Chapter 4

Databases

A computer does more than just computation. In particular, a computer can also store and retrieve large amounts of data efficiently. In this chapter, we want to understand some of the key ingredients of databases.

4.1 Dictionary

We manage a library and want to be able to quickly tell whether we carry a given book or not. We need the capability to insert, delete, and search books.

Definition 4.1 (Dictionary). A dictionary is a data structure that manages a set of objects. Each object is uniquely identified by its key. The relevant operations are

- search: find an object with a given key
- insert: put an object into the set
- delete: remove an object from the set

Remarks:

- There are alternative names for dictionary, e.g. key-value store, associative array, map, or just set.
- If the dictionary only offers search, it is called static; if it also offers insert and delete, it is dynamic.
- When discussing the algorithms, we will often ignore that we actually have a set of objects, each of which is identified by a unique key, and just talk about the set of keys. With regard to the library example, books are globally uniquely identified by a key called ISBN. Whenever we say we insert/delete/search a key, we can just drag the key's object along.
- The classic data structure for dictionaries is a binary search tree.

Algorithm 4.3: Search Tree: Search

```python
def search(self, key):
    # self is current node, initially root
    if key < self.key:
        if self.left is None:
            return None
        else:
            return self.left.search(key)
    elif key > self.key:
        if self.right is None:
            return None
        else:
            return self.right.search(key)
    return self.val
```

Remarks:

- The cost of searching in a binary search tree is proportional to the depth of the key, which is the distance between the node with the key and the root.
- There are search trees called splay trees that keep frequently searched keys close to the root for quick access. On the other hand, there may be rarely accessed keys deep in a splay tree.
- Using balanced search trees, we can maintain a dictionary with worst-case logarithmic depth for all keys, and thus worst-case logarithmic cost per insert/delete/search operation.
- Is there a way to build a dictionary with less than logarithmic cost and with keys that cannot be ordered?

4.2 Hashing

In this section we use hashing to implement an efficient dictionary.

Definition 4.4 (Universe, Key Set, Hash Table, Buckets). We consider a universe $U$ containing all possible keys. We want to maintain a subset of this universe, the key set $N \subseteq U$ with $|N| \ll n$, where $|N| \ll |U|$. We will use a hash table $M$, i.e. an array $M$ with $m$ buckets $M[0], M[1], \ldots, M[m-1]$.

Remarks:

- The standard library of almost every widely used programming language provides hash tables, sometimes by another name. In C++, they are called unordered_map, in Python dictionary, in Java HashMap.
4.2. HASHING

- The translation from virtual memory to physical memory uses a piece of hardware called translation lookaside buffer (TLB), which is a hardware implementation of a hash table. It has a fixed size and acts like a cache for frequently looked up virtual addresses.
- Compilers make use of hash tables to manage the symbol table.

**Definition 4.5 (Hash Function).** Given a universe \( U \) and a hash table \( M \), a hash function \( h : U \rightarrow M \). Given some key \( k \in U \), we call \( h(k) \) the hash of \( k \).

**Remarks:**
- A hash function should be fast to compute and distribute hashes nicely, e.g., \( h(k) = k \mod m \) for a key \( k \in U \); in contrast to Chapter 3, we do not care whether a hash function is one-way.
- If we use \( \text{BSHM} \mod m \) as our library hash function, can we insert/delete/search books in constant time?!
- What if two keys \( k \neq k' \) have \( h(k) = h(k') \)?

**Definition 4.6 (Collision).** Given a hash function \( h : U \rightarrow M \), two distinct keys \( k, k' \in U \) produce a collision if \( h(k) = h(k') \).

**Remarks:**
- Since keys may experience collisions, the key must be stored in the bucket.
- There are competing objectives we want to optimize for when hashing.
- On the one hand, we want to make the hash table small since we want to save memory. On the other hand, small tables will have more collisions. How likely is it to get a collision for a given entry? We need a good strategy for resolving collisions.

