712. Query Optimization

Optimization of i2b2 query strategy

i2b2 queries often contain multiple attributes which are combined with AND clauses (e.g. "diagnosis = diabetes AND age <= 18 years"). The i2b2 CRC cell (Clinical Research Chart) queries these attributes consecutively from the OBSERVATION_FACT table, using the result sets of each query as a base for the next query. Thus the amount of patients or encounters that needs to be evaluated decrease over each step of the query plan. In order to narrow down the search space quickly, it would be ideal to first query for attributes occurring rarely in the overall dataset. While this optimization is supported by i2b2, the platform needs to know about the frequency of each concept in the overall dataset to make use of it.

The ontology table "I2B2" contains the column "C_TOTALNUM" for this purpose. It provides the count of distinct patients for each concept in the overall dataset. The CRC cell utilizes this column to sort the order of query steps in the overall query plan in order to first query for attributes with low occurrence. However, the C_TOTALNUM column is not routinely filled during loading of the OBSERVATION_FACT table, which leads to this optimization not being applied in many cases.

The following SQL Script can be executed at the end of an ETL (extraction, transformation, loading) pathway to calculate all value for the C_TOTALNUM column automatically. The script was developed on an Oracle 11g platform but should be easily portable to other relational databases.

Coverage

Please note that the script will calculate the patient counts only over facts stored in the OBSERVATION_FACT table. If you installation contains fact data in other tables (e.g. PATIENT_DIMENSION) the script will need to be adapted to your individual configuration.

```
-- SQL Script to calculate the column i2b2.c_totalnum

-- 1. Backup the i2b2 table
CREATE TABLE t_i2b2 AS (SELECT * FROM i2b2);

-- 2a. Calculate count of distinct patients for all branches and leaves of concept hierarchy (regular concepts
CREATE TABLE t_totalnum AS
SELECT i2b.c_tablename,
       i2b.c_fullname,
       COUNT(DISTINCT obs.patient_num) AS c_totalnum
  FROM observation_fact  obs
  JOIN concept_dimension con ON obs.concept_cd   =    con.concept_cd
  JOIN i2b2              i2b ON con.concept_path LIKE i2b.c_fullname || '%'
 GROUP BY i2b.c_tablename,i2b.c_fullname;

-- 2b. (Optionally) Calculate count of distinct patients for all branches and leaves of concept hierarchy
(modifier concepts
INSERT INTO t_totalnum (c_tablename, c_fullname, c_totalnum)
SELECT i2b.c_tablename,
       i2b.c_fullname,
       COUNT(DISTINCT obs.patient_num) AS c_totalnum
  FROM observation_fact   obs
  JOIN modifier_dimension con ON obs.modifier_cd   =    con.modifier_cd
  JOIN i2b2               i2b ON con.modifier_path LIKE i2b.c_fullname || '%'
 GROUP BY i2b.c_tablename,i2b.c_fullname;

-- 3. Truncate i2b2 table
TRUNCATE TABLE i2b2;

-- 4. Repopulate the i2b2 table from its backup and the generated counts
INSERT INTO i2b2 (c_hlevel, c_fullname, c_name, c_synonym_cd,
c_visualattributes, c_totalnum, c_basecode, c_metadataxml,
c_facttablecolumn, c_tablename, c_columnname, c_columndatatype,
c_operator, c_dimcode, c_comment, c_tooltip, m_applied_path,
update_date, download_date, import_date, sourcesystem_cd, valuetype_cd,
m_exclusion_cd, c_path, c_symbol)
SELECT i2b.c_hlevel,
i2b.c_fullname, i2b.c_name, i2b.c_synonym_cd, i2b.c_visualattributes,
NVL(tnm.c_totalnum, 0), i2b.c_basecode, i2b.c_metadataxml,
i2b.c_facttablecolumn, i2b.c_tablename, i2b.c_columnname,
i2b.c_columndatatype, i2b.c_operator, i2b.c_dimcode, i2b.c_comment,
i2b.c_tooltip, i2b.m_applied_path, i2b.update_date, i2b.download_date,
i2b.import_date, i2b.sourcesystem_cd, i2b.valuetype_cd,
i2b.m_exclusion_cd, i2b.c_path, i2b.c_symbol
  FROM      t_i2b2     i2b
  LEFT JOIN t_totalnum tnm ON i2b.c_tablename = tnm.c_tablename AND i2b.c_fullname = tnm.c_fullname;

-- 5. Drop temporary tables
DROP TABLE t_totalnum;
DROP TABLE t_i2b2;

-- 6. Commit changes
COMMIT;
```

Notes

The script temporarily truncates the i2b2 table and repopulates it from a backup. This should be kept in mind if local modifications of an i2b2 installation include triggers on this table.

