Appendix B: Relational Database Documentation

1 Introduction

The purpose of this appendix is to document the FDHI Database relational schema (or structure) and provide examples of how to navigate the database using both the SQLite command-line program and supporting libraries in Python. Most users of the database will prefer to access the contents through the flatfiles, which serve as a user-friendly documentation of the database contents; however, for completeness and legacy documentation, the relational database schema is described herein in Section 2 and example database queries are in Section 3. Software versioning is documented in Section 4; Section 5 presents a summary; and references are listed in Section 6.

As discussed in the main report, a relational database uses a defined schema to store different data types in individual tables and relate the data between tables using key fields. For this project, we sought an open-source relational management system with a wide range of programming language support (e.g., Matlab toolbox, Python or R libraries) that was compatible with multiple computer operating systems (Windows, Macintosh, and Linux). We also decided a serverless management system was more appropriate because the database would not require multiple users to simultaneously update or query data entries. Based on these criteria, the SQLite database engine (Hipp, 2020) was selected as the relational database management system. The Python "sqlite3" module was used to construct the relational database schema and populate the database. (See Section 4 for software versioning.)

2 Database Structure

Several different types of data are available to document historical surface-rupturing earthquakes. For this project, we broadly grouped the data types into four categories: earthquake information, rupture information, measurement information, and an event-specific coordinate system model (Figure B.1). Each category contains a variety of information, such as metadata, geospatial data, direct observations, analysis outputs, or interpretations. To efficiently collect and systematically manage the information, we developed a custom schema to hold the information and relationships or associations between the different types of information.

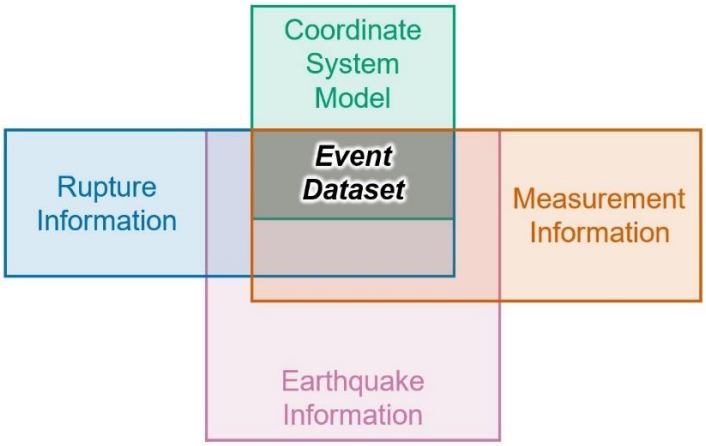

Figure B.1. Schematic showing four data-type categories that collectively describe an earthquake (event) dataset.

The core database structure is shown in the relational schema diagram on Figure B.2. Four tables are emphasized by the yellow shading in the diagram, corresponding to the four data type categories from Figure B.1: earthquake information ("METADATA_events"), rupture information ("RUPOBS_id"), measurement information ("PTOBS_id"), and an event-specific coordinate system model ("ECS_linepath"). Placeholder table names ("RUP_otherTables" and "PT_otherTables") are shown on Figure B.2 to illustrate the relationship between individual observation or interpretation tables and the core database structure. For clarity, the dataset lookup table ("METADATA_datasets"), which contains information on the "DS_ID" field, is not shown on Figure B.2 but is discussed in Section 2.1.1.

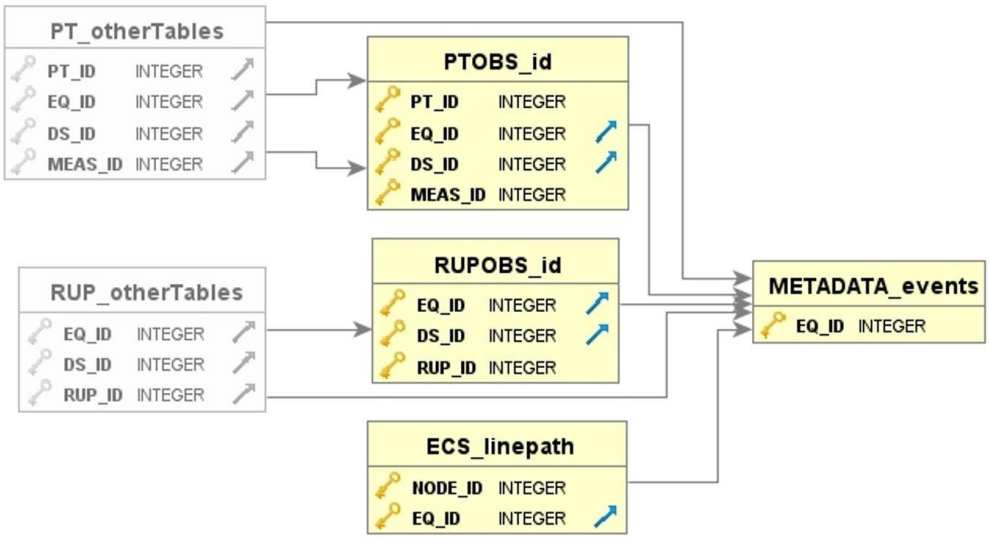

Figure B.2. Relational schema diagram showing the core FDHI Database structure.

The entire database contains 37 individual tables. Table B.1 lists the purpose and general contents of each table in the database. Table B.1 also identifies if the database table is a parent or child table and if the data are directly reported or derived. Table B.2 list every column in every table in the database and identifies primary keys and foreign key references. There are a total of 365 columns across 37 tables in the FDHI Database. The database schema can be viewed more easily online at https://fdhi.github.io/dbschema/ via the HTML documentation. The online documentation has interactive relational schema diagrams and the information in Tables B.1 and B.2.

For this project, we adopted an underscore naming convention for the tables and columns. The first part of a table name is always in capital letters and identifies the general purpose of the table [e.g., METADATA, RUPOBS (rupture observations), PTOBS (point/measurement observations)]. The second part of a table name is always in lower case letters and identifies the contents of the table [e.g., location, geology, fps (fault-parallel slip)]. Column names are formatted such that the primary keys are always in capital letters and all other column names are always in lower case letters.

Table B.1. Individual tables in the FDHI Database.

Database Table Name	Table Type	Data Type	Table Purpose/Contents
METADATA_events	Parent	Reported	Identifying data (EQ_ID) and event metadata
METADATA_datasets	Parent	Reported	Identifying data (DS_ID) and dataset metadata
METADATA_points	Child	Reported	Metadata for each event/measurement dataset combination
METADATA_ruptures	Child	Reported	Metadata for each event/rupture dataset combination
METADATA_geology	Child	Reported	Metadata for geologic map datasets
PTOBS_id	Child & Parent	Reported	Identifying data (PT_ID & MEAS_ID) and metadata on measurements
PTOBS_location	Child	Reported	Geographic coordinates for measurement sites
PTOBS_aftershock	Child	Reported	Aftershock flag for measurement sites
PTOBS_geology	Child	Derived	Geologic information for measurement sites
PTOBS_feature	Child	Reported	Information on the measured feature
PTOBS_fzw	Child	Reported	Fault zone width measurements
PTOBS_ads	Child	Reported	Dip slip component measurements
PTOBS_fps	Child	Reported	Fault-parallel slip component measurements
PTOBS_fns	Child	Reported	Fault-normal slip component measurements
PTOBS_nhs	Child	Reported	Net horizontal slip component measurements
PTOBS_vs	Child	Reported	Vertical slip component measurements
PTOBS_sh	Child	Reported	Scarp height measurements
PTOBS_tds	Child	Reported	Net (three-dimensional) slip component measurements
PTOBS_nsplays	Child	Reported	Number of fault splays in a measurement
PTOBS_confidence	Child	Reported	Dataset originator's confidence rating of measurement
PTOBS_compass	Child	Reported	Measurements other than slip measurements, such as strike, dip, and slip azimuth
PTOBS_slip_nonzero	Child	Reported	Flags for sites with unknown but nonzero slip
PTOBS_elevation	Child	Derived	Elevation, slope, terrain index, and terrain prominence data for measurement site
RUPOBS_id	Child & Parent	Reported	Identifying data (RUP_ID) and metadata on rupture lines
RUPOBS_location	Child & Parent	Reported	Identifying data (NODE_ID) and geographic coordinates for rupture line vertices
RUPOBS_aftershock	Child	Reported	Aftershock flag for rupture lines
RUPOBS_geology	Child	Reported	Geologic information for rupture line vertices
RUPOBS_length	Child	Reported	Length of rupture line
RUPOBS_confidence	Child	Reported	Dataset originator's mapping accuracy or confidence of rupture line
RUPOBS_faultname	Child	Reported	Dataset originator's fault name for rupture line
RUPOBS_sliprate	Child	Reported	Dataset originator's slip rate for rupture line

Database Table Name	Table Type	Data Type	Table Purpose/Contents
PAIRING_ptrup	Child	Derived	Information on closest rupture line for measurement sites
INTERPRETATIONS_pts	Child	Derived	Interpreted information and recommendations on measurements
INTERPRETATIONS_rups	Child	Derived	Interpreted information on rupture lines
ECS_linepath	Child	Derived	Geographic coordinates for ECS line path
ECS_pts	Child	Derived	ECS ordinates (u,t) for measurement sites
ECS_rups	Child	Derived	ECS ordinates (u,t) for rupture line vertices

Table B.2. Individual columns in the FDHI Database.