**Theorem 4.7 (Birthday Problem).** If we throw a fair \( m \)-sided dice \( n \leq m \) times, let \( D \) be the event that all throws show different numbers. Then \( D \) satisfies

\[
P(D) \leq \exp \left( -\frac{n(n-1)}{2m} \right)
\]

**Proof.** We have that

\[
P(D) = \frac{m}{m} \cdot \frac{m-1}{m} \cdot \ldots \cdot \frac{m-(n-1)}{m} = \prod_{i=0}^{n-1} \frac{m-i}{m} = \exp \left( \sum_{i=0}^{n-1} \ln \left( 1 - \frac{i}{m} \right) \right)
\]

We can use that \( \ln(1 + x) \leq x \) for all \( x > -1 \) and the monotonicity of \( e^x \):

\[
P(D) = \exp \left( \sum_{i=0}^{n-1} \ln \left( 1 - \frac{i}{m} \right) \right) \leq \exp \left( \sum_{i=0}^{n-1} \frac{1}{m} \right) = \exp \left( \frac{n(n-1)}{2m} \right)
\]

4.3. Key-Value Databases

**Definition 4.8 (Key-Value Database System).** The concept of dictionaries is used in key-value database systems. The server maintains the dictionary and clients can insert and query the stored data using the keys.

**Remarks:**
- Popular key-value databases are Redis and Memcached. They are often used for caching in web services. Dynamically generated documents or results of queries to other databases can be stored temporarily to allow fast access to often requested data.
- The data is often kept in main memory to speed up the access and only duplicated to disk to recover the database in case of a system failure.
- Depending on the used database, different data types can be stored in the value. This can be an integer, a string, or even an array.
- Document databases are an extension of simple key-value database systems. The value has to be in a format that the database understands, such as a JSON or XML document. These databases allow queries on the content of the documents. MongoDB and CouchDB are popular document databases.
4.4 Relational Databases

However, most databases offer queries beyond simple key searches. Questions like “What is the movie with the largest cast?” or “How many directors have directed more than ten movies?” should be answered without first writing a new program. Relational databases can store large amounts of structured data and answer possibly complex questions about it.

**Definition 4.9 (Table, Row, Column, Database).** A table consists of rows; so that each row (data record) contains the same fields, i.e., kinds of entries. When the rows of a table are written line by line, the fields form the columns of the table. Each column is referred to by a descriptive name, and is associated with the type of the respective field, e.g., integer, floating point, string, or a date. A database is a collection of tables.

**Remarks:**

- In the database context, tables are also called relations, because the entries in each row are related to each other, namely by belonging to the same row.

<table>
<thead>
<tr>
<th>title</th>
<th>director</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manos: The Hands of Fate</td>
<td>Harold P. Warren</td>
<td>1966</td>
</tr>
<tr>
<td>Raiders of the Lost Ark</td>
<td>Steven Spielberg</td>
<td>1981</td>
</tr>
<tr>
<td>War of the Worlds</td>
<td>Steven Spielberg</td>
<td>2005</td>
</tr>
<tr>
<td>12 Angry Men</td>
<td>Sidney Lumet</td>
<td>1957</td>
</tr>
</tbody>
</table>

Figure 4.10: A database containing a single table called “movies” storing the title, director, and year of release for each movie.

**Remarks:**

- Databases as we study them are accessed using the so-called structured query language (SQL). Thus they are referred to as SQL or relational databases.
- MySQL and PostgreSQL are two popular open source SQL database systems. SQL database systems typically run as a daemon process on some server. Client applications connect to the server and authenticate themselves via username and password. Therefore, multiple users accessing the same database may result in concurrency issues. Some form of concurrency control is necessary.
- Other database systems are tailored to single-user processing. They relieve developers from the burden of implementing efficient data structures for relational data. SQLite is one such example, and is used, e.g., in Firefox, Chrome, Android, Adobe Lightroom, and Windows 10.

4.5 SQL Basics

**Definition 4.11 (SQL Data Types).** SQL defines the following types of columns.

- **CHARACTER(m)** and **CHARACTER VARYING(m)** for fixed and variable-length strings of (maximum) length m.
- **BIT(m)** and **BIT VARYING(m)** for fixed and variable-length bit strings of (maximum) length m.
- **NUMERIC, DECIMAL, INTEGER, and SMALLINT** for fixed point and integer numbers.
- **FLOAT, REAL, and DOUBLE PRECISION** for floating point numbers.
- **DATE, TIME, and TIMESTAMPTZ** for points in time, or
- **INTERVAL** for ranges of time.

