Glossary.

Note: entries that have been revised have the latest revision mm/yy appended to their first line. A numbered comment will be added, in most cases, explaining the change.

March, 2008. We are starting to mark up this glossary to highlight its controlled vocabulary structure. Ultimately, the entries in a controlled vocabulary will become the predicates of a first-order predicate logic (FOPL) formalism, at which point software-implemented inferencing will be possible against the glossary, and against any data dictionaries or other glossaries based on it.

Every controlled vocabulary entry is itself an entry in the glossary. As soon as we can get to it, we will mark controlled vocabulary instances by italicizing their appearance in glossary definitions.

- 12/31/9999. (02/08) A value whose semantics are that of "no value", but which is treated by the DBMS as a valid date.

    1. When "12/31/9999" appears in the effectivity end date of a versioned row, it takes on additional semantics. It means "effectivity end date unknown". In addition, it also represents an assumption that the row it appears in is effective "until further notice". It is because of this interpretation that we can let the DBMS treat it as a valid date, i.e. as a date greater than any date specified in a query.

    2. However, semantically, "12/31/9999" almost never would be a valid date, because no business activities that we know of are concerned with an effectivity time period that extends up to but not beyond that date nearly eight-thousand years in the future.

- Active episode. (03/08) An episode whose most recent version is not a temporal delete version, and whose most recent version does not have an effectivity end date in the past.

    1. A past episode is not an active episode.

    2. A current episode is an active episode.

3. A future episode may or may not be an active episode. The future episodes which are active are the ones whose most recent version is not a logical delete.

- Adjacent versions. (03/8) Versions of the same object in which there are no clock ticks between the two versions.

    1. Explicitly adjacent versions. Adjacent versions in which the clock tick of the effective end date of the earlier version is exactly one clock tick earlier than the clock tick of the effective begin date of the later version.

    2. Implicitly adjacent versions. Adjacent versions in which the effectivity end date of the earlier version is "12/31/9999". Implicitly adjacent versions are the result of an episode merge.

- Child table, child row. (02/08)
    - (a) Y is a child table to an *object* table X if and only if there is a foreign key dependency from Y to X. A row in Y is a child to a row in X if and only if the row in Y has a foreign key whose value is identical to the primary key value of that related row in X.

    - (b) Y is a child table to a *version* table X if and only if there is an object foreign key dependency from Y to X. A row in Y is a child to a row in X if and only if the row in Y has an object foreign key whose value is identical to the object identifier of the related row in X, and the effectivity time period of that row in X wholly contains the effectivity time period of the row in Y.

    1. Parent/child relationships typically have a maximum cardinality of one/many, and a minimum cardinality of optional for the parent and required for the child. But it is a matter of which table contains the foreign key and which table is the reference of that foreign key that differentiates parent from child tables and rows. Cardinality constraints are *not* what make the difference.

    2. Child tables/rows are also sometimes referred to as "dependent tables" or "dependent rows", or various cognates (e.g. "RI-dependent tables", "RI-dependent rows").

    3. Revised 02/08 so the distinction applies when the dependency is to either an object or a version table.

- Clock tick. (03/08) The transition from one point in time to the next point in time, according to the "clock" which defines the granularity for all dates involved in temporal data management.

    1. The clock tick granularity used throughout these articles is a date. Thus, one clock tick is the transition from one day to the next day. In many real-world situations, the clock tick will be an hour, a minute, a second or even a full timestamp.

- Current version. (02/08) A version whose effectivity begin date is in the past, and whose effectivity end date is either unknown (represented by “12/31/9999”) or in the future.

    1. Current versions are always the most recent versions of current episodes.

- Effectivity time period. (02/08) The period of time for which a version is regarded as the truth about an object.

    1. In our version patterns, an effectivity time period is defined by an effectivity begin date and an effectivity end date, where "date" may be a calendar date or any other "tick of the clock" (as described in Part 2 of this series).

    2. Our convention is that the time period begins on the begin date, but ends one clock tick prior to the end date.

- EID, enterprise identifier. (03/08) The globally unique identifier of an object.

- Episode. (03/08) A series or one or more adjacent versions of an object in which the inaugural version is either the first version of that object, or a version which is not temporally adjacent to the version it immediately follows.

    - In the first case, it is the inaugural version of the inaugural episode of the object.

    - In the second case, it is the inaugural version of a non-inaugural episode.

        1. See “supercession”.

2. Thus, the first version of an object creates the initial episode of that object. That episode remains the current episode until a logical delete version for that object is inserted into the table, or until the effectivity end date of its most recent version is reached.

