NursingCareWare: Warehousing and Knowledge Discovery for a Nursing Care Data Set

Ray Hylock
Management Sciences Department
University of Iowa
Iowa City, IA 52242
ray-hylock@uiowa.edu

Summer Fellowship Research Progress Report

In my research proposal, I wrote that I would complete the following: 1) finish loading the data into the remaining dimension tables, 2) continue readings in cube materialization, indexing strategies, and Oracle’s data mining query language from which I will provide a literature review, 3) select the proper indexing and cube materialization strategies and begin implementing them, 4) time permitting, design and test a few data mining queries. I will address each individually below.

Point 1:

I have completed loading the data and have even adjusted the scripts to “compress” the data in the bridge tables (all code is listed at the end of this document). Data in the diseases table can be grouped together more easily due to it only having two variables: icd_id and ordering. The other three tables (nursing diagnoses, outcomes, and interventions) have an added attribute, date, which makes it far more difficult to find identical records to group. Groupings are as follows per bridge group table:

- Nursing diagnoses: 65,737/66,865 = 1.69% fewer records
- Outcomes: 128,308/128,361 = 0.041% few records
- Interventions: 117,440/117,472 = 0.027% fewer records
- Diseases: 104,554/174,492 = 40.08% fewer records

There are several potential techniques that could further the first three tables’ groupings. The first is to remove attribute(s) from the bridge and create a mini-dimension. I have already calculated the number of records that would need to be in the mini-dimensions and it is not worth the cost of joining the tables together. To create these mini-dimensions, you can either populate the tables with the exact data, that is input only data that is used (time consuming during ETL process), or you can pre-populate the table with all possible combinations. The equation for the last is as follows:

\[
\text{# of permutations} = \sum_{i=1}^{r} n^i
\]

\[
\text{# of unique rows} = \sum_{i=1}^{r} i * n^i
\]

Where \( r \) = the number of positions to be filled and \( n \) = the cardinality of the attribute to be iterated. For example, using the ordering attribute, there are at most\(^1\) \( r = 11 \) entries for a single SEQ (4 for diseases and 11 for nursing diagnoses, outcomes, and interventions) and at most \( n = 74 \) possible ordering values

---

\(^1\) By at most I mean there exist at most 11 positions and 74 ordering values in the current data set that we have. This number could increase or decrease as the data set is updated.
(4 for diseases, 74 for nursing diagnoses and interventions, and 67 for outcomes). This gives us 3.69E20 permutations and 4.06E21 unique records. These tables are definitely too big to build and therefore are not an option. If, however, the cardinality of ordering was reduced to 2 (primary and non-primary), then the total number of pre-computed records will be 40,962. The impact of this could be significant, however, the data will be generalized and thus limit some of the analysis possible.

Points 2 and 3:

I have continued to read into indexing and materialization of cuboids in data warehouses. For indexing, it depends upon the data in the tables. The best approach seems to be compressed bitmap format for our data structure with b-trees a solid second. However, Oracle does not have this option for materialized cubes which is what we will be using in the ROLAP setup. It does have a sparse option for data cubes which will translate into compressed bitmaps, but we are not using the MOLAP configuration for the following reasons:

- Data cardinality is high (translates into huge storage costs in MOLAP)
- Sparse data sets (also translates into huge storage costs in MOLAP)
- ad-hoc query support
- We will have a large number of dimensions (large being > 10)

Therefore, I will need to test bitmap and b-tree indexing as it exists in Oracle or build an indexing algorithm in PL/SQL which is also supported.

One way to cut back on the indexing issues is partitioning. Several different options are supported by Oracle, but there is no real logical way for me to break up the data. That is, I do not have sufficient knowledge of the potential queries to determine the most appropriate way to divide up the tables. The literature recommends breaking the table up into \( n \) equal sized pieces in this case. This simply allows the creation of index indexing for faster access (in theory).

Like indexing, materialized view selection is quite difficult without knowledge of the queries that will be asked or a history of queries to refer to. Traditionally, the way of figuring out the total number of cuboids for a given cube is as follows:

\[
\text{Without hierarchies: } 2^n \\
\text{With hierarchies: } \prod_{i=1}^{n} (L_i + 1)
\]

Where \( n \) = the number of dimensions and \( L_i \) = the number of levels in dimension \( i \). If we input the information for the current design, the second equation yields 138,240 cuboids. However, this equation is not appropriate when bridges are involved because a single table in the group is meaningless without the others. Therefore, I propose the following adjustments to the above equations to take this into consideration. Each set of group dimension, group, and dimension tables, should be counted as a single “dimension” when computing the number of possible cuboids. The augmented formula is:

\[
\text{Without hierarchies: } 2^b \\
\text{With hierarchies: } \prod_{i=1}^{b} \left( \prod_{j=1}^{b_i} (L_j) + 1 \right)
\]
Where \( b = \) the number of non-bridge dimensions + the number of bridge groups,  
\[
b_i = \begin{cases} 
1 & \text{# of tables in bridge group } i, \text{ if bridge group} \\
0 & \text{otherwise} 
\end{cases}, 
\]
\( L_j = \) the number of levels in bridge group \( i \)'s \( j \)th table, and \((\prod_{j=1}^{b_i} L_j)_i = \) either the total number of level combinations in bridge group \( i \) or the number of levels in dimension \( i \). Using the new equation the total number of cuboids is 540; much more manageable. However, there are still far too many possible views to build.

When it comes to cube materialization, there are two subproblems at hand. One, data cube computation and two, data cube selection. The first deals simply with aggregation of cuboids such that the number of memory/HD scans and space is minimal. That is, this problem deals with the order in which to materialize the cubes so cubes of the same type are closest to one another. The second one deals with which cuboids to actually materialize.

There are two main ways to traverse the cube lattice: top-up which starts with the most fine-grained cube and bottom-down which starts at the apex. The early cube materialization strategies all used top-down, but current materialization strategies employ bottom-up because the Apriori pruning method can be used to reduce the number of cuboids. The benefit to using top-down is that finer grained cubes are computed which can be used to compute more generalized ones. For example, cuboid AB can be computed by aggregating cuboid AB. This, however generally lead to an increase in storage space because we are storing larger cuboids. The benefit to bottom-up is the ability to prune cuboids that don’t meet a minimum support threshold (this is stated in the HAVING clause). For example, if cuboid AB doesn’t meet the minimum support, than nothing else with AB in it will, so they are ignored. The downside is, the benefit of top-down (partial order materialization i.e. AB -> A) doesn’t exist. This is, however, the choice of most new algorithms because the space savings is greater than the time savings.

Most of these techniques use greedy algorithms (space/time dependent) to pick which cuboids to materialize and in what order it will be done. There are several algorithms that use data mining to actually determine which cuboids to materialize such as clustering and density based. These, however, require substantial knowledge about space, time, and query frequencies. Also, there are hybrid algorithms that combine both top-down and bottom-up to try and use both partial order materialization and Apriori pruning.

I have decided to materialize these cuboids for demonstration purposes (I do not have any HAVING’s so I employ a top-down strategy):

- **DISEASES_GROUP_DIM**, **DISEASES_GROUP**, and **DISEASES** for the icd level (DG_I): No aggregation

```sql
DISEASES_GROUP_DIM NUMBER(10)
CCS_CCS_NAME VARCHAR2(150)
CCS_CCS_CODE VARCHAR2(5)
ICD_CCS_NAME VARCHAR2(150)
ICD_CCS_CODE VARCHAR2(5)
ICD VARCHAR2(10)
ORDERING NUMBER(10)
MV_DG_I_DIS_GROUP_FK:
FOREIGN KEY (DISEASES_GROUP_DIM) REFERENCES DISEASES_GROUP_DIM (DISEASES_GROUP_ID)

SELECT DGD.diseases_group_id, D.ccs_ccs_name, D.ccs_ccs_code, D.icd_ccs_name, D.icd_ccs_code, D.icd, DG.ordering
FROM diseases D JOIN diseases_group DG ON D.dimension_key = DG.icd_id JOIN
```
diseases_group_dim DGD ON DG.dimension_key = DGD.diseases_group_id
GROUP BY DG.diseases_group_id, D.ccs_ccs_name, D.ccs_ccs_code,
D.icd_ccs_name, D.icd_ccs_code, D.icd, DG.ordering;

- **NURSING_DIAGNOSES_GROUP_DIM, NURSING_DIAGNOSES_GROUP, and NURSING_DIAGNOSES (NDG): No aggregation**

  NURSING_DIAGNOSES_GROUP_ID NUMBER(10)
  NDCODE    NUMBER(3)
  ORDERING   NUMBER(10)
  NDDATE_ID   NUMBER(10)

  MV_NDG_ND_GROUP_FK:
  FOREIGN KEY (NURSING_DIAGNOSES_GROUP_ID) REFERENCES NURSING_DIAGNOSES_GROUP_DIM
  (NURSING_DIAGNOSES_GROUP_ID)

  SELECT NDGD.nursing_diagnoses_group_id, ND.ndcode, NDG.ordering, NDG.nddate_id
  FROM nursing_diagnoses ND JOIN nursing_diagnoses_group NDG ON ND.ndcode_id =
  NDG.ndcode_id JOIN nursing_diagnoses_group_dim NDGD ON NDG.dimension_key =
  NDGD.nursing_diagnoses_group_id
  GROUP BY NDG.nddate_id, NDGD.nursing_diagnoses_group_id, ND.ndcode,
  NDG.ordering;

- **INTERVENTIONS_GROUP_DIM, INTERVENTIONS_GROUP, and INTERVENTIONS for the interventions level (IG_I): No aggregation**

  INTERVENTIONS_GROUP_ID NUMBER(10)
  NURSING_DIAGNOSES_NICODE NUMBER(3)
  INTERVENTIONS_NICODE NUMBER(3)
  NIDEFINING_CODE  NUMBER(2)
  NICODE_GROUP   CHAR(3)
  NIDATE_ID   NUMBER(10)
  ORDERING   NUMBER(10)

  MV_IG_I_INT_GROUP_FK:
  FOREIGN KEY (INTERVENTIONS_GROUP_ID) REFERENCES INTERVENTIONS_GROUP_DIM
  (INTERVENTIONS_GROUP_ID)

  SELECT IGD.interventions_group_id, I.nursing_diagnoses_nicode,
  I.intervention_nicode, I. nidefining_code, I.nicode_group, IG.nidate_id,
  IG.ordering
  FROM interventions I JOIN interventions_group IG ON I.intervention_ni_id =
  IG.ni_id JOIN interventions_group_dim IGD ON IG.dimension_key =
  IGD.interventions_group_id
  GROUP BY IGD.interventions_group_id, I.nursing_diagnoses_nicode,
  I.intervention_nicode, I. nidefining_code, I.nicode_group,
  IG.nidate_id, IG.ordering;

- **OUTCOMES_GROUP_DIM, OUTCOMES_GROUP, and OUTCOMES for the outcomes level (OG_O): No aggregation**

  OUTCOMES_GROUP_ID  NUMBER(10)
  NURSING_DIAGNOSES_NOCODE NUMBER(3)
  NODEFINING_CODE  NUMBER(2)
  OUTCOMES_NOCODE  NUMBER(3)
  NODATE_ID   NUMBER(10)
  ORDERING   NUMBER(10)