Table Name	Column Name	Data Type	Primary Key	Foreign Key	Foreign Key References (Column, Table)
METADATA_events	EQ_ID	INTEGER	TRUE	FALSE	n/a
METADATA_events	citation_for_hypocenter	TEXT	FALSE	FALSE	n/a
METADATA_events	citation_for_magnitude	TEXT	FALSE	FALSE	n/a
METADATA_events	citation_for_style	TEXT	FALSE	FALSE	n/a
METADATA_events	epsg_for_analysis	INTEGER	FALSE	FALSE	n/a
METADATA_events	eq_date	DATETIME	FALSE	FALSE	n/a
METADATA_events	eq_name	TEXT	FALSE	FALSE	n/a
METADATA_events	hypocenter_depth_km	FLOAT	FALSE	FALSE	n/a
METADATA_events	hypocenter_latitude_degrees	FLOAT	FALSE	FALSE	n/a
METADATA_events	hypocenter_longitude_degrees	FLOAT	FALSE	FALSE	n/a
METADATA_events	magnitude	FLOAT	FALSE	FALSE	n/a
METADATA_events	magnitude_type	TEXT	FALSE	FALSE	n/a
METADATA_events	multi_event_flag	BOOLEAN	FALSE	FALSE	n/a
METADATA_events	region	TEXT	FALSE	FALSE	n/a
METADATA_events	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
METADATA_events	row_entry_source_file	TEXT	FALSE	FALSE	n/a
METADATA_events	seismic_moment_dyncm	TEXT	FALSE	FALSE	n/a
METADATA_events	style	INTEGER	FALSE	FALSE	n/a
METADATA_datasets	DS_ID	INTEGER	TRUE	FALSE	n/a
METADATA_datasets	citation	TEXT	FALSE	FALSE	n/a
METADATA_datasets	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
METADATA_datasets	row_entry_source_file	TEXT	FALSE	FALSE	n/a
METADATA_points	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
METADATA_points	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
METADATA_points	completion	TEXT	FALSE	FALSE	n/a
METADATA_points	epsg_original	INTEGER	FALSE	FALSE	n/a
METADATA_points	mapping_scale	TEXT	FALSE	FALSE	n/a
METADATA_points	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
METADATA_points	row_entry_source_file	TEXT	FALSE	FALSE	n/a

Table Name	Column Name	Data Type	Primary Key	Foreign Key	Foreign Key References (Column, Table)
METADATA_ruptures	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
METADATA_ruptures	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
METADATA_ruptures	completion	TEXT	FALSE	FALSE	n/a
METADATA_ruptures	epsg_original	INTEGER	FALSE	FALSE	n/a
METADATA_ruptures	location_basis	TEXT	FALSE	FALSE	n/a
METADATA_ruptures	mapping_scale	TEXT	FALSE	FALSE	n/a
METADATA_ruptures	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
METADATA_ruptures	row_entry_source_file	TEXT	FALSE	FALSE	n/a
METADATA_geology	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
METADATA_geology	completion	TEXT	FALSE	FALSE	n/a
METADATA_geology	epsg_original	INTEGER	FALSE	FALSE	n/a
METADATA_geology	map_scale	TEXT	FALSE	FALSE	n/a
METADATA_geology	region	TEXT	FALSE	FALSE	n/a
METADATA_geology	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
METADATA_geology	row_entry_source_file	TEXT	FALSE	FALSE	n/a
PTOBS_id	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
PTOBS_id	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
PTOBS_id	PT_ID	INTEGER	TRUE	FALSE	n/a
PTOBS_id	MEAS_ID	INTEGER	TRUE	FALSE	n/a
PTOBS_id	aperture_max_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_id	days_elapsed	INTEGER	FALSE	FALSE	n/a
PTOBS_id	location_basis	TEXT	FALSE	FALSE	n/a
PTOBS_id	measurement_uncertainty_type	TEXT	FALSE	FALSE	n/a
PTOBS_id	obs_date_approx_flag	BOOLEAN	FALSE	FALSE	n/a
PTOBS_id	obs_mody	INTEGER	FALSE	FALSE	n/a
PTOBS_id	obs_year	INTEGER	FALSE	FALSE	n/a
PTOBS_id	originator_id	TEXT	FALSE	FALSE	n/a
PTOBS_id	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
PTOBS_id	row_entry_source_file	TEXT	FALSE	FALSE	n/a
PTOBS_location	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
PTOBS_location	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)

Table Name	Column Name	Data Type	Primary Key	Foreign Key	Foreign Key References (Column, Table)
PTOBS_location	PT_ID	INTEGER	TRUE	TRUE	(PT_ID, PTOBS_id)
PTOBS_location	latitude_degrees	FLOAT	FALSE	FALSE	n/a
PTOBS_location	longitude_degrees	FLOAT	FALSE	FALSE	n/a
PTOBS_location	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
PTOBS_location	row_entry_source_file	TEXT	FALSE	FALSE	n/a
PTOBS_aftershock	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
PTOBS_aftershock	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
PTOBS_aftershock	PT_ID	INTEGER	TRUE	TRUE	(PT_ID, PTOBS_id)
PTOBS_aftershock	MEAS_ID	INTEGER	TRUE	TRUE	(MEAS_ID, PTOBS_id)
PTOBS_aftershock	aftershock_flag	BOOLEAN	FALSE	FALSE	n/a
PTOBS_aftershock	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
PTOBS_aftershock	row_entry_source_file	TEXT	FALSE	FALSE	n/a
PTOBS_geology	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
PTOBS_geology	PT_DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
PTOBS_geology	GEO_DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_geology)
PTOBS_geology	PT_ID	INTEGER	TRUE	TRUE	(PT_ID, PTOBS_id)
PTOBS_geology	distance_to_bedrock_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_geology	existing_scarp	INTEGER	FALSE	FALSE	n/a
PTOBS_geology	geology	TEXT	FALSE	FALSE	n/a
PTOBS_geology	lithology	TEXT	FALSE	FALSE	n/a
PTOBS_geology	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
PTOBS_geology	row_entry_source_file	TEXT	FALSE	FALSE	n/a
PTOBS_geology	structure	TEXT	FALSE	FALSE	n/a
PTOBS_geology	unit_age	TEXT	FALSE	FALSE	n/a
PTOBS_feature	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
PTOBS_feature	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
PTOBS_feature	PT_ID	INTEGER	TRUE	TRUE	(PT_ID, PTOBS_id)
PTOBS_feature	MEAS_ID	INTEGER	TRUE	TRUE	(MEAS_ID, PTOBS_id)
PTOBS_feature	measured_feature	TEXT	FALSE	FALSE	n/a
PTOBS_feature	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
PTOBS_feature	row_entry_source_file	TEXT	FALSE	FALSE	n/a

Table Name	Column Name	Data Type	Primary Key	Foreign Key	Foreign Key References (Column, Table)
PTOBS_fzw	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
PTOBS_fzw	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
PTOBS_fzw	PT_ID	INTEGER	TRUE	TRUE	(PT_ID, PTOBS_id)
PTOBS_fzw	MEAS_ID	INTEGER	TRUE	TRUE	(MEAS_ID, PTOBS_id)
PTOBS_fzw	fzw_central_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_fzw	fzw_high_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_fzw	fzw_low_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_fzw	fzw_meas_type	TEXT	FALSE	FALSE	n/a
PTOBS_fzw	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
PTOBS_fzw	row_entry_source_file	TEXT	FALSE	FALSE	n/a
PTOBS_ads	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
PTOBS_ads	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
PTOBS_ads	PT_ID	INTEGER	TRUE	TRUE	(PT_ID, PTOBS_id)
PTOBS_ads	MEAS_ID	INTEGER	TRUE	TRUE	(MEAS_ID, PTOBS_id)
PTOBS_ads	ads_central_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_ads	ads_downside	TEXT	FALSE	FALSE	n/a
PTOBS_ads	ads_high_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_ads	ads_low_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_ads	ads_meas_type	TEXT	FALSE	FALSE	n/a
PTOBS_ads	ads_style	TEXT	FALSE	FALSE	n/a
PTOBS_ads	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
PTOBS_ads	row_entry_source_file	TEXT	FALSE	FALSE	n/a
PTOBS_fps	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
PTOBS_fps	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
PTOBS_fps	PT_ID	INTEGER	TRUE	TRUE	(PT_ID, PTOBS_id)
PTOBS_fps	MEAS_ID	INTEGER	TRUE	TRUE	(MEAS_ID, PTOBS_id)
PTOBS_fps	fps_central_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_fps	fps_high_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_fps	fps_low_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_fps	fps_meas_type	TEXT	FALSE	FALSE	n/a
PTOBS_fps	fps_style	TEXT	FALSE	FALSE	n/a

Table Name	Column Name	Data Type	Primary Key	Foreign Key	Foreign Key References (Column, Table)
PTOBS_fps	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
PTOBS_fps	row_entry_source_file	TEXT	FALSE	FALSE	n/a
PTOBS_fns	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
PTOBS_fns	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
PTOBS_fns	PT_ID	INTEGER	TRUE	TRUE	(PT_ID, PTOBS_id)
PTOBS_fns	MEAS_ID	INTEGER	TRUE	TRUE	(MEAS_ID, PTOBS_id)
PTOBS_fns	fns_central_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_fns	fns_high_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_fns	fns_low_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_fns	fns_meas_type	TEXT	FALSE	FALSE	n/a
PTOBS_fns	fns_style	TEXT	FALSE	FALSE	n/a
PTOBS_fns	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
PTOBS_fns	row_entry_source_file	TEXT	FALSE	FALSE	n/a
PTOBS_nhs	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
PTOBS_nhs	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
PTOBS_nhs	PT_ID	INTEGER	TRUE	TRUE	(PT_ID, PTOBS_id)
PTOBS_nhs	MEAS_ID	INTEGER	TRUE	TRUE	(MEAS_ID, PTOBS_id)
PTOBS_nhs	nhs_central_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_nhs	nhs_high_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_nhs	nhs_low_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_nhs	nhs_meas_type	TEXT	FALSE	FALSE	n/a
PTOBS_nhs	nhs_style	TEXT	FALSE	FALSE	n/a
PTOBS_nhs	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
PTOBS_nhs	row_entry_source_file	TEXT	FALSE	FALSE	n/a
PTOBS_vs	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
PTOBS_vs	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
PTOBS_vs	PT_ID	INTEGER	TRUE	TRUE	(PT_ID, PTOBS_id)
PTOBS_vs	MEAS_ID	INTEGER	TRUE	TRUE	(MEAS_ID, PTOBS_id)
PTOBS_vs	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
PTOBS_vs	row_entry_source_file	TEXT	FALSE	FALSE	n/a
PTOBS_vs	vs_central_meters	FLOAT	FALSE	FALSE	n/a

