Patient level data: `SUBJECT_ID`

The following is a list of static data available in the database for `SUBJECT_ID` in the PATIENTS table:

- GENDER
- DOB
- DOD
- DOD_HOSP
- DOD_SSN

This list contains all the static data available for a single patient. Note that the `DOD_SSN` (which also contributes to the `DOD` column) is acquired from the social security death registry, i.e. an external source. It contains dates of death up to 90 days in the future for Metavision patients. It contains dates of death up to 4 years in the future for CareVue patients.

Hospital level data: `HADM_ID`

The following tables were sourced from the hospital database, and contain information recorded in the hospital, but not necessarily during the patient's ICU stay:

- ADMISSIONS
- CALLOUT
- CPTEVENTS
- DIAGNOSES_ICD
- DRGCODES
- ICUSTAYS
- LABEVENTS
- MICROBIOLOGYEVENTS
- PATIENTS
- PRESCRIPTIONS
- PROCEDURES_ICD
- SERVICES
- TRANSFERS

The following is a list of static data available in the database for `HADM_ID` in the ADMISSIONS table. This data is constant for a single hospital admission.

- ADMITTIME - The hospital admission time
- DISCHTIME - The hospital discharge time
- DEATHTIME - The date of death of the patient if they died within the hospital
- ADMISSION_TYPE - The type of admission: ELECTIVE, EMERGENCY, NEWBORN or URGENT (note that the NEWBORN value here does not perfectly identify newborns)
- ADMISSION_LOCATION - The location of the patient prior to hospital admission
- DISCHARGE_LOCATION - The location of the patient after hospital discharge
- INSURANCE - The patient's type of medical insurance
- LANGUAGE - The patient's primary language
- RELIGION - The patient's stated religion
- MARITAL_STATUS - The patient's marital status
- ETHNICITY - The patient's stated ethnicity
- DIAGNOSIS - A short description of the reason for the patient's admission

ICU level data: ICUSTAY_ID

The following tables were sourced from the ICU databases, and contain information only during a patient's ICU stay:

- DATETIMEEVENTS
- INPUTEVENTS_CV
- INPUTEVENTS_MV
- NOTEEVENTS
- OUTPUTEVENTS
- PROCEDUREEVENTS_MV

The following is a list of static data available in the database for ICUSTAY_ID in the ICUSTAYS table:

- DBSOURCE - The ICU database from which the patient exists in
- FIRST_CAREUNIT - The first unit that cared for the patient (all ICUs except NWARD)
- LAST_CAREUNIT - The last care unit that cared for the patient (all ICUs except NWARD)
- FIRST_WARDID - An integer representing the first physical location of the patient
- LAST_WARDID - An integer representing the last physical location of the patient
- INTIME - Time entered the ICU
- OUTTIME - Time left the ICU

- LOS - The patient's ICU length of stay

Types of data in the database

Data within MIMIC were recorded during routine clinical care and *not* explicitly for the purpose of retrospective data analysis. This is a key point to keep in mind when analyzing the data.

There are two types of data in the database: static data and dynamic data. Static data is recorded once for a given identifier. An example of static data is the DOB column in the PATIENTS table. Each patient has only one date of birth, which does not change over time and is not recorded with an associated timestamp. An example of dynamic data is a patient's blood pressure, which is periodically measured during a hospital stay.

This distinction between static data and dynamic data is merely a helpful conceptual construct: there is *no* strict technical distinction between date of birth and heart rate. However, static data tends to not have an associated ITEMID (as there is no need to repeatedly record values for static data), whereas dynamic data have an ITEMID to facilitate efficient storage of repeated measurements.

Static data

SUBJECT_ID

The following is a list of static data available in the database for SUBJECT_ID in the PATIENTS table:

The only static data available for patients are their gender (GENDER), date of birth (DOB) and various dates of death (DOD, DOD_HOSP, DOD_SSN). These columns all occur in the PATIENTS table.

HADM_ID

The following is a list of static data available in the database for HADM_ID in the ADMISSIONS table:

- Admission time
- Discharge time
- Death time
- Admission type

ICUSTAY_ID

The following is a list of static data available in the database for `ICUSTAY_ID` in the ICUSTAYS table:

- `INTIME` - Time entered the ICU
- `OUTTIME` - Time left the ICU
- First care unit
- Last care unit

Hospital acquired data

The following tables were sourced from the hospital database, and contain information recorded in the hospital, but not necessarily during the patient's ICU stay:

- ADMISSIONS
- CALLOUT
- CPTEVENTS
- DIAGNOSES_ICD
- DRGCODES
- ICUSTAYS
- LABEVENTS
- MICROBIOLOGYEVENTS
- PATIENTS
- PRESCRIPTIONS
- PROCEDURES_ICD
- SERVICES
- TRANSFERS

ICU acquired data

The following tables were sourced from the ICU databases, and contain information only during a patient's ICU stay:

- DATETIMEEVENTS
- INPUTEVENTS_CV
- INPUTEVENTS_MV
- NOTEEVENTS
- OUTPUTEVENTS

- PROCEDUREEVENTS_MV

Externally acquired data

The DOD_SSN (which also contributes to the DOD column) is acquired from the social security death registry. It contains dates of death up to 90 days in the future for Metavision patients. It contains dates of death up to 4 years in the future for CareVue patients.

Inputs and outputs

Inputs and outputs are extremely useful when studying intensive care unit patients. Inputs are any fluids which have been administered to the patient: such as oral or tube feedings or intravenous solutions containing medications. Outputs are fluids which have either been excreted by the patient, such as urine output, or extracted from the patient, for example through a drain. These data were the most complicated to handle technically in the MIMIC-III data.

The MIMIC-III database contains information from two distinct critical care information systems: Philips CareVue and iMDSoft Metavision. These two databases store data in different ways. In the descriptions in this document, data will be referred to as being sourced either from “CareVue” or “Metavision” to differentiate between the different systems.

INPUTEVENTS_CV, INPUTEVENTS_MV, OUTPUTEVENTS

Inputs exist in two separate tables: INPUTEVENTS_CV and INPUTEVENTS_MV. INPUTEVENTS_CV contains CareVue inputs, while INPUTEVENTS_MV contains Metavision inputs. Results from these tables can be unioned as observations are not duplicated across tables.

Outputs

Outputs are recorded in a reasonably consistent manner in both the CareVue and Metavision data. The time at which the output is measured is recorded in the CHARTTIME column. There is no start time recorded with outputs

- CHARTTIME corresponds to the time that the volume had been output by. The volume of output is recorded in the VALUE column, and the unit of measurement is provided in the VALUEUOM column (usually milliliters, or mL). It is usually reasonable to assume that any output recorded is for the interval between the current CHARTTIME and the previous CHARTTIME for the same item.

Inputs

Inputs are handled differently by CareVue and Metavision. For CareVue data, only the CHARTTIME is available. Second, the RATE and AMOUNT columns are *asynchronous*, and originally stored in different tables. Volumes of input (e.g. 50 mL of normal saline) would be stored in one table (the original IOEVENTS), and would usually be recorded every hour (though sometimes the period was longer). Conversely, the RATE of the drug would be stored separately (in MEDEVENTS), and only updated when a change or verification of the rate was made by clinical staff. As a result, the raw data looked something similar to:

CHARTTIME	VOLUME	VOLUMEUOM	CHARTTIME	RA
		09:00	1	mL/min
10:00	60	mL		
11:00	60	mL		
		11:30	0.5	mL/min
12:00	45	mL		

Here, the volume is recorded only every hour, and no start time is available. However, it's reasonable to assume that the volume measurement corresponds to an hour. Next, we can see that the rate was titrated to 0.5, and for the period between 11:00 to 12:00 there was half an hour of delivery at 1 mL/min, and half an hour of delivery at 0.5 mL/min, resulting in a total volume of 45 mL delivered for the past hour.

Summing up, for CareVue data, the rate and volume will be asynchronous, and only the CHARTTIME will be available. For rates, the CHARTTIME will correspond to a start time (when the drug was set to that rate). For volumes, the CHARTTIME will correspond to an end time.

For Metavision data, there is no concept of a volume in the database: only a RATE. All inputs are recorded with a STARTTIME and an ENDTIME. As a result, the volumes in the database for Metavision patients are *derived* from the rates. Furthermore, exact start and stop times for the drugs are easily deducible.

Details of the merging process

The difficulty in merging the databases arose due primarily to two factors: the lower resolution of information archiving in the CareVue system, and the different definition of an 'order' in the databases.

The aim of this section is to provide all the detail into how these data were merged: this information is not necessary to understand for the purposes of using the database, but will provide insight into the format of the IOEVENTS table. We welcome suggestions from the community on improving the format and usability of the table.

Philips CareVue

The CareVue system stored input/output (IO) data across five tables: IOEVENTS, MEDEVENTS, ADDITIVES, SOLUTIONS and DELIVERIES. Each time a new order for a drug was recorded in the database, the ADDITIVES, SOLUTIONS and DELIVERIES tables would be populated with information regarding the order. The data archival format is best described with an example.

Patient A has been recently admitted to the ICU and is to be administered norepinephrine to restore their blood pressure to a value of at least 60 mmHg. The route of administration is intravenous, i.e. the drug is to be pumped into the patient's blood stream directly through a line inserted in a vein. The nurse would prepare a solution of 250 mL sodium chloride (NaCl) to contain the drug. The SOLUTIONS table would consequently contain an entry of 250 mL NaCl for patient A at the time the nurse prepared the solution. The nurse would then mix in the drug into the solution: in this example let's say 8 mg of norepinephrine. The 8 mg of norepinephrine would be recorded in the ADDITIVES table, and an identifier would be recorded which linked the 8 mg of norepinephrine to the 250 mL solution of NaCl. Finally, the nurse would optionally set an initial delivery rate and route: in this case it could be 10 mL/hr intravenously. These would populate the `RATE` and `ROUTE` columns in the DELIVERIES table. Note that DELIVERIES data is less frequently present and was not consistently recorded.

