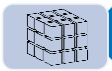
Course Outline

Architecture of Database Systems

- Towards a relational DBS architecture
- Evolution of the layer model
- The 5 Layer Model
- Properties of transactions

Transaction Management

Modern Database Technology



Data Warehouses and OLAP

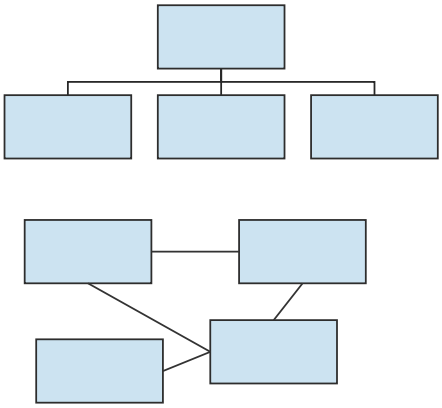
Data Mining

Big Data Analytics

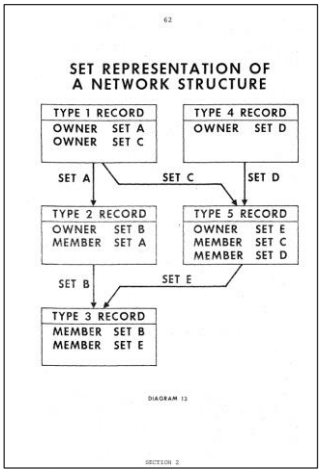
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Before The Relational Model, There Were Other Abstraction Models for Database Systems

- Hierarchical Model



- Network Model



Data base task group report to the CODASYL programming language committee, April 1971

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

8

Towards a relational DBS architecture

Hierarchical Model:

- Each record has only one parent record
- A record is a collection of fields where each field contains one value
- The fields of a record are defined by its type
- Used only in few database systems today, e.g. in IBM Information management System

Network Model:

- Proposed by the Data Base Task Group of the CODASYL programming language committee as the most general form of a data structure → No limitation to the links between records
- Enables complex data structures but only simple operations
- Widely replaced by the relational model

The Relational Model

- Developed by Codd in 1970
- Created to be simple
- Became more popular than the network model with increased computing power
- Postulates independence of the language and implementation

How can this idea be transformed into a Database System?

Information Retrieval P. BAXENDALE, Editor

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on n-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

KEY WORDS AND PHRASES: data bank, data base, data structure, data organization, hierarchies of data, networks of data, relations, derivability, redundancy, consistency, composition, join, retrieval language, predicate calculus, security, data integrity
CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22, 4.29

1. Relational Model and Normal Form

1.1. Introduction

This paper is concerned with the application of elementary relation theory to systems which provide shared access to large banks of formatted data. Except for a paper by Childs [1], the principal application of relations to data systems has been to deductive question-answering systems. Levein and Maron [2] provide numerous references to work in this area.

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for noninferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted data systems, and also the relative merits (from a logical standpoint) of competing representations of data within a single system. Examples of this clearer perspective are cited in various parts of this paper. Implementations of systems to support the relational model are not discussed.

1.2. Data Dependencies in Present Systems

The provision of data description tables in recently developed information systems represents a major advance toward the goal of data independence [5, 6, 7]. Such tables facilitate changing certain characteristics of the data representation stored in a data bank. However, the variety of data representation characteristics which can be changed *without logically impairing some application programs* is still quite limited. Further, the model of data with which users interact is still cluttered with representational properties, particularly in regard to the representation of collections of data (as opposed to individual items). Three of the principal kinds of data dependencies which still need to be removed are: ordering dependence, indexing dependence, and access path dependence. In some systems these dependencies are not clearly separable from one another.

1.2.1. *Ordering Dependence.* Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing systems which either require or permit data elements to be stored in at least one total ordering which is

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Codd, E.F. (1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM 13 (6): 377–387.