Table Name	Column Name	Data Type	Primary Key	Foreign Key	Foreign Key References (Column, Table)
PTOBS_vs	vs_downside	TEXT	FALSE	FALSE	n/a
PTOBS_vs	vs_high_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_vs	vs_low_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_vs	vs_meas_type	TEXT	FALSE	FALSE	n/a
PTOBS_vs	vs_style	TEXT	FALSE	FALSE	n/a
PTOBS_sh	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
PTOBS_sh	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
PTOBS_sh	PT_ID	INTEGER	TRUE	TRUE	(PT_ID, PTOBS_id)
PTOBS_sh	MEAS_ID	INTEGER	TRUE	TRUE	(MEAS_ID, PTOBS_id)
PTOBS_sh	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
PTOBS_sh	row_entry_source_file	TEXT	FALSE	FALSE	n/a
PTOBS_sh	sh_central_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_sh	sh_downside	TEXT	FALSE	FALSE	n/a
PTOBS_sh	sh_high_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_sh	sh_low_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_sh	sh_meas_type	TEXT	FALSE	FALSE	n/a
PTOBS_sh	sh_style	TEXT	FALSE	FALSE	n/a
PTOBS_tds	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
PTOBS_tds	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
PTOBS_tds	PT_ID	INTEGER	TRUE	TRUE	(PT_ID, PTOBS_id)
PTOBS_tds	MEAS_ID	INTEGER	TRUE	TRUE	(MEAS_ID, PTOBS_id)
PTOBS_tds	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
PTOBS_tds	row_entry_source_file	TEXT	FALSE	FALSE	n/a
PTOBS_tds	tds_central_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_tds	tds_high_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_tds	tds_low_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_tds	tds_meas_type	TEXT	FALSE	FALSE	n/a
PTOBS_tds	tds_style	TEXT	FALSE	FALSE	n/a
PTOBS_nsplays	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
PTOBS_nsplays	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
PTOBS_nsplays	PT_ID	INTEGER	TRUE	TRUE	(PT_ID, PTOBS_id)

Table Name	Column Name	Data Type	Primary Key	Foreign Key	Foreign Key References (Column, Table)
PTOBS_nsplays	MEAS_ID	INTEGER	TRUE	TRUE	(MEAS_ID, PTOBS_id)
PTOBS_nsplays	n_splays	TEXT	FALSE	FALSE	n/a
PTOBS_nsplays	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
PTOBS_nsplays	row_entry_source_file	TEXT	FALSE	FALSE	n/a
PTOBS_confidence	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
PTOBS_confidence	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
PTOBS_confidence	PT_ID	INTEGER	TRUE	TRUE	(PT_ID, PTOBS_id)
PTOBS_confidence	MEAS_ID	INTEGER	TRUE	TRUE	(MEAS_ID, PTOBS_id)
PTOBS_confidence	originator_quality_note	TEXT	FALSE	FALSE	n/a
PTOBS_confidence	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
PTOBS_confidence	row_entry_source_file	TEXT	FALSE	FALSE	n/a
PTOBS_compass	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
PTOBS_compass	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
PTOBS_compass	PT_ID	INTEGER	TRUE	TRUE	(PT_ID, PTOBS_id)
PTOBS_compass	MEAS_ID	INTEGER	TRUE	TRUE	(MEAS_ID, PTOBS_id)
PTOBS_compass	fault_dip_azimuth_central	FLOAT	FALSE	FALSE	n/a
PTOBS_compass	fault_dip_azimuth_high	FLOAT	FALSE	FALSE	n/a
PTOBS_compass	fault_dip_azimuth_low	FLOAT	FALSE	FALSE	n/a
PTOBS_compass	fault_dip_azimuth_meas_type	TEXT	FALSE	FALSE	n/a
PTOBS_compass	fault_dip_central	FLOAT	FALSE	FALSE	n/a
PTOBS_compass	fault_dip_high	FLOAT	FALSE	FALSE	n/a
PTOBS_compass	fault_dip_low	FLOAT	FALSE	FALSE	n/a
PTOBS_compass	fault_dip_meas_type	TEXT	FALSE	FALSE	n/a
PTOBS_compass	fault_strike_central	FLOAT	FALSE	FALSE	n/a
PTOBS_compass	fault_strike_high	FLOAT	FALSE	FALSE	n/a
PTOBS_compass	fault_strike_low	FLOAT	FALSE	FALSE	n/a
PTOBS_compass	fault_strike_meas_type	TEXT	FALSE	FALSE	n/a
PTOBS_compass	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
PTOBS_compass	row_entry_source_file	TEXT	FALSE	FALSE	n/a
PTOBS_compass	slip_azimuth_central	FLOAT	FALSE	FALSE	n/a
PTOBS_compass	slip_azimuth_high	FLOAT	FALSE	FALSE	n/a

Table Name	Column Name	Data Type	Primary Key	Foreign Key	Foreign Key References (Column, Table)
PTOBS_compass	slip_azimuth_low	FLOAT	FALSE	FALSE	n/a
PTOBS_compass	slip_azimuth_meas_type	TEXT	FALSE	FALSE	n/a
PTOBS_compass	slip_plunge_central	FLOAT	FALSE	FALSE	n/a
PTOBS_compass	slip_plunge_high	FLOAT	FALSE	FALSE	n/a
PTOBS_compass	slip_plunge_low	FLOAT	FALSE	FALSE	n/a
PTOBS_compass	slip_plunge_meas_type	TEXT	FALSE	FALSE	n/a
PTOBS_compass	slip_rake_central	FLOAT	FALSE	FALSE	n/a
PTOBS_compass	slip_rake_high	FLOAT	FALSE	FALSE	n/a
PTOBS_compass	slip_rake_low	FLOAT	FALSE	FALSE	n/a
PTOBS_compass	slip_rake_meas_type	TEXT	FALSE	FALSE	n/a
PTOBS_slip_nonzero	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
PTOBS_slip_nonzero	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
PTOBS_slip_nonzero	PT_ID	INTEGER	TRUE	TRUE	(PT_ID, PTOBS_id)
PTOBS_slip_nonzero	MEAS_ID	INTEGER	TRUE	TRUE	(MEAS_ID, PTOBS_id)
PTOBS_slip_nonzero	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
PTOBS_slip_nonzero	row_entry_source_file	TEXT	FALSE	FALSE	n/a
PTOBS_slip_nonzero	slip_unknown_lateral_nonzero	BOOLEAN	FALSE	FALSE	n/a
PTOBS_slip_nonzero	slip_unknown_unspecified_nonzero	BOOLEAN	FALSE	FALSE	n/a
PTOBS_slip_nonzero	slip_unknown_vertical_nonzero	BOOLEAN	FALSE	FALSE	n/a
PTOBS_elevation	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
PTOBS_elevation	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
PTOBS_elevation	PT_ID	INTEGER	TRUE	TRUE	(PT_ID, PTOBS_id)
PTOBS_elevation	elevation_meters	FLOAT	FALSE	FALSE	n/a
PTOBS_elevation	prominence_1000_meter_radius	FLOAT	FALSE	FALSE	n/a
PTOBS_elevation	prominence_125_meter_radius	FLOAT	FALSE	FALSE	n/a
PTOBS_elevation	prominence_250_meter_radius	FLOAT	FALSE	FALSE	n/a
PTOBS_elevation	prominence_500_meter_radius	FLOAT	FALSE	FALSE	n/a
PTOBS_elevation	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
PTOBS_elevation	row_entry_source_file	TEXT	FALSE	FALSE	n/a
PTOBS_elevation	slope_percent_gaussian_gradient	FLOAT	FALSE	FALSE	n/a
PTOBS_elevation	slope_percent_gdal_horns_formula	FLOAT	FALSE	FALSE	n/a

Table Name	Column Name	Data Type	Primary Key	Foreign Key	Foreign Key References (Column, Table)
PTOBS_elevation	terrain_class	INTEGER	FALSE	FALSE	n/a
PTOBS_elevation	terrain_roughness	INTEGER	FALSE	FALSE	n/a
PTOBS_elevation	terrain_ruggedness_index	FLOAT	FALSE	FALSE	n/a
PTOBS_elevation	topo_position_index	FLOAT	FALSE	FALSE	n/a
RUPOBS_id	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
RUPOBS_id	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
RUPOBS_id	RUP_ID	INTEGER	TRUE	FALSE	n/a
RUPOBS_id	observation_basis	TEXT	FALSE	FALSE	n/a
RUPOBS_id	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
RUPOBS_id	row_entry_source_file	TEXT	FALSE	FALSE	n/a
RUPOBS_location	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
RUPOBS_location	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
RUPOBS_location	RUP_ID	INTEGER	TRUE	TRUE	(RUP_ID, RUPOBS_id)
RUPOBS_location	NODE_ID	INTEGER	TRUE	FALSE	n/a
RUPOBS_location	latitude_degrees	FLOAT	FALSE	FALSE	n/a
RUPOBS_location	longitude_degrees	FLOAT	FALSE	FALSE	n/a
RUPOBS_location	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
RUPOBS_location	row_entry_source_file	TEXT	FALSE	FALSE	n/a
RUPOBS_aftershock	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
RUPOBS_aftershock	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
RUPOBS_aftershock	RUP_ID	INTEGER	TRUE	TRUE	(RUP_ID, RUPOBS_id)
RUPOBS_aftershock	aftershock_flag	BOOLEAN	FALSE	FALSE	n/a
RUPOBS_aftershock	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
RUPOBS_aftershock	row_entry_source_file	TEXT	FALSE	FALSE	n/a
RUPOBS_geology	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
RUPOBS_geology	RUP_DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
RUPOBS_geology	GEO_DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_geology)
RUPOBS_geology	RUP_ID	INTEGER	TRUE	TRUE	(RUP_ID, RUPOBS_id)
RUPOBS_geology	NODE_ID	INTEGER	TRUE	TRUE	(NODE_ID, RUPOBS_location)
RUPOBS_geology	geology	TEXT	FALSE	FALSE	n/a
RUPOBS_geology	lithology	TEXT	FALSE	FALSE	n/a