Now that the solution has been prepared, the nurse can begin administering the drug to the patient. The time is now 18:20. An initial entry of 0 mL is recorded in the IOEVENTS table at 18:00 (this usually occurs, but it has not been verified that this *always* occurs). The nurse begins at a rate of 1 mcg/kg/min. The MEDEVENTS contains an entry at a `CHARTTIME` of 18:20 for a rate of 1 mcg/kg/min. Five minutes later, at 18:25, the nurse notes that the blood pressure is still lower than the desired 60 mmHg and increases the dose. The nurse raises the dose to 2

mcg/kg/min, and MEDEVENTS records the new dose of 2 mcg/kg/min at 18:25. The nurse checks again and notes that the blood pressure has reached the target value of 60 mmHg and ceases titration of the drug. At 19:00, the volume of drug administered to the patient is recorded. If the patient weighed 100 kg, then the amount of drug administered would be equal to:

5 min * 1 mcg/kg/min * 100 kg + 25 min * 2 mcg/kg/min * 100 kg = 500 mcg + 5000 mcg = 5500 mcg = 5.5 mg

The amount of solution administered would be equal to:

(5 min * 1 mcg/kg/min * 100 kg + 25 min * 2 mcg/kg/min * 100 kg) * 250 mL / 8 mg = 5.5 mg * 250 mL / 8 mg = 171.875 mL

Consequently, IOEVENTS would record 171.875 mL at 19:00. If the patient continued at the same rate for this drug, then IOEVENTS would record (60 * 2 * 100 * 250 / 8000) = 375 mL at 20:00. Note that:

- Unless the rate is updated, no new entry in MEDEVENTS exists
- IOEVENTS always records a value on an hourly basis, regardless of how long the IO event has been present
- 375 mL is larger than the original 250 mL bag: somewhere during administration the nurse would need to replace the empty bag with a new solution of the same formulation

Note that the changing of the bag is sometimes, but not always, recorded in the ADDITIVES/SOLUTIONS/DELIVERIES table as a new order. It usually occurs if the drug has been discontinued, and then re-prescribed later in the patient's stay.

Metavision

Metavision records IO data using two tables: RANGESIGNALS and ORDERENTRY. These tables do not appear in MIMIC-III as they have been merged to form the INPUTEVENTS_MV table. RANGESIGNALS contains recorded data elements which last for a fixed period of time. Furthermore, the RANGESIGNALS table recorded information for each component of the drug separately. For example, for a norepinephrine administration there would be two components: a main order component (norepinephrine) and a solution component (NaCl).
The STARTTIME and ENDTIME of RANGESIGNALS indicated when the drug started and finished. *Any* change in the drug rate would result in the current infusion ending, and a new STARTTIME being created.

Let's return to our example above of the patient being given norepinephrine. The STARTTIME for the solution (NaCl) and the drug (norepinephrine) would be 18:20. The rate of the drug would be listed as 1 mcg/kg/min, and the rate of the

solution would be listed as 10 mL/hr. The nurse, as before, decides to increase the drug rate at 18:25 to 2 mcg/kg/min. As a result, the ENDTIME for the two rows corresponding to the solution (NaCl and norepinephrine) is set to 18:25. Two new rows are generated with a STARTTIME of 18:25. These two new rows would continue until either (i) the drug rate was changed or (ii) the drug was delivery was discontinued. The ORDERID column would be identical for each instantiation of NaCl and norepinephrine which corresponded to the same solution/rate. That is, for the infusion given between 18:20 and 18:25, both NaCl and norepinephrine would have the same ORDERID. LINKORDERID would further link the drug across all administrations, even when the rate is changed. The following table demonstrates these concepts:

Item	STARTTIME	ENDTIME	RATE	RATEUOM	ORDERID
Norepinephrine	18:20	18:25	1	mcg/kg/min	8003
NaCl	18:20	18:25	10	ml/hr	8003
Norepinephrine	18:25	20:00	2	mcg/kg/min	8020
NaCl	18:25	20:00	20	ml/hr	8020

Note that ORDERID links items occurring at the same time which correspond to the same solution, while LINKORDERID links all these solutions together across time. Note also that LINKORDERID is equal to the first ORDERID which occurs for the solution, as above.

Time types

Time in the database is stored with one of two suffixes: TIME and DATE. If a column has TIME as the suffix, e.g. CHARTTIME, then the data resolution is down to the minute. If the column has DATE as the suffix, e.g. CHARTDATE, then the data resolution is down to the day. That means that measurements in a CHARTDATE column will always have 00:00:00 has the hour, minute, and second values. This does *not* mean it was recorded at midnight: it indicates that we do not have the exact time, only the date.

Date shifting

All dates in the database have been shifted to protect patient confidentiality. Dates will be internally consistent for the same patient, but randomly distributed

in the future. This means that if measurement A is made at `2150-01-01 14:00:00`, and measurement B is made at `2150-01-01 15:00:00`, then measurement B was made 1 hour after measurement A.

The date shifting preserved the following:

- Time of day - a measurement made at `15:00:00` was actually made at `15:00:00` local standard time.
- Day of the week - a measurement made on a Sunday will appear on a Sunday in the future.
- Seasonality - a measurement made during the winter months will appear during a winter month.

The date shifting *removed* the following:

- Year - The year is randomly distributed between 2100 - 2200.
- Day of the month - The absolute day of the month is *not* preserved.
- Inter-patient information - Two patients in the ICU on `2150-01-01` were *not* in the ICU at the same time.

Dates of birth

Dates of birth which occur in the present time are *not* true dates of birth. Furthermore, dates of birth which occur before the year 1900 occur if the patient is older than 89. In these cases, the patient's age at their first admission has been fixed to 300.

Time columns in the database

CHARTTIME vs STORETIME

Most data, with the exception of patient related demographics, are recorded with a time indicating when the observation was made: CHARTTIME. CHARTTIME dates back to the use of paper charts: in order to facilitate efficient observations by nursing staff, the day was separated into hourly blocks, and observations were recorded within these hourly blocks. Thus, any time one performed a measurement between the hours of 04:00 and 05:00, the data would be charted in the 04:00 block, and so on. This concept has carried forward into the electronic recording of data: even if data is recorded at 04:23, in many cases it is still charted as occurring at 04:00.

STORETIME provides information on the recording of the data element itself. All observations in the database must be validated before they are archived into the patient medical record. The STORETIME provides the exact time that this validation

occurred. For example, a heart rate may be charted at 04:00, but only validated at 04:40. This indicates that the care provider validated the measurement at 4:40 and indicated that it was a valid observation of the patient at 04:00. Conversely, it's also possible that the STORETIME occurs *before* the CHARTTIME. While a Glasgow Coma Scale may be charted at a CHARTTIME of 04:00, the observation may have been made and validated slightly before (e.g. 3:50). Again, the validation implies that the care staff believed the measurement to be an accurate reflection of the patient status at the given CHARTTIME.

Summing up: CHARTTIME vs. STORETIME

CHARTTIME is the time at which a measurement is *charted*. In almost all cases, this is the time which best matches the time of actual measurement. In the case of continuous vital signs (heart rate, respiratory rate, invasive blood pressure, non-invasive blood pressure, oxygen saturation), the CHARTTIME is usually exactly the time of measurement. STORETIME is the time at which the data is recorded in the database: logically it occurs after CHARTTIME, often by hours, but usually not more than that.

CHARTDATE

CHARTDATE is equivalent to CHARTTIME, except it does not contain any information on the time (all hour, minute, and seconds are 0 for these measurements).

ADMITTIME, DISCHTIME, DEATHTIME

ADMITTIME and DISCHTIME are the hospital admission and discharge times, respectively. DEATHTIME is the time of death of a patient if they died *in* hospital. If the patient did not die within the hospital for the given hospital admission, DEATHTIME will be null.

CREATETIME, UPDATETIME, ACKNOWLEDGETIME, OUTCOMETIME, FIRSTRESERVATIONTIME, CURRENTRESERVATIONTIME

CREATETIME is the time at which an ICU discharge was requested for a given patient. UPDATETIME is the time which the ICU discharge request was updated. ACKNOWLEDGETIME was the time at which the discharge request was acknowledged by the transfers team. OUTCOMETIME is the time at which the ICU discharge request was completed (with an outcome of 'Discharged' or 'Canceled'). FIRSTRESERVATIONTIME and CURRENTRESERVATIONTIME only occur for patients who require certain locations in the hospital.

INTIME, OUTTIME

INTIME and OUTTIME provide the time at which a patient entered and exited the given unit. In the ICUSTAYS table, the unit is always an ICU. In the TRANSFERS table, the unit can be any ward in the hospital.

STARTTIME, ENDTIME

For events which occur over a period of time, STARTTIME and ENDTIME provide the beginning and end time of the event. For medical infusions, these columns indicate the period over which the substance was administered.

COMMENTS_DATE

COMMENTS_DATE provides the time at which a cancel or edit comment was made for a given order.

DOB, DOD, DOD_HOSP, DOD_SSN

DOB is the patient's date of birth. If the patient is older than 89, their date of birth is set to 300 at their first admission. DOD is the patient's date of death: sourced either from the hospital database (DOD_HOSP) or the social security database (DOD_SSN).

TRANSFERTIME

TRANSFERTIME is the time at which the patient's service changes.

MIMIC-III v1.4

MIMIC-III v1.4 was released on 2 September 2016. It was a major release enhancing data quality and providing a large amount of additional data for Metavision patients.

