7.0.0 Schema Changes

Last Modified on 07/01/2025 11:31 am ACDT

Schema changes between CareRight V6.99.1 and 7.0.0

New Tables/Views

- service_provider_external_identifiers
- admission_fim_scores

New Columns

- entitlement_usages.archived_at
- transaction_means.description_label
- transaction_means.description_generic_table_code
- transaction_means.bank_label
- transaction_means.bank_generic_table_code
- transaction_means.payment_label
- transaction_means.payment_generic_table_code
- transaction_means.show_bank_date
- transaction_means.seeded
- transaction_means.payment_group
- events.patient_account_id
- saved_reports.last_run
- users.merge_user_token
- admission_categories.hide_snap
- users.merge_user_sent_at
- admission_categories.hide_fim

New Indexes

- users.email - unique
- users.staff_member_id - unique

```
CREATE UNIQUE NONCLUSTERED INDEX idx_user_staff_unique
        ON users(staff_member_id)
        WHERE staff_member_id IS NOT NULL
```

Changed Views

- None

Changed Columns/Indexes

- admission_categories.hide_transports
- admission_categories.hide_non_drg_morbidities

Changed Tables

- ihc_transports renamed to ihc_transport_services

Deleted Tables/Views

- None

Deleted Columns/Indexes

- locations.medisecure_facility_code
- f_prof_categry.medisecure_occupation_code

OTHER

```
UPDATE people a
SET current_information_classification_level = m.current_classification
from people b
join (
      select a.id,
            coalesce(b.highest_classification_level, 0) current_classification#{" "}
      from people a#{" "}
        left join (select b.id, max(classification_level) highest_classification_level#{" "}
                  from information_classifications a#{" "}
                    join people b on b.id = a.classificationable_id and a.classificationable_type = 'Person'
                  where (a.archived is null or a.archived = 0)
                    and a.classified_from < CURRENT_TIMESTAMP
                    and (a.classified_until IS NULL OR a.classified_until > CURRENT_TIMESTAMP)
                  group by b.id
                 ) b on a.id = b.id


    ) m on m.id = b.id
WHERE a.id = b.id
```

```
execute "update transaction_means set seeded=#{Healthsolve::SQLHelper.sql_true},
                                payment_group='Cash'
          where code = 'C'"
      execute "update transaction_means set description_label='BSB/Account Number',
                                bank_label='Cheque Number',
                                payment_label='Drawer',
                                seeded=#{Healthsolve::SQLHelper.sql_true},
                                payment_group='Cheque'
          where code = 'Q'"
      execute "update transaction_means set description_label='Details',
                                bank_label='Name on Card',
                                payment_label='Card Number',
                                description_generic_table_code='p_card',
                                seeded=#{Healthsolve::SQLHelper.sql_true},
                                payment_group='Card'
          where code = 'B'"
      execute "update transaction_means set description_label='Details',
                                seeded=#{Healthsolve::SQLHelper.sql_true},
                                payment_group='Eftpos'
          where code = 'E'"
      execute "update transaction_means set description_label='Details',
                               bank_label='Name on Card',
                               payment_label='Card Number',
                               description_generic_table_code='p_card',
                               seeded=#{Healthsolve::SQLHelper.sql_true},
                               payment_group='Card'
          where code = 'R'"
      execute "update transaction_means set description_label='Details',
                                bank_label='Branch',
                                payment_label='Drawer',
                                description_generic_table_code='p_bank',
                                bank_generic_table_code='p_branch',
                                show_bank_date=#{Healthsolve::SQLHelper.sql_true},
                                seeded=#{Healthsolve::SQLHelper.sql_true},
                                payment_group='Bank'
          where code = 'D'"
      execute "update transaction_means set description_label='Details',
                                bank_label='Branch',
                                payment_label='Drawer',
                                description_generic_table_code='p_bank',
                                bank_generic_table_code='p_branch',
                                seeded=#{Healthsolve::SQLHelper.sql_true},
                                payment_group='Bank'
          where code = 'X'"
      execute "update transaction_means set seeded=#{Healthsolve::SQLHelper.sql_true},
                                payment_group='Bank'
          where code = 'N'"
    end
```

CareRight 7.0.7 Changes

Receipt Numbers

We have swapped from referring to the *f_counters* table and the *ct_rec_no* field to a proper database sequence (*receipt_number*).

- Removed column *ct_counters.ct_rec_no*
- Create sequence receipt_numbers

This does not change the definition of affected columns that referenced this number. However it changed the **behaviour**.

Importantly, where a transaction rolled back; **previously this would discard any attempts to assign a new receipt number.**

From this release, this will **continually increment the receipt numbers even if the transaction is rolled back** This may manifest as "missing" receipt numbers; but is an expected behaviour.