Table Name	Column Name	Data Type	Primary Key	Foreign Key	Foreign Key References (Column, Table)
RUPOBS_geology	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
RUPOBS_geology	row_entry_source_file	TEXT	FALSE	FALSE	n/a
RUPOBS_geology	unit_age	TEXT	FALSE	FALSE	n/a
RUPOBS_length	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
RUPOBS_length	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
RUPOBS_length	RUP_ID	INTEGER	TRUE	TRUE	(RUP_ID, RUPOBS_id)
RUPOBS_length	line_length_meters	TEXT	FALSE	FALSE	n/a
RUPOBS_length	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
RUPOBS_length	row_entry_source_file	TEXT	FALSE	FALSE	n/a
RUPOBS_confidence	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
RUPOBS_confidence	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
RUPOBS_confidence	RUP_ID	INTEGER	TRUE	TRUE	(RUP_ID, RUPOBS_id)
RUPOBS_confidence	confidence	TEXT	FALSE	FALSE	n/a
RUPOBS_confidence	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
RUPOBS_confidence	row_entry_source_file	TEXT	FALSE	FALSE	n/a
RUPOBS_faultname	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
RUPOBS_faultname	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
RUPOBS_faultname	RUP_ID	INTEGER	TRUE	TRUE	(RUP_ID, RUPOBS_id)
RUPOBS_faultname	fault_name	TEXT	FALSE	FALSE	n/a
RUPOBS_faultname	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
RUPOBS_faultname	row_entry_source_file	TEXT	FALSE	FALSE	n/a
RUPOBS_sliprate	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
RUPOBS_sliprate	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
RUPOBS_sliprate	RUP_ID	INTEGER	TRUE	TRUE	(RUP_ID, RUPOBS_id)
RUPOBS_sliprate	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
RUPOBS_sliprate	row_entry_source_file	TEXT	FALSE	FALSE	n/a
RUPOBS_sliprate	slip_rate	TEXT	FALSE	FALSE	n/a
PAIRING_ptrup	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
PAIRING_ptrup	PT_DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
PAIRING_ptrup	RUP_DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
PAIRING_ptrup	PT_ID	INTEGER	TRUE	TRUE	(PT_ID, PTOBS_id)

Table Name	Column Name	Data Type	Primary Key	Foreign Key	Foreign Key References (Column, Table)
PAIRING_ptrup	RUP_ID	INTEGER	FALSE	TRUE	(RUP_ID, RUPOBS_id)
PAIRING_ptrup	distance_to_rupture_meters	FLOAT	FALSE	FALSE	n/a
PAIRING_ptrup	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
PAIRING_ptrup	row_entry_source_file	TEXT	FALSE	FALSE	n/a
INTERPRETATIONS_pts	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
INTERPRETATIONS_pts	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
INTERPRETATIONS_pts	PT_ID	INTEGER	TRUE	TRUE	(PT_ID, PTOBS_id)
INTERPRETATIONS_pts	MEAS_ID	INTEGER	TRUE	TRUE	(MEAS_ID, PTOBS_id)
INTERPRETATIONS_pts	hwfw_flag	TEXT	FALSE	FALSE	n/a
INTERPRETATIONS_pts	group_id	TEXT	FALSE	FALSE	n/a
INTERPRETATIONS_pts	location_id	INTEGER	FALSE	FALSE	n/a
INTERPRETATIONS_pts	measurement_category	TEXT	FALSE	FALSE	n/a
INTERPRETATIONS_pts	quality_code	INTEGER	FALSE	FALSE	n/a
INTERPRETATIONS_pts	rank	TEXT	FALSE	FALSE	n/a
INTERPRETATIONS_pts	rank_alternative_europe	TEXT	FALSE	FALSE	n/a
INTERPRETATIONS_pts	rank_confidence	INTEGER	FALSE	FALSE	n/a
INTERPRETATIONS_pts	recommended_net_high_for_analysis_meters	FLOAT	FALSE	FALSE	n/a
INTERPRETATIONS_pts	recommended_net_low_for_analysis_meters	FLOAT	FALSE	FALSE	n/a
INTERPRETATIONS_pts	recommended_net_preferred_for_analysis_meters	FLOAT	FALSE	FALSE	n/a
INTERPRETATIONS_pts	recommended_net_preferred_notes	TEXT	FALSE	FALSE	n/a
INTERPRETATIONS_pts	recommended_net_preferred_usage_flag	TEXT	FALSE	FALSE	n/a
INTERPRETATIONS_pts	recommended_net_preferred_vector_basis	TEXT	FALSE	FALSE	n/a
INTERPRETATIONS_pts	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
INTERPRETATIONS_pts	row_entry_source_file	TEXT	FALSE	FALSE	n/a
INTERPRETATIONS_pts	rupture_rank	TEXT	FALSE	FALSE	n/a

Table Name	Column Name	Data Type	Primary Key	Foreign Key	Foreign Key References (Column, Table)
INTERPRETATIONS_rups	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
INTERPRETATIONS_rups	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
INTERPRETATIONS_rups	RUP_ID	INTEGER	TRUE	TRUE	(RUP_ID, RUPOBS_id)
INTERPRETATIONS_rups	rank	TEXT	FALSE	FALSE	n/a
INTERPRETATIONS_rups	rank_confidence	INTEGER	FALSE	FALSE	n/a
INTERPRETATIONS_rups	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
INTERPRETATIONS_rups	row_entry_source_file	TEXT	FALSE	FALSE	n/a
ECS_linepath	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
ECS_linepath	NODE_ID	INTEGER	TRUE	FALSE	n/a
ECS_linepath	latitude_degrees	FLOAT	FALSE	FALSE	n/a
ECS_linepath	longitude_degrees	FLOAT	FALSE	FALSE	n/a
ECS_linepath	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
ECS_linepath	row_entry_source_file	TEXT	FALSE	FALSE	n/a
ECS_linepath	t	FLOAT	FALSE	FALSE	n/a
ECS_linepath	u	FLOAT	FALSE	FALSE	n/a
ECS_linepath	utm_x	FLOAT	FALSE	FALSE	n/a
ECS_linepath	utm_y	FLOAT	FALSE	FALSE	n/a
ECS_pts	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
ECS_pts	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
ECS_pts	PT_ID	INTEGER	TRUE	TRUE	(PT_ID, PTOBS_id)
ECS_pts	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
ECS_pts	row_entry_source_file	TEXT	FALSE	FALSE	n/a
ECS_pts	t	FLOAT	FALSE	FALSE	n/a
ECS_pts	u	FLOAT	FALSE	FALSE	n/a
ECS_rups	EQ_ID	INTEGER	TRUE	TRUE	(EQ_ID, METADATA_events)
ECS_rups	DS_ID	INTEGER	TRUE	TRUE	(DS_ID, METADATA_datasets)
ECS_rups	RUP_ID	INTEGER	TRUE	TRUE	(RUP_ID, RUPOBS_id)
ECS_rups	NODE_ID	INTEGER	TRUE	TRUE	(NODE_ID, RUPOBS_location)
ECS_rups	row_entry_date_updated	DATETIME	FALSE	FALSE	n/a
ECS_rups	row_entry_source_file	TEXT	FALSE	FALSE	n/a
ECS_rups	t	FLOAT	FALSE	FALSE	n/a

Table Name	Column Name	Data Type	Primary Key	Foreign Key	Foreign Key References (Column, Table)
ECS_rups	u	FLOAT	FALSE	FALSE	n/a

2.1 METADATA TABLES

There are five metadata tables in the FDHI Database:

- METADATA_events
- METADATA_datasets
- METADATA_points
- METADATA_ruptures
- METADATA_geology

The "METADATA_events" table stores the earthquake information, and it is one of five parent tables in the database because the unique event identifier ("EQ_ID") is first assigned in this table. The “EQ_ID” column is in 35 of the 37 tables in the FDHI Database and serves as a primary key in those tables, as well as a foreign key references the “EQ_ID” column in the “METADATA_events” table. Two tables in the database do not use an “EQ_ID” column because the contents are unrelated to a specific earthquake ("METADATA_datasets" and “METADATA_geology”).

The other metadata tables store dataset-specific and event-specific information. The "METADATA_datasets" table stores the dataset information, and it is another parent table in the database because the unique dataset identifier ("DS_ID") is first assigned in this table (Table B.1; also https://fdhi.github.io/dbschema/). This information is separated from the earthquake information because some datasets report information for multiple earthquakes, and some datasets do not contain earthquake-specific information (e.g., geologic maps). The “METADATA_points” and “METADATA_ruptures” tables store information specific to each dataset-event combination; therefore, these tables use the “EQ_ID” and “DS_ID” columns together as composite primary keys. The “METADATA_geology” table stores information on the geologic dataset and is independent of the events.

2.2 OBSERVATION-BASED DATA TABLES

There are 26 observation-based data tables in the FDHI Database: eight for rupture data and 18 for point/measurement data. These children tables with their respective parent tables, “RUPOBS_id” and “PTOBS_id”.

The "RUPOBS_id" table assigns a unique identifier ("RUP_ID") that is carried throughout other rupture information tables. There are seven other rupture information tables, represented by the placeholder "RUP_otherTables" on Figure B.2. The purpose and contents of the other rupture information tables are listed in Table B.2. All rupture observation tables at minimum use the “EQ_ID”, “DS_ID”, and “RUP_ID” columns together as composite primary keys. Some rupture

observation tables also use “NODE_ID” as an additional primary key if the table contains information on each vertex in a rupture line.

Similarly, the "PTOBS_id" table assigns unique identifiers ("PT_ID" and "MEAS_ID") that are carried throughout 17 other measurement observation tables, represented by the placeholder "PT_otherTables" on Figure B.2. The purpose and contents of the other rupture information tables are listed in Table B.2. Two unique identifiers ("PT_ID" and "MEAS_ID") are required because some datasets report multiple measurements (“MEAS_ID”) at the same site (“PT_ID”). All point/measurement observation tables use the “EQ_ID”, “DS_ID”, “PT_ID”, and “MEAS_ID” columns together as composite primary keys.