Issues addressed include:

- #88 - Text data sourced from drop down menus or "pick lists" has been added to the database for Metavision patients. Users interested in the exact concepts added can query them as follows:
  ```
  select * from d_items
  where linksto = 'chartevents'
  and dbsource = 'metavision'
  and param_type = 'Text'.
  ```

All data was added to the `chartevents` table, increasing the raw (uncompressed) size of the datafile by 5 GB.
- #201 - Data in tables may now only occur within 1 year of a patient's hospital admission `admittime` and `dischtime`. Observations outside this range corresponded to typographical errors in the `charttime`, and this unreliable data has been removed.
- #197 - A small issue where a single observation was duplicated and assigned to two `icustay_id` (and sometimes two `hadm_id`) has been corrected: this affected 37 `icustay_id` and 80 `hadm_id`. Thanks @matteobonvini on GitHub for the report.
- #191 - A subset of patients were missing data due to the transition from CareVue to Metavision: this data has been restored.
- #193 - Some laboratory concepts (particularly CD counts) were conflated (i.e. percentages mixed with counts) - these have been corrected. Thanks Yuqi Si for the report (via StackExchange).
- #196 - The `amount` column in `inputevents_mv` did not correspond to `amountuom` (unit of measurement) - instead it corresponded to a "base" unit (e.g. grams instead of milligrams). Data in the `amount` has been updated to match the `amountuom` where appropriate. This also effected the `procedureevents_mv` table. Thanks @ngageorange on GitHub for the report.
- #198 - NULL values for character fields are now properly represented in the CSV file as empty fields with no double quotes. Database systems such as PostgreSQL will now recognize these fields as null when copying with the NULL '' qualifier.
- #199 - Addendums to discharge summaries in the notes can now be distinguished in the `noteevents` table: their `category` has remained 'Discharge summary', but their description has been changed from 'Report' to 'Addendum'
- #194 - The LOINC code for Total CO2 has been corrected (was 1959-6 total bicarb, is now 34728-6 total CO2)
- #186 - Data has been harmonized into EST.
- #180 - Some patients were erroneously labelled as in care unit `TSICU`: these have been corrected to `CSRU`
- #189 - Schema change: The `has_ioevents_data` column in `admissions` has been removed as it was no longer meaningful.
- #185 - Three text concepts (GCS, code status, RASS) had a null `linksto` field in `d_items`, even though the data existed in `chartevents`. This has been corrected. Note these were the only three text concepts available for Metavision patients in MIMIC-III v1.3 (see #88).
- #60 - The date shift for dates in the `text` field of `noteevents` has been corrected: before it corresponded to a different date shifted used in MIMIC-II (easily identifiable by years occurring between 2500-3500). It now matches the dates in the structured data.
- #188 - Negative microbiology cultures are now recorded in `micriobiologyevents` - negative cultures can be identified by a NULL value for the organism name (`org_name`).

Be sure to validate the checksum of the resultant file to ensure you have the correct version.

Past versions

This section lists past versions in reverse chronological order.

MIMIC-III v1.3

MIMIC-III v1.3 was released on 10 December 2015. It was a minor release enhancing the consistency of the dataset.

Issues addressed include:

- #175 - A new value for `DRG_VERSION` was added to the `DRGCODES` table to clarify why the same code matched to multiple descriptions.
- #174 - The `EDTIMEOUT` column was renamed to `EDOUTTIME` in the `ADMISSIONS` table for consistency with other timestamp columns.
- #173 - The `UOM` column was renamed to `VALUEUOM` in the `CHARTEVENTS`, `DATETIMEEVENTS`, and `LABEVENTS` tables for consistency with other `UOM` columns.
- #172, #177 - Several careunit acronyms were merged in the `TRANSFERS` and `ICUSTAYS` tables for ease of interpretation.
- #168 - A set of `ITEMID`s in the `INPUTEVENTS_CV` table were inappropriately low (<30000), they have no been corrected.
- #167 - Duplicate radiology reports were removed from the `NOTEEVENTS` table. These duplicates were present in the raw data.
- #166 - The `DBSOURCE` column was corrected from Metavision to CareVue for a set of patients in the `TRANSFERS` and `ICUSTAYS` tables.

MIMIC-III v1.2

MIMIC-III v1.2 was released on 10 November 2015. MIMIC-III v1.2 was a major release providing both bug fixes and a large amount of additional data.

Major issues addressed, including additional data made available:

- #130 and #135 - Duplicate data in various events tables with `CGID` has been removed
- #132 - Hospital expire flag was in the wrong table - now moved to `ADMISSIONS` table. `EXPIRE_FLAG` added to `PATIENTS` table.
- #137 - The `ITEMID` for input/output items has been properly shifted to range between 30000-40000. Previously it incorrectly ranged between 1-5000, and as a result did not match the dictionary entries in `D_ITEMS`.
- #141 - `CHARTTIME` and `STORETIME` have been added to `NOTEEVENTS`, if available. No times were available for ECG and echo reports (only the date).
- #144 - Dates of birth for patients > 89 are now shifted by 300 years for clarity.
- #151 - The time of emergency department registration and exit has been added to the admissions table, where available.
- #154 - A new table, `PROCEDUREEVENTS_MV`, has been added. This table contains information regarding the start and stop time for various procedures for Metavision patients. Procedures include x-rays, ventilation, dialysis, and others.
- #158 - The `IOEVENTS` table has been split into three tables: `INPUTEVENTS_CV` (CareVue patients only), `INPUTEVENTS_MV` (Metavision patients only), and `OUTPUTEVENTS` (all patients).
- #162 - 5,795,842 rows of data corresponding to yes/no answers have been added for Metavision patients

- #164 - 10,140 rows of “Non Iv meds” (e.g. vancomycin) have been added for Metavision patients

Minor issues addressed:

- #126 - ROW_ID, CGID, ORDERID and LINKORDERID now stored as INT instead of BIGINT
- #134 - The CR/LF characters which prefixed notes in NOTEEVENTS have been removed
- #136 - The VOLUME column in the events tables for inputs has been changed to AMOUNT
- #139 - The units for certain solution volumes, erroneously recorded as rate units, have been corrected
- #140 - The ORIGINALAMOUNT column no longer exists for rows with drug rates, only with drug volumes
- #148 - Removed RECORD_ID from NOTEEVENTS as it was redundant
- #149 - ICUSTAYEVENTS has been renamed ICUSTAYS
- #152 - The STARTTIME and ENDTIME columns have been renamed STARTDATE and ENDDATE for the PRESCRIPTIONS table to reflect the lack of time information.
- #155 - LINKSTO column in D_ITEMS has been corrected: now correctly refers to PROCEDUREEVENTS_MV, INPUTEVENTS_MV, INPUTEVENTS_CV, and OUTPUTEVENTS.
- #156 - The SEQUENCE column in DIAGNOSES_ICD and PROCEDURES_ICD has been renamed SEQ_NUM.
- #159 - A CONCEPTID column has been added to D_ITEMS for future ontology mapping/data harmonization.
- #163 - The CODE column has been removed from MICROBIOLOGYEVENTS and D_ITEMS as it was redundant to ITEMID

MIMIC-III v1.1

MIMIC-III v1.1 was released on 24 September 2015. It was primarily a bug fix release, and addresses the following issues:

- #116 - CGID was incorrect in the DATETIMEEVENTS, CHARTEVENTS, IOEVENTS and NOTEEVENTS tables. It has now been corrected.
- #117 - VALUENUM for GCS verbal response measurements has been corrected for Metavision (it was offset by -1).
- #118 - VALUENUM for all GCS measurements in CareVue is no longer null, and contains the appropriate value in the scale.
- #120 - DOD was incorrectly set to DOB - this has been fixed.
- #121 - IOEVENTS contained incorrect units for certain drugs (sometimes the unit was a rate when the actual observation was an amount, e.g. listed as “mcgkgmin” when it should have been “mg”).
- #122 - DBSOURCE in the TRANSFERS and ICUSTAYEVENTS tables has been corrected - originally it only contained ‘metavision’ when a patient was in the ICU, so the same patient would be listed as ‘carevue’ when out of the ICU and ‘metavision’ when inside the ICU.
- #123 - Precision in the IOEVENTS table has been fixed at 10 decimal places.

MIMIC-III v1.0

MIMIC-III v1.0 was released on 25 August 2015. It was a preliminary version and not widely publicized to allow for internal testing. As this was the first release of the database, and the successor of the MIMIC-II database, no changes are listed here. An overview of changes between MIMIC-II and MIMIC-III is provided here.

Feedback

MIMIC-II vs MIMIC-III

MIMIC-III is an extension of MIMIC-II: it incorporates the data contained in MIMIC-II (collected between 2001 - 2008) and augments it with newly collected data between 2008 - 2012. In addition, many data elements have been regenerated from the raw data in a more robust manner to improve the quality of the underlying data.

One of the challenges of adding new data resulted from a change in data management software at the Beth Israel Deaconess Medical Center. The original Philips CareVue system (which archived data from 2001 - 2008) was replaced with the new Metavision data management system (which continues to be used to the present). This page aims to facilitate the transition for researchers familiar with MIMIC-II who would like to continue their research with the updated MIMIC-III.

ITEMID mapping

In MIMIC-II there were multiple tables containing the same column name, `ITEMID`, but referring to different concepts. In attempt to alleviate confusion, we have merged all these tables into a single table, as follows:

Old table	Me
D_CHARTITEMS	D_ITEMS
D_IOITEMS	D_ITEMS

Old table	Me
D_MEDITEMS	D_ITEMS
D_CODEDITEMS	*deleted*
D_PARAMMAP_ITEMS	*deleted*
D_LABITEMS	D_LABITEMS

D_CHARTITEMS, D_IOITEMS, and D_MEDITEMS were all sourced from the same data source: the ICU database (specfically Philips CareVue). In contrast, iMDSoft Metavision only has a single table to define most `ITEMID` concepts. In order to simplify the schema and coalesce the databases, it was decided to merge together all the `ITEMID` dictionary tables into a single table, *except* D_LABITEMS (see below).

As the `ITEMID` ranges in the original D_ tables overlapped, they were offset by constant values. The following table maps the old `ITEMID` value ranges to the new ranges:

MIMIC-II source	Old range	New range
D_CHARTITEMS	1 - 20009	1-20009
D_MEDITEMS	1 - 405	30001 - 30405
D_IOITEMS	1 - 6807	40001 - 46807

For example, the charted item “Heart Rate” had an `ITEMID` of 211 in MIMIC-II (in D_CHARTITEMS). The new `ITEMID` for this is, again, 211, as there was no offset for D_CHARTITEMS. This means that any ITEMIDs used in the D_CHARTITEMS table will be directly portable to MIMIC-III (caveat: you will still need to extract additional `ITEMID` for the new metavision patients).

Conversely, if we look at the `ITEMID` = 51 in MIMIC-II D_MEDITEMS (vasopressin), we can find the same concept in MIMIC-III as 51 + 30000 = 30051.

D_CODEDITEMS

D_CODEDITEMS contained many concepts - most of these have been unchanged, simply moved to the new table D_ITEMS. The following table provides details.

