Návrh standardů pro tvorbu datového modelu

Topic	**Mandatory / Recommended**	**Description of the rule**
Identifiers of the database fields and tables	Mandatory	Write them in capital letters divided by underlines. Example of the identifier: THIS_IS_THE_NAME_OF_THE_TABLE There should be used always names that correspond to the field content and that describe the content as good as possible. (e.g. PERSON.REGISTRATION_STATE is preferred before the general name PERSON.STATE)
Naming consistency	Mandatory	Keep the naming in the database consistent. Use same names for the same things, use same data types for the same things. Incl. naming of primary keys, foreign keys, constraints, indexes for foreign keys, …
Character set	Recommended	Keep the database ready for the special characters. Use the default database character set Unicode.
Primary key	Mandatory	Define a primary key in each table.
Primary key naming	Recommended	Use always one field for the primary key called ID_XYZ where XYZ is the table name. The exception are the intermediate tables realizing a M:N relation where the primary key consists from two fields. In this case it is possible to create an additional ID_XYZ field as primary key, but some frameworks (like Hibernate) prefer to have a primary key from two fields in such a case.
Foreign key	Mandatory	Define foreign keys for relations between the tables. We create an index for each foreign key. Example: table PERSON has two fields ID_PERSON and NAME table BUILDING has fields ID_BUILDING and ID_PERSON (as owner), … If it is necessary to have more relations from BUILDING to PERSON we add "_AS_" in English "_ALS_" in German and "MEANING". E.g. HOUSE.ID_PERSON_AS_OWNER We can use it also in case of one field to make the meaning of the field more clear.
Create user /change user / timestamp	Mandatory	Create in each table 5 technical fields for logging who and when created the record, who and when updated the record for the last time and a timestamp to prevent concurrency conflicts (e.g. two users changing same record at the same time). "CREATE_DATE" Timestamp(3), "CREATE_USER" Number(38,0), "CHANGE_DATE" Timestamp(3), "CHANGE_USER" Number(38,0), "CHANGE_TIMESTAMP" Number(38,0) NOT NULL Set these values – by the triggers or by the application. (E.g. a trigger may automatically set CHANGE_TIMESTAMP according to a Oracle-sequence at the record creation and at each record modification). Example of usage of CHANGE_TIMESTAMP Application loads data about a person (incl. PERSON.CHANGE_TIMSTAMP). User can see the person on the screen. Let's imagine another user is changing the given person at this moment. Later the first person modifies the "old" data on the screen and tries to save them in DB. The application has to test at the saving whether there is in DB still the originally read value in the field CHANGE_TIMESTAMP. If not it means somebody else has changed the data in the meantime. And the saving should failure (rollback, no change in DB).

Archive table	Mandatory	Create an archive table (ARC_XYZ) for each original table (XYZ). Log each data change in the ARC table via (generated) triggers. It (in the combination with the 5 technical fields – see above) gives the user a chance to answer any time a question who and when set a certain value in some field or who and when deleted a certain record.
Archive table – additional fields	Recommended	CREATE TABLE "ARC_XYZ"("CHANGE_NR" NUMBER(38,0), -- filled from a global sequence for all ARC_ tables "ID_XYZ" Number(38,0), … "CREATE_DATE" Timestamp(3), "CREATE_USER" Number(38,0), "CHANGE_DATE" Timestamp(3), "CHANGE_USER" Number(38,0), "CHANGE_TIMESTAMP" Number(38,0), "ARC_CREATE_DATE" TIMESTAMP(3), "ARC_CREATE_USER" NUMBER(38,0), "WHY_IN_ARC" CHAR(1 CHAR)) -- 'U' = updated or 'D' = deleted
Boolean representation	Mandatory	Use a DBMS specific boolean type or numeric values 0/1 or text values T/F.
Views	Mandatory	Name the views V_*
Views for external systems	Mandatory	If some external systems need to work with your database, do not allow them to work directly with the tables, but always create views for them. It gives you a chance to restructure your database later and just change the views for the external systems. Otherwise the future restructuring of your database would lead to a necessity to change (and pay) changes of the external systems.
Stored procedures	Mandatory	Name the stored procedures up_* (up = user procedure, because of MSSQL performance problems with the names sp_* we use generally in all DBMSs up_)
Stored functions	Mandatory	Name the stored functions uf_* (uf = user function)
CODES, STATES, …	Recommended	Let's imagine - we have a field REGISTRATION_STATE with three possible values: Candidate, Member, Deactivated - we know it won't be necessary to define the registration states dynamically - we know the set of the three possible values is stable, we do not see any necessity to define additional possible values In such a case we usually do not create a special table for the registration states, but create just a field in the corresponding table (e.g. PERSON.REGISTRATION_STATE). We ensure via CONSTRAINT on DB level that there are allowed only values: 'C', -- for Candidate 'M', -- for Member 'D' -- for Deactivated Such codes are usually of the type char(1) and the chosen values (e.g. 'C') correspond to the meaning they represent (e.g. Candidate).
PROGRAM_CODE	Mandatory	Let's imagine there are ten records in some base table (e.g. a table with some states) and there is connected a special application functionality with few certain records. Do not hardcode in the application the IDs of these records, but create a special field PROGRAM_CODE and write there a special text value (unique in the given table). Then it is clear to everybody the application contains a special functionality for the given record.

Indexes and statistics	Recommended	Ensure a regular automatic actualization of the indexes and statistics (e.g. via “MSSQL jobs”).
Automatic backup	Recommended	Ensure a regular automatic database backuping (e.g. via “MSSQL jobs”).
Indexes / performance	Recommended	Create indexes for the expected DML-statements. It is necessary to check the performance of the real application and prepare indexes according to the real statements. Use e.g. profiler and index tuning wizard for that.
Document the DB structures	Mandatory	Write a detail comment to each table, view, column, stored procedure, trigger. A missing or not clear comment is a common reason of the misunderstandings. When the DBMS supports it we create the comments also in DB. Exception: It is not necessary to write comment to each archive table and its fields, because it has “same/similar” structure as the original one. Exception: It is not necessary to document the 5 technical fields, because they have a same meaning in the each table.
Sequences, identity fields, …	Mandatory	Select a DBMS specific solution how to assign the next IDs to the records. Use identity fields for MSSQL, sequences for Oracle, … But never use the algorithm (MAX(ID_TBL) from TBL) + 1. It is not safe when more people are working at the same time and it ignores the fact that some records may be deleted (they are logged in the archive table ARC_TBL).