed.

Example: `1.25`

playback_rate_delta

A number representing the change in playback rate.

Example: `-0.25`

subtitle_language

An IETF language tag representing the language of the subtitles.

Example: `zh-CN`

tech

`flash` or `html5`

timecode

The time within the video when the event was sent.

Example: `731.793587`

video_name

The video name, given when the video was uploaded. seek Example: `What is Bioconductor`

volume

A number between 0 and 1 describing the volume of the player.

Example: `0.706600341796875`

volume_delta

A number representing the volume change.

Example: `-0.029599609374999946`

Access

Access events are recorded when users accesses any course material through the browser, or our mobile applications.

We only export access events from the browser (including mobile browser) presently. Learners using our iOS and Android application will not be reflected in the exports. Please contact us at data-support@coursera.org if this is a need of yours.

As a reminder, each event has common metadata, is identified by a key field, and contains a value field with a JSON object.

Common Value Fields

Currently access events do not have any value fields, as all the information is available in the `url` field.

Keys

pageview

This event is fired when a user accesses Coursera's website through a browser. The browser may be a desktop browser (e.g `Chrome`), or a mobile browser (e.g `Chrome Mobile`)

More specifically, the event is fired when the user downloads all the Javascript required to render the page. If a user hits `coursera.org` but then closes the browser, or goes to another page before all the Javascript is downloaded, that user will not trigger a `pageview` event.

page_requested

This event is fired when a user makes a request to Coursera's website through a browser. This differs from the `pageview` event in that it is fired immediately after the user requests the page, not when the user downloads the HTML for rendering the page. This event is not fired when the user navigates within a single page application. There should be less `page_requested` events than `pageview` events.

page_preloaded

This event is fired the first time the user downloads Coursera's Javascript for a given page load. This differs from the `pageview` event in that it is fired before all the Javascript required to render the page is downloaded. This event is not fired when the user navigates within a single page application. There should be less `page_preloaded` events than `pageview` events.

URL Examples

- `https://www.coursera.org/learn/SLUG/discussions/weeks/1` - Discussion forums for week 1 of course identified by SLUG.
- `https://www.coursera.org/learn/SLUG/lecture/ITEM_ID` - A lecture video for course identified by SLUG, for a given ITEM_ID. Item ids may be linked via the course content tables, and is also linked to video clickstream events.
- `https://www.coursera.org/learn/SLUG/exam/ITEM_ID` - An exam ("summative quiz"). URLs of this type may also contain the exam name at the end of the URL. Item ids may be linked via the quiz tables.
- `https://www.coursera.org/learn/SLUG/quiz/ITEM_ID/QUIZ_NAME`. A quiz ("formative quiz"). URLs of this type may also contain the quiz name at the end of the URL. Item ids may be linked via the quiz tables

URL Structure

All pageviews are of the form `...coursera.org/learn/YOUR_COURSE_SLUG...`.

Looking at the examples above, we see the URL have a structure: we can identify the item id for a given pageview by splitting on `/`, and looking at the 7th split. Looking at the 6th split gives us the type of access. (E.g. was it for a lecture? Was it for a quiz? A discussion?)

Below, I list the most common types of accesses:

Access Type
home
lecture
supplement
peer
exam
discussions
outline
quiz
module
programming
forum
discussionPrompt
gradedLti
resources
ungradedLti

These access types correspond roughly to the item types described in the course item types table, but the mapping isn't exact.

Complications

Query Strings and Hash Fragments

A url may contain a query string, and a fragment identifier. For example, the url `https://www.coursera.org/learn/MY_COURSE_SLUG?query=123#hash` has both a query string, and a hash identifier. When analyzing events in aggregate, they may safely be ignored. When analyzing events from individual users, the query string may include UTM parameters, and other useful tracking values.

Data Quality Issues

There have been periods in the past when `pageview` events were not being recorded properly due to programming errors. One period Coursera is aware of is between 18:00 UTC on May 17 to around May 20. Please let us know of any additional data quality issues, and we'd be happy to investigate them with you.

