DESIGN OF RELATIONAL DATABASE

Roman Danel

Using **structural analysis**, basic components for designing the database model are:

- ERD (Entity Relationship Diagram) – describes database structure and integrity constraints
- DFD (Data Flow Diagram) – gives functional model – describes data flow and functions
- STD (State Diagram) – describes state changes in time
- DD (Data Dictionary)

ERD (Entity Relationship Diagram)

ERD describes in graphical form the structure of the database.

The aim of ERD is **modeling data stored into the database and analyse their relationship**.

Logical structure of a database consists from:

- **Entity** – object where data is stored (in database a table)
- **Attribute** – fundamental element of data that characterizes the entity (in database a column)
- **Relation** – relationship between the entities (two tables)
- **Cardinality** – type of relationship between entities: 1:1, 1:N, N:1, M:N

Notes

Relationship is the information that the system needs to remember, it can not be inferred.

Note that there is a difference between ERD (designed by Chen) and relational data model (theory by Codd). ERD does not plan physical tables and rows, that's the ERD implementation using RDM (Relational Data Model). That's the difference between the terms “Relationship” and “Relation”.

DFD (Data Flow Diagram)

The aim of DFD is modelling of **data or control flows** in the information system (in graphical form).

DFD diagrams describe functionality of the system.

DeMarco's notation is the most commonly used diagram within DFD.

DFD consists of the following elements: "process", "database", "input/output" and "flow".

DFD diagrams have hierarchical structure. **Context diagram** is at the highest level – the system is presented as object with links to the neighbourhood. At the other levels of DFD hierarchy there are descriptions of the individual processes and data or control flows.

DFD decomposition to lower levels up to the level of elementary functions:

- There shouldn't be a process that has no entries yet produces data streams
- There shouldn't be a process that consumes data only and has no output

STD (State Transition Diagram)

STD contains a sequence of states in which the system can find itself and under what conditions it may change its state. It models the system's changes over a period of time.

- It is important in terms of understanding the logic of the system
- States are static, change of state is often a consequence of an event.
- It can be shown by state diagram or flowcharts
- It can be hierarchical

DD (Data Dictionary)

A data dictionary is used to describe formalized system data from the user's perspective.

DD works with a **metadata** (data describes user data):

- Metadata are managed by database system
- Read-only access

Metadata can, for example, be stored in a table that contains definitions of user data tables (table column names, their data type, size, etc).

SQL - Structured Query Language

Current relational databases support standards of the SQL programming language which is key for database design and data manipulation.

SQL is a declarative **programming language** whose primary purpose is to enable working with data in a relational database.

Note:

The term "declarative language" means that the programmer declares what he wants to do and not how it should be done.

SQL commands can be divided into two groups:

- **DDL - Data Definition Language** (CREATE, DROP, GRANT)
- **DML - Data Manipulation Language** (SELECT, INSERT, UPDATE, DELETE)

SQL is standardized; there are several SQL standards such as:

- SQL86
- SQL89 - included referential integrity
- SQL92
- SQL99 – extended with OOP (object oriented programming)

DATABASE NORMALIZATION

The aim of normalization is to achieve ideal database data structure. Data should be stored in the simplest form and without a redundancy (recurrence the same information).

The so-called „normal forms“ of database are:

- **1st Normal Form** – record does not contain any recurring item
- **2nd Normal Form** – record has only PK (Primary Key, for details see next paragraph “Integrity”) and non-key fields are dependent on the entire PK
- **3rd Normal Form** – excludes transitive (delegated) dependence - the table does not include records that are not part of the key (i. e. the content of the columns are simple, indivisible information)
- **BCNF (Boyce-Codd Normal Form)**

Although there are fourth and fifth Normal Forms, they are not widely used. The most often used one is the 3rd NF.

Bad design of relational model results in:

- Data redundancy

- Inability to convey certain information

Denormalization

• Maximum normalization is not always ideal - too meticulous standardization can reduce the processing speed

• Denormalization - aggregated data values for canned reports lead to faster outputs

INTEGRITY

- **Entity integrity**
  - Realized by **Primary Key** – unique identification of entities (without NULL) ; every row has a unique value
- **Referential integrity**
  - Realized by **Foreign Key** – column value that refers to the value in the column of another table. The foreign key enables links in the relational database.

Referential integrity:

- **Restrictive**
  Parent record cannot be deleted if there is a link to it from a child record.
- **Set null**
  If the parent record is deleted, the child record sets the value of the foreign key to NULL.
- **Cascade**
  If the parent record is deleted, all child records that refer to it are deleted too.

TRANSACTIONS

As mentioned in the "Transaction Processing System" chapter, the basic idea of transactional systems is the transaction - a sequence of operations that need to be done to reach the goal while the database remains consistent.

The transaction is indivisible, must be executed either as a whole or, in the case of failure, it must cancel the results of the transaction and return the system to its original state.

Consistency of the database may be violated e. g. with parallel processing of multiple SQL commands that operate over the same data.

Transaction attributes:

A C I D = Atomicity, Consistence, Isolation, Durability

- **Atomicity** – transaction is atomic, i. e. indivisible, it is understood as a whole.
- **Consistency** - the result of the completed transaction must be the data in a consistent state (i. e. it must comply with all integrity and referential constraints)
- **Isolation** – data changed by transactions are invisible to other users until the transaction is completed
- **Durability** – result of the transaction must be permanent (the data changed as the result of that transaction have to be physically recorded on a disk, not only in the database engine cache)

The reason for introducing transactions was to troubleshoot multi-user access to databases and to prevent users from mutually overwriting data.

Closing of the transaction is possible by two ways: **COMMIT** (validation) or **ROLLBACK** (cancel the transaction or return the system to the state before the transaction was started).

RELATION BETWEEN THE DATABASE TABLES - CONCLUSION

- The database contains a number of tables
- All tables are not necessarily linked
- Related tables produce "schemes"
- A database can contain several schemes

Note:
In the era of file-based databases such as DBase or FoxPro, a single table corresponded to a single database file.

INDEXES

Indexes are defined on a column or group of columns in the table and are used to accelerate searching and reading data from a table. If data needs to be found in a column that is part of an index, the database engine doesn't need to retrieve the entire table (sequential scan) but only loads the data page according to the information stored in the index, which leads to faster response during the query.

INTERFACES TO ACCESS THE DATA FROM DATABASE SYSTEMS

Data stored in a database system are accessed by the database engine interface. In addition to interfaces provided by database system vendors there are also standardized interfaces. The most common data interfaces used in information systems are:

ODBC	Open Database Connectivity
JDBC	Java Database Connectivity
ADO	ActiveX Data Objects (from Microsoft)
ADO.NET	ADO for .NET framework