The script creates and drops 2 temporary tables during execution.

The script should be invoked with the user credentials of the schema containing the i2b2 ontology table.

As an additional advantage, the i2b2 web client displays values of C_TOTALNUM in the concept hierarchy. Please note that i2b2 currently does not show counts for modifier concepts and to our knowledge also does not utilize the patient counts for modifier concepts for query optimization. Step 2b of the above SQL script (which is runtime intensive) can thus be commented out without negative effects.

Calculation of database table statistics

Many relational databases keep internal statistics about the composition of data in all tables of the system. These provide information about the total number of values in each column as well as their distribution, among other details. The database query optimizer leverages these statistics to construct the potentially best query strategy, including the order in which to query joined tables or the order in which to apply selection criteria. Depending on database configuration, these statistics may be updated automatically, e.g. after table content is updated, periodically (e.g. each night) or manually. Out-of-date statistics, which do no longer reflect the current content of a table, may lead to notable performance impacts because of inappropriate query strategies proposed by the optimizer. Thus, unless the database system is generally configured to update table statistics immediately after loading, it is recommended to include a procedure to explicitly update table statistics at the end of an i2b2 ETL pathway.

The following Oracle PL/SQL script invokes the function to gather the statistics of the i2b2 schema. The placeholder [SCHEMA] needs to be replaced with the name of the schema containing the i2b2 fact and/or dimension tables.

```
BEGIN
DBMS_STATS.GATHER_SCHEMA_STATS([SCHEMA]);
END;
```

Index optimization of the OBSERVATION_FACT table

Column indexes are a long established method to improve query performance by speeding up the selection of relevant rows as well as join operations. In order to achieve this effect, the distinct values of selected table columns are stored separately, together with pointers to table rows containing each value. The database can leverage the often smaller size of these index structures to more quickly locate relevant data rows, instead of having to scan the full original table. Indexes are often stored in specialized structures (e.g. binary trees) which are optimized for quick access. Indexes can be constructed on single columns as well as multiple columns (composite indexes), which are concatenated to form the actual index value. Composite indexes provide an additional performance benefit when multiple columns are queried that are all part of the composite index, as only a single index structure needs to be examined, instead of several. However, composite indexes can only be applied if the provided search criteria are matching the order of columns in the index (e.g. if a composite index is concatenated out of columns A, B and C, it cannot be applied if column A is not part of the query).

Whether the database actually uses indexes or not is decided by the query optimizer at runtime, depending on table statistics and cached performance data from previous queries, and explicit "optimizer hints" that can be inludced in SQL queries, among others. The query strategy actually chosen can be examined at runtime (e.g. with the Oracle EXPLAIN PLAN statement) or by inspecting query logs.

The OBSERVATION_FACT table as it is setup in a default i2b2 installation already contains several indexes for query optimization:

Index Name	Task	Contained columns
OBSERVATION_FACT_PK	primary key	ENCOUNTER_NUM, CONCEPT_CD, PROVIDER_ID, START_DATE
FACT_CNPT_PAT_ENCT_IDX	optimize queries by concept code	CONCEPT_CD, INSTANCE_NUM, PATIENT_NUM, ENCOUNTER_NUM
FACT_NOLOB	optimize queries by patient id and start date	PATIENT_NUM, START_DATE, CONCEPT_CD, ENCOUNTER_NUM
FACT_PATCON_DATE_PRVD_IDX	optimize queries by patient id and concept code	PATIENT_NUM, CONCEPT_CD, START_DATE, END_DATE
OF_CTX_BLOB	optimize fulltext queries to BLOB column	OBSERVATION_BLOB

With the exception of the OF_CTX_BLOB index, all default indexes are composites covering multiple columns. As mentioned above, composite indexes can only be applied when query criteria are included that match the order in which the composite index is concatenated. E.g. the FACT_NOLOB index would speed up searching for a concept code only when at least a patient number and a start date is also included (which precedes the concept code in the concatenation). This leaves only the FACT_CNPT_PAT_ENCT_INDEX to support queries by concept code without the necessity to include additional mandatory columns. The OF_CTX_BLOB column is a special "fulltext" index that relates only to the OBSERVATION_BLOB column, which can contain long texts (e.g. discharge letters, radiology findings) and is not filled for the majority of typical i2b2 data items.