ITEMID range	Concept	Where is it in MIMIC-I
60001 - 61018	DRG codes	Deleted - DRGs are stored with descriptions in DRG

ITEMID range	Concept	Where is it in MIMIC-I
70001 - 70093	Microbiology specimens	D_ITEMS, these ITEMID values are **unchanged**
80001 - 80312	Microbiology organisms	D_ITEMS, these ITEMID values are **unchanged**
90001 - 90031	Microbiology antibacterium	D_ITEMS, these ITEMID values are **unchanged**
100001 - 101885	ICD-9 procedure codes	Deleted - ICD9 codes are stored with descriptions

Lab ITEMIDs - not merged into D_ITEMS

The ITEMID for laboratory measurements in the D_LABITEMS and LABEVENTS tables in MIMIC-II do *not* match the ITEMID for laboratory measurements in MIMIC-III. We have provided a mapping table to facilitate the updating of queries which use this table. The mapping can be found in the MIMIC Code Repository.

Much of the data has been mapped to LOINC codes, which provide a standard ontology for recorded lab values. Careful inspection shows that the LOINC code for an ITEMID in MIMIC-III is, in rare occasions, different from the LOINC code for the same concept in MIMIC-II. This is usually attributable to the laboratory assigning a new LOINC code, which is done for many reasons, including changing the reagents of a laboratory test, changing the technique used to acquire the result or because the previous LOINC code was discontinued.

New ITEMIDs

While almost all concepts are unified into a single table (D_ITEMS), the concepts within that table are *not* unified. As patients admitted between 2001-2008 used a different ICU system to patients admitted between 2008-2012, the ITEMID values for the same concept differ in these two time periods. That is, for heart rates between 2001-2008, the ITEMID value 211 is appropriate. For heart rates for patients admitted between 2008 and onward, the ITEMID 220045 is appropriate. Most concepts will require the use of multiple ITEMID values in order to completely extract the data.

Schema changes

ADMISSIONS

ADMISSIONS is now sourced from the hospital database, rather than the ICU database. Changes include:

- Admission and discharge dates now have the time component
- Discharge location is now available
- Diagnosis on admission is now available

- ED registration and exit time are now available

CENSUSEVENTS replaced by TRANSFERS

The CENSUSEVENTS table was used in MIMIC-II to track patient hospital admissions. The original source of this table was the ICU database which recorded when patients were admitted and discharged from the ICU. In MIMIC-III, this table has been removed and replaced with the TRANSFERS table. The TRANSFERS table is sourced from the hospital admission, discharge, transfer (ADT) data. This has a number of advantages:

- The ADT data tracks a patient throughout the *entire* hospital stay, providing greater granularity and easier tracking of a patient's hospital course
- The ADT data provides information regarding ward location
- The ADT data has fewer erroneous admissions: frequently the ICU database contained erroneous entries corresponding to accidental admission/discharges
- The ADT data is available for all patients in the ICU database

DEMOGRAPHIC_DETAIL merged into ADMISSIONS

The DEMOGRAPHIC_DETAIL provided extra static information regarding a patient which rarely changed throughout an admission (e.g. age, ethnicity). This data was originally sourced from the ICU database. The new ADMISSIONS table is sourced entirely from the hospital database, and contained the same set of demographics. Instead of creating a new table for these demographics, it was decided to maintain the demographics within the ADMISSIONS table in the same format as it exists in the raw data.

DRGEVENTS renamed DRGCODES

The DRGEVENTS table has been renamed DRGCODES. The table still corresponds to diagnostic related groups, however the clarity of the data has been improved. First, a column corresponding to the DRG system has been provided. Second, if available, additional information regarding the DRG severity and mortality risk have been added. Finally, the DRG code has been mapped to a description based upon the underlying version. Due to frequent updates in the DRG coding system, it is non-trivial to map DRG codes to a description. To ease the use of this data, each DRG code has been mapped to a free text description which has been added to the table.

ICD9 renamed DIAGNOSES_ICD

To clarify the content of the table, the ICD9 table has been renamed to DIAGNOSES_ICD. The table contains diagnoses sourced from the hospital databased codified by ICD, usually ICD-9.

IOEVENTS, ADDITIVES, DELIVERIES and MEDEVENTS

Data in the IOEVENTS and MEDEVENTS tables is now contained in the OUTPUTEVENTS, INPUTEVENTS_CV and INPUTEVENTS_MV tables. As all the medications in the MEDEVENTS table were continuous infusions, they were all associated with an entry in IOEVENTS. MEDEVENTS would specify the drug rate, while IOEVENTS would specify the volume given. These tables have been consolidated to ease querying for drug deliveries.

POE_MED and POE_ORDER merged into PRESCRIPTIONS

The term POE, or provider order entry, references a hospital specific database which users may not be familiar with. To clarify the content of these tables, they have been merged into a single table named PRESCRIPTIONS.

Identifier changes

SUBJECT_ID, HADM_ID and ICUSTAY_ID

SUBJECT_ID between MIMIC-II v2.6 and MIMIC-III have been kept consistent, for example, SUBJECT_ID 788 is corresponds to the same patient in MIMIC-II v2.6 as it does in MIMIC-III.

HADM_ID have been regenerated in MIMIC-III. HADM_ID in MIMIC-II v2.6 will *not* match any HADM_ID in MIMIC-III. The newly generated HADM_ID range from 100,000 - 199,999 to help differentiate these IDs from others.

ICUSTAY_ID have been regenerated in MIMIC-III. ICUSTAY_ID in MIMIC-II v2.6 will *not* match any ICUSTAY_ID in MIMIC-III. Note that the newly generated ICUSTAY_ID range is between 200,000 - 299,999 to prevent confusion with other IDs.

New tables

CALLOUT

The CALLOUT table contains information about ICU discharge planning and execution. Each patient is “called out” of the ICU: which involves alerting hospital administrative staff that a bed, usually on the floor, is required for a patient currently in the ICU. The call out event also includes any precautions that the patient may require (such as susceptibility to MRSA or respiratory support). The

table provides information both on when the patient was deemed ready for discharge and when the patient actually left the ICU.

PROCEDURES_ICD

ICD-9 codes for procedures are now available in the PROCEDURES_ICD table.

INPUTEVENTS_CV and INPUTEVENTS_MV

Two different monitoring systems were operating in the hospital over the data collection period. The systems - Metavision and CareVue - recorded data in very different ways. We therefore made the decision not to merge the data, and instead provided two separate tables (INPUTEVENTS_CV and INPUTEVENTS_MV).

OUTPUTEVENTS

Data about outputs were recorded in a consistent fashion for the Metavision and CareVue databases. Therefore, we merged this data into one table: OUTPUTEVENTS.

Removed tables

There are many tables in MIMIC-II which are no longer present in MIMIC-III. In most cases, these tables were generated from the raw data for user convenience. We have transitioned from the approach of creating flat files of these tables to providing the source code necessary to regenerate them. This has two advantages: first it is much more efficient in terms of data transfer, and second it clarifies that these data are not “raw” in that they are not acquired directly from the databases but rather synthesized views of this data.

COMORBIDITY_SCORES

This table is frequently used to define the comorbid status for patients. Code for generating this table will be provided in the MIMIC Code Repository. The comorbidities will be defined using ICD-9 codes and DRG codes as proposed by Elixhauser et al.

DEMOGRAPHICEVENTS, D_DEMOGRAPHICITEMS

These tables were specific to the older CareVue database. As these tables are not present in the Metavision data, and the same information has been acquired from the hospital database (see the ADMISSIONS and D_PATIENTS tables), these tables have been removed.

D_CAREUNITS

The care unit identifier, CUID, has been removed from the various tables as it was unavailable in the Metavision ICU database. Care unit can be determined at the patient level, rather than the observation level, using the ICUSTAYEVENTS and TRANSFERS tables.

D_CODEDITEMS

This was a generated table to facilitate the interpretation of various coding systems, including microbiology, DRG, etc. The database has been restructured to have explicit definitions for these codes where appropriate.

D_PARAMMAP_ITEMS

This table is no longer needed as `ITEMID` concepts have been consolidated in the D_ITEMS table.

ICUSTAY_DETAILS

A script to generate ICUSTAY_DETAILS will be provided in the MIMIC Code Repository shortly.

PARAMETER_MAPPING

This table is no longer needed as `ITEMID` concepts have been consolidated in the D_ITEMS table.

WAVEFORM_*, D_WAVEFORM_SIGNALS

D_WAVEFORM_SIGNALS, WAVEFORM_METADATA, WAVEFORM_SEGMENTS, WAVEFORM_SEG_SIG, WAVEFORM_SIGNALS, WAVEFORM_TRENDS,

WAVEFORM_TREND_SIGNALS have been removed. The mapping to the waveform data is no longer provided within the relative database for clarity.

Querying MIMIC-III in Postgres

Prerequisites: *This tutorial assumes that you have an active connection to an instance of MIMIC-III running on PostgreSQL*. Note also that all the queries are written assuming that the MIMIC-III database is on your default search path. To change this in PostgreSQL, run the following command:

```
set search_path to mimiciii;
```

Note in the above we have assumed MIMIC-III is installed under the schema `mimiciii` - it may be different on your system.

1. Overview

This tutorial provides an introduction to the structure and content of the MIMIC-III database. By the end of this tutorial you will be able to:

- Obtain metadata from the various database objects (tables, views, etc);
- Perform basic queries on a single table;
- Perform basic 'joins' to combine tables and extract useful information;
- Use database 'views' to extract high-level information.

2. Database metadata

Metadata for a particular table can be obtained in Postgres with `\d+ <schema>.<tablename>`. For example, the following command prints metadata for the admissions table:

```
\d+ MIMICIII.ADMISSIONS;

-- returns the following metadata

       Column        |            Type             | Modifiers | ...
---------------------+-----------------------------+-----------+ ...
 row_id              | integer                     | not null  | ...
 subject_id          | integer                     | not null  | ...
 hadm_id             | integer                     | not null  | ...
 admittime           | timestamp(0) without time zone | not null  | ...
 dischtime           | timestamp(0) without time zone | not null  | ...
 ...
```

Try looking at the metadata for other tables such as `patients` and `chartevents`.

3. Patient numbers

Ensure that the 'Query...' tab at the top of the screen is selected. SQL queries can be entered in the top panel and the results will be displayed at the bottom when the 'Execute query' button is pressed. Enter the following SQL in the box and press the 'Execute query' button.

```
SELECT *
FROM patients;
```

At the bottom of the screen you will see three columns: subject_id, gender, and date of birth. 50 records are retrieved at a time and you can page through the results using the controls at the bottom of the screen.