2.3 INTERPRETED DATA TABLES

There are three interpreted data tables in the FDHI Database:

- INTERPRETATIONS_rups
- INTERPRETATIONS_pts
- PAIRING_ptrup

The interpretation tables store the results of specific geologic assessments of the data that were requested by the model development teams. The assessments relate to rupture and point/measurement rank (principal, distributed, etc.), hangingwall/footwall flags for distributed measurements, and recommended net slip values/usage. Details bearing on the assessments are discussed in the main report. Composite primary keys for the “INTERPRETATIONS_rups” table are “EQ_ID”, “DS_ID”, and “RUP_ID”. Composite primary keys for the “INTERPRETATIONS_pts” tables are the same as the points/measurement observation tables (“EQ_ID”, “DS_ID”, “PT_ID”, and “MEAS_ID”).

The "PAIRING_ptrup" table stores derived information on the spatial relationships between measurement sites ("PT_ID") and mapped rupture linework ("RUP_ID"), considering the rank of the rupture line and measurement site. Details bearing on the geospatial analysis techniques applied to derive the paired information is discussion in the main report. The composite primary keys for this table are the "EQ_ID", "PT_DS_ID", "RUP_DS_ID", "PT_ID", and "RUP_ID" columns.

2.4 EVENT-SPECIFIC COORDINATE SYSTEM (ECS) TABLES

There are three tables in the FDHI Database that store information for the event-specific coordinate system (ECS). This information includes the location of the ECS line in geographic coordinates (“ECS_linepath” table) and the u-t coordinates of each measurement site and each node of each rupture vertex (“ECS_pts” and “ECS_rups” tables, respectively). The "ECS_linepath" table does

not create unique identifiers that are used in any other tables. The "ECS_pts" table uses the "EQ_ID", "DS_ID", "PT_ID", and "MEAS_ID" columns together as composite primary keys, and the "ECS_rups" table "EQ_ID", "DS_ID", "RUP_ID", and "NODE_ID" together as composite primary keys. Details bearing on the algorithm used to compute the ECS are discussed in the main report.

2.5 POPULATING THE DATABASE

The FDHI Database contents were prepared in separate data files and imported into the database according to the relational constraints enforced by the schema. This approach has been used in other similar projects, such as the NGA-Subduction Database (Mazzoni et al., 2020). The source data files included spreadsheets or text files with event and dataset metadata, flatfiles containing analysis results, and ESRI shapefiles with point/measurement and rupture line observations and interpretations. As discussed in the main report, utilizing geographic information system (GIS) software was necessary to perform geospatial analysis and geologic assessment of the data; therefore, most of the source data files were prepared in GIS software as ESRI shapefiles.

Contents of a relational database are added using Structured Query Language (SQL). The relational schema of a database enforces constraints on added data. These constraints are table- and column-specific and are defined when the table is created. The constraints can include data type, prohibiting null values, requiring unique values, and prohibiting a foreign key that has not been assigned in the parent table. Failure to comply with data entry constraints returns an error, which provides an element of quality assurance. For example, considering the FDHI Database schema (Table B.2; also https://fdhi.github.io/dbschema/, a rupture line ("RUP_ID") cannot be added to the "RUPOBS_id" table for EQ_ID = 92 if this earthquake identifier (92) is not assigned in the "METADATA_events" table. Consequently, the order in which data were imported into the database was considered in populating the database. For this project, custom Python functions were used with Python's "sqlite3" module to construct SQL statements for importing data from the source files into the appropriate tables in the FDHI Database.

3 Retrieving Database Contents

The FDHI Database *.db file is a binary file that can only be opened in software that can read the file format. We created the database in SQLite; therefore, software that can read the SQLite database file format is required to view or retrieve the contents of the database. Examples include the SQLite command-line program and graphical user interface (GUI) software or programming language-specific libraries compatible with the SQLite engine (see Section 4).

Contents of a relational database are both added and retrieved using Structured Query Language (SQL). Standard SQL is a declarative (or non-procedural) language, which means the syntax or commands are relatively simple logic statements and the database engine handles the execution procedure (or control flow). Different database engines (or relational management systems) provide various features that add some procedural functionality, but Standard SQL commands (e.g., *create table*, *insert into*, *select*, *from*, *where*, *group by*, *order by*, *left join*, *inner join*) are applicable in any database software. Retrieving the contents of any database requires knowledge of the database schema and writing SQL queries. Using a GUI to interact with a database can be helpful because most GUIs generate table relationship diagrams (e.g., https://fdhi.github.io/dbschema/) for specific tables or the entire database, and some GUIs also have point-and-click interfaces to help build SQL queries.

We recommend most users of the FDHI Database to use the flatfiles (which are in a user-friendly *.csv format) to access the contents of the database. For this project, custom Python functions were used with Python's "sqlite3" module and "pandas" library to construct SQL queries to retrieve data from individual tables in the database, join the data from multiple tables in the database, and aggregate the results into flatfiles. We used our knowledge of the database schema to produce the flatfiles and check for errors and inconsistencies. Therefore, the flatfiles are the formal documentation of the database contents that have been reviewed for accuracy and completeness. As described in the main report, three flatfiles were produced from the FDHI Database:

1. Measurements flatfile
2. Ruptures flatfile
3. Event-specific coordinate system (ECS) flatfile.

Three flatfiles were required to represent the three distinct information types contained in the database (Figure B.1).

For completeness and practical legacy documentation, the following sections provide simple examples on querying the FDHI Database structure and contents. We present two alternative software options to investigate the database. Section 3.1 uses the SQLite command-line program to work through the examples. The same examples are repeated in Section 3.2 using Python's "sqlite3" module and "pandas" library. The examples are not exhaustive, and SQL is a versatile tool in which there are often multiple query structures that will perform the same task. Writing effective queries depends on the desired outputs, knowledge of the database schema, and user-specific preferences on code readability.

In the following sections, it is assumed that the user has a basic working knowledge of SQL syntax, the Python language, and the Python's "pandas" data structures library. While SQL commands are not case sensitive and generally ignore whitespace, we follow the common convention of capitalizing SQL command keywords and splitting statements across multiple lines for readability.

3.1 EXAMPLE FDHI DATABASE QUERIES IN SQLITE COMMAND-LINE

The following subsections use the SQLite command-line program (see Section 4, Data and Resources) to show how to open the database, retrieve information on the database schema, and perform common SQL queries on the database. Additional information on the SQLite program can be found in the online documentation at https://sqlite.org/docs.html (Hipp, 2020). See Section 4 for software versioning.

3.1.1 Starting out: how to open the database in SQLite and investigate the database schema

This section shows how to open the database in the SQLite command-line program and investigate the database schema by printing information to the console or exporting the information to a text file. Most of the syntax in this section is specific to the SQLite program.

The following commands will open the FDHI Database in SQLite. The user will need to update their folder path and database name as needed. Comments are denoted with a dual dash symbol (--) and gray font. The comments are shown for clarity and do not need to be entered in the command line. An unusual feature of SQLite is that foreign key constraints must be explicitly activated (Hipp, 2020), as shown below. The other commands shown below are recommended to improve the readability of console outputs.

```
.open folderpath/FDHI_ph17_rev0.db
-- opens database in specific directory
-- alternatively can open in read only mode with this
```

```
-- syntax:
-- .open file:folderpath/FDHI_ph17_rev0.db?mode=ro

PRAGMA foreign_keys = ON;
-- per docs, foreign keys must still be enabled by the
-- application at runtime

.mode columns
.header on
-- optional, this makes console outputs easier to read

.nullvalue 'NULL'
-- specify how to display null values; default is 0
```

3.1.1.1 Example 1

The following query will print a list of the tables in the database to the console. The output is shown in blue font. (If the output does not print to console, the user can first enter the command "`.output stdout`" to reset the standard output.)

```
SELECT NAME
FROM sqlite_master
WHERE type = 'table'
;
-- print to console list of all tables in database
```

```
ECS_linepath           PTOBS_geology
ECS_pts                PTOBS_id
ECS_rups               PTOBS_location
INTERPRETATIONS_pts    PTOBS_nhs
INTERPRETATIONS_rups   PTOBS_nsplays
METADATA_datasets      PTOBS_sh
METADATA_events        PTOBS_slip_nonzero
METADATA_geology       PTOBS_elevation
METADATA_points        PTOBS_tds
METADATA_ruptures      PTOBS_vs
PAIRING_ptrup          RUPOBS_aftershock
PTOBS_ads              RUPOBS_confidence
PTOBS_aftershock       RUPOBS_faultname
PTOBS_compass          RUPOBS_geology
PTOBS_confidence       RUPOBS_id
PTOBS_feature          RUPOBS_length
PTOBS_fns              RUPOBS_location
PTOBS_fps              RUPOBS_sliprate
PTOBS_fzw...
```

Alternatively, SQLite has a built-in helper command "`.tables`" that can be used in lieu of the query to return the same result.

3.1.1.2 Example 2

The following command will retrieve the information of a specific table, including the column names, data types, null constraints, and primary keys; here we use the "METADATA_points" table as an example. The output is shown in blue font. Default column widths in the console output are set to 10 characters, so the names of some fields are truncated in the display.

```
PRAGMA table_info(METADATA_points);
-- print to console table information for specific table

cid   name        type       notnull   dflt_value  pk
----  ----------  ---------  --------  ----------  ---
0     completion  TEXT       0                     0
1     mapping_sc  TEXT       0                     0
2     DS_ID       INTEGER    1                     2
3     row_entry_  TEXT       0                     0
4     epsg_origi  INTEGER    0                     0
5     row_entry_  DATETIME   0                     0
6     EQ_ID       INTEGER    1                     1
```

3.1.1.3 Example 3

The following command will retrieve the foreign key constraints a specific table; here we use the "METADATA_points" table as an example. The output is shown in blue font.

```
PRAGMA foreign_key_list(METADATA_points);
-- print to console foreign key information for specific
table

id   seq   table              from    to
---  ----  -----------------  ------  ------
0    0     METADATA_datasets  DS_ID   DS_ID
1    0     METADATA_events    EQ_ID   EQ_ID
```

3.1.1.4 Example 4

The following command will retrieve the schema, or SQL syntax, that was used to construct a specific table; here we use the "METADATA_points" table as an example. The output is shown in blue font.

```
.schema METADATA_points
-- print to console schema for specific table

CREATE   TABLE   METADATA_points   (completion   TEXT,
mapping_scale   TEXT,   DS_ID   INTEGER   NOT   NULL,
row_entry_source_file   TEXT,   epsg_original   INTEGER,
row_entry_date_updated   DATETIME,   EQ_ID   INTEGER   NOT
NULL, PRIMARY KEY (EQ_ID, DS_ID), FOREIGN KEY (EQ_ID)
REFERENCES METADATA_events (EQ_ID), FOREIGN KEY (DS_ID)
REFERENCES METADATA_datasets (DS_ID));
```

3.1.1.5 Example 5

The following command will retrieve and export the schema for the entire database to a *.sql text file.

```
.output folderpath/schema.sql
-- specify output directory and filename
-- this will override console output
.schema
-- print to file schema for entire database
```

3.1.2 Performing common SQL queries on the database in SQLite

This section provides simple examples of common SQL queries on the database using the SQLite command-line program. The syntax here is not specific to SQLite, but it is assumed that the user has enabled foreign key constraints, opened the database, and formatted the console outputs as described above in 3.1.1. The examples and alternatives here are not exhaustive as there are often multiple query structures that will return the same output.