**Remarks:**

- The range of each type includes the special value NULL. Note that NULL is different from the string "NULL", the empty string, and from the number 0 (zero). NULL indicates that the row has no value for the corresponding field.
- Many database systems implement more types, e.g., geographic coordinates, IP addresses, geometric objects, or large integers.
- All SQL statements end with a semicolon. The SQL language is case insensitive, but by convention keywords are often typed in upper case.
- The SQL-92 specification is over 600 pages long, newer versions of the standard even longer. To add insult to injury there are lots of vendor specific "SQL dialects", i.e., modifications and extensions. However, the basic set of commands for creating, manipulating, and querying tables are largely the same across database implementations.

**CREATE DATABASE database-name:**

Additional parameters allow to set database-specific options, e.g., user-based permissions, or default character sets for text strings. How a database is opened depends on the implementation.

**CREATE TABLE table-name (field-name type, field-name type, ... );**

To enforce that all rows have a value for a particular field, one can add NOT NULL to the type when creating the table. Fields have a default value, which is NULL if not specified by adding DEFAULT value to the type description.

**Remarks:**

- There are also GUI and web-based client applications (that execute locally or on an http-server, respectively) and offer access to the database in a more intuitive manner than the classic command line tools. Examples for PostgreSQL are pgAdmin, DataGrip and DBBeaver.
- Such tools are especially helpful for creating the databases and tables and often support multiple database systems. They also feature importing data from various formats, e.g., CSV files, instead of using SQL statements to populate the tables.
INSERT INTO table-name (field-name, . . . ) VALUES (value, . . . );
Values must be listed in the same order as the corresponding field names.
When a field name (and thus its value) is omitted the field’s default value is assumed.
When the list of field names is omitted the field’s values must be listed in the same order that was used when creating the table.
To insert more than one row in one statement, multiple rows may be separated by commas.

SELECT * FROM movies;
SELECT * FROM movies WHERE director = 'Spielberg, Steven';
SELECT title FROM movies WHERE year BETWEEN 1990 AND 1999;
SELECT * FROM movies WHERE title IS NULL OR director IS NULL;
SELECT title, director FROM movies WHERE title LIKE '%Star%';

SELECT MIN(year) FROM movies;
SELECT AVG(year) FROM movies WHERE director='Lumet, Sidney';
SELECT COUNT(+) FROM movies;
SELECT COUNT(DISTINCT director) FROM movies;

SELECT field-name FROM table-name WHERE condition;
Lists all specified fields of all rows in the table that fulfill the condition.
The special field * lists all fields. The WHERE condition may be omitted
to list the whole table. A condition can include comparisons (<, >, =, >)
between fields constants. The special value NULL can be tested with IS
NULL. Conditions can be joined using parenthesis and logic operators like
AND, OR, and NOT. Strings can be matched with patterns using field-
name LIKE pattern. In the pattern, an underscore (_) matches a single
character, whereas % matches arbitrarily many.

SELECT field-name, . . . FROM table-name WHERE condition;
Lists all specified fields of all rows in the table that fulfill the condition.
In an aggregation, the keyword DISTINCT indicates that only distinct values
should be considered.

SELECT field-name, COUNT(field-name) FROM table-name GROUP BY field-name;
Aggregations may be partitioned using the group-by clause. Similar to
before, the query result can only include aggregates and fields by which
the result is partitioned.
Since WHERE clauses are applied before GROUP BY the result of aggre-
gations cannot appear in them. When the result should be conditioned
on the result of an aggregation, a HAVING clause can be used.

SELECT . . . ORDER BY field-name, . . .
After each field-name, the keyword ASC or DESC can be used to deter-
mine ascending or descending sorting order, respectively.

SELECT year, director, COUNT(title) FROM movies
GROUP BY director, year
ORDER BY year DESC, director ASC;

SELECT field-name-aggregate, . . . GROUP BY field-name,. . .
Aggregations may be partitioned using the group-by clause. Similar to
before, the query result can only include aggregates and fields by which
the result is partitioned.

UPDATE table-name SET field-name = value, . . . WHERE condition;
Updates the specified fields in all rows fulfilling the condition.