  MV_OG_O_OUT_GROUP_FK:
  FOREIGN KEY (OUTCOMES_GROUP_ID) REFERENCES OUTCOMES_GROUP_DIM
  (OUTCOMES_GROUP_ID)
SELECT OGD.outcomes_group_id, O.nursing_diagnoses_nocode, O.nodefining_code, 
O.outcomes_no_id, OG.nodate_id, OG.ordering 
FROM outcomes O JOIN outcomes_group OG ON O.outcomes_no_id = OG.no_id JOIN 
outcomes_group_dim OGD ON OG.dimension_key = OGD.outcomes_group_id 
GROUP BY OGD.outcomes_group_id, O.nursing_diagnoses_nocode, O.nodefining_code, 
O.outcomes_no_id, OG.nodate_id, OG.ordering;

• VISITS and DG_I (VDG_I_AGE): avg(age), count(*)

    DISEASE_GROUP_ID NUMBER(10)
    CCS_CCS_NAME VARCHAR2(150)
    CCS_CCS_CODE VARCHAR2(5)
    ICD_CCS_NAME VARCHAR2(150)
    ICD_CCS_CODE VARCHAR2(5)
    ICD VARCHAR2(10)
    ORDERING NUMBER(10)
    AVG_AGE NUMBER(5,2)
    CNT NUMBER(10)

MV_VDG_I_AGE__DIS_GROUP_FK:
FOREIGN KEY (DISEASES_GROUP_ID) REFERENCES DISEASES_GROUP_DIM (DISEASES_GROUP_ID)

SELECT M.diseases_group_id, M.ccs_ccs_name, M.ccs_ccs_code, M.icd_ccs_name,
M.icd_ccs_code, M.icd, M.ordering, AVG(V.age) as avg_age, COUNT(*) as cnt
FROM visits V JOIN mv_dg_i M ON V.diseases_group_id = M.diseases_group_id
GROUP BY M.diseases_group_id, M.ccs_ccs_name, M.ccs_ccs_code, M.icd_ccs_name,
M.icd_ccs_code, M.icd, M.ordering;

• VISITS and NDG (VNDG_AGE): avg(age), count(*)

    NURSING_DIAGNOSES_GROUP_ID NUMBER(10)
    ND_CODE    NUMBER(3)
    ORDERING   NUMBER(10)
    NDDATE_ID   NUMBER(10)
    AVG_AGE   NUMBER(5,2)
    CNT    NUMBER(10)

MV_VNDG_AGE__ND_GROUP_FK:
FOREIGN KEY (NURSING_DIAGNOSES_GROUP_ID) REFERENCES NURSING_DIAGNOSES_DIM (NURSING_DIAGNOSES_GROUP_ID)

SELECT M.ndcode, M.ordering, M.nddate_id, AVG(V.age) as avg_age, COUNT(*) as cnt
FROM visits V JOIN mv_ndg M ON V.nursing_diagnoses_group_id =
M.nursing_diagnoses_group_id
GROUP BY M.ndcode, M.ordering, M.nddate_id;

• VISITS and IG_I (VIG_I_AGE): avg(age), count(*)

    INTERVENTIONS_GROUP_ID NUMBER(10)
    NURSING_DIAGNOSES_NICODE NUMBER(3)
    INTERVENTIONS_NICODE NUMBER(3)
    NIDEFINING_CODE NUMBER(2)
    NICODE_GROUP CHAR(3)
    NIDATE_ID NUMBER(10)
    ORDERING NUMBER(10)
    AVG_AGE NUMBER(5,2)
    CNT NUMBER(10)

MV_VIG_I_AGE__INT_GROUP_FK:
FOREIGN KEY (INTERVENTIONS_GROUP_ID) REFERENCES INTERVENTIONS_GROUP_DIM
(INTERVENTIONS_GROUP_ID)

SELECT M.interventions_group_id, M.nursing_diagnoses_nicode,
M.intervention_nicode, M.nidefining_code, M.nicode_group, M.nidate_id,
M.ordering, AVG(V.age) as avg_age, COUNT(*) as cnt
FROM visits V JOIN mv_ig_i M ON V.interventions_group_id =
M.interventions_group_id
GROUP BY M.interventions_group_id, M.nursing_diagnoses_nicode,
M.intervention_nicode, M.nidefining_code, M.nicode_group, M.nidate_id,
M.ordering;

• VISITS and OG_O (VOG_O_AGE): avg(age), count(*)

OUTCOMES_GROUP_ID NUMBER(10)
NURSING_DIAGNOSES_NOCODE NUMBER(3)
NODEFINING_CODE NUMBER(2)
OUTCOMES_NOCODE NUMBER(3)
NODEDATE_ID NUMBER(10)
ORDERING NUMBER(10)
AVG_AGE NUMBER(5,2)
CNT NUMBER(10)

MV_VOG_O_AGE__OUT_GROUP_FK:
FOREIGN KEY (OUTCOMES_GROUP_ID) REFERENCES OUTCOMES_GROUP_DIM
(OUTCOMES_GROUP_ID)

SELECT M.outcomes_group_id, M.nursing_diagnoses_nocode, M.nodefining_code,
M.outcomes_nocode, M.nodate_id, M.ordering, AVG(V.age) as avg_age,
COUNT(*) as cnt
FROM visits V JOIN mv_og_o M ON V.outcomes_group_id = M.outcomes_group_id
GROUP BY M.outcomes_group_id, M.nursing_diagnoses_nocode, M.nodefining_code,
M.outcomes_nocode, M.nodate_id, M.ordering;

• VISITS, OG_O, and NDG (V_O_ND_AGE)

OUTCOMES_GROUP_ID NUMBER(10)
NODEFINING_CODE NUMBER(2)
OUTCOMES_NOCODE NUMBER(3)
NURSING_DIAGNOSES_GROUP_ID NUMBER(10)
NDCODE NUMBER(3)
AGE NUMBER(5,2)

MV_V_O_ND__OUT_GROUP_FK:
FOREIGN KEY (OUTCOMES_GROUP_ID) REFERENCES OUTCOMES_GROUP_DIM
(OUTCOMES_GROUP_ID)

MV_V_O_ND__ND_GROUP_FK:
FOREIGN KEY (NURSING_DIAGNOSES_GROUP_ID) REFERENCES NURSING_DIAGNOSES_GROUP_DIM
(NURSING_DIAGNOSES_GROUP_ID)

SELECT MO.outcomes_group_id, MO.nodefining_code, MO.outcomes_nocode,
MD.nursing_diagnoses_group_id, MD.ndcode, V.age
FROM visits V JOIN mv_og_o MO ON V.outcomes_group_id = MO.outcomes_group_id
JOIN mv_ndg MD ON V.nursing_diagnoses_group_id = MD.nursing_diagnoses_group_id
GROUP BY MO.outcomes_group_id, MO.nodefining_code, MO.outcomes_nocode,
MD.nursing_diagnoses_group_id, MD.ndcode, V.age;
• VISITS, OG_O, and NDG (V_O_ND_AGE): \text{avg(age)}, \text{count(*)}

\begin{verbatim}
SELECT MO.outcomes_group_id, MO.nursing_diagnoses_nocode, MO.nodefining_code,
       MO.outcomes_nocode, MO.nodate_id, MO.ordering,
       MD.nursing_diagnoses_group_id, MD.ndcode, MD.ordering, MD.nddate_id,
       AVG(V.age) as avg_age, COUNT(*) as cnt
FROM mv_og_o MO JOIN visits V ON V.outcomes_group_id = MO.outcomes_group_id
JOIN mv_ndg MD ON V.nursing_diagnoses_group_id = MD.nursing_diagnoses_group_id
GROUP BY MO.outcomes_group_id, MO.nursing_diagnoses_nocode, MO.nodefining_code,
         MO.outcomes_nocode, MO.nodate_id, MO.ordering,
         MD.nursing_diagnoses_group_id, MD.ndcode, MD.ordering, MD.nddate_id;
\end{verbatim}

• VISITS, IG_I, and NDG (V_I_ND_AGE)

\begin{verbatim}
SELECT MI.interventions_group_id, MI.intervention_nicode, MI.nidefining_code,
       MI.nicode_group, MND.nursing_diagnoses_group_id, MND.ndcode, V.age
FROM mv_ig_i MI JOIN visits V ON MI.interventions_group_id = V.interventions_group_id
JOIN mv_ndg MND ON V.nursing_diagnoses_group_id = MND.nursing_diagnoses_group_id
GROUP BY MI.interventions_group_id, MI.intervention_nicode, MI.nidefining_code,
         MI.nicode_group, MND.nursing_diagnoses_group_id, MND.ndcode, V.age;
\end{verbatim}
VISITS, IG_I, and NDG (V_I_ND_AGE): avg(age), count(*)

INTERVENTIONS_GROUP_ID NUMBER(10)
NURSING_DIAGNOSES_NICODE NUMBER(3)
INTERVENTIONS_NICODE NUMBER(3)
NIDEFINING_CODE NUMBER(2)
NICODE_GROUP CHAR(3)
NIDATE_ID NUMBER(10)
I_ORDERING NUMBER(10)
NURSING_DIAGNOSES_GROUP_ID NUMBER(10)
ND_CODE NUMBER(3)
ND_ORDERING NUMBER(10)
ND_DATE_ID NUMBER(10)
AVG_AGE NUMBER(5,2)
CNT NUMBER(10)

MV_V_I_ND_AGE__INT_GROUP_FK:
FOREIGN KEY (INTERVENTIONS_GROUP_ID) REFERENCES INTERVENTIONS_GROUP_DIM (INTERVENTIONS_GROUP_ID)

MV_V_I_ND_AGE__ND_GROUP_FK:
FOREIGN KEY (NURSING_DIAGNOSES_GROUP_ID) REFERENCES NURSING_DIAGNOSES_GROUP_DIM (NURSING_DIAGNOSES_GROUP_ID)

SELECT MI.interventions_group_id, MI.nursing_diagnoses_nicode,
       MI.intervention_nicode, MI.nidefining_code, MI.nicode_group,
       MI.nidate_id, MI.ordering, MND.nursing_diagnoses_group_id, MND.ndcode,
       MND.ordering, MND.nddate_id, AVG(V.age) as avg_age, COUNT(*) as cnt
FROM mv_ig_i MI JOIN visits V ON MI.interventions_group_id = V.interventions_group_id
                                             JOIN mv_ndg MND ON V.nursing_diagnoses_group_id = MND.nursing_diagnoses_group_id
GROUP BY MI.interventions_group_id, MI.nursing_diagnoses_nicode,
       MI.intervention_nicode, MI.nidefining_code, MI.nicode_group,
       MI.nidate_id, MI.ordering, MND.nursing_diagnoses_group_id, MND.ndcode,
       MND.ordering, MND.nddate_id;

Point 4:

I have read about data mining and have run some “tests” to see how it works, but this part should really be left up to Lian and Nick and their more advanced algorithms. As of now, I do not know for what purpose Der-Fa or anyone else would use the data miner since Nick and Lian are working on the web interface. However, if Der-Fa wants to use the data miner for her research questions, then we can sit down and figure out a plan. Basically, I can use it, but do not have an in depth understanding of the software (just like everything Oracle, it is extremely convoluted, ambiguous, and the documentation is terrible if you do not already know how to use the software).
Oracle DataMiner Example

Oracle DataMiner has the following built in functions:

- Unsupervised: Association, Clustering, and Feature Extraction
- Supervised: Attribute Importance, Classification, and Regression

Example using Association: (took approximately 13 minutes to run on my laptop)

- Function Type: Association Rules
- Algorithm: Apriori (only option for association rules)
- Table/View: MV_V_I_ND
- Item ID: NIDEFINING_CODE (the attribute used for association)
- Transaction Identifier: INTERVENTIONS_GROUP_ID (based on SEQ, see ETL scripts)
- Stored as: DEMO__MV_V_I_ND
- Advanced settings:
  - Sampling: Disabled
  - Minimum support: 5% (default)
  - Minimum confidence: 10% (default)
  - Limit number of attributes in each rule: 10 (default 3)

The results: Minimum support = 5% (can be changed after the fact), minimum confidence = 10% (can be changed after the fact as well), and selected number of results to view is 10.