For the examples in this section, a portion of the output will be shown in blue font. Default column widths in the console output are set to 10 characters, so the names of some fields are truncated in the display. If the output does not print to the console, the user should enter the command "`.output stdout`" to reset the standard output and then proceed with the queries.

3.1.2.1 Example 6

The following query will retrieve specific information from the "METADATA_events" table and print it to the console.

```
SELECT EQ_ID, eq_name, magnitude, eq_date
FROM METADATA_events
ORDER BY EQ_ID
;

EQ_ID   eq_name     magnitude   eq_date
------  ----------  ----------  -------------------
1       Landers     7.28        1992-06-28 00:00:00
2       Hector      7.13        1999-10-16 00:00:00
3       EMC         7.2         2010-04-04 00:00:00
4       Balochista  7.7         2013-09-24 00:00:00
5       Izmit_Koca  7.51        1999-08-17 00:00:00
6       Borrego     6.63        1968-04-09 00:00:00
7       Imperial2   6.53        1979-10-15 00:00:00
8       Superstiti  6.54        1987-11-24 00:00:00
9       Kobe        6.9         1995-01-16 00:00:00
10      Denali      7.9         2002-11-03 00:00:00
11      Duzce       7.14        1999-11-12 00:00:00
12      Wenchuan    7.9         2008-05-12 00:00:00
...
```

3.1.2.2 Example 7

Several earthquakes in the database contain data from multiple datasets. The following query will generate an output table showing the measurement datasets available ("DS_ID" field) for each earthquake ("EQ_ID" field; "eq_name" field also included for convenience). This information is contained in the "METADATA_events" and "METADATA_points" tables in the FDHI Database, and the "EQ_ID" field serves as the primary and foreign keys to relate information between the tables. In this example, a *left outer join* operation is used to perform the query.

```
SELECT METADATA_events.EQ_ID,
           METADATA_events.eq_name,
           METADATA_points.DS_ID
FROM METADATA_events
LEFT OUTER JOIN METADATA_points
     ON METADATA_events.EQ_ID = METADATA_points.EQ_ID
ORDER BY METADATA_events.EQ_ID
;
```

```
EQ_ID       eq_name     DS_ID
----------  ----------  ----------
1           Landers     3
1           Landers     6
2           Hector      2
2           Hector      3
2           Hector      6
2           Hector      99
3           EMC         18
4           Balochista  23
4           Balochista  75
5           Izmit_Koca  6
...
```

Alternatively, the above result can be achieved with a *where* clause in lieu of the *left outer join*:

```
SELECT METADATA_events.EQ_ID,
           METADATA_events.eq_name,
           METADATA_points.DS_ID
FROM METADATA_events, METADATA_points
WHERE METADATA_events.EQ_ID = METADATA_points.EQ_ID
ORDER BY METADATA_events.EQ_ID
;
```

3.1.2.3 Example 8

The following query will generate an output table showing which earthquakes have measurement information from DS_ID = 6. This information is contained in the "METADATA_events" and "METADATA_points" tables in the FDHI Database, and the "EQ_ID" field serves as the primary

and foreign keys to relate information between the tables. This example uses a *where* clause, but the same result could also be achieved with a *left outer join* operation.

```
SELECT METADATA_events.EQ_ID,
           METADATA_events.eq_name,
           METADATA_points.DS_ID
FROM METADATA_events, METADATA_points
WHERE METADATA_points.DS_ID = 6
     AND METADATA_events.EQ_ID = METADATA_points.EQ_ID
;
```

```
EQ_ID       eq_name     DS_ID
----------  ----------  ----------
1           Landers     6
2           Hector      6
5           Izmit_Koca  6
6           Borrego     6
7           Imperial2   6
9           Kobe        6
```

3.1.2.4 Example 9

The following query will return the citation for DS_ID = 6.

```
SELECT citation
FROM METADATA_datasets
WHERE METADATA_datasets.DS_ID = 6
;
```

```
pers. comm., Dawson, T. to Sarmiento, A. in support of:
Petersen, M. D., Dawson, T. E., Chen, R., Cao, T., Wills,
C. J., Schwartz, D. P., & Frankel, A. D. (2011). Fault
displacement hazard for strike-slip faults. Bulletin of
the Seismological Society of America, 101(2), 805-825.
```

3.1.2.5 Example 10

Information from multiple tables will need to be joined to retrieve measurement locations and amplitudes for a specific slip component. The example below generates an output table with the following information for EQ_ID = 7:

- earthquake, dataset, and measurement identifiers
- measurement location (latitude/longitude)
- central/preferred fault-parallel slip measurement
- central/preferred vertical slip measurement

The identifier information is contained in the following three tables: (1) "METADATA_events" (2) "METADATA_points" (3) "PTOBS_id". Location information is in the "PTOBS_location" table, and the slip measurements are in the "PTOBS_fps" and PTOBS_vs" tables. Four key fields serve as the primary and foreign keys to relate information between the tables: "EQ_ID", "DS_ID", "PT_ID", and "MEAS_ID".

The query below uses a *left outer join* operation, but the same result could also be achieved with other syntax, such as a *where* clause, nested inner joins, or table aliases.

```
SELECT METADATA_events.EQ_ID,
           METADATA_points.DS_ID,
           PTOBS_id.PT_ID,
           PTOBS_id.MEAS_ID,
           PTOBS_location.longitude_degrees,
           PTOBS_location.latitude_degrees,
           PTOBS_fps.fps_central_meters,
           PTOBS_vs.vs_central_meters
FROM METADATA_events
LEFT OUTER JOIN METADATA_points
     ON METADATA_events.EQ_ID = METADATA_points.EQ_ID
LEFT OUTER JOIN PTOBS_id
     ON METADATA_events.EQ_ID = PTOBS_id.EQ_ID
     AND METADATA_points.DS_ID = PTOBS_id.DS_ID
LEFT OUTER JOIN PTOBS_location
     ON METADATA_events.EQ_ID = PTOBS_location.EQ_ID
     AND METADATA_points.DS_ID = PTOBS_location.DS_ID
     AND PTOBS_id.PT_ID = PTOBS_location.PT_ID
LEFT OUTER JOIN PTOBS_fps
     ON METADATA_events.EQ_ID = PTOBS_fps.EQ_ID
     AND METADATA_points.DS_ID = PTOBS_fps.DS_ID
     AND PTOBS_id.PT_ID = PTOBS_fps.PT_ID
     AND PTOBS_id.MEAS_ID = PTOBS_fps.MEAS_ID
LEFT OUTER JOIN PTOBS_vs
     ON METADATA_events.EQ_ID = PTOBS_vs.EQ_ID
     AND METADATA_points.DS_ID = PTOBS_vs.DS_ID
     AND PTOBS_id.PT_ID = PTOBS_vs.PT_ID
     AND PTOBS_id.MEAS_ID = PTOBS_vs.MEAS_ID
WHERE METADATA_events.EQ_ID = 7
ORDER BY METADATA_events.EQ_ID
     AND METADATA_points.DS_ID
     AND PTOBS_id.PT_ID
     AND PTOBS_id.MEAS_ID
;

EQ_ID DS_ID PT_ID MEAS_ID longitud latitu fps_ce vs_cen
----- ----- ----- ------- -------- ------ ------ -----
7     6     1     1       -115.391 32.714 NULL   0.0
7     6     1     2       -115.391 32.714 0.55   NULL
7     6     1     3       -115.391 32.714 0.55   NULL
7     6     1     4       -115.391 32.714 0.55   0.0
7     6     2     1       -115.389 32.711 0.04   0.0
```

```
7     6     2     2       -115.389 32.711 0.04   NULL
7     6     2     3       -115.389 32.711 0.04   0.0
7     6     3     1       -115.400 32.723 0.58   0.09
7     6     3     2       -115.400 32.723 0.634  NULL
...
```

The above query prints several hundred rows of information to the console. Alternatively, the query result could be saved to a *.csv file with the following commands:

```
.mode csv
-- set output mode to CSV
.output folderpath/myfileout.csv
-- specify output directory and filename
-- this will override console output
SELECT ...
-- enter query here
```

3.2 EXAMPLE FDHI DATABASE QUERIES IN PYTHON

Default Python installations include the "sqlite3" module, which serves as an alternative to the SQLite engine accessed in the command-line. While the same syntax described above can be used within the Python "sqlite3" module in most cases, the advantage of executing database queries in Python is the convenience of the added procedural functionality in the Python language, such as the string format method, general ease of creating and passing variables, and returning output tables as "pandas" series or dataframes.

The following subsections use Python's "sqlite3" module and the "pandas" library to show how to open the database, retrieve information on the database schema, and perform common SQL queries on the database. See Section 4 for software versioning. For convenience, the string-like outputs will be direct results from the "sqlite3" module, and the "pandas" library will be used to generate the table-like outputs as series or dataframes. Most of the syntax in the following subsections are specific to the SQLite program (and therefore specific to the Python "sqlite3" module) and the "pandas" library, although universal SQL statements are still embedded in most commands. SQL and the "pandas" library are both versatile tools and there are usually multiple approaches to perform queries and aggregate results.

3.2.1 Starting out: how to open the database in Python and investigate the database schema

This section shows how to open the database in Python and investigate the database schema by printing information to the console or exporting the information to a text file.

