Physical database design

- The learning objectives for this week are:
    - Knowing the purpose of physical database design
    - Knowing the basic types of integrity constrains
    - Knowing how to create a basic set of tables with integrity constraints in SQL Server
    - Knowing the advantages and disadvantages of database indexes
    - Knowing how to create indexes on database tables in SQL Server
    - Knowing what is meant by database performance and database security

Physical database design

- Once we have designed a logical database schema based on a specific data model, we can start designing the *physical implementation of the database*
- The *physical database design* is concerned with the *target DBMS product* and all *physical-level details*
- The target DMBS product could be for example SQL Server, PostgreSQL, or MySQL
- The designer should be fully aware of the functionality of the target DBMS and must know how the computer system hosting the DBMS operates
- For example in RDMSs, the common SQL syntax and functionality is the same, but there are still important differences
- In practice, physical design *must be tailored to a particular DBMS*

Creating tables

Database integrity

Database security

Database performance

Database indexes

- Suppose that you are librarian in a library that has hundreds of books and you need to find the book by title "Dune"
- If you don't have any information on the locations of different books, you have to go through *every single book* to find the book you are looking for
- If you have an ordered list of book titles and their locations you can quickly skip to letter "D" and find the location of the book you are looking for quite quickly
- In a database such "ordered list of book titles and their locations" resembles an *index*

Database indexes

- *Indexes* are critical for database performance. A basic fact of database processing is that *disk access* is extremely slow compared to *main memory access*
- An index is implemented with a compact data structure that at least partly fit into the main memory
- This makes it possible for the DBMS to find data in the database with fewer disk accesses
- Indexes are created on columns or a group of columns to speed up queries that filter rows based on them
- If we would have a database table `Book` and column `title`, we could create an index on the `title` column to speed up queries that *search for rows based on the column's value*, such as:

```sql
SELECT title FROM Book
WHERE title = 'Dune'
```

Database indexes

- *Without an index*, the DBMS has to go through each row in the table, which requires many disk accesses (slow)
- *With an index*, the DBMS just has to find the corresponding *index entry* in the index data structure. In most cases the index entry can be found in the main memory (fast)
- The index entry will point to data on the disk, which minimizes the required disk accesses

Creating an index

- We can create an index in the SQL Server using the `CREATE INDEX` statement:

```
CREATE INDEX employee_deptno_index ON Employee(deptno)
```

- In this example, we create an index with name `employee_deptno_index` for the `Employee` table's `deptno` column
- We can delete an index with a specific name in a table using the `DROP INDEX` statement:

```
DROP INDEX employee_deptno_index ON Employee
```

Advantages and disadvantages of indexes

- If indexes are the magical way to search for rows based on a certain column value faster, then *why not add a index for every column on table?*
- Indexes have many *advantages*, such as:
  - Minimise the number of disk accesses needed to locate data in the database
  - Enforce `PRIMARY KEY` and `UNIQUE` constraints. If a suitable index exists, then the DBMS
    can use the index to find out if a duplicate value already exists for the key
  - Enforce `FOREIGN KEY` constraints. If a suitable index exists, then the DBMS can use the index to find out if any child row exists for a row that is being deleted
  - Accelerate `JOIN` operations
  - Avoid sorts for `DISTINCT`, `GROUP BY`, `ORDER BY` and `UNION` clauses

Advantages and disadvantages of indexes

- But there are also *disadvantages*, such as:
  - More disk space is needed for indexes. Indexes need some maintenance, too
  - The DBMS has to update all indexes on the table when a new row is inserted or a new
    row is deleted
  - When an existing row is updated the DBMS has to update related indexes accordingly. That is, if there are too many indexes on a table they might start decreasing performance on inserts, updates, and deletes

When to use indexes?

- In general, indexes are important for tables with a *large number of rows* and *frequently repeated queries*
- If the number of rows in a table is very small, then an index does not improve performance
- Commonly indexes are considered when a certain database query execution in the system becomes too slow
- In such case, a database designer will analyze the database query and creates indexes suitable for speeding it up