The selected rule states: If NIDEFINING_CODE.5 = 1 AND NIDEFINING_CODE.9 = 1 AND NIDEFINING_CODE.13 = 1, THEN NIDEFINING_CODE.4 = 1, Confidence (%) = 98.2, Support (%) = 6.53.

Oracle Data Miner uses the notation “= 1” to mean “is present in the basket.” The example therefore means, a basket that contains one or more NIDEFINING_CODE = 5, one or more NIDEFINING_CODE = 9, and one or more NIDEFINING_CODE = 13, is likely to contain one or more NIDEFINING_CODE = 4.
OracleBI Discoverer Example:

OracleBI Discoverer Administrator is used to set up OracleBI Discoverer Desktop. Once the tables are deployed in the warehouse, you connect to them. Next, you create the hierarchies. These are NOT automatically transferred over (except for the date hierarchies). Therefore, you have to manually create the hierarchies and not just for the attributes, but on EACH table including materialized views.

OracleBI Discoverer Desktop is an ad-hoc query tool that I’ve been using to view example cubes. Below is an example of the following:

- **Tables:** INTERVENTIONS_GROUP_DIM, INTERVENTIONS_GROUP, and INTERVENTIONS (used materialized view MV_JG_I), NURSING_DIAGNOSES_GROUP_DIM, NURSING_DIAGNOSES_GROUP, and NURSING_DIAGNOSES (used materialized view MV_NDG), and VISITS. This is stored as materialized view MV_V_I_ND.

- **X-Axis:** Nursing diagnoses code (ndcode)
- **Y-Axis (hierarchical):** interventions defining code (nidefining_code) -> interventions code group (nicode_group) -> interventions code (interventions_nicode)
• Measure: Variance (one of the many built in functions)
• Constraints:
  o 65 <= age <= 74
  o COUNT(age) > 1
  o Variance >= 20
  o NICODE_GROUP = “MBZ”

• The graph type is “Surface” format “3D” with Y-Axis [min:max] = [20:40]
  o Right-click on graph to drill (they call it drill up/down instead of roll-up/drill-down)

The arrows on the left-hand side on the Y-Axis of the cube are used for drilling based on predefined hierarchies, attributes in other tables, or attributes in that table that are not present in the cube.
ETL Scripts:

I have thoroughly tested these scripts and packaged them in such a way that writing a few lines of code will allow the user to access all functionality.

Some results are as follows:

To load an empty data warehouse takes approximately 11 minutes (using my laptop with both the database and data warehouse in the same Oracle instance)

To reload an existing data warehouse takes approximately 25 minutes (same constraints as above)

GRANTS

CREATE OR REPLACE PROCEDURE grants AS
BEGIN
EXECUTE IMMEDIATE 'grant all on system.patients to rep_owner';
EXECUTE IMMEDIATE 'grant all on system.ccs to rep_owner';
EXECUTE IMMEDIATE 'grant all on system.ccs_icd to rep_owner';
EXECUTE IMMEDIATE 'grant all on system.diseases to rep_owner';
EXECUTE IMMEDIATE 'grant all on system.interventions to rep_owner';
EXECUTE IMMEDIATE 'grant all on system.outcomes to rep_owner';
EXECUTE IMMEDIATE 'grant all on system.nursing_diagnoses to rep_owner';
EXECUTE IMMEDIATE 'grant all on system.visits to rep_owner';
commit;
END grants;
/

DROP SCRIPTS (sequences and temp table)

CREATE OR REPLACE PROCEDURE drops AS
BEGIN
EXECUTE IMMEDIATE 'DROP SEQUENCE DISEASES_GROUP_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE DISEASES_GROUP_DIM_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE DISEASES_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE INTERVENTIONS_GROUP_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE INTERVENTIONS_GROUP_DIM_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE INTERVENTIONS_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE NURSING_DIAGNOSES_GROUP_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE NURSING_DIAGNOSES_GROUP_D_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE NURSING_DIAGNOSES_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE OUTCOMES_GROUP_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE OUTCOMES_GROUP_DIM_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE OUTCOMES_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE PATIENTS_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE VISITS_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE DISEASES_GROUPINGS_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE NURSING_DIAGNOSES_GS_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE INTERVENTIONS_GROUPINGS_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE OUTCOMES_GROUPINGS_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE DISEASES_GROUPING_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE NURSING_DIAGNOSES_GROUPING_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE INTERVENTIONS_GROUPING_SEQ';
EXECUTE IMMEDIATE 'DROP SEQUENCE OUTCOMES_GROUPING_SEQ';
EXECUTE IMMEDIATE 'DROP TABLE groupings';
commit;
END drops;
/
CREATION SCRIPTS (sequences and temp table)

CREATE OR REPLACE PROCEDURE creates AS BEGIN
EXECUTE IMMEDIATE 'CREATE SEQUENCE DISEASES_GROUP_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE DISEASES_GROUP_DIM_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE DISEASES_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE INTERVENTIONS_GROUP_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE INTERVENTIONS_GROUP_DIM_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE INTERVENTIONS_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE NURSING_DIAGNOSES_GROUP_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE NURSING_DIAGNOSES_GROUP_D_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE NURSING_DIAGNOSES_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE OUTCOMES_GROUP_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE OUTCOMES_GROUP_DIM_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE OUTCOMES_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE PATIENTS_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE VISITS_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE DISEASES_GROUPINGS_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE NURSING_DIAGNOSES_GS_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE INTERVENTIONS_GROUPINGS_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE OUTCOMES_GROUPINGS_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE DISEASES_GROUPING_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE NURSING_DIAGNOSES_GROUPING_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE INTERVENTIONS_GROUPING_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';
EXECUTE IMMEDIATE 'CREATE SEQUENCE OUTCOMES_GROUPING_SEQ MINVALUE 1 MAXVALUE 999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER NOCYCLE ';

EXECUTE IMMEDIATE 'CREATE TABLE groupings ( key NUMBER, seq NUMBER, code1 NUMBER, order1 NUMBER, date1 NUMBER, code2 NUMBER, order2 NUMBER, date2 NUMBER, code3 NUMBER, order3 NUMBER, date3 NUMBER, code4 NUMBER, order4 NUMBER, date4 NUMBER, code5 NUMBER, order5 NUMBER, date5 NUMBER, code6 NUMBER, order6 NUMBER, date6 NUMBER, code7 NUMBER, order7 NUMBER, date7 NUMBER, code8 NUMBER, order8 NUMBER, date8 NUMBER, code9 NUMBER, order9 NUMBER, date9 NUMBER, code10 NUMBER, order10 NUMBER, date10 NUMBER, code11 NUMBER, order11 NUMBER, date11 NUMBER)';
committ;
END creates;
/

TRIGGERS (sequencers)

CREATE OR REPLACE TRIGGER patients_sequencer
BEFORE INSERT ON patients
FOR EACH ROW
DECLARE
  v_next number;
BEGIN
  SELECT patients_seq.nextval into v_next
  FROM dual;
  :new.dimension_key := v_next;
END;
/

CREATE OR REPLACE TRIGGER interventions_sequencer
BEFORE INSERT ON interventions
FOR EACH ROW
DECLARE
    v_next number;
BEGIN
    SELECT interventions_seq.nextval into v_next FROM dual;
    :new.dimension_key := v_next;
    :new.nursing_diagnoses_ni_id := v_next;
    :new.intervention_ni_id := v_next;
END;
/

CREATE OR REPLACE TRIGGER nursing_diagnoses_sequencer
BEFORE INSERT ON nursing_diagnoses
FOR EACH ROW
DECLARE
    v_next number;
BEGIN
    SELECT nursing_diagnoses_seq.nextval into v_next FROM dual;
    :new.dimension_key := v_next;
    :new.ndcode_id := v_next;
END;
/

CREATE OR REPLACE TRIGGER diseases_sequencer
BEFORE INSERT ON diseases
FOR EACH ROW
DECLARE
    v_next number;
BEGIN
    SELECT diseases_seq.nextval into v_next FROM dual;
    :new.dimension_key := v_next;
    :new.ccs_icd_id := v_next;
    :new.icd_icd_id := v_next;
END;
/

CREATE OR REPLACE TRIGGER outcomes_sequencer
BEFORE INSERT ON outcomes
FOR EACH ROW
DECLARE
    v_next number;
BEGIN
    SELECT outcomes_seq.nextval into v_next FROM dual;
    :new.dimension_key := v_next;
    :new.nursing_diagnoses_no_id := v_next;
    :new.outcomes_no_id := v_next;
END;
/
commit;
PROCEDURES

CREATE OR REPLACE PROCEDURE diseases_groupings_insert AS

    CURSOR c1 is SELECT seq, dimension_key, ordering
               FROM system.diseases SD JOIN diseases D ON SD.icd = D.icd
               ORDER BY seq, dimension_key;

    resultRow c1%rowtype;
    v_oldSEQ groupings.seq%type;
    v_next number;
    v_index number;

    BEGIN
        open c1;
        fetch c1 into resultRow;
        v_oldSEQ:=0;
        v_index:=2;

        SELECT diseases_groupings_seq.nextval into v_next
        FROM dual;

        while c1%found loop
            if(resultRow.seq = v_oldSEQ) then
                if(v_index = 2) then
                    UPDATE groupings
                    SET code2 = resultRow.dimension_key, order2 =
                    resultRow.ordering
                    WHERE seq = resultRow.seq;
                elsif(v_index = 3) then
                    UPDATE groupings
                    SET code3 = resultRow.dimension_key, order3 =
                    resultRow.ordering
                    WHERE seq = resultRow.seq;
                elsif(v_index = 4) then
                    UPDATE groupings
                    SET code4 = resultRow.dimension_key, order4 =
                    resultRow.ordering
                    WHERE seq = resultRow.seq;
                elsif(v_index = 5) then
                    UPDATE groupings
                    SET code5 = resultRow.dimension_key, order5 =
                    resultRow.ordering
                    WHERE seq = resultRow.seq;
                elsif(v_index = 6) then
                    UPDATE groupings
                    SET code6 = resultRow.dimension_key, order6 =
                    resultRow.ordering
                    WHERE seq = resultRow.seq;
                elsif(v_index = 7) then
                    UPDATE groupings
                    SET code7 = resultRow.dimension_key, order7 =
                    resultRow.ordering
                    WHERE seq = resultRow.seq;
                elsif(v_index = 8) then
                    UPDATE groupings
                    SET code8 = resultRow.dimension_key, order8 =
                    resultRow.ordering
                    WHERE seq = resultRow.seq;
                elsif(v_index = 9) then
                    UPDATE groupings
                    SET code9 = resultRow.dimension_key, order9 =
                    resultRow.ordering
                    WHERE seq = resultRow.seq;

            end if;
            v_oldSEQ:=resultRow.seq;
            v_index:=v_index+1;
        end while;