3. If, when a version Y is inserted into the table, the most recent version of that object already in the table is a logical delete version, or is a version whose effectivity end date is in the past, then Y inaugurates a non-initial episode of that object. That remains the current episode until a logical delete version for that object is inserted into the table, or until the effectivity end date of its current version is reached.

4. Related terminology:
    a. Current episode. An episode of an object whose inaugural (oldest) version has an effectivity begin date not in the future.

    b. Terminated episode. An episode of an object whose most recent version is a logical delete.

    c. Past (non-terminated) episode. An episode of an object whose most recent version is in the past, but is not a logical delete.

    d. Future episode. An episode of an object whose inaugural version has an effectivity begin date in the future.

    e. Future terminated episode. An future episode of an object whose most recent version is a logical delete.

    f. Temporally adjacent episodes. A series of temporally adjacent versions of the same object in which at least one version, which is neither the earliest nor the latest version of the object, is a logical delete version.[1]

- Episode merge. The process by which a current and a future episode of the same object become one episode when the effectivity end date of the most recent

[1] Those who have read C. J. Date, Hugh Darwen, Nikos Lorentzos, *Temporal Data and the Relational Model* (Morgan-Kaufmann, San Francisco, 2002) may notice that their approach does not permit temporally adjacent versions. (see pp. xxxx). (They do not use the concept of an episode in their discussions, however.) They believe that the Closed World Assumption (CWA) means that adjacent episodes are illegitimate. Also, they claim that requiring episodes to be non-adjacent is "a reasonable assumption". In later articles, we will explain why we think their interpretation of the CWA is wrong. As for whether their assumption is "reasonable", we know from real-world experience that it is not.

version of the current episode is “12/31/9999”, and the current date reaches the effectivity begin date of the inaugural version of the future episode.

1. This process merges the two versions because it creates a chronological sequence of versions in which there are no clock ticks between two chronologically successive versions.

2. When two episodes are merged, the result is an episode in which the most recent version of the earlier episode and the inaugural version of the later episode are *implicitly* adjacent. (See “Adjacent versions”.)

- Future episode. (02/08) An episode whose first version has an effectivity begin date in the future.

- Future version. (02/08) Any version of a future episode.

- Inaugural version (of an episode). (02/08) The version of an episode of an object with the chronologically earliest effectivity begin date.

- Most recent version. (03/08) A version of an episode of an object whose effectivity begin date is chronologically the latest across all versions of that episode.

    1. Until we begin discussing Version Pattern 7, we will assume that no two versions of the same object can have the same effectivity begin date. After all, between that date and the earliest effectivity end date between them, that would mean that we had two different statements of what was true about an object during that period of time. There can only be one truth, and should be only one version of it.

    2. With Version Pattern 7, we will discuss cases in which two or more versions of the same object might have the same effectivity begin date.

        - One source of such cases is when there is a business requirement to correct an error in a version table, but also to retain, as queryable history, the version discovered to be in error.

        - Another source of such cases is when one of several incompatible descriptions might be true of an object, in a given time period, but we don’t know which one.

3. Changed the definition substantively, by making most current versions relative to the episodes they occur in.

- Object foreign key (OFK). (02/08) A column in a versioned or non-versioned table which contains the object identifier used by one or more rows in a (not necessarily distinct) version table.

    1. For example, in Figure 2 of Part 20, client-nbr in the Policy table is an OFK.

    2. As an illustration of temporal RI, the referencing Policy table row must have the value of a client-nbr in a row in the Client table such that the effectivity period of the referenced row wholly contains the effectivity period of that row, or is itself an object row (thus without an effectivity period).

    3. "versioned table" changed to "versioned or non-versioned" in Part 21.

- Object referential integrity, object RI (02/08). The constraint that a row containing an OFK must reference a row whose object identifier value is identical to that OFK value, and whose effectivity time period wholly contains the effectivity time period of the referencing row.

    1. Object RI cannot be enforced by today's RDBMSs. This follows from the fact that OFKs reference an object only indirectly, by means of the one or more versions that implement it. In those versions, the referenced object identifier is only part of the primary key of its table, and is thus not necessarily (or even usually) unique.

    2. Object RI requires that the referenced (parent) table be a versioned table. But the referencing (child) table, the one that contains the OFK, may itself be either a versioned or an object (non-versioned) table.

- Object table, object. (02/08) A table whose rows represent persistent objects. Sometimes called a "non-versioned" table.

    1. Persistent objects are things that exist over time and can change over time, such as vendors, customers, employees, regulatory agencies, products, services, bills of material, invoices, purchase orders, claims, certifications, etc.

2. In an OLAP, star-schema database, dimension tables are tables of persistent objects.[2]

3. In an OLTP database, assuming that there are no version tables in the database, object tables are all the tables which are not transaction tables.