See also: https://learn.microsoft.com/en-us/sql/relational-databases/sequence-numbers/sequence-numbers?view=sql-server-ver16

```
existing_data = CounterSet.where(ct_number: 1).first
 start = 1
 start = CounterSet.next_counter_value("CT_REC_NO") if existing_data
 # historically went into:
 #   f_statement.stat_ref
 #   refunds.number
 #   receipt.number
 check = (Statement.maximum(:stat_ref) || 0) + 1
 start = check if check > start
 check = (Receipt.maximum(:number) || 0) + 1
 start = check if check > start
 check = (Refund.maximum(:number) || 0) + 1
 start = check if check > start
 case ActiveRecord::Base.connection.adapter_name.downcase
 when "sqlserver"
   execute <<~END_OF_SQL
     CREATE SEQUENCE receipt_number START WITH #{start} INCREMENT BY 1
   END_OF_SQL
 when "postgresql"
   execute <<~END_OF_SQL
     CREATE SEQUENCE receipt_number START #{start}
   END_OF_SQL
 else
   raise "Unhandled database type"
 end
 remove_column :f_counters, :ct_rec_no
end
```

While this change is small and should be routine, we are recommending customers have:

- Appropriate database backups in place
- A regression test plan to validate receipt creation post deployment functions as expected.

CareRight 7.0.8 Changes

Transaction Numbers

Similar to the Receipt number changes.

```
existing_data = CounterSet.where(ct_number: 1).first
start = 1
start = CounterSet.next_counter_value("CT_TRANS_NO") if existing_data

check = (Transaction.maximum(:trans_no) || 0) + 1
start = check if check > start


case ActiveRecord::Base.connection.adapter_name.downcase
when "sqlserver"
  execute <<~END_OF_SQL
    CREATE SEQUENCE transaction_number START WITH #{start} INCREMENT BY 1
  END_OF_SQL


when "postgresql"
  execute <<~END_OF_SQL
    CREATE SEQUENCE transaction_number START #{start}
  END_OF_SQL

else
  raise "Unhandled database type"
end
remove_column :f_counters, :ct_trans_no
CounterSet.reset_column_information
```

Referral Numbers

Similar to the Receipt number changes.

```
existing_data = CounterSet.where(ct_number: 1).first
start = 1
start = CounterSet.next_counter_value("CT_F_REF_NUMBER") if existing_data


check = (Referral.maximum(:f_ref_number) || 0) + 1
start = check if check > start


case ActiveRecord::Base.connection.adapter_name.downcase
when "sqlserver"
  execute <<~END_OF_SQL
    CREATE SEQUENCE referral_number START WITH #{start} INCREMENT BY 1
  END_OF_SQL


when "postgresql"
  execute <<~END_OF_SQL
    CREATE SEQUENCE referral_number START #{start}
  END_OF_SQL


else
  raise "Unhandled database type"
end
remove_column :f_counters, :ct_f_ref_number
CounterSet.reset_column_information
```

Patient Account Numbers

Similar to the Receipt number changes.

```
existing_data = CounterSet.where(ct_number: 1).first
start = 1
start = CounterSet.next_counter_value("CT_P_ACC_REF") if existing_data


check = (PatientAccount.maximum(:p_acc_ref) || 1) + 1
start = check if check > start


case ActiveRecord::Base.connection.adapter_name.downcase
when "sqlserver"
  execute <<~END_OF_SQL
    CREATE SEQUENCE patient_account_number START WITH #{start} INCREMENT BY 1
  END_OF_SQL


when "postgresql"
  execute <<~END_OF_SQL
    CREATE SEQUENCE patient_account_number START #{start}
  END_OF_SQL


else
  raise "Unhandled database type"
end
remove_column :f_counters, :ct_p_acc_ref
CounterSet.reset_column_information
```

Invoice Numbers

Similar to the Receipt number changes.

```
existing_data = CounterSet.where(ct_number: 1).first
start = 1
start = CounterSet.next_counter_value("CT_INV_NO") if existing_data
check = (Invoice.maximum(:inv_no) || 0) + 1
start = check if check > start


case ActiveRecord::Base.connection.adapter_name.downcase
when "sqlserver"
  execute <<~END_OF_SQL
    CREATE SEQUENCE invoice_number START WITH #{start} INCREMENT BY 1
  END_OF_SQL


when "postgresql"
  execute <<~END_OF_SQL
    CREATE SEQUENCE invoice_number START #{start}
  END_OF_SQL


else
  raise "Unhandled database type"
end
remove_column :f_counters, :ct_inv_no
CounterSet.reset_column_information
```