DELETE FROM table-name WHERE condition;
Removes all rows fulfilling the condition from the table.

4.6 Modeling
The way our example table from Figure 4.10 is designed results in lots of dupli-
cate data—the director’s name is stored anew for each row, and two directors
with the same name cannot be distinguished. The situation worsens when we
want to store the cast of each movie. In other words, the way we modeled our
data can be improved. Entity-Relationship (ER) diagrams are a tool to find
good representations for data.

Definition 4.16 (Entity-Relationship Diagram). Rectangles denote entities
(tables), and diamonds with edges to entities indicate relations between those
entities. On such an edge, the number 1 or the letter n denotes whether the
corresponding entity takes part once or arbitrarily many times in the relation.
Entities and relations can have attributes (columns) with a name, drawn as
eclipses. Indicated attributes are key attributes which must be unique for each
such entity.
4.6. MODELING

Figure 4.17: Model for a movie database. Movies and directors are in a 1-to-n relation: Each movie is directed by 1 director, and a director may work on many movies. Movies and actors are in a n-to-n relation, which has an additional attribute: An actor may appear in many movies, and each appearance is associated with a character in that movie, played by that actor.

Remarks:

• It is standard practice to assign a so-called key attribute, often named id, to every entity.

• What do ER diagrams have to do with SQL? Primarily, ER diagrams are for conceptually modeling the kind of data and relations one wishes to store. They can be translated into databases. Each entity corresponds to a table with the corresponding attributes as columns. An n-to-n relation is represented by a table with columns for each attribute, and a column for the key attribute of each entity in the relation.

• A close relative of the ER diagram is the Unified Modeling Language (UML). UML is used to represent the tables of a database (or classes of object oriented software) accurately, with detailed information, e.g. fields.

Figure 4.18: The actor table and a table capturing the acting relation.

Remarks:

• The same scheme can be used for 1-to-1 and 1-to-n relations. However, one may also include the relation in the table storing the entity on the 1-side.

4.7 Keys & Constraints

What is stopping us from inserting a row in the acting table that contains an actor id or a movie id that does not exist? Or from creating a director with a duplicate id?

Definition 4.20 (Key). In a table, a column (or set of columns) is a unique key if the corresponding values uniquely identify the rows within the table. The primary key of a table is a designated unique key. A foreign key is a column (or set of columns) that references the primary key of another table.

Remarks:

• SQL databases can automatically enforce these constraints. For example, a row containing a foreign key can only be inserted if it references an existing primary key. Vice versa, a row may only be removed if its primary key is not referenced by any foreign key.

```
ALTER TABLE table
ADD CONSTRAINT UNIQUE (field-name, . . .);

→ notebook
Any two rows must differ in at least one of the specified fields.

ALTER TABLE table
ADD PRIMARY KEY (field-name, . . .);

→ notebook
Sets the specified fields as the primary key for the table. Any two rows must differ in at least one of the specified fields. The entries in these fields must not be NULL.

ALTER TABLE left-table
ADD FOREIGN KEY (field-name, . . .)
REFERENCES right-table;

→ notebook
Ensures that the values in the specified fields in the left table are the primary key of a row in the right table.
```
4.8 Joins

How can we access the data, which is now scattered across multiple tables?

```sql
SELECT movie.title, director.name AS director, movie.year
FROM movie INNER JOIN director ON movie.director_id = director.id;
```

Listing 4.21: Example query that returns the table depicted in Figure 4.22.

### Remarks:
- In a query, one can create aliases for field and table names using the AS keyword, see Listing 4.21.
- The result of a JOIN clause can be ordered, fields can be aggregated and grouped, and conditions can be added using WHERE clauses.
- For example, we can combine joins and aggregations to answer our initial question of which movie has the largest cast.

```sql
SELECT movie.title, director.name AS director, movie.year
FROM movie RIGHT OUTER JOIN director ON movie.director_id = director.id;
```

Listing 4.24: Example query that returns the table depicted in Figure 4.25.