4. Roughly speaking, object tables are the tables which are the concern of Master Data Management.

- Original delete. (03/08) A delete transaction against a versioned object.

    1. A delete transaction written as though its target is an object table, not a version table.

    2. Thus, in our ongoing example, a business user submitting a "delete client" transaction will think in terms of a Client table and a delete of one of its rows, not in terms of a Client Version table and the supercession of one of its rows.

- Original insert. (03/08) A insert transaction against a versioned object.

    1. An insert transaction written as though its target is an object table, not a version table.

    2. Thus, in our ongoing example, a business user submitting an "insert client" transaction will think in terms of a Client table and an insert of a row representing a new client, not in terms of a Client Version table and the insert of a row representing a version of that client.

- Original (row-level) update. (03/08) A row-level update transaction against a versioned object.

    1. An update transaction written as though its target is an object table, not a version table.

    2. Thus, in our ongoing example, a business user submitting an "update client" transaction will think in terms of a Client table and an update of

[2] For a more in-depth discussion of different types of tables, see the articles "An Ontology of Tables", at MindfulData.com.

one of its rows, not in terms of a Client Version table and a supercession of one of its rows.

- Original update. (03/08) A update transaction (insert, row-level update, delete or upsert) against a versioned object

    1. An update transaction written as though its target is an object table, not a version table.

    2. Note that "update", in its first occurrence, is used generically, to mean any transaction that alters the state of the database; while in its second occurrence (qualified by "row-level"), it is used specifically in the sense of a transaction that affects one or more rows that already exist in the database.

    3. In these discussions, all original updates, unless otherwise noted, will be current transactions. Later in this series, we will describe original updates that are future-dated, meaning that the effectivity begin date is a date later than the date of the physical transaction itself.

    4. Later in this series, we will also describe original updates against hypothetical versions.

    5. Later in this series, we will also describe original updates that create versions that correct other versions.

- Original upsert. (03/08) An upsert transaction against a versioned object.

    1. An upsert transaction written as though its target is an object table, not a version table.

    2. Thus, in our ongoing example, a business user submitting an "upsert client" transaction will think in terms of a Client table and either an insert of a new row representing a new client if the client does not already exist on the database, or else an update of an existing row if he does.

    3. An upsert transaction against a non-versioned table, of course, is one that will be translated into either a physical insert against that table or else a physical update.

- Parent table, parent row. (02/08)

- (a) X is an *object* table which is a parent table to a table Y if and only if there is a foreign key dependency from Y to X. A row in Y is a child to a row in X if and only if the row in Y has a foreign key whose value is identical to the primary key value of that related row in X.

- (b) X is a *version* table which is a parent table to a table Y if and only if there is an object foreign key dependency from Y to X. A row in Y is a child to a row in X if and only if the row in Y has an object foreign key whose value is identical to the object identifier of the related row in X, and the effectivity time period of that row in X wholly contains the effectivity time period of the row in Y.

- Query-enforced temporal referential integrity. (02/08) Temporal referential integrity enforced by filtering out any violations at query time.

    1. Unnecessary if update-enforced temporal referential integrity is used instead.

- Queryable history. (02/08) Data about an object which was valid at some time in the past, which is no longer currently valid, but which as easily and rapidly accessible as current data.

    1. "As easily and rapidly accessible as current data" means what it says. Our way of providing this access is to use version tables. In such tables, production queries against current data (the most common kind of query) can be used to retrieve historical data simply by adding a date to a BETWEEN clause of a SQL statement that would, without that addition, retrieve the corresponding current data from an object table.

    2. As we have mentioned before (and will again), providing queryable history, in this manner, can significantly lower the development and operations cost of accessing historical data, and significantly improve the currency of the historical data retrieved.

- Standard referential integrity, standard RI. (02/08) The referential integrity constraint among non-versioned tables that today's RDBMSs can enforce.

    1. Since we are calling these non-versioned tables "object tables", we erroneously equated object RI with standard RI in Part 19.

- Supercede, supercession. (02/08) In these articles, we use these terms to refer to the replacement of a current version with a new current version.

    1. Supercession is a *logical* function. Physically, supercession is done by inserting a new row in a version table.

    2. Deletion in a version table is always done via supercession.

    3. Versionable updates, in a version table, are always done via supercession.

    4. However, creating the first version of an object does *not* involve supercession, as there is no current version of that object to supercede.

    5. A more subtle point. Creating the first version of an *episode* of an object also does not involve supercession. Even if other versions of an object exist, the last version of every non-current episode is either a delete version, or a version whose effectivity end date is in the past. Thus, when a new episode is started, the most recent prior version of that object is not a current version. Thus, there is no current version of the object to supercede when that new episode begins.