    END;
WHERE seq = resultRow.seq;
elsif(v_index = 10) then
  UPDATE groupings
  SET code10 = resultRow.dimension_key, order10 = resultRow.ordering
  WHERE seq = resultRow.seq;
elsif(v_index = 11) then
  UPDATE groupings
  SET code11 = resultRow.dimension_key, order11 = resultRow.ordering
  WHERE seq = resultRow.seq;
end if;
v_index:= v_index +1;
else
  SELECT diseases_groupings_seq.nextval into v_next
  FROM dual;
  v_oldSEQ := resultRow.seq;
  INSERT INTO groupings (key, seq, code1, order1)
  VALUES (v_next, resultRow.seq, resultRow.dimension_key,
          resultRow.ordering);
  v_index:=2;
end if;
fetch c1 into resultRow;
end loop;
close c1;
commit;
END diseases_groupings_insert;
/
CREATE OR REPLACE PROCEDURE diseases_grouping_insert AS
CURSOR c1 is SELECT seq, code1, order1, code2, order2, code3, order3, code4,
  order4, code5, order5, code6, order6, code7, order7, code8,
  order8, code9, order9, code10, order10, code11, order11,
  seq
FROM groupings
ORDER BY code1, order1, code2, order2, code3, order3, code4,
  order4, code5, order5, code6, order6, code7, order7, code8,
  order8, code9, order9, code10, order10, code11, order11;
resultRow c1%rowtype;
v_total number;
v_next number;
v_icd_id number;
v_code1 number;
v_order1 number;
v_code2 number;
v_order2 number;
v_code3 number;
v_order3 number;
v_code4 number;
v_order4 number;
v_code5 number;
v_order5 number;
v_code6 number;
v_order6 number;
v_code7 number;
v_order7 number;
v_code8 number;
v_order8 number;
v_code9 number;
v_order9 number;
v_code10 number;
v_order10 number;
v_code11 number;
v_order11 number;
begin
open c1;
fetch c1 into resultRow;
v_total:=0;
v_icd_id:=0;
v_code1:=0;
v_order1:=0;
v_code2:=0;
v_order2:=0;
v_code3:=0;
v_order3:=0;
v_code4:=0;
v_order4:=0;
v_code5:=0;
v_order5:=0;
v_code6:=0;
v_order6:=0;
v_code7:=0;
v_order7:=0;
v_code8:=0;
v_order8:=0;
v_code9:=0;
v_order9:=0;
v_code10:=0;
v_order10:=0;
v_code11:=0;
v_order11:=0;
while c1%found loop
if(nvl(resultRow.code1,0) = nvl(v_code1,0) AND nvl(resultRow.order1,0) =
nvl(v_order1,0) AND nvl(resultRow.code2,0) = nvl(v_code2,0) AND
nvl(resultRow.order2,0) = nvl(v_order2,0)
AND nvl(resultRow.code3,0) = nvl(v_code3,0) AND
nvl(resultRow.order3,0) = nvl(v_order3,0) AND nvl(resultRow.code4,0) =
nvl(v_code4,0) AND nvl(resultRow.order4,0) = nvl(v_order4,0)
AND nvl(resultRow.code5,0) = nvl(v_code5,0) AND
nvl(resultRow.order5,0) = nvl(v_order5,0) AND nvl(resultRow.code6,0) =
nvl(v_code6,0) AND nvl(resultRow.order6,0) = nvl(v_order6,0)
AND nvl(resultRow.code7,0) = nvl(v_code7,0) AND
nvl(resultRow.order7,0) = nvl(v_order7,0) AND nvl(resultRow.code8,0) =
nvl(v_code8,0) AND nvl(resultRow.order8,0) = nvl(v_order8,0)
AND nvl(resultRow.code9,0) = nvl(v_code9,0) AND
nvl(resultRow.order9,0) = nvl(v_order9,0) AND nvl(resultRow.code10,0) =
nvl(v_code10,0) AND nvl(resultRow.order10,0) = nvl(v_order10,0)
AND nvl(resultRow.code11,0) = nvl(v_code11,0) AND
nvl(resultRow.order11,0) = nvl(v_order11,0)) then
null;
else
SELECT diseases_grouping_seq.nextval into v_next
FROM dual;
INSERT INTO diseases_group_dim (dimension_key, diseases_group_id)
VALUES (v_next, v_next);
if(resultRow.code1 > 0) then
INSERT INTO diseases_group (dimension_key, dg_id, icd_ID, ordering)
VALUES (v_next, v_next, resultRow.code1, resultRow.order1);
end if;

if(resultRow.code2 > 0) then
    INSERT INTO diseases_group (dimension_key, dg_id, icd_ID, ordering)
    VALUES (v_next, v_next, resultRow.code2, resultRow.order2);
end if;

if(resultRow.code3 > 0) then
    INSERT INTO diseases_group (dimension_key, dg_id, icd_ID, ordering)
    VALUES (v_next, v_next, resultRow.code3, resultRow.order3);
end if;

if(resultRow.code4 > 0) then
    INSERT INTO diseases_group (dimension_key, dg_id, icd_ID, ordering)
    VALUES (v_next, v_next, resultRow.code4, resultRow.order4);
end if;

if(resultRow.code5 > 0) then
    INSERT INTO diseases_group (dimension_key, dg_id, icd_ID, ordering)
    VALUES (v_next, v_next, resultRow.code5, resultRow.order5);
end if;

if(resultRow.code6 > 0) then
    INSERT INTO diseases_group (dimension_key, dg_id, icd_ID, ordering)
    VALUES (v_next, v_next, resultRow.code6, resultRow.order6);
end if;

if(resultRow.code7 > 0) then
    INSERT INTO diseases_group (dimension_key, dg_id, icd_ID, ordering)
    VALUES (v_next, v_next, resultRow.code7, resultRow.order7);
end if;

if(resultRow.code8 > 0) then
    INSERT INTO diseases_group (dimension_key, dg_id, icd_ID, ordering)
    VALUES (v_next, v_next, resultRow.code8, resultRow.order8);
end if;

if(resultRow.code9 > 0) then
    INSERT INTO diseases_group (dimension_key, dg_id, icd_ID, ordering)
    VALUES (v_next, v_next, resultRow.code9, resultRow.order9);
end if;

if(resultRow.code10 > 0) then
    INSERT INTO diseases_group (dimension_key, dg_id, icd_ID, ordering)
    VALUES (v_next, v_next, resultRow.code10, resultRow.order10);
end if;

if(resultRow.code11 > 0) then
    INSERT INTO diseases_group (dimension_key, dg_id, icd_ID, ordering)

VALUES (v_next, v_next, resultRow.code11,
resultRow.order11);
end if;

v_code1:=resultRow.code1;
v_order1:=resultRow.order1;
v_code2:=resultRow.code2;
v_order2:=resultRow.order2;
v_code3:=resultRow.code3;
v_order3:=resultRow.order3;
v_code4:=resultRow.code4;
v_order4:=resultRow.order4;
v_code5:=resultRow.code5;
v_order5:=resultRow.order5;
v_code6:=resultRow.code6;
v_order6:=resultRow.order6;
v_code7:=resultRow.code7;
v_order7:=resultRow.order7;
v_code8:=resultRow.code8;
v_order8:=resultRow.order8;
v_code9:=resultRow.code9;
v_order9:=resultRow.order9;
v_code10:=resultRow.code10;
v_order10:=resultRow.order10;
v_code11:=resultRow.code11;
v_order11:=resultRow.order11;
end if;

UPDATE visits
SET diseases_group_id = v_next
WHERE seq = resultRow.seq;

fetch c1 into resultRow;
end loop;
close c1;
END diseases_grouping_insert;
/

CREATE OR REPLACE PROCEDURE nd_groupings_insert AS

CURSOR c1 is SELECT D.dimension_key as DD, N.dimension_key as ND, SN.ordering,
SN.seq
FROM nursing_diagnoses N JOIN system.nursing_diagnoses SN ON N.ndcode =
SN.nndcode JOIN date_out D ON SN.nnddate = D.day
ORDER BY SN.seq;

resultRow c1%rowtype;
v_oldSEQ groupings.seq%type;
v_next number;
v_index number;

BEGIN
open c1;
fetch c1 into resultRow;
v_oldSEQ:=0;
v_index:=2;

SELECT nursing_diagnoses_gs_seq.nextval into v_next
FROM dual;

while c1%found loop
if(resultRow.seq = v_oldSEQ) then
if (v_index = 2) then
    UPDATE groupings
    SET code2 = resultRow.ND, order2 = resultRow.ordering,
        date2 = resultRow.DD
    WHERE seq = resultRow.seq;
elsif (v_index = 3) then
    UPDATE groupings
    SET code3 = resultRow.ND, order3 = resultRow.ordering,
        date3 = resultRow.DD
    WHERE seq = resultRow.seq;
elsif (v_index = 4) then
    UPDATE groupings
    SET code4 = resultRow.ND, order4 = resultRow.ordering,
        date4 = resultRow.DD
    WHERE seq = resultRow.seq;
elsif (v_index = 5) then
    UPDATE groupings
    SET code5 = resultRow.ND, order5 = resultRow.ordering,
        date5 = resultRow.DD
    WHERE seq = resultRow.seq;
elsif (v_index = 6) then
    UPDATE groupings
    SET code6 = resultRow.ND, order6 = resultRow.ordering,
        date6 = resultRow.DD
    WHERE seq = resultRow.seq;
elsif (v_index = 7) then
    UPDATE groupings
    SET code7 = resultRow.ND, order7 = resultRow.ordering,
        date7 = resultRow.DD
    WHERE seq = resultRow.seq;
elsif (v_index = 8) then
    UPDATE groupings
    SET code8 = resultRow.ND, order8 = resultRow.ordering,
        date8 = resultRow.DD
    WHERE seq = resultRow.seq;
elsif (v_index = 9) then
    UPDATE groupings
    SET code9 = resultRow.ND, order9 = resultRow.ordering,
        date9 = resultRow.DD
    WHERE seq = resultRow.seq;
elsif (v_index = 10) then
    UPDATE groupings
    SET code10 = resultRow.ND, order10 = resultRow.ordering,
        date10 = resultRow.DD
    WHERE seq = resultRow.seq;
elsif (v_index = 11) then
    UPDATE groupings
    SET code11 = resultRow.ND, order11 = resultRow.ordering,
        date11 = resultRow.DD
    WHERE seq = resultRow.seq;
end if;

v_index:= v_index +1;
else
    SELECT nursing_diagnoses_gs_seq.nextval into v_next
    FROM dual;

    v_oldSEQ := resultRow.seq;