### Remarks:
- The query from Listing 4.23 uses a LIMIT clause to return only the ten first entries of the sorted results.
- An INNER JOIN where the condition is TRUE returns the Cartesian product of both tables. This special case can also be obtained with the CROSS JOIN clause.
- An inner join will only return those rows of one table that have a matching row (that satisfies the condition) in the other table. For example, in Listing 4.21, a director with id 5 would not appear in the result if there are no movies which have director id=5.
- If you want unmatched rows to appear in the result, you need to use an OUTER JOIN.

```sql
SELECT movie.title, COUNT(*) AS cast_size
FROM acting INNER JOIN movie ON acting.movie_id = movie.id
GROUP BY movie.id ORDER BY cast_size DESC LIMIT 10;
```

Listing 4.23: Finding the 10 movies with the largest cast.

### Remarks:
- The query from Listing 4.23 uses a LIMIT clause to return only the ten first entries of the sorted results.
- An INNER JOIN where the condition is TRUE returns the Cartesian product of both tables. This special case can also be obtained with the CROSS JOIN clause.
- An inner join will only return those rows of one table that have a matching row (that satisfies the condition) in the other table. For example, in Listing 4.21, a director with id 5 would not appear in the result if there are no movies which have director id=5.
- If you want unmatched rows to appear in the result, you need to use an OUTER JOIN.

### Remarks:
- In a query, one can create aliases for field and table names using the AS keyword, see Listing 4.21.
- The result of a JOIN clause can be ordered, fields can be aggregated and grouped, and conditions can be added using WHERE clauses.
- For example, we can combine joins and aggregations to answer our initial question of which movie has the largest cast.
OUTER JOIN also returns all rows from the left or right table that have no matching row on the opposite table, respectively. The fields in unmatched rows cannot be filled from the other table and are filled with NULL values. A FULL OUTER JOIN returns both of the above.

Remarks:
- A LEFT OUTER JOIN in Listing 4.24 would include the movies with no director instead of the directors who have not directed any movie.
- Queries may use more than one JOIN clause.

```sql
SELECT movie.title
FROM actor JOIN movie
ON actor.actor_id = movie.actor_id AND actor.name = 'Ford, Harrison'
RIGHT OUTER JOIN movie ON movie.movie_id = movie_id
WHERE actor.actor_id IS NULL;
```

Listing 4.26: Finding all movies that Harrison Ford did not appear in.

Remarks:
- The conditions for the first join in Listing 4.26 ensure that only movies with Harrison Ford are taken into account for the second OUTER JOIN. That second join in turn delivers all movies that cannot be matched, yielding a NULL entry for the actor_id for movies without Harrison Ford.

Chapter Notes

Dictionaries based on search trees are useful for providing additional operations such as nearest neighbor queries or range queries, where we want to find all keys in a certain range. Binary search trees were first published by three independent groups in 1960 and 1962 [for references, see Knuth [13]]. The first instance of a self-balancing search tree that guarantees logarithmic cost for insert/search/delete in the AVL-tree, named after its inventors Adelson-Velskii and Landis [1]. For multidimensional keys, e.g. geographic data or images, there are specialized tree structures such as k-d-trees [2] or RR-trees [4].

Hashing has a long history and was initially used and validated based on empirical results. One of the first publications was Peterson’s 1957 article [14] where he defined an idealized version of probing and empirically analyzed linear probing. Universal hashing was introduced two decades later by Carter and Wegman in 1979 [5]. Perfect static hashing was invented in 1984 by Fredman et al. [16] and is sometimes also referred to as FKS hashing after its inventors. Its dynamization by Dietzfelbinger et al. took another decade until 1994 [9].

In 1970, Edgar F. Codd proposed the relational database model [8] while working at IBM research. Later in the 70s, another group at IBM developed SQL’s predecessor SEQUEL (Structured English Query Language) [6]. After being renamed SQL due to trademark issues, it was standardized by the ISO in 1987 and later revised [12]. Other companies started developing relational database systems, and nowadays there are many SQL databases implementing different feature sets to choose from.

Around the same time, ER diagrams were conceived as a modeling tool [3, 7]. The Unified Modeling Language (UML), first standardized by the ISO in 1995 [12] and revised in 2012, also includes diagrams that model databases.

This chapter was written in collaboration with Georg Bachmeier and Jochen Seidel.

Bibliography