- In the 70s, implementations of the relational model were too slow for productive use
- Computing power grew rapidly
- Now the relational model is the standard abstraction model for database systems

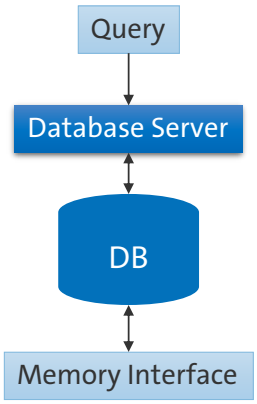
- Declarative queries, the use of values, and set orientation make it easy to use compared to the network model which uses pointers and is record oriented

How are Relational Database Systems Built?

```
SELECT s.firstname, s.lastname, COUNT(l.name)
FROM Student s
INNER JOIN Program p ON s.programId = p.id
INNER JOIN Attendance a ON a.studentId = s.studentId
INNER JOIN Lecture l ON a.lectureId = l.id
GROUP BY s.firstname, s.lastname WHERE p.name='DSE'
```

```
byte[] b = read(File f, int pos, int length)
```

The Monolithic Approach

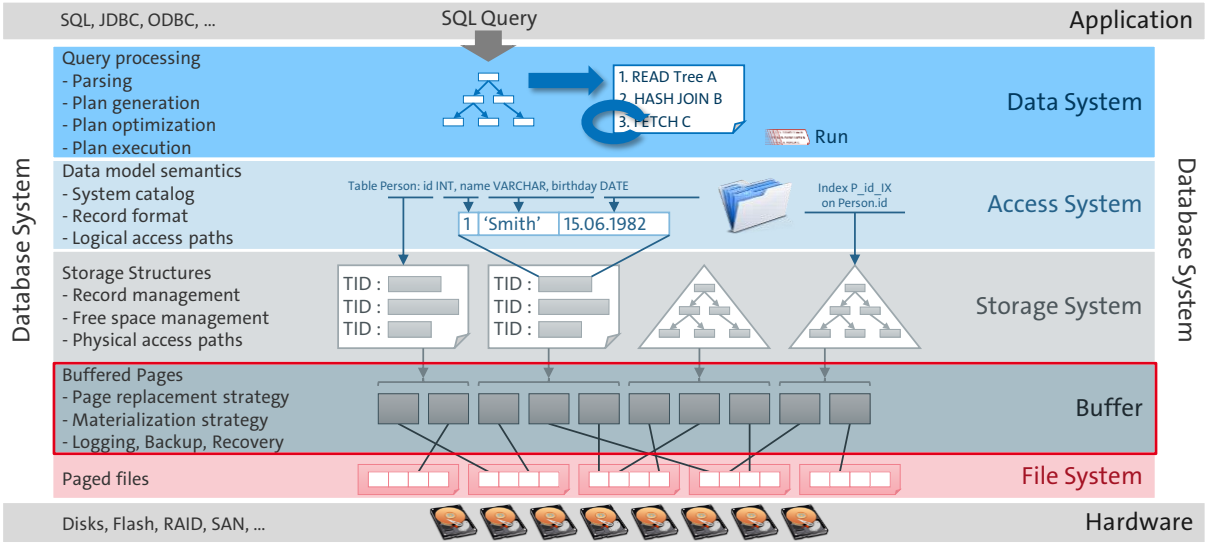


Evolution of Databases introduces more challenges

- New storage structures and access methods
- Changes of storage media
- Additional object types
- Integrity constraints
- Changing interfaces and application programs

Encapsulation via a hierarchical structure

The 5 Layer Model



The 5 Layer Model

Additional literature: Härder, Theo. "DBMS Architecture–the Layer Model and its Evolution." *Datenbank-Spektrum* 13 (2005): 45-57.