    INSERT INTO groupings (key, seq, code1, order1, date1)
    VALUES (v_next, resultRow.seq, resultRow.ND, resultRow.ordering,
        resultRow.DD);

    v_index:=2;
end if;

fetch c1 into resultRow;
end loop;
close c1;
commit;
END nd_groupings_insert;
/

CREATE OR REPLACE PROCEDURE nd_grouping_insert AS

CURSOR c1 is SELECT seq, code1, order1, date1, code2, order2, date2, code3,
    order3, date3, code4, order4, date4, code5, order5, date5, code6, order6,
    date6, code7, order7, date7, code8, order8, date8, code9, order9, date9,
    code10, order10, date10, code11, order11, date11
FROM groupings
ORDER BY code1, order1, date1, code2, order2, date2, code3,
    order3, date3, code4, order4, date4, code5, order5, date5,
    code6, order6, date6, code7, order7, date7, code8, order8,
    date8, code9, order9, date9, code10, order10, date10,
    code11, order11, date11, seq;
resultRow c1%rowtype;
v_total number;
v_next number;
v_icd_id number;
v_code1 number;
v_date1 number;
v_code2 number;
v_date2 number;
v_code3 number;
v_order3 number;
v_date3 number;
v_code4 number;
v_order4 number;
v_date4 number;
v_code5 number;
v_order5 number;
v_date5 number;
v_code6 number;
v_order6 number;
v_date6 number;
v_code7 number;
v_order7 number;
v_date7 number;
v_code8 number;
v_order8 number;
v_date8 number;
v_code9 number;
v_order9 number;
v_date9 number;
v_code10 number;
v_order10 number;
v_date10 number;
v_code11 number;
v_order11 number;
v_date11 number;
BEGIN
    open c1;
    fetch c1 into resultRow;
v_total:=0;
v_icd_id:=0;
v_code1:=0;
v_order1:=0;
v_date1:=0;
v_code2:=0;
v_order2:=0;
v_date2:=0;
v_code3:=0;
v_order3:=0;
v_date3:=0;
v_code4:=0;
v_order4:=0;
v_date4:=0;
v_code5:=0;
v_order5:=0;
v_date5:=0;
v_code6:=0;
v_order6:=0;
v_date6:=0;
v_code7:=0;
v_order7:=0;
v_date7:=0;
v_code8:=0;
v_order8:=0;
v_date8:=0;
v_code9:=0;
v_order9:=0;
v_date9:=0;
v_code10:=0;
v_order10:=0;
v_date10:=0;
v_code11:=0;
v_order11:=0;
v_date11:=0;

while c1%found loop
if(nvl(resultRow.code1,0) = nvl(v_code1,0) AND nvl(resultRow.order1,0) = nvl(v_order1,0) AND nvl(resultRow.date1,0) = nvl(v_date1,0)
AND nvl(resultRow.code2,0) = nvl(v_code2,0) AND nvl(resultRow.order2,0) = nvl(v_order2,0) AND nvl(resultRow.date2,0) = nvl(v_date2,0)
AND nvl(resultRow.code3,0) = nvl(v_code3,0) AND nvl(resultRow.order3,0) = nvl(v_order3,0) AND nvl(resultRow.date3,0) = nvl(v_date3,0)
AND nvl(resultRow.code4,0) = nvl(v_code4,0) AND nvl(resultRow.order4,0) = nvl(v_order4,0) AND nvl(resultRow.date4,0) = nvl(v_date4,0)
AND nvl(resultRow.code5,0) = nvl(v_code5,0) AND nvl(resultRow.order5,0) = nvl(v_order5,0) AND nvl(resultRow.date5,0) = nvl(v_date5,0)
AND nvl(resultRow.code6,0) = nvl(v_code6,0) AND nvl(resultRow.order6,0) = nvl(v_order6,0) AND nvl(resultRow.date6,0) = nvl(v_date6,0)
AND nvl(resultRow.code7,0) = nvl(v_code7,0) AND nvl(resultRow.order7,0) = nvl(v_order7,0) AND nvl(resultRow.date7,0) = nvl(v_date7,0)
AND nvl(resultRow.code8,0) = nvl(v_code8,0) AND nvl(resultRow.order8,0) = nvl(v_order8,0) AND nvl(resultRow.date8,0) = nvl(v_date8,0)
AND nvl(resultRow.code9,0) = nvl(v_code9,0) AND nvl(resultRow.order9,0) = nvl(v_order9,0) AND nvl(resultRow.date9,0) = nvl(v_date9,0)
AND nvl(resultRow.code10,0) = nvl(v_code10,0) AND  
nvl(resultRow.order10,0) = nvl(v_order10,0) AND nvl(resultRow.date10,0) =  
nvl(v_date10,0)
AND nvl(resultRow.code11,0) = nvl(v_code11,0) AND 

nvl(resultRow.order11,0) = nvl(v_order11,0) AND nvl(resultRow.date11,0) =  
nvl(v_date11,0)) then
    null;
else
    SELECT nursing_diagnoses_grouping_seq.nextval into v_next 
    FROM dual;

    INSERT INTO nursing_diagnoses_group_dim (dimension_key, 
nursing_diagnoses_group_id) 
VALUES (v_next, v_next);

if(resultRow.code1 > 0) then
    INSERT INTO nursing_diagnoses_group (dimension_key, ndg_id, 
ndcode_ID, ordering, nddate_id) 
VALUES (v_next, v_next, resultRow.code1, resultRow.order1, 
resultRow.date1);
end if;

if(resultRow.code2 > 0) then
    INSERT INTO nursing_diagnoses_group (dimension_key, ndg_id, 
ndcode_ID, ordering, nddate_id) 
VALUES (v_next, v_next, resultRow.code2, resultRow.order2, 
resultRow.date2);
end if;

if(resultRow.code3 > 0) then
    INSERT INTO nursing_diagnoses_group (dimension_key, ndg_id, 
ndcode_ID, ordering, nddate_id) 
VALUES (v_next, v_next, resultRow.code3, resultRow.order3, 
resultRow.date3);
end if;

if(resultRow.code4 > 0) then
    INSERT INTO nursing_diagnoses_group (dimension_key, ndg_id, 
ndcode_ID, ordering, nddate_id) 
VALUES (v_next, v_next, resultRow.code4, resultRow.order4, 
resultRow.date4);
end if;

if(resultRow.code5 > 0) then
    INSERT INTO nursing_diagnoses_group (dimension_key, ndg_id, 
ndcode_ID, ordering, nddate_id) 
VALUES (v_next, v_next, resultRow.code5, resultRow.order5, 
resultRow.date5);
end if;

if(resultRow.code6 > 0) then
    INSERT INTO nursing_diagnoses_group (dimension_key, ndg_id, 
ndcode_ID, ordering, nddate_id) 
VALUES (v_next, v_next, resultRow.code6, resultRow.order6, 
resultRow.date6);
end if;

if(resultRow.code7 > 0) then
    INSERT INTO nursing_diagnoses_group (dimension_key, ndg_id, 
ndcode_ID, ordering, nddate_id) 
VALUES (v_next, v_next, resultRow.code7, resultRow.order7, 
resultRow.date7);
end if;

if(resultRow.code8 > 0) then
    INSERT INTO nursing_diagnoses_group (dimension_key, ndg_id, ndcode_ID, ordering, nddate_id)
    VALUES (v_next, v_next, resultRow.code8, resultRow.order8, resultRow.date8);
end if;

if(resultRow.code9 > 0) then
    INSERT INTO nursing_diagnoses_group (dimension_key, ndg_id, ndcode_ID, ordering, nddate_id)
    VALUES (v_next, v_next, resultRow.code9, resultRow.order9, resultRow.date9);
end if;

if(resultRow.code10 > 0) then
    INSERT INTO nursing_diagnoses_group (dimension_key, ndg_id, ndcode_ID, ordering, nddate_id)
    VALUES (v_next, v_next, resultRow.code10, resultRow.order10, resultRow.date10);
end if;

if(resultRow.code11 > 0) then
    INSERT INTO nursing_diagnoses_group (dimension_key, ndg_id, ndcode_ID, ordering, nddate_id)
    VALUES (v_next, v_next, resultRow.code11, resultRow.order11, resultRow.date11);
end if;

v_code1:=resultRow.code1;
v_order1:=resultRow.order1;
v_date1:=resultRow.date1;
v_code2:=resultRow.code2;
v_order2:=resultRow.order2;
v_date2:=resultRow.date2;
v_code3:=resultRow.code3;
v_order3:=resultRow.order3;
v_date3:=resultRow.date3;
v_code4:=resultRow.code4;
v_order4:=resultRow.order4;
v_date4:=resultRow.date4;
v_code5:=resultRow.code5;
v_order5:=resultRow.order5;
v_date5:=resultRow.date5;
v_code6:=resultRow.code6;
v_order6:=resultRow.order6;
v_date6:=resultRow.date6;
v_code7:=resultRow.code7;
v_order7:=resultRow.order7;
v_date7:=resultRow.date7;
v_code8:=resultRow.code8;
v_order8:=resultRow.order8;
v_date8:=resultRow.date8;
v_code9:=resultRow.code9;
v_order9:=resultRow.order9;
v_date9:=resultRow.date9;
v_code10:=resultRow.code10;
v_order10:=resultRow.order10;
v_date10:=resultRow.date10;
v_code11:=resultRow.code11;
v_order11:=resultRow.order11;
v_date11:=resultRow.date11;
UPDATE visits
SET nursing_diagnoses_group_id = v_next
WHERE seq = resultRow.seq;

fetch c1 into resultRow;
end loop;
close c1;

END nd_grouping_insert;
/

CREATE OR REPLACE PROCEDURE interventions_groupings_insert AS

CURSOR c1 is SELECT SI.seq, I.dimension_key as InD, D.dimension_key as DD,
    SI.ordering
    FROM system.interventions SI, interventions I, date_out D
    WHERE SI.nicode_group = I.nicode_group
    AND SI.nicode = I.nursing_diagnoses_nicode
    AND SI.nidefining_code = I.nidefining_code
    AND SI.nidate = D.day
    ORDER BY SI.seq;

resultRow c1%rowtype;
v_oldSEQ groupings.seq%type;
v_next number;
v_index number;

BEGIN
open c1;
fetch c1 into resultRow;
v_oldSEQ:=0;
v_index:=2;

SELECT interventions_groupings_seq.nextval into v_next
FROM dual;

while c1%found loop
    if(resultRow.seq = v_oldSEQ) then
        if(v_index = 2) then
            UPDATE groupings
            SET code2 = resultRow.InD, order2 = resultRow.ordering,
                date2 = resultRow.DD
            WHERE seq = resultRow.seq;
        elsif(v_index = 3) then
            UPDATE groupings
            SET code3 = resultRow.InD, order3 = resultRow.ordering,
                date3 = resultRow.DD
            WHERE seq = resultRow.seq;
        elsif(v_index = 4) then
            UPDATE groupings
            SET code4 = resultRow.InD, order4 = resultRow.ordering,
                date4 = resultRow.DD
            WHERE seq = resultRow.seq;
        elsif(v_index = 5) then
            UPDATE groupings
            SET code5 = resultRow.InD, order5 = resultRow.ordering,
                date5 = resultRow.DD
            WHERE seq = resultRow.seq;
        elsif(v_index = 6) then
            UPDATE groupings
            SET code6 = resultRow.InD, order6 = resultRow.ordering,
date6 = resultRow.DD
WHERE seq = resultRow.seq;
elsif(v_index = 7) then
  UPDATE groupings
  SET code7 = resultRow.InD, order7 = resultRow.ordering,
      date7 = resultRow.DD
  WHERE seq = resultRow.seq;
elsif(v_index = 8) then
  UPDATE groupings
  SET code8 = resultRow.InD, order8 = resultRow.ordering,
      date8 = resultRow.DD
  WHERE seq = resultRow.seq;
elsif(v_index = 9) then
  UPDATE groupings
  SET code9 = resultRow.InD, order9 = resultRow.ordering,
      date9 = resultRow.DD
  WHERE seq = resultRow.seq;
elsif(v_index = 10) then
  UPDATE groupings
  SET code10 = resultRow.InD, order10 = resultRow.ordering,
       date10 = resultRow.DD
  WHERE seq = resultRow.seq;
elsif(v_index = 11) then
  UPDATE groupings
  SET code11 = resultRow.InD, order11 = resultRow.ordering,
       date11 = resultRow.DD
  WHERE seq = resultRow.seq;
end if;
v_index:= v_index +1;
else
  SELECT interventions_groupings_seq.nextval into v_next
  FROM dual;
v_oldSEQ := resultRow.seq;
  INSERT INTO groupings (key, seq, code1, order1, date1)
  VALUES (v_next, resultRow.seq, resultRow.InD, resultRow.ordering,
      resultRow.DD);
v_index:=2;
end if;
fetch c1 into resultRow;
end loop;
close c1;
commit;
END interventions_groupings_insert;
/