Redirect

For redirects, we will fire pageview both on the initial location, and on the redirected location. For example, `https://www.coursera.org/learn/r-programming/week/1/discussions` redirects to `https://www.coursera.org/learn/r-programming/discussions/weeks/1`, so users visiting the first URL will fire 2 pageview events.

Duplicate Pageviews

Sometimes multiple pageviews may fire for the same base URL (after stripping query parameter and the hash fragment). This is because we may redirect to a page with a different query parameter, which causes an extra pageview. Additionally, there may have been instrumentation errors on our end.

How to Analyze

Raw clickstream data can be difficult to work with in its very fine granularity. This presents analytical challenges that we describe below. At a high level, the goal of this document is to outlines ways in which one can answer questions using clickstream data.

There are two common approaches to analyzing clickstream data: via a relational manner, or a non-relational manner.

Relational

Clickstream data may be loaded into a relational database. This approach can be very useful for a particular subset of common queries against the data (i.e., “I want to see all of this user’s video interactions in order”), and for many the familiarity and usability advantages will make this an attractive option. This also gives the ability to JOIN between clickstream data and other Coursera data sources.

However, the large size of the data may strain the default memory and time constraints imposed by such software, and fine tuning will likely be necessary to ensure that your preferred database software can handle the load.

Other types of problems are poorly served by a relational approach. For instance, to answer the question, “How long did a user spend watching a video?” one would need to calculate the difference between successive rows in the result set of the previous example, excluding unmatched play-pause pairs and using special logic to handle video seeking events.

To write this query, one would typically need an unwieldy array of self joins, aggregative functions and conditionals. Beyond being difficult to describe, such queries would typically be slow without fine tuning. Faced with this kind of question, one can turn to non relational solutions.

Non Relational

R

Example parsing clickstream data.

```
library(tidyverse)

# Read data and expand out JSON field as columns in a data frame
video_df_unparsed_json_value <- read_delim("~/video-example.gz", ",",
                                col_names = c("user_id",
                                              "hashed_session_cookie_id",
                                              "server_timestamp",
                                              "hashed_ip",
                                              "user_agent",
                                              "url",
                                              "initial_referrer_url",
                                              "browser_language",
                                              "course_id",
                                              "country_cd",
                                              "region_cd",
                                              "timezone",
                                              "os",
                                              "browser",
                                              "key",
                                              "value"),
                                escape_double = FALSE,
                                escape_backslash = TRUE)
json_value_df <- data.frame(jsoncol = video_df_unparsed_json_value$value, stringsAsFac
tors=F)
parsed_json_value_df <- do.call(rbind.data.frame, lapply(json_value_df$jsoncol, FUN=fu
nction(x){ as.list(jsonlite::fromJSON(x))}))
parsed_json_value_df$timecode <- as.double(as.character(parsed_json_value_df$timecode)
)
video_with_expanded_json <- cbind(video_df_unparsed_json_value, parsed_json_value_df)
```

Here, data processing is done in a programmatic fashion using general purpose computation frameworks.

MapReduce

MapReduce is a common approach: rather than attempt to handle all of the data at once, extract lines from the data file one at a time, parse the line into a data structure, and apply a map function to this structure. The map function can perform any necessary processing, and then group and store the results in file or in-memory stores until input completes.

Once all events in the file have been processed, a reduce function then takes the stored results and performs another set of operations on the resultant groups to combine them into a usable output.

MapReduce is very flexible, and Hadoop is an open source implemention of its processing model. On top of Hadoop exists a multitude of wrappers and tools advertising ease of development, expressibility, testability, etc. Examples include Cascading, and Pig.

Apache Spark

MapReduce is done with disk storage as the intermediate storage. This allows for scalability at the cost of speed. Spark is a is a fast and general engine for large-scale data processing. Spark takes advantage of the orders of magnitude speed increase when using memory as compared to disk.

Coursera has not worked with Spark in a production setting, so our experiences here is limited. Any contributions to this documentation would be appreciated.

Frequently Asked Questions

Can I query for the same learners across courses on the new platform?

Yes, when using data exports across multiple courses, the user ID's across the different CSV files are consistent for one learner.