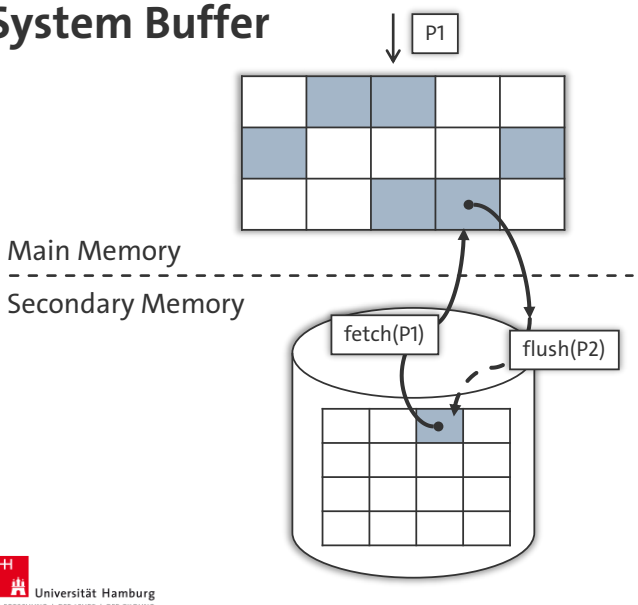
Received requests from upper layer (examples):
- Data System: Select * from mytable;
- Access System: FIND NEXT/STORE record
- Storage System: insert into B-Tree, store internal record
- Buffer: fetch page
- File System: Read/write block

Managed objects:
- Data System: Relations, Views
- Access System: External (logical) records, Index Structures
- Storage System: Internal (physical) records, Hash tables, Trees
- Buffer: Segments, Pages
- File System: Files, Blocks

Layers are ideally independent of each other → For performance reasons they are sometimes merged in a DBS

The following slides show examples of the work of each layer. They are not a complete representation of the layer's tasks.

System Buffer



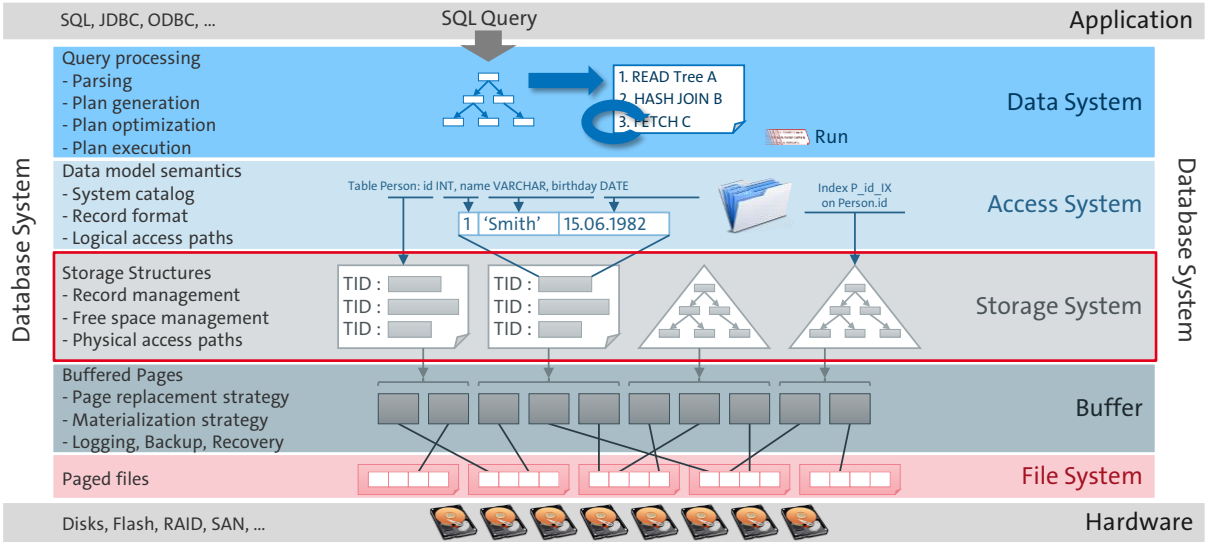
Calls of the System Buffer

- Provide page (logical reference), might require to replace another page
- FIX – Fix page in the buffer, such that the content can be accessed, usually done when a page is provided
- UNFIX – Release a FIX, such that page can be replaced
- Revision mark – A note that the content has been changed and needs to be physically written to disc when it is replaced !!!The note must be written before applying the change!!!
- Write – Write the buffer to main memory/disc