Obtain the number of patients with the following query:

```
SELECT COUNT(*)
FROM patients;
```

The 'gender' column identifies the gender of the patient. We can obtain the distinct values used to indicate gender using the following query:

```
SELECT DISTINCT(gender)
FROM patients;
```

We can see that 'M' and 'F' are the two characters used to indicate patient gender. We can use this information to obtain the number of female patients by adding a condition to select rows where the gender is 'F':

```
SELECT COUNT(*)
FROM patients
WHERE gender = 'F';
```

And the numbers of male and female patients can be obtained using the following query:

```
SELECT gender, COUNT(*)
FROM patients
GROUP BY gender;
```

4. Mortality and admissions

A flag which records whether or not a patient died in the hospital is stored in the patients table. Count the number of patients who died using the following query:

```
SELECT expire_flag, COUNT(*)
FROM patients
GROUP BY expire_flag;
```

The database also contains date of death for patients who died inside the hospital in the column 'dod_hosp' and the date of death found in social security death records in 'dod_ssn'. This information from both columns is merged in the 'dod' column with priority given to 'dod_hosp'. Note that this database contains

adult and neonatal patients which will affect the mortality statistics. Categorizing patients into different age groups is carried out in the next section.

5. Patient age and mortality

To determine the adult mortality rate we must first select the adult patients. We define adults as those patients who are 15 years old or above at the date of their first admission. To perform this query, we must combine the patients and admissions tables to find patient admission dates and dates of birth. We have denoted 'admissions' with the alias 'a' and 'patients' with alias 'p':

```
SELECT p.subject_id, p.dob, a.hadm_id,
    a.admittime, p.expire_flag
FROM admissions a
INNER JOIN patients p
ON p.subject_id = a.subject_id;
```

Next we find the earliest admission date for each patient. This requires the use of two functions: the 'MIN' function, which obtains the minimum value, and the 'PARTITION BY' function, which determines the groups over which the minimum value is obtained. To determine the earliest admission time for each patient:

```
SELECT p.subject_id, p.dob, a.hadm_id,
    a.admittime, p.expire_flag,
    MIN (a.admittime) OVER (PARTITION BY p.subject_id) AS first_admittime
FROM admissions a
INNER JOIN patients p
ON p.subject_id = a.subject_id
ORDER BY a.hadm_id, p.subject_id;
```

A patient's age is given by the difference between their date of birth and the date of their first admission. We can obtain this by combining the above query with another query to provide the ages. Furthermore, we assign categories to different ages: >= 15 years old are adults and the rest are assigned to other categories. Note the use of the `WITH` clause, which allows us to make a temporary view which we can query against in subsequent lines.

```
WITH first_admission_time AS
(
  SELECT
      p.subject_id, p.dob, p.gender
      , MIN (a.admittime) AS first_admittime
      , MIN( ROUND( (cast(admittime as date) - cast(dob as date)) /
365.242,2) )
          AS first_admit_age
  FROM patients p
  INNER JOIN admissions a
  ON p.subject_id = a.subject_id
  GROUP BY p.subject_id, p.dob, p.gender
  ORDER BY p.subject_id
```

```
)
SELECT
    subject_id, dob, gender
    , first_admittime, first_admit_age
    , CASE
        -- all ages > 89 in the database were replaced with 300
        WHEN first_admit_age > 89
            then '>89'
        WHEN first_admit_age >= 14
            THEN 'adult'
        WHEN first_admit_age <= 1
            THEN 'neonate'
        ELSE 'middle'
        END AS age_group
FROM first_admission_time
ORDER BY subject_id
```

The above query can now be combined with the **WHERE** and **COUNT** functions described earlier to determine the number of adult patients, whether or not they died, and therefore, their mortality rate.

```
WITH first_admission_time AS
(
  SELECT
      p.subject_id, p.dob, p.gender
      , MIN (a.admittime) AS first_admittime
      , MIN( ROUND( (cast(admittime as date) - cast(dob as date)) /
365.242,2) )
          AS first_admit_age
  FROM patients p
  INNER JOIN admissions a
  ON p.subject_id = a.subject_id
  GROUP BY p.subject_id, p.dob, p.gender
  ORDER BY p.subject_id
)
, age as
(
  SELECT
      subject_id, dob, gender
      , first_admittime, first_admit_age
      , CASE
          -- all ages > 89 in the database were replaced with 300
          -- we check using > 100 as a conservative threshold to ensure we
capture all these patients
          WHEN first_admit_age > 100
              then '>89'
          WHEN first_admit_age >= 14
              THEN 'adult'
          WHEN first_admit_age <= 1
              THEN 'neonate'
          ELSE 'middle'
          END AS age_group
  FROM first_admission_time
)
select age_group, gender
```

```
  , count(subject_id) as NumberOfPatients
from age
group by age_group, gender
```

Note that no 'middle' patients show up - this reflects the fact that MIMIC-III does not contain data from paediatric patients.

6. ICU stays

In the MIMIC-III database, we define an ICU stay to be continuous if a patient is returned to an ICU room within 24 hours of being moved to a ward. Patient ICU movements are recorded in the transfers table:

```
SELECT *
FROM transfers;
```

The columns should be fairly self explanatory. Click on the transfers table on the left hand side if you need more information about the columns and the data they contain. The 'prev_careunit' and 'curr_careunit' contain the names of the previous and current careunit respectively. The transfers table also includes 'prev_wardid' and 'curr_wardid' columns, which contain the IDs of the previous and current careunit respectively. Ward IDs, which specify the room within a ward, have no corresponding key in order to protect patient health information.

The transfers table may have multiple entries per patient to provide detail of all movement between various careunits of the hospital. The first entry in the transfers table for a patient who comes into the ICU will have nothing in the 'prev_careunit' column. Similarly, the last entry for a patient will have nothing in the 'curr_careunit'. Entries that have nothing in both previous and current careunit columns indicate that the patient has been transfered between non intensive care units. An example query for one patient and result from the transfers table is shown below. Note that columns 'intime', 'outtime', and 'los' have been truncated.

```
SELECT *
FROM transfers
WHERE HADM_ID = 112213;
```

row_id	subject_id	hadm_id	
54	12	112213	
55	12	112213	
56	12	112213	232669
57	12	112213	
58	12	112213	232669

row_id	subject_id	hadm_id
59	12	112213
60	12	112213

7. Services

Services is a newly added table in MIMIC-III which contains information about the transfers from being under one service to another during a patient's stay. The services table contains columns including 'prev_service' and 'curr_service' which contain the names of previous and current services respectively. 'transfertime' is the time at which the patient was moved from 'prev_service' to 'curr_service'.

8. Tutorial problem

How would you gather useful information about patients admitted to the ICU? The problem can be broken down into several parts:

Step 1

First start with retrieving 'subject_id', 'hadm_id', 'icustay_id', 'intime', and 'outtime' from the 'icustays' table.

Step 2

Using the patients table retrieve the calculated age of patients.

Step 3

Separate neonates from adult patients.

Step 4

By incorporating the admissions table, find how long each stay was **BEFORE** the patients were admitted to the ICU

Step 5

Next find the date of the patient's death if applicable.

Step 6

Then find those deaths that occurred while the patients were in the hospital.

Step 7

Find how many of those deaths occurred within the ICU.

Solutions to the problems in section 8

Solution to step 1

```
SELECT ie.subject_id, ie.hadm_id, ie.icustay_id,
    ie.intime, ie.outtime
FROM icustays ie;
```

Solution to step 2

```
SELECT ie.subject_id, ie.hadm_id, ie.icustay_id,
    ie.intime, ie.outtime,
    ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242, 2) AS age
FROM icustays ie
INNER JOIN patients pat
ON ie.subject_id = pat.subject_id;
```

Solution to step 3

```
SELECT ie.subject_id, ie.hadm_id, ie.icustay_id,
    ie.intime, ie.outtime,
    ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242, 2) AS
age,
    CASE
        WHEN ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242,
2) <= 1
            THEN 'neonate'
        WHEN ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242,
2) <= 14
            THEN 'middle'
        -- all ages > 89 in the database were replaced with 300
        WHEN ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242,
2) > 100
            then '>89'
        ELSE 'adult'
        END AS ICUSTAY_AGE_GROUP
FROM icustays ie
INNER JOIN patients pat
ON ie.subject_id = pat.subject_id;
```

Solution to step 4

```
SELECT ie.subject_id, ie.hadm_id, ie.icustay_id,
    ie.intime, ie.outtime,
    ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242, 2) as
age,
    ROUND((cast(ie.intime as date) - cast(adm.admittime as date))/365.242, 2)
as preiculos,
    CASE
        WHEN ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242,
2) <= 1
            THEN 'neonate'
        WHEN ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242,
2) <= 14
            THEN 'middle'
        -- all ages > 89 in the database were replaced with 300
        WHEN ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242,
2) > 100
            THEN '>89'
```

```
        ELSE 'adult'
        END AS ICUSTAY_AGE_GROUP
FROM icustays ie
INNER JOIN patients pat
ON ie.subject_id = pat.subject_id
INNER JOIN admissions adm
ON ie.hadm_id = adm.hadm_id;
```

Solution to step 5

```
SELECT ie.subject_id, ie.hadm_id, ie.icustay_id,
    ie.intime, ie.outtime, adm.deathtime,
    ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242, 2) as
age,
    ROUND((cast(ie.intime as date) - cast(adm.admittime as date))/365.242, 2)
AS preiculos,
    CASE
        WHEN ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242,
2) <= 1
            THEN 'neonate'
        WHEN ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242,
2) <= 14
            THEN 'middle'
        -- all ages > 89 in the database were replaced with 300
        WHEN ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242,
2) > 100
            THEN '>89'
        ELSE 'adult'
        END AS ICUSTAY_AGE_GROUP
FROM icustays ie
INNER JOIN patients pat
ON ie.subject_id = pat.subject_id
INNER JOIN admissions adm
ON ie.hadm_id = adm.hadm_id;
```