As described above, only one index is useful for queries by concept code. The application of this index is also hardcoded into CRC query SQL by means of an optimizer hint forcing the query optimizer to use this index even if it would usually decide otherwise. However, the index only covers concept codes, not modifier codes. As modifiers have gained increasing importance since i2b2 v1.6, it is recommended to drop this index and replace it with 2 new indexes covering both concept as well as modifier codes. Dropping renaming the original index allows the query optimizer to freely choose appropriate indexes without having to remove the offending optimizer hint from the CRC cell source code.

The following Oracle SQL script drops the FACT_CNPT_PAT_ENCT_IDX and replaces it with 2 separate indexes for concept and modifier codes:

```
DROP INDEX fact_cnpt_pat_enct_idx;
CREATE INDEX idrt_fact_cnpt_pat_enct_idx ON observation_fact (concept_cd, instance_num, patient_num,
encounter_num);
CREATE INDEX idrt_fact_mdf_pat_enct_idx  ON observation_fact (modifier_cd, instance_num, patient_num,
encounter_num);
COMMIT;
```

The script should be invoked with the user credentials of the schema containing the i2b2 ontology table.

Related information: Partitioning of tables and indexes can provide further performance improvements. The following section contains a modified script to create partitioned indexes. Also, the ETL process can be sped up by dropping indexes before loading and recreating them afterwards.

Partitioning of the OBSERVATION_FACT Table

Some relational databases provide a partitioning feature, which can be applied to break up large tables into separate segments (partitioning, however, often is a separately licensed "enterprise-level" feature). The goal is to group data elements which are usually queried together in distinct partitions. While it would also be possible to explicitly place such groups into separate individual tables, the advantage of partitioning is that the overall table can still be accessed transparently under a single name, thus obviating the need to change program code accessing the table.

A prerequisite for partitioning is a "partitioning key" column that can be used to unequivocally assign each record to a single partition. Partitions can be defined by ranges over the partition key (e.g. years from 2001-2005, 2006-2010, ...) or by mapping individual values to partitions (list-based partitioning), and other methods. At runtime, the query optimizer can identify search-criteria over the partitioning key to preselect relevant partitions to access (e.g. only the 2001-2005 partition for a query limited to 2004), allowing it to skip over all other (irrelevant) partitions. Besides this kind of implicit partition selection (by including the partition key in query criteria), partitions can also be explicitly referenced in FROM clauses. Partitioning can be applied not only to table data, but also to related indexes, thus speeding up access to indexes as well. Partitioned indexes are also called "local" indexes.

Partitioning can also provide additional benefits during the ETL phase, as it simplifies the isolated truncating and loading of individual segments of data, without having to reload all data or needing to use slower DELETE statements for removing data to be reloaded.

In i2b2, the OBSERVATION_FACT table and its related indexes are obvious candidates for partitioning. As mentioned in the previous section, the volume of data types can be very heterogeneous, and partitioning can be leveraged e.g. to skip large volumes of laboratory findings for queries that do not address them. The CONCEPT_CD column can be used as a partitioning key: it is usually prefixed with a schema code that relates to the type of data referenced (e.g. ICD for diagnoses, LAB for lab values). Due to the high volume of distinct concept codes, however, list-based partitioning cannot be applied, but rather range-based partitioning that include all concepts within a schema key or group of schema keys. The definition of the partitions and key ranges needs to be adapted to each local i2b2 installation, as it reflects the schema keys and data volumes locally used.

Warning

Please note that existing tables cannot be partitioned. It is necessary to drop the OBSERVATION_FACT table, re-create it with appropriate partitions and reload the data (e.g. through the regular ETL process or from a prior copy of the table)