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

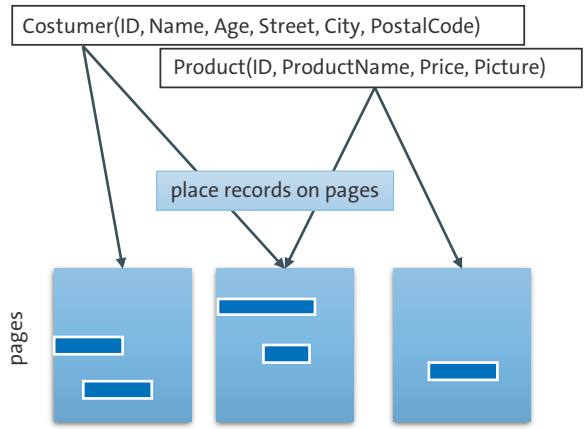
- Read and write operations use the buffer of a DBS
- Buffer can only hold a fraction of the whole dataset
  → Page replacement strategies are used to manage buffer (FIFI, LIFO, LRU, LFU)
  → Ring buffers exist but are usually applied for stream processing
- Buffer consists of page structured segments and buffer control block (holds information necessary for managing buffer)
- Special feature of DB buffer in contrast to general buffer management (e.g. by the operating system): Application knowledge can be used for buffer management

The 5 Layer Model



The 5 Layer Model

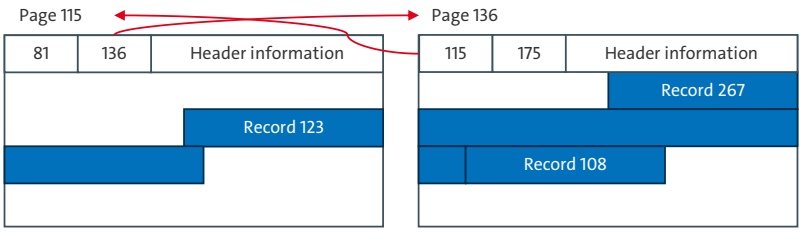
Record Management



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

- Representation of complete records on pages
- Pages have a fixed length, records can have a variable length
- Addressing of Records
    - Allocation Table
    - TID-concept (Tuple Identifier)
- Heap management

Organization of Pages



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

The 5 Layer Model

Addresses of records are created when they are inserted. They provide a way to access the records later.

Concatenation

Pages are linked together by double linked lists
Recording of free pages: heap management

Page Header

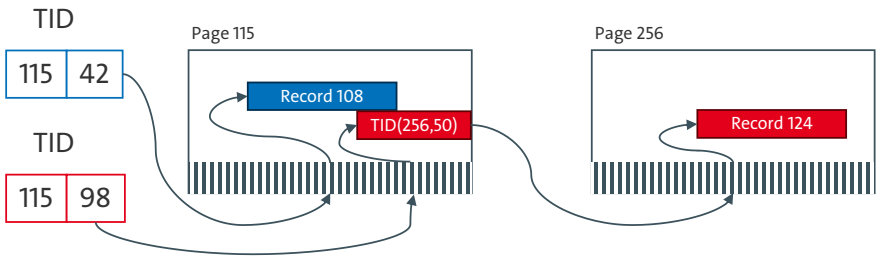
Information about previous and and following page
Optionally also number of the page itself
Information about the type of record (Table Directory)
Information about free space

Issues and challenges

- Distinct addressing of records for the entire lifetime of the record
- Support of data migration
- (Runtime) stability against relocation within a page
- Fast access, access as direct as possible
- No necessity for frequent reorganizations

TIDs

- Address is the tuple (Page ID, index in this page)
- Array in page holds the position of the records
- Migration possible without change of TID → Required when a record in a page becomes too big