As an example, if your Specialization consists of four launched courses and an upcoming capstone, you could:

1. Generate a data export for each of the courses
2. Unzip each export, and view/load each course_grades.csv
3. Query for those learners who reached a passing state in each course
4. Query to count the number of learners who have passed all four courses, who will be enabled to enroll in the upcoming capstone

What data is absent from data exports on the New Platform?

Here are some data sources which we have not included in the data exports.

- Mobile Clickstream Data: Data exports do not provide clickstream data for Coursera's iOS and Android application. If you wish for mobile clickstream data exports to be priortized, please contact us at data-support@coursera.org.

Changelog

As this guide updates, the list below describes the major changes.

May 2017

- The Learner Tables section was updated to reflect the addition of three columns of employment status, educational attainment, and student status.
- For exports after 2017-05-18, all activity from all mentors are added. Exports before this data included only mentors that started as learners in the course. This update will include data for those users, usually on-campus TAs, who jumped directly to a mentor role in the course without ever being a learner.

January 2017

- The Learner Tables section was updated to reflect the addition of Gender data.
- For exports after 2017-01-24, all activity data originating from instructors' Coursera accounts are now added. The table with the most impact was **discussion_questions**, with the addition of discussion threads and discussion prompts that an instructor had authored.

November 2016

- Coursera released new types of discussion forums: Mentor Forums, Custom Forums, and Subforums. The existing Course Forums were also upgraded, which results in a minor data model change. The older **discussion_forums** table is removed and is replaced with a newer **discussion_course_forums** table, which operates in the same manner. The Discussion Tables section has been updated.

August 2016

- A section on Clickstream Data was added to document the now available clickstream exports.
- A section on Command Line Access was added to describe a command line library for interacting with Coursera research exports.
- A section on research export API was added to document Coursera Data Exports API.
- SQL scripts are now included with data export for setting up a Postgres database (see Loading Data Exports for further usage).

- The **assessment_scopes** table, which was always empty, has been removed.
- The chapter on Peer Review Assignments Table was expanded to include screenshots from a course and in-depth details about the submission, review, and scoring process in peer review assignments.

July 2016

- This guide was updated to reflect the Course Versioning feature. This changelog article provides for more details for existing users of Data Exports, especially on tables marked for deprecation. The Course Content Tables and Course Grades Tables sections were changed, and a few other sections had minor updates.

April 2016

- This guide was updated with a new chapter on Summary Tables and its first section about the Course Certificate Payments export.

March 2016

- The first version of this guide was written and released during the Research Track event at Coursera's 2016 Partners Conference.

Course Versioning (Jul 2016)

Your data exports will change due to Course Versioning regardless of whether this feature is used in any of your courses. See below for table changes and deprecations.

Course Versioning is a new Coursera feature launched in early 2016 that allows instructors to create multiple versions, or “branches” as we call them in the data model, of their course. More information can found in our Partner Help Center article on Course Versioning.

This feature comes with a few table changes and additions, which we have detailed in subsequent sections:

1. a new table, **course_branches**
2. a new column, `course_branch_id`, in the **on_demand_sessions** table
3. three new tables to replace existing course content tables:
    - **course_modules → course_branch_modules**
    - **course_lessons → course_branch_lessons**
    - **course_items → course_branch_items**
4. three new tables to replace existing assessments mapping tables:
    - **course_item_assessments → course_branch_item_assessments**
    - **course_item_peer_assignments → course_branch_item_peer_assignments**
    - **course_item_programming_assignments → course_branch_item_programming_assignments**
5. a new table, **course_branch_grades**

The existing **course_progress** table and the **course_item_grades** table have not changed.

1. New course_branches Table

The list of all course versions can be found in the **course_branches** table, which contains three columns:

1. `course_branch_id`: the unique id for a course version
2. `course_id`: the course tied to each branch
3. `course_branch_changes_description`: A user-facing summary of the changes between this branch and the previous branch, based on instructor's input.