The following commands will open the FDHI Database in Python. The user will need to update their folder path and database name as needed. Comments are denoted with a hash symbol

(#) and gray font. The comments are shown for clarity and do not need to be entered in the command. As discussed in Section 3.1.1, an unusual feature of SQLite is that foreign key constraints must be explicitly activated (Hipp, 2020), as shown below.

```
import sqlite3
import pandas as pd
# import required libraries

db = r'C:\folderpath\FDHI_ph17_rev0.db'
conn = sqlite3.connect(db)
# connects to database in specific directory

conn.execute('PRAGMA foreign_keys=ON')
# per docs, foreign keys must still be enabled by the
# application at runtime
```

3.2.1.1 Example 1

The following commands will print a list of the tables in the database to the console. The output is shown in blue font.

```
cursor = conn.cursor()
query = (
    "SELECT NAME "
    "FROM sqlite_master "
    "WHERE type = 'table' "
)

tables = cursor.execute(query).fetchall()
# returns a list of tuples as unicode characters

tables = map(str, [t[0] for t in tables])
# clean up list of tuples into list and encode unicode
# characters

print tables
# print to console list of all tables in database

['METADATA_events', 'METADATA_datasets',
'METADATA_points', 'PTOBS_id', 'PTOBS_nsplays',
'PTOBS_confidence', 'PTOBS_feature', 'PTOBS_location',
'PTOBS_aftershock', 'PTOBS_compass',
'PTOBS_elevation', 'PTOBS_fzw', 'PTOBS_fps',
'PTOBS_fns', 'PTOBS_nhs', 'PTOBS_ads', 'PTOBS_vs',
'PTOBS_sh', 'PTOBS_tds', 'METADATA_ruptures',
'RUPOBS_id', 'RUPOBS_faultname', 'RUPOBS_confidence',
'RUPOBS_sliprate', 'RUPOBS_length',
'RUPOBS_aftershock', 'RUPOBS_location',
'METADATA_geology', 'PTOBS_geology', 'RUPOBS_geology',
'PAIRING_ptrup', 'PTOBS_slip_nonzero', 'ECS_pts',
```

```
'ECS_rups', 'ECS_linepath', 'INTERPRETATIONS_pts',
'INTERPRETATIONS_rups']
```

3.2.1.2 Example 2

The following command will retrieve the information of a specific table, including the column names, data types, null constraints, and primary keys, as a "pandas" dataframe; here we use the "METADATA_points" table as an example. The output is shown in blue font. Some column widths are truncated here for clarity, but the full text will be displayed in the Python console. As well, the default dataframe index generated by "pandas" is also omitted for clarity but will still be displayed in the Python console.

```
df = pd.read_sql(
    "PRAGMA table_info('METADATA_points')", conn
)
print df
# print to console table information for specific table

cid         name      type  notnull dflt_value  pk
  0   completion      TEXT        0       None   0
  1  mapping_sca      TEXT        0       None   0
  2        DS_ID   INTEGER        1       None   2
  3   row_entry_      TEXT        0       None   0
  4   epsg_origi   INTEGER        0       None   0
  5   row_entry_  DATETIME        0       None   0
  6        EQ_ID   INTEGER        1       None   1
```

Alternatively, the above command can be written using Python's string format method, and a list of table names can be passed. The syntax below will retrieve information for the "METADATA_points" and "METADATA_events" tables and print the information to the console.

```
tables = ['METADATA_points', 'METADATA_events']
# list tables of interest enclosed in quotes

for t in tables:
    df = pd.read_sql(
        "PRAGMA table_info({})".format(t), conn
    )
    print df
    # print to console table information for specific
    # table
```

3.2.1.3 Example 3

The following command will retrieve the foreign key constraints a specific table as a "pandas" dataframe; here we use the "METADATA_points" table as an example. The output is shown in blue font. As with the above example, the syntax below could also be written using Python's string format method to pass the table name as a variable.

```
df = pd.read_sql(
```

```
    "PRAGMA foreign_key_list('METADATA_points')", conn
)
print df
# print to console foreign key information for specific
table

id  seq              table    from     to
 0    0  METADATA_datasets  DS_ID  DS_ID
 1    0    METADATA_events  EQ_ID  EQ_ID
```

3.2.1.4 Example 4

While the SQLite command-line program has a built-in helper function ("`.schema`") to retrieve the schema, or SQL syntax, there is no built-in equivalent in the Python "sqlite3" module. Instead, the following commands can be used retrieve the schema, or SQL syntax, that was used to construct a specific table; here we use the "METADATA_points" table as an example. Python's string format method to pass the table name as a variable; alternatively, the table name could be embedded in the query. The output is shown in blue font.

```
query = (
    "SELECT sql "
    "FROM sqlite_master "
    "WHERE name = '{t}' "
).format(t='METADATA_points')
# use string format method to pass table name

output = conn.execute(query).fetchall()
# perform query

output = u''.join(i[0] for i in output)
# clean up output into string

print output
.schema METADATA_points
# print to console schema for specific table

CREATE TABLE METADATA_points (completion TEXT,
mapping_scale TEXT, DS_ID INTEGER NOT NULL,
row_entry_source_file TEXT, epsg_original INTEGER,
row_entry_date_updated DATETIME, EQ_ID INTEGER NOT
NULL, PRIMARY KEY (EQ_ID, DS_ID), FOREIGN KEY (EQ_ID)
REFERENCES METADATA_events (EQ_ID), FOREIGN KEY
(DS_ID) REFERENCES METADATA_datasets (DS_ID))
```

3.2.1.5 Example 5

As with Example 4, there is no built-in equivalent to the "`.schema`" helper function in the Python "sqlite3" module, but the following commands will retrieve and export the schema for the entire database to a *.sql text file.

```
query = (
    "SELECT sql "
    "FROM sqlite_master "
)

with open(r'C:\folderpath\schema.sql', 'w') as writer:
# specify output directory and filename
    for r in conn.execute(query):
        writer.write("%s;\r" % str(r[0]))
           # print to file schema for database
```

3.2.2 Performing common SQL queries on the database in Python

This section provides simple examples of common SQL queries on the database using the Python, the "sqlite3" module, and the "pandas" library. It is assumed that the user has enabled foreign key constraints and opened the database, as described above in Section 3.2.1.

For the examples in this section, a portion of the output will be shown in blue font. Some column widths are truncated herein for clarity, but the full text will be displayed in the Python console. The default dataframe index generated by "pandas" is also omitted for clarity but will still be displayed in the Python console.

3.2.2.1 Example 6

The following query will retrieve specific information from the "METADATA_events" table as a "pandas" dataframe and print it to the console. The output is shown in blue font.

```
query = (
    "SELECT EQ_ID, eq_name, magnitude, eq_date "
    "FROM METADATA_events "
    "ORDER BY EQ_ID "
)

df = pd.read_sql_query(query, conn)

df.replace([None], "NULL", inplace=True)
# specify how to display null values; default is empty

print df
```

```
EQ_ID            eq_name  magnitude          eq_date
    1            Landers       7.28  1992-06-28 00:0
    2             Hector       7.13  1999-10-16 00:0
    3                EMC       7.20  2010-04-04 00:0
    4        Balochistan       7.70  2013-09-24 00:0
    5      Izmit_Kocaeli       7.51  1999-08-17 00:0
    6            Borrego       6.63  1968-04-09 00:0
    7           Imperial2       6.53  1979-10-15 00:0
    8  SuperstitionHills       6.54  1987-11-24 00:0
    9               Kobe       6.90  1995-01-16 00:0
```

```
   10             Denali       7.90  2002-11-03 00:0
   11              Duzce       7.14  1999-11-12 00:0
   12           Wenchuan       7.90  2008-05-12 00:0
...
```

3.2.2.2 Example 7

Several earthquakes in the database contain data from multiple datasets. The following query will generate an output table, as a "pandas" dataframe, showing the measurement datasets available ("DS_ID" field) for each earthquake ("EQ_ID" field; "eq_name" field also included for convenience). This information is contained in the "METADATA_events" and "METADATA_points" tables in the FDHI Database, and the "EQ_ID" field serves as the primary and foreign keys to relate information between the tables. In this example, a *left outer join* operation is used to perform the query. Alternatively, the same result can be achieved with a *where* clause in lieu of the *left outer join* (Section 3.1.2.2).

```
query = (
    "SELECT METADATA_events.EQ_ID, "
        "METADATA_events.eq_name, "
        "METADATA_points.DS_ID "
    "FROM METADATA_events "
    "LEFT OUTER JOIN METADATA_points "
    "ON METADATA_events.EQ_ID = METADATA_points.EQ_ID "
    "ORDER BY METADATA_events.EQ_ID "
)

df = pd.read_sql_query(query, conn)

df.replace([None], "NULL", inplace=True)
# specify how to display null values; default is empty

print df
```

```
EQ_ID            eq_name  DS_ID
    1            Landers      3
    1            Landers      6
    2             Hector      2
    2             Hector      3
    2             Hector      6
    2             Hector     99
    3                EMC     18
    4        Balochistan     23
    4        Balochistan     75
    5      Izmit_Kocaeli      6
...
```

Alternatively, the same result can be achieved leveraging various built-in functions in the "pandas" library. The following example uses a SQL query to retrieve all entries in the tables of interest ("METADATA_events" and "METADATA_points") as dataframes, merge the

dataframes on the specified key fields (“DS_ID”), extract the fields of interest (“EQ_ID”; “eq_name”; and “DS_ID”), and removes duplicate rows. This is one of many ways to utilize the “pandas” library for this application, and this simple approach works when the key field names are the same in both tables.

```
# list tables, keys of interest
tables = ['METADATA_events', 'METADATA_points']
keys = ['EQ_ID']

# perform queries
df = pd.DataFrame()
for t in tables:
    query = (
        "SELECT * "
        "FROM '{table}' "
    ).format(table=t)

    if df.empty:
        df = pd.read_sql_query(query, conn)
    else:
        df = pd.merge(df,
                      pd.read_sql_query(query, conn),
                      how='outer', on=keys
                      )

# list fields of interest
fields = ['EQ_ID', 'eq_name', 'DS_ID']

# extract fields of interest
# remove duplicate rows
# specify how to display null values; default is empty
df = df[fields].copy()\
    .drop_duplicates()\
    .replace([None], "NULL")

print df
```