Line Item Numbers

Similar to the Receipt number changes.

```
existing_data = CounterSet.where(ct_number: 1).first
start = 1
start = CounterSet.next_counter_value("CT_LINE_NO") if existing_data
check = (LineItem.maximum(:line_no) || 0) + 1
start = check if check > start


case ActiveRecord::Base.connection.adapter_name.downcase
when "sqlserver"
  execute <<~END_OF_SQL
    CREATE SEQUENCE li_number START WITH #{start} INCREMENT BY 1
  END_OF_SQL


when "postgresql"
  execute <<~END_OF_SQL
    CREATE SEQUENCE li_number START #{start}
  END_OF_SQL


else
  raise "Unhandled database type"
end
remove_column :f_counters, :ct_line_no
CounterSet.reset_column_information
```

Estimate Numbers

Similar to the Receipt number changes.

```
existing_data = CounterSet.where(ct_number: 1).first
start = 1
start = CounterSet.next_counter_value("CT_EST_NO") if existing_data
check = (Estimate.maximum(:est_no) || 0) + 1
start = check if check > start


case ActiveRecord::Base.connection.adapter_name.downcase
when "sqlserver"
  execute <<~END_OF_SQL
    CREATE SEQUENCE estimate_number START WITH #{start} INCREMENT BY 1
  END_OF_SQL


when "postgresql"
  execute <<~END_OF_SQL
    CREATE SEQUENCE estimate_number START #{start}
  END_OF_SQL


else
  raise "Unhandled database type"
end
remove_column :f_counters, :ct_est_no
CounterSet.reset_column_information
```

Estimate Line Item Numbers

Similar to the Receipt number changes.

```
existing_data = CounterSet.where(ct_number: 1).first
start = 1
start = CounterSet.next_counter_value("CT_HOLD_NO") if existing_data
check = (EstimateItem.maximum(:hold_no) || 0) + 1
start = check if check > start


case ActiveRecord::Base.connection.adapter_name.downcase
when "sqlserver"
  execute <<~END_OF_SQL
    CREATE SEQUENCE estimate_line_item_number START WITH #{start} INCREMENT BY 1
  END_OF_SQL


when "postgresql"
  execute <<~END_OF_SQL
    CREATE SEQUENCE estimate_line_item_number START #{start}
  END_OF_SQL


else
  raise "Unhandled database type"
end
remove_column :f_counters, :ct_hold_no
CounterSet.reset_column_information
```

Professional Contact Numbers

Similar to the Receipt number changes.

```
existing_data = CounterSet.where(ct_number: 1).first
start = 1
start = CounterSet.next_counter_value("CT_PR_NUMB") if existing_data


check = (ProfessionalContact.maximum(:pr_numb) || 0) + 1
start = check if check > start


case ActiveRecord::Base.connection.adapter_name.downcase
when "sqlserver"
  execute <<~END_OF_SQL
    CREATE SEQUENCE professional_contact_number START WITH #{start} INCREMENT BY 1
  END_OF_SQL


when "postgresql"
  execute <<~END_OF_SQL
    CREATE SEQUENCE professional_contact_number START #{start}
  END_OF_SQL


else
  raise "Unhandled database type"
end
remove_column :f_counters, :ct_pr_numb
CounterSet.reset_column_information
```

Professional Practice Numbers

Similar to the Receipt number changes.

```
existing_data = CounterSet.where(ct_number: 1).first
start = 1
start = CounterSet.next_counter_value("CT_PC_NUMB") if existing_data


check = (ProfessionalPractice.maximum(:pc_numb) || 0) + 1
start = check if check > start


case ActiveRecord::Base.connection.adapter_name.downcase
when "sqlserver"
  execute <<~END_OF_SQL
    CREATE SEQUENCE professional_practice_number START WITH #{start} INCREMENT BY 1
  END_OF_SQL


when "postgresql"
  execute <<~END_OF_SQL
    CREATE SEQUENCE professional_practice_number START #{start}
  END_OF_SQL


else
  raise "Unhandled database type"
end
remove_column :f_counters, :ct_pc_numb
CounterSet.reset_column_information
```

Statement Numbers

Similar to the Receipt number changes.

```
existing_data = CounterSet.where(ct_number: 1).first
start = 1
start = CounterSet.next_counter_value("CT_STAT_NO") if existing_data
check = (Statement.maximum(:stat_no) || 0) + 1
start = check if check > start


case ActiveRecord::Base.connection.adapter_name.downcase
when "sqlserver"
  execute <<~END_OF_SQL
    CREATE SEQUENCE statement_number START WITH #{start} INCREMENT BY 1
  END_OF_SQL


when "postgresql"
  execute <<~END_OF_SQL
    CREATE SEQUENCE statement_number START #{start}
  END_OF_SQL


else
  raise "Unhandled database type"
end
remove_column :f_counters, :ct_stat_no
CounterSet.reset_column_information
```