When an instructor drafts a new course, Coursera creates an initial course version that is labeled as the original version. The `course_branch_id` for this course version is simply populated with the same value as the `course_id`. For every new course version created afterwards, the `course_branch_id` is populated with a prefix of "branch~" and a new alphanumeric suffix. Here is an illustration:

Version	course_id	course_branch_id
original version	abcdefghijklm123456789	abcdefghijklm123456789
version 2	abcdefghijklm123456789	branch~AqdxTg241aaikKsUGu4132
version 3	abcdefghijklm123456789	branch~aABr1r38535JadfAbasdwA

2. New Column in on_demand_sessions

Course versions can only be applied to session-based courses. The existing **on_demand_sessions** table lists a course's schedule of historic and future sessions. A new column, `course_branch_id`, is added to this table and denotes each session's course version.

When an instructor drafts a new course version, it is not automatically scheduled in future sessions. A new `course_branch_id` value exists in the **course_branches** table and not in the **on_demand_sessions** table. When the instructor assigns the course version to future session(s), the new `course_branch_id` value will appear in the **on_demand_sessions** table.

3. New Course Content Tables

The older **course_items** table contains the list of items (e.g. lectures, assessments, etc.) for a course. If your course does not use the Course Versioning feature, then the new **course_branch_items** table can be interpreted in the same way as before, though some column names have changed.

If your course does use versions, the **course_branch_items** table will have a list of items for each version. Consider a course on its original version of 5 modules, 10 lessons, and 40 items. The instructor then created a new course version that adds a new module containing two lessons and four items. The **course_branch_items** table would contain 84 rows of items that spans across the two versions.

course_branch_id	count_items
abcdefghijklm123456789	40
branch~AqdxTg241aaikKsUGu4132	44

Similarly, the **course_branch_lessons** table would contain 22 rows and the **course_branch_modules** table would contain 11 rows.

Of note, the older versions of these tables will only contain the number of items, lessons, and modules of the original version.

4. New Assessments Mapping Tables

The older **course_item_assessments** table provided the mapping between the `course_item_id` in the **course_items** table (and also the **course_branch_items** table) and the `assessment_id` column in the family of the **assessments** tables. The new **course_branch_item_assessments** table will provide the same information with additional course version data.

Consider an instructor that created the original version of a course with a quiz. The first saved assessment-version of that quiz had six questions, and then the instructor added two more questions and saved a new assessment-version. The **course_branch_item_assessments** was populated as such:

course_branch_id	course_item_id	assessment_id
abcdefghijklm123456789	Ye441	polkjhgfsdafasfzxcv@1
abcdefghijklm123456789	Ye441	polkjhgfsdafasfzxcv@2

The course then launched, and learners were tested on that quiz with eight questions. The **assessments** tables, such as **assessment_responses**, were populated with the `assessment_id` value of "polkjhgfsdafasfzxcv@2" to record learner activity with this quiz.

After a few sessions, the instructor decided to use the Course Versioning feature. In a new course version, the instructor removed a confusing question and also added five additional questions. The **course_branch_item_assessments** was populated as such:

course_branch_id	course_item_id	assessment_id
abcdefghijklm123456789	Ye441	polkjhgfsdafasfzxcv@1
abcdefghijklm123456789	Ye441	polkjhgfsdafasfzxcv@2
branch~AqdxTg241aaikKsUGu4132	Ye441	qazwscefvFreVatereA@1

This illustration shows that the `course_item_id` value did not change in the third row, and there is a new `course_branch_id` value with a new `assessment_id` value. In the **assessment_questions** table, it would reflect the differing number of questions contained across those three `assessment_id` 's.

Similarly, the new **course_branch_item_peer_assignments** table maps the `course_item_id` of peer assignments to the family of **peer** tables. Lastly, the new **course_branch_item_programming_assignments** table maps the `course_item_id` of programming assignments to the family of **programming** tables.

Of note, the older versions of these tables will only contain the mapping of `course_item_id` to the original version of quizzes, peer assignments, and programming assignments.

5. New course_branch_grades Table

The existing **course_grades** table contains the data when a learner reached his or her highest grade for the course across all branches. A new table, **course_branch_grades**, contains the data when a learner reached his or her highest grade for each course version.

There will be no deprecation of the **course_grades** tables; both will be continue to be included in data exports.

In the **course_grades** table, the columns `course_grade_overall_passed_items` and `course_grade_verified_passed_items` currently only pertain to graded items in the original version of the course and not to any other versions. Coursera will either upgrade this column with better logic OR will deprecate and remove the two columns.