3.2.2.3 Example 8

The following query will generate an output table, as a “pandas” dataframe, showing which earthquakes have measurement information from DS_ID = 6. This information is contained in the “METADATA_events” and “METADATA_points” tables in the FDHI Database, and the “EQ_ID” field serves as the primary and foreign keys to relate information between the tables. This example performs two separate queries, collecting each output in a dataframe, merges the two dataframes together, and removes duplicate and empty rows.

```
# set up and perform query 1
table = 'METADATA_events'

query = (
```

```
    "SELECT * "
    "FROM '{t}' "
).format(t=table)

df1 = pd.read_sql_query(query, conn)

# set up and perform query 2
table, dsid = 'METADATA_points', 6

query = (
    "SELECT * "
    "FROM '{t}' "
    "WHERE '{t}'.DS_ID = ? "
).format(t=table)

df2 = pd.read_sql_query(
    sql=query, con=conn, params=(dsid,)
)

# merge results
df_final = pd.merge(
    df1, df2, how='outer', on=['EQ_ID']
)

# list fields of interest
fields = ['EQ_ID', 'eq_name', 'DS_ID']

# extract fields of interest
# remove duplicate and empty rows
# specify how to display null values; default is empty
df_final = df_final[fields].copy()\
    .drop_duplicates().dropna()\
    .replace([None], "NULL")

print df_final
```

```
EQ_ID        eq_name  DS_ID
    1        Landers      6
    2         Hector      6
    5  Izmit_Kocaeli      6
    6        Borrego      6
    7      Imperial2      6
    9           Kobe      6
```

Alternatively, the same result can be achieved by directly passing a full SQL query. As discussed in Section 3.1.2.3, the example below uses a *where* clause, but the same result could also be achieved with a *left outer join* operation.

```
query = (
    "SELECT METADATA_events.EQ_ID, "
        "METADATA_events.eq_name, "
```

```
        "METADATA_points.DS_ID "
    "FROM METADATA_events, METADATA_points "
    "WHERE METADATA_points.DS_ID = 6 "
        "AND METADATA_events.EQ_ID = "
            "METADATA_points.EQ_ID "
)

df = pd.read_sql_query(query, conn)
print df
```

3.2.2.4 Example 9

The following commands will return the citation for DS_ID = 6. Python's string format method to pass the dataset id (6) as a variable; alternatively, this value could be embedded in the query. The output is shown in blue font.

```
query = (
    "SELECT citation "
    "FROM METADATA_datasets "
    "WHERE METADATA_datasets.DS_ID = {i} "
).format(i=6)

output = conn.execute(query).fetchall()
# perform query

output = u''.join(i[0] for i in output)
# clean up output into string
print output

pers. comm., Dawson, T. to Sarmiento, A. in support
of: Petersen, M. D., Dawson, T. E., Chen, R., Cao, T.,
Wills, C. J., Schwartz, D. P., & Frankel, A. D.
(2011). Fault displacement hazard for strike-slip
faults. Bulletin of the Seismological Society of
America, 101(2), 805-825.
```

3.2.2.5 Example 10

Information from multiple tables will need to be joined to retrieve measurement locations and amplitudes for a specific slip component. The example below generates an output table with the following information for EQ_ID = 7:

- earthquake, dataset, and measurement identifiers
- measurement location (latitude/longitude)
- central/preferred fault-parallel slip measurement
- central/preferred vertical slip measurement

The identifier information is contained in the following three tables: (1) “METADATA_events” (2) “METADATA_points” (3) “PTOBS_id”. Location information is in the “PTOBS_location” table, and the slip measurements are in the “PTOBS_fps” and PTOBS_vs” tables. Four key fields serve as the primary and foreign keys to relate information between the tables: “EQ_ID”, “DS_ID”, “PT_ID”, and “MEAS_ID”.

This same example performed in Section 3.1.2.5 shows the syntax for the full SQL query using a *left outer join* operation (alternatively, other syntax, such as a *where* clause, nested inner joins, or table aliases could also be used). In this example below, various built-in functions in the “pandas” library are used instead, in which we will perform multiple queries and multiple merges of the results. The queries are separated based on tables with the same keys and incrementally merged on the common keys. The final steps extract the fields of interest, remove duplicate rows, and sort the output. However, as with the alternative query in Example 9 in Section 3.2.2.4, the full SQL query from 3.1.2.5 could also be used.

```
# set up and perform query 1
tables = ['METADATA_events', 'METADATA_points']
keys = ['EQ_ID']
eqid = 7

df1 = pd.DataFrame()
for t in tables:
    query = (
        "SELECT * "
        "FROM '{table}' "
        "WHERE '{table}'.EQ_ID = ? "
    ).format(table=t)

    if df1.empty:
        df1 = pd.read_sql_query(
            sql=query, con=conn,
            params=(eqid,)
        )
    else:
        df1 = pd.merge(
            df1, pd.read_sql_query(
                sql=query, con=conn, params=(eqid,)
            ), how='outer', on=keys
        )

# set up and perform query 2
tables = ['PTOBS_location']
keys = ['EQ_ID', 'DS_ID', 'PT_ID']

df2 = pd.DataFrame()
for t in tables:
    query = (
        "SELECT * "
        "FROM '{table}' "
```

```
        "WHERE '{table}'.EQ_ID = ? "
    ).format(table=t)

    if df2.empty:
        df2 = pd.read_sql_query(
            sql=query, con=conn,
            params=(eqid,)
        )
    else:
        df2 = pd.merge(
            df2, pd.read_sql_query(
                sql=query, con=conn, params=(eqid,)
            ), how='outer', on=keys
        )

# merge results of queries 1 and 2
df2   =   df1.merge(df2,   how='outer',   on=['EQ_ID',
'DS_ID'])\
    .drop_duplicates()

# set up and perform query 3
tables = ['PTOBS_id', 'PTOBS_fps', 'PTOBS_vs']
keys = ['EQ_ID', 'DS_ID', 'PT_ID', 'MEAS_ID']

df3 = pd.DataFrame()
for t in tables:
    query = (
        "SELECT * "
        "FROM '{table}' "
        "WHERE '{table}'.EQ_ID = ? "
    ).format(table=t)

    if df3.empty:
        df3 = pd.read_sql_query(
            sql=query, con=conn, params=(eqid,)
        )
    else:
        df3 = pd.merge(
            df3, pd.read_sql_query(
                sql=query, con=conn, params=(eqid,)
            ), how='outer', on=keys
        )

# merge all results
keys = ['EQ_ID', 'DS_ID', 'PT_ID']
df_final = df3.merge(df2, how='outer', on=keys)

# list fields of interest
fields = [
    'EQ_ID',     'DS_ID',     'PT_ID',     'MEAS_ID',
'longitude_degrees',
```

```
    'latitude_degrees',          'fps_central_meters',
'vs_central_meters'
]

# extract fields of interest
# remove duplicate rows
# specify how to display null values; default is empty
df_final = df_final[fields].copy().drop_duplicates()\
    .replace([None], "NULL")

# sort output
keys.extend(['MEAS_ID'])
df_final.sort_values(keys, inplace=True)

print df_final
```

```
EQ_ID DS_ID PT_ID MEAS_ID longitud latitu fps_ce vs_cen
    7     6     1       1 -115.391 32.714   NULL      0
    7     6     1       2 -115.391 32.714   0.55   NULL
    7     6     1       3 -115.391 32.714   0.55   NULL
    7     6     1       4 -115.391 32.714   0.55      0
    7     6     2       1 -115.389 32.711   0.04      0
    7     6     2       2 -115.389 32.711   0.04   NULL
    7     6     2       3 -115.389 32.711   0.04      0
    7     6     3       1 -115.400 32.723   0.58   0.09
    7     6     3       2 -115.400 32.723  0.634   NULL
...
```

4 Software

The following software versions were used to build, populate, and query the FDHI Database:

- SQLite database engine version 3.14.2 (Hipp, 2020; https://www.sqlite.org)
- Python version 2.7.15 (https://www.python.org)
- Python "sqlite3" module version 2.6.0 (https://docs.python.org)
- Python "pandas" library version 0.18.1 (https://pandas.pydata.org/)

The HTML documentation available at https://fdhi.github.io/dbschema/ was generated using SchemaSpy software (Petzäll and Kasa, 2019). GUI software compatible with SQLite databases includes, but is not limited to, "DB Browser for SQLite" (https://sqlitebrowser.org/) and "DbVisualizer" (https://www.dbvis.com/).

5 Summary

This appendix documents the relational schema (or structure) of the FDHI Database. The database contains 37 individual tables (Table B.1) that can generally be categorized as storing metadata (5 tables), observations (26 tables), interpreted data (3 tables) and event-specific coordinate system information (3 tables). The tables are interrelated by the database schema (see the online HTML documentation at https://fdhi.github.io/dbschema/ and also Table B.2).

This appendix also provides guidance and examples on retrieving information from the database. We recommend most users of the database access the contents through the flatfiles because these products are the formal documentation of the database contents and have been reviewed for accuracy and completeness. However, for documentation purposes, examples of how to navigate the database using the SQLite command-line program and Python are provided in this appendix.

Collectively, the database schema descried in Section 2 and the example database queries in Section 3 document the FDHI Database structure and general contents. This information can be used in future generations of the FDHI Database to guide the addition of new event datasets and types of data.

6 References

Hipp, R. D. (2020). SQLite. Retrieved from https://www.sqlite.org/index.html.

Mazzoni S., Kishida T., Ahdi S.K., Contreras V., Darragh R.B., Chiou B.S.-J., Kuehn N., Bozorgnia Y., Stewart J.P. (2020). Chapter 2: Relational Database, in Data Resources for NGA-Subduction Project, PEER Report No. 2020/02, *Pacific Earthquake Engineering Research Cente*r, University of California, Berkeley, CA.

Petzäll, N. and Kasa, R. (2019). Schema Spy, version 6.1.0. Github. URL: https://github.com/schemaspy/schemaspy.