- Temporal delete. (03/08) The result of translating an original delete into a physical transaction that supercedes all active versions of the original object, and also all active versions of all temporally RI dependent objects.

    1. Just as a normal delete is an invalid transaction if the row representing the object in question is not on the database, a temporal delete is an invalid transaction if a row representing the current version of the object in question is not on the database.

- Temporal insert. (03/08) The result of translating an original insert into a physical transaction against a version table.

    1. Just as a normal insert is an invalid transaction if the row representing the object in question is already on the database, a temporal insert is an invalid transaction if a row representing the current version of the object in question is already on the database.

    2. These temporal inserts are always physical inserts of a new version of the object in question.

3. Note that temporal inserts may be of either current versions or of future-dated versions. In other words, the effective begin date of the temporal insert may be either the date of the physical transaction itself, or some future date.

4. If a temporal insert is a valid transaction, either the object was never represented in the database, or else it was represented but its most current episode is terminated at the time of the temporal insert. In either case, there will be no non-deleted future-dated versions to worry about.

- Temporal (row-level) update. (03/08) The result of translating an original update into a physical transaction against a version table.

    1. Just as a normal update is an invalid transaction if the row representing the object in question is not on the database, a temporal update is an invalid transaction if a row representing the current version of the object in question is not on the database.

    2. These temporal updates are always physical inserts of a new current version that supercedes the version current at the time the transaction began.

    3. In addition, if there are any future-dated versions of that object, they are also updated in the same manner. Each one is superceded by a new version based on the version it is replacing, updated with the temporal update.

    4. There will be no clock ticks lying between the effectivity end date of the superceded version, and the effectivity begin date of the superceding version. The latter immediately follows the former.

- Temporal transaction. A transaction against the active episodes of an object.

    1. A temporal transaction is always the result of a mapping from an original transaction.

- Temporal update. (03/08) The result of translating an original update into a physical transaction that affects one or more rows of one or more version tables.

- Temporal upsert. (03/08) The result of translating an original upsert into a physical transaction against a version table.

1. These temporal upserts are always inserts of a current version. If a current version already exists on the database when the transaction begins, the temporal upsert becomes a temporal update. Otherwise, it becomes a temporal insert.

2. A temporal update inserts a non-initial version into a current episode. A temporal insert inserts the inaugural version of a new episode.

3. Any clock-tick gap between two versions exists between the last version of one episode and the first version of its successor episode. As a corollary, any pair of versions with no clock-tick gap between them belong to the same episode.

4. For what happens at that point, see "temporal insert" and "temporal update".

- Temporal referential integrity, temporal RI. (03/08) If there is a temporal RI dependency from table Y to version table X (not necessarily distinct), then no exposed state of the database is valid in which any row in table Y is not object-RI linked to a row in table X whose effectivity time period wholly contains the effectivity time period of the row in table Y (if table Y is a version table). No queries are valid that would expose such a state.

    1. Revised the definition to be based on the concept of an "exposed state" of the database. See the introduction to Part 21 for the motivation for this change.

    2. See the entries "update-enforced temporal referential integrity", and "query-enforced temporal referential integrity".

- Terminating an episode. (03/08) Ending an episode by superceding its most recent version with a temporal delete version.

- Update-enforced temporal referential integrity. (03/08) Temporal referential integrity applied to transactions which update the database, and which therefore prevents violations of temporal referential integrity from appearing on the database.

    1. Ill-advised if this form of integrity checking would incur a large number of physical I/Os.

- Version, version table. (03/08) A table whose rows represent versions of persistent objects.

    1. A version of a persistent object is a time-slice of that object. A row in a version table represents a version, and describes that object as it was, is, will be and/or might be during its effectivity time period.

    2. A row in a version table is what the custodians of the table believe is the truth about the object it represents, during the indicated effectivity time period.

- Versionable updates. (03/08) Original updates to a versioned table which cause a new version to be inserted.

    1. In these articles, we have assumed that all original updates to versioned tables will cause a new version to be inserted. But in reality, the business is likely to want to track changes to only selected columns of a table. For the other columns, an original update can be applied as an update in place, and does not require creation of a superceding version.

- Wholly contained in (for effectivity time periods). (03/08) Time period 2 is wholly contained in time period 1 if and only if the effectivity begin date of time period 2 is equal to or later than the effectivity begin date of time period 1, and the effectivity end date of time period 2 is equal to or earlier than the effectivity end date of time period 1.

    1. In order to implement this constraint in today's production databases, we rely on SQL data types and operators. Specifically, we rely on dates and/or timestamps, and the DBMS implementation of comparison operators for those data types.