CREATE OR REPLACE PROCEDURE interventions_grouping_insert AS

CURSOR c1 is SELECT seq, code1, order1, date1, code2, order2, date2, code3,
      order3, date3, code4, order4, date4, code5, order5, date5, code6,
      order6, date6, code7, order7, date7, code8, order8, date8, code9,
      order9, date9, code10, order10, date10, code11, order11, date11
  FROM groupings
ORDER BY code1, order1, date1, code2, order2, date2, code3,
      order3, date3, code4, order4, date4, code5, order5, date5, code6,
      order6, date6, code7, order7, date7, code8, order8, date8, code9,
      order9, date9, code10, order10, date10, code11, order11, date11
FROM groupings
ORDER BY code1, order1, date1, code2, order2, date2, code3,
      order3, date3, code4, order4, date4, code5, order5, date5, code6,
      order6, date6, code7, order7, date7, code8, order8, date8, code9,
      order9, date9, code10, order10, date10, code11, order11, date11
FROM groupings

resultRow c1%rowtype;
v_total number;
v_next number;
v_icd_id number;
v_code1 number;
v_order1 number;
v_date1 number;
v_code2 number;
v_order2 number;
v_date2 number;
v_code3 number;
v_order3 number;
v_date3 number;
v_code4 number;
v_order4 number;
v_date4 number;
v_code5 number;
v_order5 number;
v_date5 number;
v_code6 number;
v_order6 number;
v_date6 number;
v_code7 number;
v_order7 number;
v_date7 number;
v_code8 number;
v_order8 number;
v_date8 number;
v_code9 number;
v_order9 number;
v_date9 number;
v_code10 number;
v_order10 number;
v_date10 number;
v_code11 number;
v_order11 number;
v_date11 number;

BEGIN
open c1;
fetch c1 into resultRow;
  v_total:=0;
  v_icd_id:=0;
  v_code1:=0;
  v_order1:=0;
  v_date1:=0;
  v_code2:=0;
  v_order2:=0;
  v_date2:=0;
  v_code3:=0;
  v_order3:=0;
  v_date3:=0;
  v_code4:=0;
  v_order4:=0;
  v_date4:=0;
  v_code5:=0;
  v_order5:=0;
  v_date5:=0;
  v_code6:=0;
  v_order6:=0;
  v_date6:=0;
  v_code7:=0;
  v_order7:=0;
  v_date7:=0;
v_code8:=0;
v_order8:=0;
v_date8:=0;
v_code9:=0;
v_order9:=0;
v_date9:=0;
v_code10:=0;
v_order10:=0;
v_date10:=0;
v_code11:=0;
v_order11:=0;
v_date11:=0;

while c1%found loop
  if(nvl(resultRow.code1,0) = nvl(v_code1,0) AND nvl(resultRow.order1,0) = nvl(v_order1,0) AND nvl(resultRow.date1,0) = nvl(v_date1,0)
      AND nvl(resultRow.code2,0) = nvl(v_code2,0) AND nvl(resultRow.order2,0) = nvl(v_order2,0) AND nvl(resultRow.date2,0) = nvl(v_date2,0)
      AND nvl(resultRow.code3,0) = nvl(v_code3,0) AND nvl(resultRow.order3,0) = nvl(v_order3,0) AND nvl(resultRow.date3,0) = nvl(v_date3,0)
      AND nvl(resultRow.code4,0) = nvl(v_code4,0) AND nvl(resultRow.order4,0) = nvl(v_order4,0) AND nvl(resultRow.date4,0) = nvl(v_date4,0)
      AND nvl(resultRow.code5,0) = nvl(v_code5,0) AND nvl(resultRow.order5,0) = nvl(v_order5,0) AND nvl(resultRow.date5,0) = nvl(v_date5,0)
      AND nvl(resultRow.code6,0) = nvl(v_code6,0) AND nvl(resultRow.order6,0) = nvl(v_order6,0) AND nvl(resultRow.date6,0) = nvl(v_date6,0)
      AND nvl(resultRow.code7,0) = nvl(v_code7,0) AND nvl(resultRow.order7,0) = nvl(v_order7,0) AND nvl(resultRow.date7,0) = nvl(v_date7,0)
      AND nvl(resultRow.code8,0) = nvl(v_code8,0) AND nvl(resultRow.order8,0) = nvl(v_order8,0) AND nvl(resultRow.date8,0) = nvl(v_date8,0)
      AND nvl(resultRow.code9,0) = nvl(v_code9,0) AND nvl(resultRow.order9,0) = nvl(v_order9,0) AND nvl(resultRow.date9,0) = nvl(v_date9,0)
      AND nvl(resultRow.code10,0) = nvl(v_code10,0) AND nvl(resultRow.order10,0) = nvl(v_order10,0) AND nvl(resultRow.date10,0) = nvl(v_date10,0)
      AND nvl(resultRow.code11,0) = nvl(v_code11,0) AND nvl(resultRow.order11,0) = nvl(v_order11,0) AND nvl(resultRow.date11,0) = nvl(v_date11,0)) then
    null;
  else
    SELECT interventions_grouping_seq.nextval into v_next
    FROM dual;

    INSERT INTO interventions_group_dim (dimension_key, interventions_group_id)
    VALUES (v_next, v_next);

    if(resultRow.code1 > 0) then
      INSERT INTO interventions_group (dimension_key, nig_id, ni_ID, ordering, nidate_id)
      VALUES (v_next, v_next, resultRow.code1, resultRow.order1, resultRow.date1);
    end if;
  end if;
if(resultRow.code2 > 0) then
    INSERT INTO interventions_group (dimension_key, nig_id, ni_id, ordering, nidate_id)
    VALUES (v_next, v_next, resultRow.code2, resultRow.order2, resultRow.date2);
end if;

if(resultRow.code3 > 0) then
    INSERT INTO interventions_group (dimension_key, nig_id, ni_id, ordering, nidate_id)
    VALUES (v_next, v_next, resultRow.code3, resultRow.order3, resultRow.date3);
end if;

if(resultRow.code4 > 0) then
    INSERT INTO interventions_group (dimension_key, nig_id, ni_id, ordering, nidate_id)
    VALUES (v_next, v_next, resultRow.code4, resultRow.order4, resultRow.date4);
end if;

if(resultRow.code5 > 0) then
    INSERT INTO interventions_group (dimension_key, nig_id, ni_id, ordering, nidate_id)
    VALUES (v_next, v_next, resultRow.code5, resultRow.order5, resultRow.date5);
end if;

if(resultRow.code6 > 0) then
    INSERT INTO interventions_group (dimension_key, nig_id, ni_id, ordering, nidate_id)
    VALUES (v_next, v_next, resultRow.code6, resultRow.order6, resultRow.date6);
end if;

if(resultRow.code7 > 0) then
    INSERT INTO interventions_group (dimension_key, nig_id, ni_id, ordering, nidate_id)
    VALUES (v_next, v_next, resultRow.code7, resultRow.order7, resultRow.date7);
end if;

if(resultRow.code8 > 0) then
    INSERT INTO interventions_group (dimension_key, nig_id, ni_id, ordering, nidate_id)
    VALUES (v_next, v_next, resultRow.code8, resultRow.order8, resultRow.date8);
end if;

if(resultRow.code9 > 0) then
    INSERT INTO interventions_group (dimension_key, nig_id, ni_id, ordering, nidate_id)
    VALUES (v_next, v_next, resultRow.code9, resultRow.order9, resultRow.date9);
end if;

if(resultRow.code10 > 0) then
    INSERT INTO interventions_group (dimension_key, nig_id, ni_id, ordering, nidate_id)
    VALUES (v_next, v_next, resultRow.code10, resultRow.order10, resultRow.date10);
end if;
if(resultRow.code11 > 0) then
    INSERT INTO interventions_group (dimension_key, nig_id,
        ni_ID, ordering, nidate_id)
VALUES (v_next, v_next, resultRow.code11,
        resultRow.order11, resultRow.date11);
end if;

v_code1:=resultRow.code1;
v_order1:=resultRow.order1;
v_date1:=resultRow.date1;
v_code2:=resultRow.code2;
v_order2:=resultRow.order2;

v_date2:=resultRow.date2;
v_code3:=resultRow.code3;
v_order3:=resultRow.order3;
v_date3:=resultRow.date3;
v_code4:=resultRow.code4;
v_order4:=resultRow.order4;
v_date4:=resultRow.date4;
v_code5:=resultRow.code5;
v_order5:=resultRow.order5;
v_date5:=resultRow.date5;
v_code6:=resultRow.code6;
v_order6:=resultRow.order6;
v_date6:=resultRow.date6;
v_code7:=resultRow.code7;
v_order7:=resultRow.order7;
v_date7:=resultRow.date7;
v_code8:=resultRow.code8;
v_order8:=resultRow.order8;
v_date8:=resultRow.date8;
v_code9:=resultRow.code9;
v_order9:=resultRow.order9;
v_date9:=resultRow.date9;
v_code10:=resultRow.code10;
v_order10:=resultRow.order10;
v_date10:=resultRow.date10;
v_code11:=resultRow.code11;
v_order11:=resultRow.order11;
v_date11:=resultRow.date11;
end if;

UPDATE visits
SET interventions_group_id = v_next
WHERE seq = resultRow.seq;

fetch c1 into resultRow;
end loop;
close c1;

END interventions_grouping_insert;
/

CREATE OR REPLACE PROCEDURE outcomes_groupings_insert AS

    CURSOR c1 is SELECT SO.seq, O.dimension_key as OD, D.dimension_key as DD,
        SO.ordering
    FROM system.outcomes SO, outcomes O, date_out D
    WHERE SO.nocode = O.nursing_diagnoses_nocode
        AND SO.nodefining_code = O.nodefining_code
        AND SO.nodate = D.day
    ORDER BY SO.seq;