The following Oracle SQL script drops the OBSERVATION_FACT table, re-creates it with a sample partitioning scheme and re-creates all necessary indexes, including an option to also partition the indexes. NB: the indexes creation statement already includes the optimizations described in the previous section.

```
DROP TABLE "OBSERVATION_FACT" cascade constraints;
CREATE TABLE "OBSERVATION_FACT"
    (           "ENCOUNTER_NUM" NUMBER(38,0) NOT NULL ENABLE,
                "PATIENT_NUM" NUMBER(38,0) NOT NULL ENABLE,
                "CONCEPT_CD" VARCHAR2(50 BYTE) NOT NULL ENABLE,
                "PROVIDER_ID" VARCHAR2(50 BYTE) NOT NULL ENABLE,
                "START_DATE" DATE NOT NULL ENABLE,
                "MODIFIER_CD" VARCHAR2(100 BYTE) NOT NULL ENABLE,
                "INSTANCE_NUM" NUMBER(18,0) NOT NULL ENABLE,
                "VALTYPE_CD" VARCHAR2(50 BYTE),
                "TVAL_CHAR" VARCHAR2(255 CHAR),
                "NVAL_NUM" NUMBER(18,5),
                "VALUEFLAG_CD" VARCHAR2(50 BYTE),
                "QUANTITY_NUM" NUMBER(18,5),
                "UNITS_CD" VARCHAR2(50 BYTE),
                "END_DATE" DATE,
                "LOCATION_CD" VARCHAR2(50 BYTE),
                "OBSERVATION_BLOB" CLOB,
                "CONFIDENCE_NUM" NUMBER(18,5),
                "UPDATE_DATE" DATE,
                "DOWNLOAD_DATE" DATE,
                "IMPORT_DATE" DATE,
                "SOURCESYSTEM_CD" VARCHAR2(50 BYTE),
                "UPLOAD_ID" NUMBER(38,0),
                CONSTRAINT "OBSERVATION_FACT_PK" PRIMARY KEY ("ENCOUNTER_NUM", "CONCEPT_CD", "PROVIDER_ID",
"START_DATE", "MODIFIER_CD", "INSTANCE_NUM") USING INDEX LOCAL ENABLE
   )
LOB ("OBSERVATION_BLOB") STORE AS BASICFILE (ENABLE STORAGE IN ROW CHUNK 8192 RETENTION)
PARTITION BY RANGE ("CONCEPT_CD")
   (
   PARTITION BIO VALUES LESS THAN ('DEM'), -- "BIO" partition for biomaterials
   PARTITION DEM VALUES LESS THAN ('ENC'), -- "DEM" partition for demographics
   PARTITION ENC VALUES LESS THAN ('ICD'), -- "ENC" partition for encounter data
   PARTITION ICD VALUES LESS THAN ('LAB'), -- "ICD" partition for diagnoses
   PARTITION LAB VALUES LESS THAN ('OPS'), -- "LAB" partition for lab findings
   PARTITION OPS VALUES LESS THAN ('ORG'), -- "OPS" partition for procedures
   PARTITION ORG VALUES LESS THAN ('PAT'), -- "ORG" partition for organizational data
   PARTITION PAT VALUES LESS THAN ('RAD'), -- "PAT" partition for pathology findings
   PARTITION RAD VALUES LESS THAN ('THE'), -- "RAD" partition for radiology findings
   PARTITION THE VALUES LESS THAN ('ZZZ')  -- "THE" partition for therapy data
   );
CREATE INDEX of_ctx_blob                ON observation_fact (observation_blob) indextype is ctxsys.context
parameters ('sync (on commit)');
CREATE INDEX fact_nolob                 ON observation_fact (patient_num, start_date, concept_cd,
encounter_num, instance_num, nval_num, tval_char, valtype_cd, modifier_cd, valueflag_cd, provider_id,
quantity_num, units_cd, end_date, location_cd, confidence_num, update_date, download_date, import_date,
sourcesystem_cd, upload_id) LOCAL;
CREATE INDEX fact_patcon_date_prvd_idx   ON observation_fact (patient_num, concept_cd, start_date, end_date,
encounter_num, instance_num, provider_id, nval_num, valtype_cd) LOCAL;
CREATE INDEX idrt_fact_cnpt_pat_enct_idx ON observation_fact (concept_cd, instance_num, patient_num,
encounter_num) LOCAL;
CREATE INDEX idrt_fact_mdf_pat_enct_idx  ON observation_fact (modifier_cd, instance_num, patient_num,
encounter_num) LOCAL;
```

The script should be invoked with the user credentials of the schema containing the i2b2 ontology table.

Partition ranges are defined with an "alphanumeric less-than" syntax, i.e. the BIO partition contains all concept codes up to (but not including) the prefix DEM, which starts the demographics partition. "ZZZ" is given as a dummy upper limit for the last partition, catching all alphanumeric prefixes after "THE" for the therapy partition.

Please note that the fulltext index "OF_CTX_BLOB" cannot be partitioned (not supported by the database) and is thus created as a regular "global" index. All other indexes are partitioned using the "LOCAL" option.