Solution to step 6

```
SELECT ie.subject_id, ie.hadm_id, ie.icustay_id,
    ie.intime, ie.outtime, adm.deathtime,
    ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242, 2) AS
age,
    ROUND((cast(ie.intime as date) - cast(adm.admittime as date))/365.242, 2)
AS preiculos,
    CASE
        WHEN ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242,
2) <= 1
            THEN 'neonate'
        WHEN ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242,
2) <= 14
            THEN 'middle'
        -- all ages > 89 in the database were replaced with 300
        WHEN ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242,
2) > 100
            THEN '>89'
        ELSE 'adult'
        END AS ICUSTAY_AGE_GROUP,
    -- note that there is already a "hospital_expire_flag" field in the
admissions table which you could use
```

```
    CASE
        WHEN adm.hospital_expire_flag = 1 then 'Y'
    ELSE 'N'
    END AS hospital_expire_flag
FROM icustays ie
INNER JOIN patients pat
ON ie.subject_id = pat.subject_id
INNER JOIN admissions adm
ON ie.hadm_id = adm.hadm_id;
```

Solution to step 7

```
SELECT ie.subject_id, ie.hadm_id, ie.icustay_id,
    ie.intime, ie.outtime, adm.deathtime,
    ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242, 2) AS
age,
    ROUND((cast(ie.intime as date) - cast(adm.admittime as date))/365.242, 2)
AS preiculos,
    CASE
        WHEN ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242,
2) <= 1
            THEN 'neonate'
        WHEN ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242,
2) <= 14
            THEN 'middle'
        -- all ages > 89 in the database were replaced with 300
        WHEN ROUND((cast(ie.intime as date) - cast(pat.dob as date))/365.242,
2) > 100
            THEN '>89'
        ELSE 'adult'
        END AS ICUSTAY_AGE_GROUP,
    -- note that there is already a "hospital_expire_flag" field in the
admissions table which you could use
    CASE
        WHEN adm.hospital_expire_flag = 1 then 'Y'
    ELSE 'N'
    END AS hospital_expire_flag,
    -- note also that hospital_expire_flag is equivalent to "Is adm.deathtime
not null?"
    CASE
        WHEN adm.deathtime BETWEEN ie.intime and ie.outtime
            THEN 'Y'
        -- sometimes there are typographical errors in the death date, so
check before intime
        WHEN adm.deathtime <= ie.intime
            THEN 'Y'
        WHEN adm.dischtime <= ie.outtime
            AND adm.discharge_location = 'DEAD/EXPIRED'
            THEN 'Y'
        ELSE 'N'
        END AS ICUSTAY_EXPIRE_FLAG
FROM icustays ie
INNER JOIN patients pat
ON ie.subject_id = pat.subject_id
INNER JOIN admissions adm
ON ie.hadm_id = adm.hadm_id;
```

BigQuery

BigQuery is a columnar, distributed relational database management system. BigQuery accesses only the columns specified in the query, making it ideal for data analysis workflows. BigQuery can be used to query a cloud based instance of MIMIC-III through the web browser. To access MIMIC-III on BigQuery, see the cloud data access guide.

MIMIC-III on BigQuery

MIMIC-III is organized into three "datasets" on BigQuery. BigQuery uses datasets to organize data into subgroups. If you are familiar with the database concept of a schema, then you can understand BigQuery datasets as the logical equivalent of a schema. The three datasets on BigQuery are:

- `mimiciii_clinical` - Almost all of MIMIC-III is stored here.
- `mimiciii_notes` - The NOTEEVENTS table is stored here.
- `mimiciii_derived` - Useful derivations which have been extracted from MIMIC-III are stored here for reuse by the community. All code to generate these views are open source and publicly available on the mimic-code repository.

This tutorial will focus on querying the bulk of MIMIC-III data stored in `mimiciii_clinical`.

1. Overview

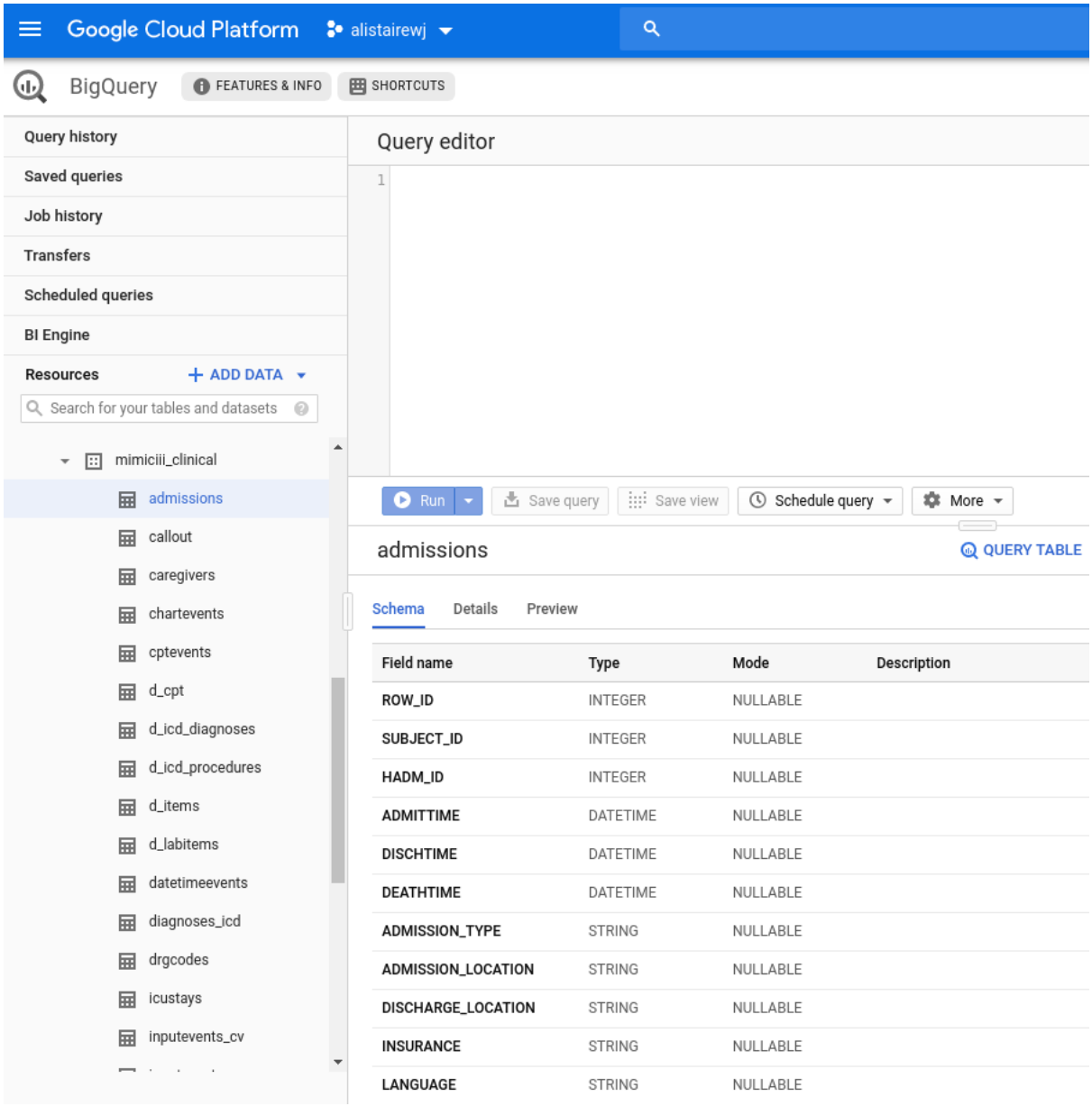
This tutorial provides an introduction to the structure and content of the MIMIC-III database. By the end of this tutorial you will be able to:

- Obtain metadata from the various database objects (tables, views, etc);
- Perform basic queries on a single table;
- Perform basic 'joins' to combine tables and extract useful information;
- Use database 'views' to extract high-level information.

First, navigate to BigQuery: https://console.cloud.google.com/bigquery

2. Database metadata

Metadata for a particular table can be obtained by clicking the table on the left sidebar on BigQuery, as below:



If you do not see the `mimiciii_clinical` dataset on BigQuery, you may need to request access to MIMIC-III on BigQuery, or pin the project to your sidebar.

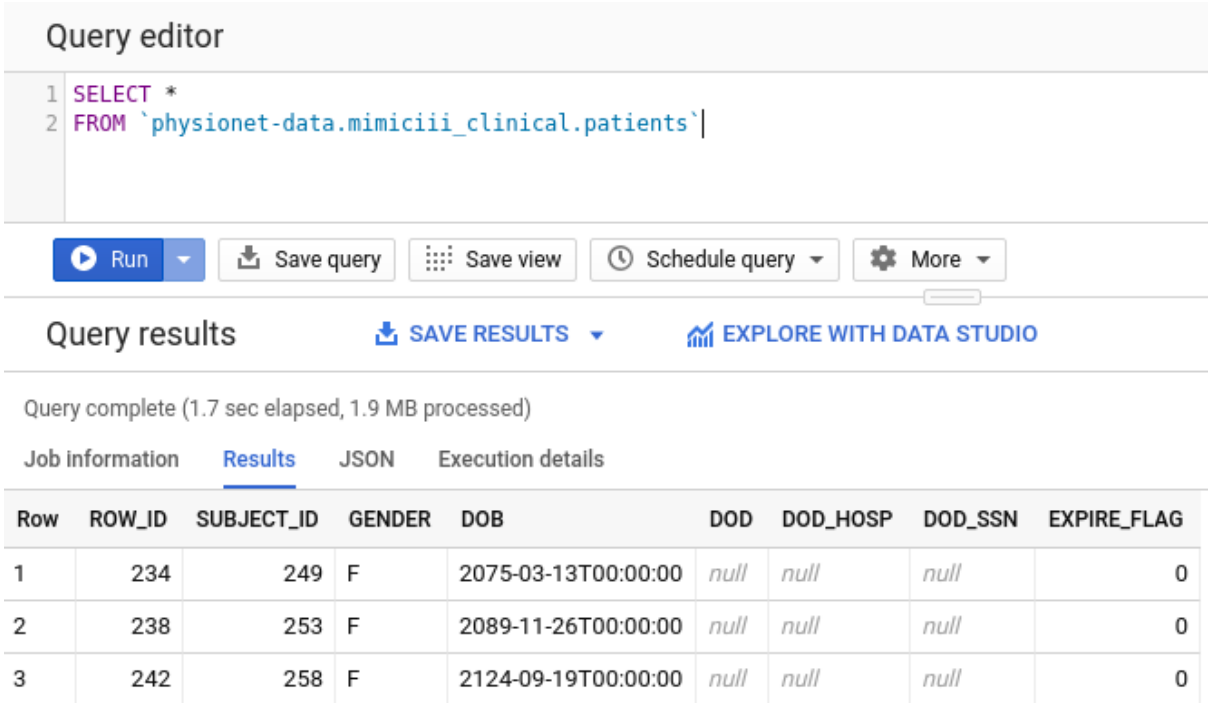
Try looking at the metadata for other tables such as `patients` and `chartevents`.