resultRow c1%rowtype;
v_oldSEQ groupings.seq%type;
v_next number;
v_index number;
BEGIN
open c1;
fetch c1 into resultRow;
v_oldSEQ:=0;
v_index:=2;
SELECT outcomes_groupings_seq.nextval into v_next
FROM dual;
while c1%found loop
  if(resultRow.seq = v_oldSEQ) then
    if(v_index = 2) then
      UPDATE groupings
      SET code2 = resultRow.OD, order2 = resultRow.ordering,
          date2 = resultRow.DD
      WHERE seq = resultRow.seq;
    elseif(v_index = 3) then
      UPDATE groupings
      SET code3 = resultRow.OD, order3 = resultRow.ordering,
          date3 = resultRow.DD
      WHERE seq = resultRow.seq;
    elseif(v_index = 4) then
      UPDATE groupings
      SET code4 = resultRow.OD, order4 = resultRow.ordering,
          date4 = resultRow.DD
      WHERE seq = resultRow.seq;
    elseif(v_index = 5) then
      UPDATE groupings
      SET code5 = resultRow.OD, order5 = resultRow.ordering,
          date5 = resultRow.DD
      WHERE seq = resultRow.seq;
    elseif(v_index = 6) then
      UPDATE groupings
      SET code6 = resultRow.OD, order6 = resultRow.ordering,
          date6 = resultRow.DD
      WHERE seq = resultRow.seq;
    elseif(v_index = 7) then
      UPDATE groupings
      SET code7 = resultRow.OD, order7 = resultRow.ordering,
          date7 = resultRow.DD
      WHERE seq = resultRow.seq;
    elseif(v_index = 8) then
      UPDATE groupings
      SET code8 = resultRow.OD, order8 = resultRow.ordering,
          date8 = resultRow.DD
      WHERE seq = resultRow.seq;
    elseif(v_index = 9) then
      UPDATE groupings
      SET code9 = resultRow.OD, order9 = resultRow.ordering,
          date9 = resultRow.DD
      WHERE seq = resultRow.seq;
    elseif(v_index = 10) then
      UPDATE groupings
      SET code10 = resultRow.OD, order10 = resultRow.ordering,
          date10 = resultRow.DD
      WHERE seq = resultRow.seq;
    elseif(v_index = 11) then
      UPDATE groupings
      SET code11 = resultRow.OD, order11 = resultRow.ordering,
          date11 = resultRow.DD
      WHERE seq = resultRow.seq;
  end if;
  v_index:=v_index+1;
v_oldSEQ:=resultRow.seq;
end loop;
```
SET code11 = resultRow.OD, order11 = resultRow.ordering, 
date11 = resultRow.DD
WHERE seq = resultRow.seq;
end if;
else
v_index := v_index +1;
SELECT outcomes_groupings_seq.nextval into v_next
FROM dual;
v_oldSEQ := resultRow.seq;
INSERT INTO groupings (key, seq, code1, order1, date1)
VALUES (v_next, resultRow.seq, resultRow.OD, resultRow.ordering, 
resultRow.DD);
v_index:=2;
end if;

fetch c1 into resultRow;
end loop;
close c1;
commit;
END outcomes_groupings_insert;
/

CREATE OR REPLACE PROCEDURE outcomes_grouping_insert AS

CURSOR c1 is SELECT seq, code1, order1, date1, code2, order2, date2, code3, 
order3, date3, code4, order4, date4, code5, order5, date5, 
code6, order6, date6, code7, order7, date7, code8, order8, 
date8, code9, order9, date9, code10, order10, date10, 
code11, order11, date11 
FROM groupings 
ORDER BY code1, order1, date1, code2, order2, date2, code3, 
order3, date3, code4, order4, date4, code5, order5, date5, 
code6, order6, date6, code7, order7, date7, code8, order8, 
date8, code9, order9, date9, code10, order10, date10, 
code11, order11, date11, seq;
resultRow c1%rowtype;

v_total number;
v_next number;
v_icd_id number;
v_code1 number;
v_order1 number;
v_date1 number;
v_code2 number;
v_order2 number;
v_date2 number;
v_code3 number;
v_order3 number;
v_date3 number;
v_code4 number;
v_order4 number;
v_date4 number;
v_code5 number;
v_order5 number;
v_date5 number;
v_code6 number;
v_order6 number;
v_date6 number;
v_code7 number;
v_order7 number;
v_code8 number;
v_order8 number;
v_code9 number;
v_order9 number;
v_code10 number;
v_order10 number;
v_code11 number;

END outcomes_grouping_insert;
/```
v_code8 number;
v_order8 number;
v_date8 number;
v_code9 number;
v_order9 number;
v_date9 number;
v_code10 number;
v_order10 number;
v_date10 number;
v_code11 number;
v_order11 number;
v_date11 number;

BEGIN
open c1;
fetch c1 into resultRow;
v_total:=0;
v_icd_id:=0;
v_code1:=0;
v_order1:=0;
v_date1:=0;
v_code2:=0;
v_order2:=0;
v_date2:=0;
v_code3:=0;
v_order3:=0;
v_date3:=0;
v_code4:=0;
v_order4:=0;
v_date4:=0;
v_code5:=0;
v_order5:=0;
v_date5:=0;
v_code6:=0;
v_order6:=0;
v_date6:=0;
v_code7:=0;
v_order7:=0;
v_date7:=0;
v_code8:=0;
v_order8:=0;
v_date8:=0;
v_code9:=0;
v_order9:=0;
v_date9:=0;
v_code10:=0;
v_order10:=0;
v_date10:=0;
v_code11:=0;
v_order11:=0;
v_date11:=0;

while c1%found loop
if(nvl(resultRow.code1,0) = nvl(v_code1,0) AND nvl(resultRow.order1,0) = nvl(v_order1,0) AND nvl(resultRow.date1,0) = nvl(v_date1,0)
    AND nvl(resultRow.code2,0) = nvl(v_code2,0) AND nvl(resultRow.order2,0) = nvl(v_order2,0) AND nvl(resultRow.date2,0) = nvl(v_date2,0)
    AND nvl(resultRow.code3,0) = nvl(v_code3,0) AND nvl(resultRow.order3,0) = nvl(v_order3,0) AND nvl(resultRow.date3,0) = nvl(v_date3,0)

    THEN
        v_total:=v_total+1;
    END if;
end loop;

END;
AND nvl(resultRow.code4, 0) = nvl(v_code4, 0) AND nvl(resultRow.order4, 0) = nvl(v_order4, 0) AND nvl(resultRow.date4, 0) = nvl(v_date4, 0)
    AND nvl(resultRow.code5, 0) = nvl(v_code5, 0) AND nvl(resultRow.order5, 0) = nvl(v_order5, 0) AND nvl(resultRow.date5, 0) = nvl(v_date5, 0)
    AND nvl(resultRow.code6, 0) = nvl(v_code6, 0) AND nvl(resultRow.order6, 0) = nvl(v_order6, 0) AND nvl(resultRow.date6, 0) = nvl(v_date6, 0)
    AND nvl(resultRow.code7, 0) = nvl(v_code7, 0) AND nvl(resultRow.order7, 0) = nvl(v_order7, 0) AND nvl(resultRow.date7, 0) = nvl(v_date7, 0)
    AND nvl(resultRow.code8, 0) = nvl(v_code8, 0) AND nvl(resultRow.order8, 0) = nvl(v_order8, 0) AND nvl(resultRow.date8, 0) = nvl(v_date8, 0)
    AND nvl(resultRow.code9, 0) = nvl(v_code9, 0) AND nvl(resultRow.order9, 0) = nvl(v_order9, 0) AND nvl(resultRow.date9, 0) = nvl(v_date9, 0)
    AND nvl(resultRow.code10, 0) = nvl(v_code10, 0) AND nvl(resultRow.order10, 0) = nvl(v_order10, 0) AND nvl(resultRow.date10, 0) = nvl(v_date10, 0)
    AND nvl(resultRow.code11, 0) = nvl(v_code11, 0) AND nvl(resultRow.order11, 0) = nvl(v_order11, 0) AND nvl(resultRow.date11, 0) = nvl(v_date11, 0)) then
    null;
else
    SELECT outcomes_grouping_seq.nextval into v_next
    FROM dual;
    INSERT INTO outcomes_group_dim (dimension_key, outcomes_group_id)
    VALUES (v_next, v_next);
    if(resultRow.code1 > 0) then
        INSERT INTO outcomes_group (dimension_key, og_id, no_ID, ordering, nodate_id)
        VALUES (v_next, v_next, resultRow.code1, resultRow.order1, resultRow.date1);
    end if;
    if(resultRow.code2 > 0) then
        INSERT INTO outcomes_group (dimension_key, og_id, no_ID, ordering, nodate_id)
        VALUES (v_next, v_next, resultRow.code2, resultRow.order2, resultRow.date2);
    end if;
    if(resultRow.code3 > 0) then
        INSERT INTO outcomes_group (dimension_key, og_id, no_ID, ordering, nodate_id)
        VALUES (v_next, v_next, resultRow.code3, resultRow.order3, resultRow.date3);
    end if;
    if(resultRow.code4 > 0) then
        INSERT INTO outcomes_group (dimension_key, og_id, no_ID, ordering, nodate_id)
        VALUES (v_next, v_next, resultRow.code4, resultRow.order4, resultRow.date4);
    end if;
    if(resultRow.code5 > 0) then

\begin{verbatim}
INSERT INTO outcomes_group (dimension_key, og_id, no_ID, ordering, nodate_id)
VALUES (v_next, v_next, resultRow.code5, resultRow.order5, resultRow.date5);
end if;

if(resultRow.code6 > 0) then
    INSERT INTO outcomes_group (dimension_key, og_id, no_ID, ordering, nodate_id)
    VALUES (v_next, v_next, resultRow.code6, resultRow.order6, resultRow.date6);
end if;

if(resultRow.code7 > 0) then
    INSERT INTO outcomes_group (dimension_key, og_id, no_ID, ordering, nodate_id)
    VALUES (v_next, v_next, resultRow.code7, resultRow.order7, resultRow.date7);
end if;

if(resultRow.code8 > 0) then
    INSERT INTO outcomes_group (dimension_key, og_id, no_ID, ordering, nodate_id)
    VALUES (v_next, v_next, resultRow.code8, resultRow.order8, resultRow.date8);
end if;

if(resultRow.code9 > 0) then
    INSERT INTO outcomes_group (dimension_key, og_id, no_ID, ordering, nodate_id)
    VALUES (v_next, v_next, resultRow.code9, resultRow.order9, resultRow.date9);
end if;

if(resultRow.code10 > 0) then
    INSERT INTO outcomes_group (dimension_key, og_id, no_ID, ordering, nodate_id)
    VALUES (v_next, v_next, resultRow.code10, resultRow.order10, resultRow.date10);
end if;

if(resultRow.code11 > 0) then
    INSERT INTO outcomes_group (dimension_key, og_id, no_ID, ordering, nodate_id)
    VALUES (v_next, v_next, resultRow.code11, resultRow.order11, resultRow.date11);
end if;

v_code1:=resultRow.code1;
v_order1:=resultRow.order1;
v_date1:=resultRow.date1;
v_code2:=resultRow.code2;
v_order2:=resultRow.order2;
v_date2:=resultRow.date2;
v_code3:=resultRow.code3;
v_order3:=resultRow.order3;
v_date3:=resultRow.date3;
v_code4:=resultRow.code4;
v_order4:=resultRow.order4;
v_date4:=resultRow.date4;
v_code5:=resultRow.code5;
v_order5:=resultRow.order5;
v_date5:=resultRow.date5;
\end{verbatim}
### INSERT SCRIPTS (these do the work)

```sql
CREATE OR REPLACE PROCEDURE insertScripts AS
BEGIN
    EXECUTE IMMEDIATE 'INSERT INTO patients (pt_id, ssn, gender, race, zipcode, marstat, relig, birthdate_id)
    SELECT P.pt_id, P.ssn, P.gender, P.race, P.zipcode, P.marstat, P.relig, D.dimension_key
    FROM system.patients P JOIN date_out D ON P.birthdate = D.day';
    commit;

    EXECUTE IMMEDIATE 'INSERT INTO visits (seq, pt_id, services, los, disstate, age, admdate_id, disdate_id)
    SELECT V.seq, V.pt_id, V.service, V.los, V.disstate, V.age, D.dimension_key, D2.dimension_key
    FROM system.visits V JOIN date_out D ON V.admdate = D.day JOIN date_out D2 ON V.disdate = D2.day';
    commit;

    EXECUTE IMMEDIATE 'INSERT INTO diseases (icd, ccs_ccs_code, icd_ccs_code, ccs_ccs_name, icd_ccs_name)
    SELECT DISTINCT D.icd, C.ccs_code, C.ccs_code, C.ccs_name, C.ccs_name
    FROM system.diseases D JOIN system.ccs_icd I ON D.icd = I.ICD JOIN system.ccs C
    ON C.ccs_code = I.ccs_code';
    commit;

    diseases_groupings_insert;
    diseases_grouping_insert;
    EXECUTE IMMEDIATE 'delete from groupings';
    commit;
END
```

---