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

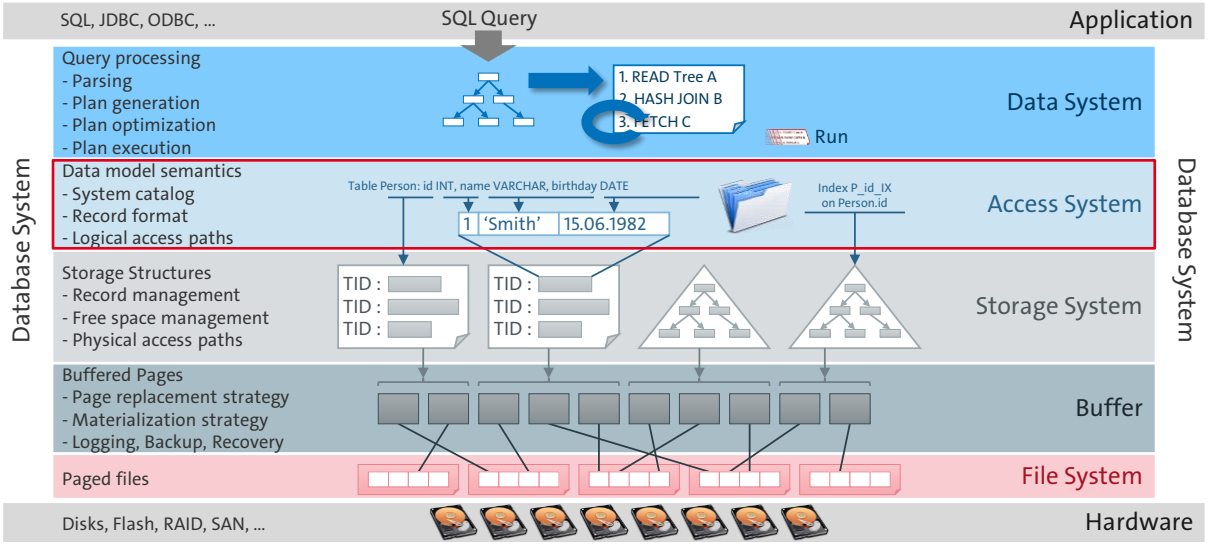
Deleting a record:

- The entry in the page array is marked as invalid
- All other records on the same page can be moved to maximize the free continuous space → only the positions are changed, not their index in the page array
- This way, record addresses are not changed, the same TIDs can still be used

Changing a record:

- **Case 1:** Record becomes smaller → all records are moved wihting the same page, positions in the page array are changed (same as with a deleted record)
- **Case 2:** Record becomes larger and space on the page is sufficient to store it → see case 1
- **Case 3:** Record becomes larger and space on the page is not sufficient to store it → Record is moved to another page and TID is stored on the original page (see Figure), If record is moved again later, TID in original page is changed again → only one additional reference necessary even if record is changed multiple times

The 5 Layer Model

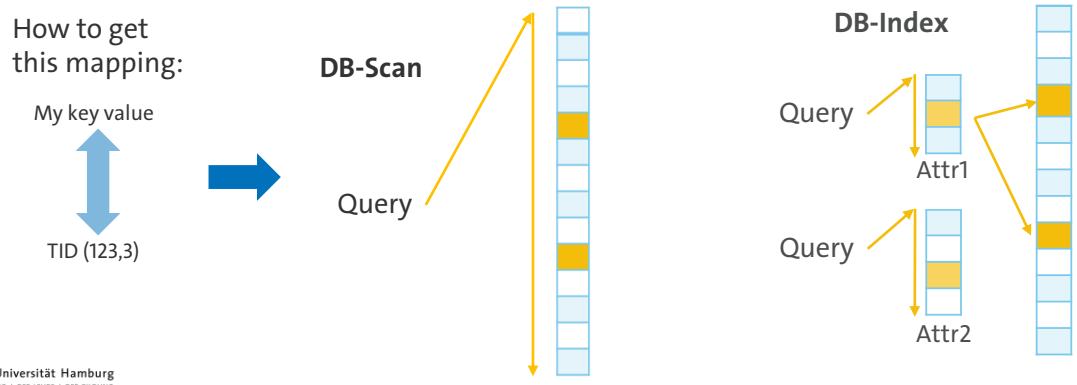


The 5 Layer Model

Use of Indexes

- To avoid full scans, indexes are used

  → Extra step to avoid unnecessary memory access and comparisons



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Types of access:

- Sequential access (scan) → not sufficiently fast for large record types or small result sets
- Sequenial access on a sorted attribute
- Direct access via primary key, foreign key(s), composite keys, or search terms
- Navigation from a record to a set of associated records

Requirements:

- Efficient way of finding records, i.e. as direct as possible
- Avoiding sequential search of a dataset
- Facilitate access control by predefined access paths (constraints)
- Keep topological relations

B-Tree revisited

https://www.cs.usfca.edu/~galles/visualization/BTree.html

B-Trees were covered in DB Grundlagen (not only at UHH, but also at other universities)

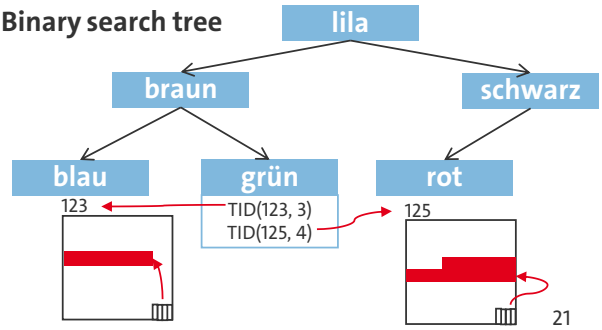
Classification of Indexes

- Primary or Secondary Index (additional access path)
- Simple index or multilevel index
- Storage structure:
  - Tree-structured, e.g. B-Tree
  - Sequential, e.g. Sequential lists
  - Scattered, e.g. Hash regions
- Access method:
  - Key comparison
  - Key transformation, e.g. hashing

Sequential List

blau	
braun	
grün	TID(123, 3) TID(125, 4)
lila	
rot	
schwarz	

Binary search tree



- Keys do not always fit into main memory, i. e. it can be useful to chose an index that minimalizes disc accesses (e.g. B-Trees and its variants)
- Extension of binary trees: trees with multiple children per node → B-Trees were designed for use in DB systems
- Features: Dynamic reorganization by splitting and merging of pages, direct key access
- Extensions, e.g. B+ tree, B*-tree add more features, e.g. sorted sequential access

Creating an Index in SQL

- `CREATE INDEX my_index ON myTable(myAttr);`
- `CREATE INDEX my_index ON myTable(myAttr,myAttr2);`
- `CREATE UNIQUE INDEX my_index ON myTable (myAttr) INCLUDE (myAttr2)`
- `CREATE UNIQUE INDEX index ON myTable (myAttr) [DIS]ALLOW REVERSE SCANS`
- Some Systems allow computed indexes:
  - `CREATE INDEX my_index ON myTable (a + b * (c - 1), a, b);`

- The order of the attributes in a multilevel index is not commutative
  → An existing index on (myAttr1,myAttr2) can be used if both attributes are queried or only myAttr1. It cannot be used (efficiently) if only my Attr2 is queried.
- UNIQUE does not allow duplicate values
- INCLUDE includes the attribute in the key, but does not use it for the creation of the index, i.e. in a search tree it is only part of the leaf nodes but not necessary for the other nodes
  → Useful if an attribute is part of the column list in a query but not of the WHERE clause
- [DIS]ALLOW REVERSE SCANS
  → Indexes are usually scanned in the order they were created in. This can be used to allow the opposite behavior. The default depends on the DB system.

- Not all of these are available on every system. Always check the docs!