3. Patient numbers

SQL queries can be entered in the top "Query editor" panel and the results will be displayed at the bottom when the 'Run' button is pressed. Enter the following SQL in the box and press the 'Execute query' button.

```
SELECT *
FROM `physionet-data.mimiciii_clinical.patients`
```

You should see the following:



Row	ROW_ID	SUBJECT_ID	GENDER	DOB	DOD	DOD_HOSP	DOD_SSN	EXPIRE_FLAG
1	234	249	F	2075-03-13T00:00:00	null	null	null	0
2	238	253	F	2089-11-26T00:00:00	null	null	null	0
3	242	258	F	2124-09-19T00:00:00	null	null	null	0

50 records are retrieved at a time and you can page through the results using the controls at the bottom of the screen.

Obtain the number of patients with the following query:

```
SELECT COUNT(*)
FROM `physionet-data.mimiciii_clinical.patients`;
```

The 'gender' column identifies the gender of the patient. We can obtain the distinct values used to indicate gender using the following query:

```
SELECT DISTINCT(gender)
FROM `physionet-data.mimiciii_clinical.patients`;
```

We can see that 'M' and 'F' are the two characters used to indicate patient gender. We can use this information to obtain the number of female patients by adding a condition to select rows where the gender is 'F':

```
SELECT COUNT(*)
FROM `physionet-data.mimiciii_clinical.patients`
WHERE gender = 'F';
```

And the numbers of male and female patients can be obtained using the following query:

```
SELECT gender, COUNT(*)
FROM `physionet-data.mimiciii_clinical.patients`
GROUP BY gender;
```

4. Mortality and admissions

A flag which records whether or not a patient died in the hospital is stored in the patients table. Count the number of patients who died using the following query:

```
SELECT expire_flag, COUNT(*)
FROM `physionet-data.mimiciii_clinical.patients`
GROUP BY expire_flag;
```

The database also contains date of death for patients who died inside the hospital in the column 'dod_hosp' and the date of death found in social security death records in 'dod_ssn'. This information from both columns is merged in the 'dod' column with priority given to 'dod_hosp'. Note that this database contains adult and neonatal patients which will affect the mortality statistics. Categorizing patients into different age groups is carried out in the next section.

5. Patient age and mortality

To determine the adult mortality rate we must first select the adult patients. We define adults as those patients who are 15 years old or above at the date of their first admission. To perform this query, we must combine the patients and admissions tables to find patient admission dates and dates of birth. We have denoted 'admissions' with the alias 'a' and 'patients' with alias 'p':

```
SELECT p.subject_id, p.dob, a.hadm_id,
    a.admittime, p.expire_flag
FROM `physionet-data.mimiciii_clinical.admissions` a
INNER JOIN `physionet-data.mimiciii_clinical.patients` p
  ON p.subject_id = a.subject_id;
```

Next we find the earliest admission date for each patient. This requires the use of two functions: the 'MIN' function, which obtains the minimum value, and the 'PARTITION BY' function, which determines the groups over which the minimum value is obtained. To determine the earliest admission time for each patient:

```
SELECT p.subject_id, p.dob, a.hadm_id,
    a.admittime, p.expire_flag,
    MIN (a.admittime) OVER (PARTITION BY p.subject_id) AS first_admittime
FROM `physionet-data.mimiciii_clinical.admissions` a
INNER JOIN `physionet-data.mimiciii_clinical.patients` p
  ON p.subject_id = a.subject_id
ORDER BY a.hadm_id, p.subject_id;
```

A patient's age is given by the difference between their date of birth and the date of their first admission. In BigQuery, we must use special functions to do mathematical operations on datetime columns (note: as most columns in MIMIC-III have dates *and* times, they are often called `datetime` columns to differentiate them from `date` columns, which only store the date). The reference page for [datetime operations is available on the BigQuery documentation website](#).

Let's calculate the age of patients at their time of admission:

```
SELECT p.subject_id, p.dob, a.hadm_id,
    a.admittime, p.expire_flag,
    DATETIME_DIFF(admittime, dob, YEAR) as age
FROM `physionet-data.mimiciii_clinical.admissions` a
INNER JOIN `physionet-data.mimiciii_clinical.patients` p
  ON p.subject_id = a.subject_id
ORDER BY p.subject_id, a.hadm_id;
```

You should see the following:

Query editor

```
1
2 SELECT p.subject_id, p.dob, a.hadm_id,
3     a.admittime, p.expire_flag,
4     DATETIME_DIFF(admittime, dob, YEAR) as age
5 FROM `physionet-data.mimiciii_clinical.admissions` a
6 INNER JOIN `physionet-data.mimiciii_clinical.patients` p
7   ON p.subject_id = a.subject_id
8 ORDER BY p.subject_id, a.hadm_id;
```

Run | Save query | Save view | Schedule query | More

Query results | SAVE RESULTS | EXPLORE WITH DATA STUDIO

Query complete (1.8 sec elapsed, 2.4 MB processed)

Job information | Results | JSON | Execution details

Row	subject_id	dob	hadm_id	admittime	expire_flag	age
1	2	2138-07-17T00:00:00	163353	2138-07-17T19:04:00	0	0
2	3	2025-04-11T00:00:00	145834	2101-10-20T19:08:00	1	76
3	4	2143-05-12T00:00:00	185777	2191-03-16T00:28:00	0	48
4	5	2103-02-02T00:00:00	178980	2103-02-02T04:31:00	0	0
5	6	2109-06-21T00:00:00	107064	2175-05-30T07:15:00	0	66
6	7	2121-05-23T00:00:00	118037	2121-05-23T15:05:00	0	0
7	8	2117-11-20T00:00:00	159514	2117-11-20T10:22:00	0	0
8	9	2108-01-26T00:00:00	150750	2149-11-09T13:06:00	1	41
9	10	2103-06-28T00:00:00	184167	2103-06-28T11:36:00	0	0
10	11	2128-02-22T00:00:00	194540	2178-04-16T06:18:00	1	50
11	12	2032-03-24T00:00:00	112213	2104-08-07T10:15:00	1	72
12	13	2127-02-27T00:00:00	143045	2167-01-08T18:43:00	0	40
13	16	2178-02-03T00:00:00	103251	2178-02-03T06:35:00	0	0
14	17	2087-07-14T00:00:00	161087	2135-05-09T14:11:00	0	48
15	17	2087-07-14T00:00:00	194023	2134-12-27T07:15:00	0	47

Note at the bottom we have calculated the age for `subject_id` = 17 twice, once for each of their hospital admissions (the `hadm_id` is different between the rows).

If we examine the same patient more than once when calculating a statistic such as mortality, then our estimate will contain "repeated measures". Unless we handle this phenomenon explicitly, our calculation will be biased.

A simple solution is to only examine the first hospitalization for each patient, which we can do with a `GROUP BY` clause.

```
SELECT
    p.subject_id, p.dob, p.gender
    , MIN(a.admittime) AS first_admittime
    , MIN( DATETIME_DIFF(admittime, dob, YEAR) )
        AS first_admit_age
FROM `physionet-data.mimiciii_clinical.patients` p
INNER JOIN `physionet-data.mimiciii_clinical.admissions` a
  ON p.subject_id = a.subject_id
GROUP BY p.subject_id, p.dob, p.gender
ORDER BY p.subject_id
```

Note in the above:

1. Our group is defined by columns listed after `GROUP BY`, in this case: `p.subject_id`, `p.dob`, `p.gender`. BigQuery will return a row for each unique combination of these three columns. Since we know that each patient only has one `dob` and one `gender` in the PATIENTS table, we know that this will return one row per `subject_id`.
2. We have removed `hadm_id`, as we will collapse multiple `hadm_id` into a single row.
3. We have wrapped `admittime` in a `MIN()` aggregate function. For dates, `MIN()` returns the earliest date.
4. We have wrapped age in a `MIN()` aggregate function. This tells BigQuery to extract the minimum age across all hospital admissions.

Now that we have a set of unique patients with their age, we can group them into sensible categories based upon age and calculate the mortality rate in these categories. Patients with an age >= 15 years old are adults and the rest are assigned to other categories. Note the use of the `WITH` clause, which allows us to make a temporary view which we can query against in subsequent lines.

```
WITH first_admission_time AS
(
  SELECT
      p.subject_id, p.dob, p.gender
      , MIN (a.admittime) AS first_admittime
      , MIN( DATETIME_DIFF(admittime, dob, YEAR) )
          AS first_admit_age
  FROM `physionet-data.mimiciii_clinical.patients` p
  INNER JOIN `physionet-data.mimiciii_clinical.admissions` a
  ON p.subject_id = a.subject_id
  GROUP BY p.subject_id, p.dob, p.gender
  ORDER BY p.subject_id
)
SELECT
```

```
    subject_id, dob, gender
    , first_admittime, first_admit_age
    , CASE
        -- all ages > 89 in the database were replaced with 300
        WHEN first_admit_age > 89
            then '>89'
        WHEN first_admit_age >= 15
            THEN 'adult'
        WHEN first_admit_age <= 1
            THEN 'neonate'
        ELSE 'middle'
        END AS age_group
FROM first_admission_time
ORDER BY subject_id
```

The above query can now be combined with the **WHERE** and **COUNT** functions described earlier to determine the number of adult patients, whether or not they died, and therefore, their mortality rate.

```
WITH first_admission_time AS
(
  SELECT
      p.subject_id, p.dob, p.gender
      , MIN (a.admittime) AS first_admittime
      , MIN( DATETIME_DIFF(admittime, dob, YEAR) )
          AS first_admit_age
  FROM `physionet-data.mimiciii_clinical.patients` p
  INNER JOIN `physionet-data.mimiciii_clinical.admissions` a
  ON p.subject_id = a.subject_id
  GROUP BY p.subject_id, p.dob, p.gender
  ORDER BY p.subject_id
)
, age as
(
  SELECT
      subject_id, dob, gender
      , first_admittime, first_admit_age
      , CASE
          -- all ages > 89 in the database were replaced with 300
          WHEN first_admit_age > 89
              then '>89'
          WHEN first_admit_age >= 14
              THEN 'adult'
          WHEN first_admit_age <= 1
              THEN 'neonate'
          ELSE 'middle'
          END AS age_group
  FROM first_admission_time
)
select age_group, gender
  , count(subject_id) as NumberOfPatients
from age
group by age_group, gender
```

Note that no 'middle' patients show up - this reflects the fact that MIMIC-III does not contain data from paediatric patients.