```sql
v_code6:=resultRow.code6;
v_order6:=resultRow.order6;
v_date6:=resultRow.date6;
v_code7:=resultRow.code7;
v_order7:=resultRow.order7;
v_date7:=resultRow.date7;
v_code8:=resultRow.code8;
v_order8:=resultRow.order8;
v_date8:=resultRow.date8;
v_code9:=resultRow.code9;
v_order9:=resultRow.order9;
v_date9:=resultRow.date9;
v_code10:=resultRow.code10;
v_order10:=resultRow.order10;
v_date10:=resultRow.date10;
v_code11:=resultRow.code11;
v_order11:=resultRow.order11;
v_date11:=resultRow.date11;
end if;

UPDATE visits
SET outcomes_group_id = v_next
WHERE seq = resultRow.seq;

fetch c1 into resultRow;
end loop;
close c1;

END outcomes_grouping_insert;
/
```
EXECUTE IMMEDIATE 'INSERT INTO nursing_diagnoses(ndcode)
SELECT DISTINCT nndcode
FROM system.nursing_diagnoses';
commit;

nd_groupings_insert;
nd_grouping_insert;
EXECUTE IMMEDIATE 'delete from groupings';
commit;

EXECUTE IMMEDIATE 'INSERT INTO interventions(nicode_group,
    nursing_diagnoses_nicode, intervention_nicode, nidefining_code)
SELECT DISTINCT nicode_group, nicode, nicode, nidefining_code
FROM system.interventions';
commit;

interventions_groupings_insert;
interventions_grouping_insert;
EXECUTE IMMEDIATE 'delete from groupings';
commit;

EXECUTE IMMEDIATE 'INSERT INTO outcomes(nursing_diagnoses_nocode,
    outcomes_nocode, nodefining_code)
SELECT DISTINCT nocode, nocode, nodefining_code
FROM system.outcomes';
commit;

outcomes_groupings_insert;
outcomes_grouping_insert;
EXECUTE IMMEDIATE 'delete from groupings';
commit;

mvRefresh;
commit;
END insertScripts;
/

MATERIALIZED VIEW REFRESH SCRIPTS

CREATE OR REPLACE PROCEDURE mvRefresh AS
BEGIN
  DBMS_MVIEW.REFRESH('mv_dg_i');
  DBMS_MVIEW.REFRESH('mv_ig_i');
  DBMS_MVIEW.REFRESH('mv_ndg');
  DBMS_MVIEW.REFRESH('mv_og_o');
  DBMS_MVIEW.REFRESH('mv_v_i_nd');
  DBMS_MVIEW.REFRESH('mv_v_i_nd_age');
  DBMS_MVIEW.REFRESH('mv_v_o_nd');
  DBMS_MVIEW.REFRESH('mv_v_o_nd_age');
  DBMS_MVIEW.REFRESH('mv_vdg_i_age');
  DBMS_MVIEW.REFRESH('mv_vig_i_age');
  DBMS_MVIEW.REFRESH('mv_vndg_age');
  DBMS_MVIEW.REFRESH('mv_vog_o_age');
commit;
END mvRefresh;
/
CLEAN UP SCRIPTS

CREATE OR REPLACE PROCEDURE cleanup AS
BEGIN
    EXECUTE IMMEDIATE 'DROP TRIGGER diseases_group_insert';
    EXECUTE IMMEDIATE 'DROP TRIGGER interventions_group_insert';
    EXECUTE IMMEDIATE 'DROP TRIGGER nursing_diagnoses_group_insert';
    EXECUTE IMMEDIATE 'DROP TRIGGER outcomes_group_insert';
    EXECUTE IMMEDIATE 'DROP PROCEDURE diseases_groupings_insert';
    EXECUTE IMMEDIATE 'DROP PROCEDURE diseases_grouping_insert';
    EXECUTE IMMEDIATE 'DROP PROCEDURE nd_groupings_insert';
    EXECUTE IMMEDIATE 'DROP PROCEDURE nd_grouping_insert';
    EXECUTE IMMEDIATE 'DROP PROCEDURE interventions_groupings_insert';
    EXECUTE IMMEDIATE 'DROP PROCEDURE interventions_grouping_insert';
    EXECUTE IMMEDIATE 'DROP PROCEDURE outcomes_groupings_insert';
    EXECUTE IMMEDIATE 'DROP PROCEDURE outcomes_grouping_insert';
    EXECUTE IMMEDIATE 'DROP TABLE groupings';
    commit;
END cleanup;
/

CREATE OR REPLACE PROCEDURE cleanup AS
BEGIN
    EXECUTE IMMEDIATE 'DROP TABLE groupings';
    commit;
END cleanup;
/

REVOKE SCRIPTS

CREATE OR REPLACE PROCEDURE revokes AS
BEGIN
    revoke all from rep_owner;
    commit;
END revokes;
/

DROP ALL TABLES EXCEPT DATES

CREATE OR REPLACE PROCEDURE clearAllData_leaveDates AS
BEGIN
    DELETE FROM visits;
    DELETE FROM diseases_group;
    DELETE FROM interventions_group;
    DELETE FROM outcomes_group;
    DELETE FROM nursing_diagnoses_group;
    DELETE FROM diseases_group_dim;
    DELETE FROM interventions_group_dim;
    DELETE FROM nursing_diagnoses_group_dim;
    DELETE FROM outcomes_group_dim;
    DELETE FROM outcomes;
    DELETE FROM interventions;
    DELETE FROM diseases;
    DELETE FROM nursing_diagnoses;
    DELETE FROM patients;
    commit;
END clearAllData_leaveDates;
/
DROP ALL DATA INCLUDING DATES

CREATE OR REPLACE PROCEDURE clearAllData AS
BEGIN
    DELETE FROM visits;
    DELETE FROM diseases_group;
    DELETE FROM interventions_group;
    DELETE FROM outcomes_group;
    DELETE FROM nursing_diagnoses_group;
    DELETE FROM diseases_group_dim;
    DELETE FROM interventions_group_dim;
    DELETE FROM nursing_diagnoses_group_dim;
    DELETE FROM outcomes_group_dim;
    DELETE FROM outcomes;
    DELETE FROM outcomes_group_dim;
    DELETE FROM interventions;
    DELETE FROM diseases;
    DELETE FROM nursing_diagnoses;
    DELETE FROM patients;
    DELETE FROM date_out;
    commit;
END clearAllData;
/

CHECK SCRIPTS

these are simple copy and paste checks / all statements in each section should equal
------------------------------------------------------------------------------
DISEASES
------------------------------------------------------------------------------
total number of records
-----------------------------
SELECT count(*)
FROM system.diseases SD JOIN diseases D ON SD.icd = D.icd
ORDER BY seq, dimension_key;

SELECT count(*)
FROM visits V JOIN diseases_group_dim DGD ON V.diseases_group_id = DGD.dimension_key
    JOIN diseases_group DG ON DGD.dimension_key = DG.dimension_key;

total number of unique id's
-----------------------------
SELECT count(cnt)
FROM (SELECT diseases_group_id, count(*) as cnt
    FROM visits
    WHERE diseases_group_id > 0
    GROUP BY diseases_group_id);

SELECT count(*)
FROM diseases_group_dim;
compression first/second

SELECT count(*)  
FROM diseases_group;

SELECT count(*)  
FROM system.diseases SD JOIN diseases D ON SD.icd = D.icd  
ORDER BY seq, dimension_key;

-------------------------------------------------------------------------
nursing_diagnoses  
-------------------------------------------------------------------------

SELECT count(*)  
FROM nursing_diagnoses N JOIN system.nursing_diagnoses SN ON N.ndcode = SN.nndcode  
JOIN date_out D ON SN.nnddate = D.day;

SELECT count(*)  
FROM visits V JOIN nursing_diagnoses_group_dim NDGD ON V.nursing_diagnoses_group_id =  
NDGD.dimension_key  
JOIN nursing_diagnoses_group NDG ON NDGD.dimension_key = NDG.dimension_key;

SELECT count(*)  
FROM nursing_diagnoses_group_dim;

-------------------------------------------------------------------------
interventions  
-------------------------------------------------------------------------

SELECT count(*)  
FROM system.interventions SI, interventions I, date_out D  
WHERE SI.nicode_group = I.nicode_group  
AND SI.nicode = I.nursing_diagnoses_nicode  
AND SI.nidefining_code = I.nidefining_code  
AND SI.nidate = D.day;
SELECT count(*)
FROM visits V JOIN interventions_group_dim IGD ON V.interventions_group_id = IGD.dimension_key
    JOIN interventions_group IG ON IGD.dimension_key = IG.dimension_key;

------------------------
total number of unique id's
------------------------
SELECT count(cnt)
FROM (SELECT interventions_group_id, count(*) as cnt
    FROM visits
    WHERE interventions_group_id > 0
    GROUP BY interventions_group_id);

SELECT count(*)
FROM interventions_group_dim;

------------------------
compression first/second
------------------------
SELECT count(*)
FROM interventions_group;

SELECT count(*)
FROM system.interventions SI, interventions I, date_out D
WHERE SI.nicode_group = I.nicode_group
AND SI.nicode = I.nursing_diagnoses_nicode
AND SI.nidefining_code = I.nidefining_code
AND SI.nidate = D.day;

-------------------------------------------------------------------------
outcomes
-------------------------------------------------------------------------
------------------------
total number of records
------------------------
SELECT count(*)
FROM system.outcomes SO, outcomes O, date_out D
WHERE SO.nocode = O.nursing_diagnoses_nocode
AND SO.nodefining_code = O.nodefining_code
AND SO.nodate = D.day;

SELECT count(*)
FROM visits V JOIN outcomes_group_dim OGD ON V.outcomes_group_id = OGD.dimension_key
    JOIN outcomes_group OG ON OGD.dimension_key = OG.dimension_key;

------------------------
total number of unique id's
------------------------
SELECT count(cnt)
FROM (SELECT outcomes_group_id, count(*) as cnt
    FROM visits
    WHERE outcomes_group_id > 0
    GROUP BY outcomes_group_id);

SELECT count(*)
FROM outcomes_group_dim;
---------------------------
compression first/second
---------------------------

SELECT count(*)
FROM outcomes_group;

SELECT count(*)
FROM system.outcomes SO, outcomes O, date_out D
WHERE SO.nocode = O.nursing_diagnoses_nocode
AND SO.nodefining_code = O.nodefining_code
AND SO.nodate = D.day;

GRANTS FOR ALLOWING DATA MINING (not included in regular grants)

GRANT create mining model TO rep_owner;
GRANT create procedure TO rep_owner;
GRANT create session TO rep_owner;
GRANT create table TO rep_owner;
GRANT create sequence TO rep_owner;
GRANT create view TO rep_owner;
GRANT create job TO rep_owner;
GRANT create type TO rep_owner;
GRANT create synonym TO rep_owner;
GRANT execute on ctxsys.ctx_ddl TO rep_owner;

CLEAN LOAD

CREATE OR REPLACE PROCEDURE clean_import_from_genesis AS
BEGIN
creates;
insertScripts;
cleanup;
commit;
END clean_import_from_genesis;
/

RELOAD

CREATE OR REPLACE PROCEDURE re_import_from_genesis AS
BEGIN
  clearAllData_leaveDates;
drops;
  creates;
  insertScripts;
  cleanup;
  commit;
END re_import_from_genesis;
/