6. ICU stays

In the MIMIC-III database, we define an ICU stay to be continuous if a patient is returned to an ICU room within 24 hours of being moved to a ward. Patient ICU movements are recorded in the transfers table:

```
SELECT *
FROM `physionet-data.mimiciii_clinical.transfers`;
```

The columns should be fairly self explanatory. Click on the transfers table on the left hand side if you need more information about the columns and the data they contain. The 'prev_careunit' and 'curr_careunit' contain the names of the previous and current careunit respectively. The transfers table also includes 'prev_wardid' and 'curr_wardid' columns, which contain the IDs of the previous and current careunit respectively. Ward IDs, which specify the room within a ward, have no corresponding key in order to protect patient health information.

The transfers table may have multiple entries per patient to provide detail of all movement between various careunits of the hospital. The first entry in the transfers table for a patient who comes into the ICU will have nothing in the 'prev_careunit' column. Similarly, the last entry for a patient will have nothing in the 'curr_careunit'. Entries that have nothing in both previous and current careunit columns indicate that the patient has been transfered between non intensive care units. An example query for one patient and result from the transfers table is shown below. Note that columns 'intime', 'outtime', and 'los' have been truncated.

```
SELECT *
FROM `physionet-data.mimiciii_clinical.transfers`
WHERE HADM_ID = 112213;
```

row_id	subject_id	hadm_id	
54	12	112213	
55	12	112213	
56	12	112213	232669
57	12	112213	
58	12	112213	232669
59	12	112213	
60	12	112213	

7. Services

Services is a newly added table in MIMIC-III which contains information about the transfers from being under one service to another during a patient's stay. The services table contains columns including 'prev_service' and 'curr_service' which contain the names of previous and current services respectively. 'transfertime' is the time at which the patient was moved from 'prev_service' to 'curr_service'.

8. Tutorial problem

How would you gather useful information about patients admitted to the ICU? The problem can be broken down into several parts:

Step 1

First start with retrieving 'subject_id', 'hadm_id', 'icustay_id', 'intime', and 'outtime' from the 'icustays' table.

Step 2

Using the patients table retrieve the calculated age of patients.

Step 3

Separate neonates from adult patients.

Step 4

By incorporating the admissions table, find how long each stay was **BEFORE** the patients were admitted to the ICU

Step 5

Next find the date of the patient's death if applicable.

Step 6

Then find those deaths that occurred while the patients were in the hospital.

Step 7

Find how many of those deaths occurred within the ICU.

Solutions to the problems in section 8

Solution to step 1

```
SELECT ie.subject_id, ie.hadm_id, ie.icustay_id,
    ie.intime, ie.outtime
```

```
FROM `physionet-data.mimiciii_clinical.icustays` ie;
```

Solution to step 2

```
SELECT ie.subject_id, ie.hadm_id, ie.icustay_id,
    ie.intime, ie.outtime,
    DATETIME_DIFF(admittime, dob, YEAR) AS age
-- we use 'ie' as an alias for the icustays table
FROM `physionet-data.mimiciii_clinical.icustays` ie
-- we use 'pat' as an alias for the patients table
INNER JOIN `physionet-data.mimiciii_clinical.patients` pat
-- since subject_id is unique for every row in patients, we will get
-- one row for every row in icustays (ie)
  ON ie.subject_id = pat.subject_id;
```

Solution to step 3

```
SELECT ie.subject_id, ie.hadm_id, ie.icustay_id,
    ie.intime, ie.outtime,
    DATETIME_DIFF(admittime, dob, YEAR) AS age,
    CASE
        WHEN DATETIME_DIFF(admittime, dob, YEAR) <= 1
            THEN 'neonate'
        WHEN DATETIME_DIFF(admittime, dob, YEAR) <= 14
            THEN 'middle'
        -- all ages > 89 in the database were replaced with 300
        WHEN DATETIME_DIFF(admittime, dob, YEAR) > 89
            then '>89'
        ELSE 'adult'
        END AS ICUSTAY_AGE_GROUP
FROM `physionet-data.mimiciii_clinical.icustays` ie
INNER JOIN `physionet-data.mimiciii_clinical.patients` pat
  ON ie.subject_id = pat.subject_id;
```

Solution to step 4

```
SELECT ie.subject_id, ie.hadm_id, ie.icustay_id,
    ie.intime, ie.outtime,
    DATETIME_DIFF(adm.admittime, pat.dob, YEAR) as age,
    DATETIME_DIFF(ie.intime, adm.admittime, DAY) as preiculos,
    CASE
        WHEN DATETIME_DIFF(adm.admittime, pat.dob, YEAR) <= 1
            THEN 'neonate'
        WHEN DATETIME_DIFF(adm.admittime, pat.dob, YEAR) <= 14
            THEN 'middle'
        WHEN DATETIME_DIFF(adm.admittime, pat.dob, YEAR) > 89
            THEN '>89'
        ELSE 'adult'
        END AS ICUSTAY_AGE_GROUP
FROM `physionet-data.mimiciii_clinical.icustays` ie
INNER JOIN `physionet-data.mimiciii_clinical.patients` pat
  ON ie.subject_id = pat.subject_id
INNER JOIN `physionet-data.mimiciii_clinical.admissions` adm
  ON ie.hadm_id = adm.hadm_id;
```

Solution to step 5

```
SELECT ie.subject_id, ie.hadm_id, ie.icustay_id,
    ie.intime, ie.outtime,
    -- patient death in hospital is stored in the admissions table
```

```
    adm.deathtime,
    DATETIME_DIFF(adm.admittime, pat.dob, YEAR) as age,
    DATETIME_DIFF(ie.intime, adm.admittime, DAY) as preiculos,
    CASE
        WHEN DATETIME_DIFF(adm.admittime, pat.dob, YEAR) <= 1
            THEN 'neonate'
        WHEN DATETIME_DIFF(adm.admittime, pat.dob, YEAR) <= 14
            THEN 'middle'
        WHEN DATETIME_DIFF(adm.admittime, pat.dob, YEAR) > 89
            THEN '>89'
        ELSE 'adult'
        END AS ICUSTAY_AGE_GROUP
FROM `physionet-data.mimiciii_clinical.icustays` ie
INNER JOIN `physionet-data.mimiciii_clinical.patients` pat
  ON ie.subject_id = pat.subject_id
INNER JOIN `physionet-data.mimiciii_clinical.admissions` adm
  ON ie.hadm_id = adm.hadm_id;
```

Solution to step 6

```
SELECT ie.subject_id, ie.hadm_id, ie.icustay_id,
    ie.intime, ie.outtime,
    -- patient death in hospital is stored in the admissions table
    adm.deathtime,
    DATETIME_DIFF(adm.admittime, pat.dob, YEAR) as age,
    DATETIME_DIFF(ie.intime, adm.admittime, DAY) as preiculos,
    CASE
        WHEN DATETIME_DIFF(adm.admittime, pat.dob, YEAR) <= 1
            THEN 'neonate'
        WHEN DATETIME_DIFF(adm.admittime, pat.dob, YEAR) <= 14
            THEN 'middle'
        WHEN DATETIME_DIFF(adm.admittime, pat.dob, YEAR) > 89
            THEN '>89'
        ELSE 'adult'
    END AS ICUSTAY_AGE_GROUP,
    -- the "hospital_expire_flag" field in the admissions table indicates if a
patient died in-hospital
    CASE
        WHEN adm.hospital_expire_flag = 1 then 'Y'
    ELSE 'N'
    END AS hospital_expire_flag
FROM `physionet-data.mimiciii_clinical.icustays` ie
INNER JOIN `physionet-data.mimiciii_clinical.patients` pat
  ON ie.subject_id = pat.subject_id
INNER JOIN `physionet-data.mimiciii_clinical.admissions` adm
  ON ie.hadm_id = adm.hadm_id;
```

Solution to step 7

```
SELECT ie.subject_id, ie.hadm_id, ie.icustay_id,
    ie.intime, ie.outtime,
    -- patient death in hospital is stored in the admissions table
    adm.deathtime,
    DATETIME_DIFF(adm.admittime, pat.dob, YEAR) as age,
    DATETIME_DIFF(ie.intime, adm.admittime, DAY) as preiculos,
    CASE
        WHEN DATETIME_DIFF(adm.admittime, pat.dob, YEAR) <= 1
            THEN 'neonate'
```

```
        WHEN DATETIME_DIFF(adm.admittime, pat.dob, YEAR) <= 14
            THEN 'middle'
        WHEN DATETIME_DIFF(adm.admittime, pat.dob, YEAR) > 89
            THEN '>89'
        ELSE 'adult'
    END AS ICUSTAY_AGE_GROUP,
    -- the "hospital_expire_flag" field in the admissions table indicates if a
patient died in-hospital
    CASE
        WHEN adm.hospital_expire_flag = 1 then 'Y'
    ELSE 'N'
    END AS hospital_expire_flag,
    -- note also that hospital_expire_flag is equivalent to "Is adm.deathtime
not null?"
    CASE
        WHEN adm.deathtime BETWEEN ie.intime and ie.outtime
            THEN 'Y'
        -- sometimes there are typographical errors in the death date, so
check before intime
        WHEN adm.deathtime <= ie.intime
            THEN 'Y'
        WHEN adm.dischtime <= ie.outtime
            AND adm.discharge_location = 'DEAD/EXPIRED'
            THEN 'Y'
        ELSE 'N'
        END AS ICUSTAY_EXPIRE_FLAG
FROM `physionet-data.mimiciii_clinical.icustays` ie
INNER JOIN `physionet-data.mimiciii_clinical.patients` pat
  ON ie.subject_id = pat.subject_id
INNER JOIN `physionet-data.mimiciii_clinical.admissions` adm
  ON ie.hadm_id = adm.hadm_id;
```