2.1) Allgemeines

2.1.1) Definitionen

2.1.1.1) Datenbanksystem:

Ein Datenbanksystem ist ein computergestütztes System bestehend aus einer Datenbasis zur Beschreibung eines Ausschnitts der realen Welt sowie Programmen zum geregelten Zugriff auf die Datenbasis.

2.1.1.2) Datenbankverwaltungssystem (DBMS-data base management system)

Ist jener Teil der Software die zwischen den eigentlichen Daten und den Benutzern der Daten liegt und alle Anfragen der Benutzer verarbeitet. Sie stellt jene Einrichtung zur Verfügung die notwendig sind um neue Daten anzulegen, zu löschen, abzufragen und zu verändern

2.1.2) Motivation

Die ersten Computer unterstützten Informationssysteme, wurden in Form von Einzellösungen, d.h. durch einzelne Anwendungsprogramme mit privaten Dateien realisiert. Diese Programme verwendeten unmittelbar das zugrunde liegende Dateisystem auf den jeweiligen Rechner. Gleichartige Daten wurden in separaten Dateien gespeichert, die selbst wieder aus einzelnen Datensätzen bestanden.

Produktion
Angestellte
Teile

Verkauf
Kunden
Teile

Fakturierung
Kunden
Teile

Personalabteilung
Kunden
Gehalt

Diese Systeme waren schwer wartbar da mehrfach verwendete Daten auch mehrfach gespeichert wurden, deshalb entstand die integrierte Datenverarbeitung bei der Dateien in mehreren Anwendungsprogrammen verwendet werden. Doch auch die separate Abspeicherung von teilweise in Beziehung stehenden Daten würde zu schwerwiegenden Problemen führen:

- **Redundanz** - Dieselben Informationen werden doppelt gespeichert.
- **Inkonsistenz** - Dieselben Informationen werden in unterschiedlichen Versionen gespeichert.
- **Integritätsverletzung** - Die Einhaltung komplexer Intigritätsbedingungen fällt schwer.
- **Verknüpfungseinschränkung** - Logisch verwandte Daten sind schwer zu verknüpfen wenn sie in isolierten Dateien liegen.
- **Mehrbenutzerprobleme** - Gleichzeitiges Editieren der Datei führt zur Anomalie (lost update).
- **Verlust von Daten** - Außer einem kompletten Backup ist kein Recovery-Mechanismus vorhanden.
- **Sicherheitsprobleme** - Abgestufte Zugriffsrechte können nicht implementiert werden.
- **Hohe Entwicklungskosten** - Für jedes Anwendungsprogramm müssen die Fragen zur Dateiverwaltung erneut gelöst werden.

Heute werden Informationssysteme meist mit Hilfe von Datenbanksystemen realisiert.

Einerseits ermöglicht diese Trennung zwischen Anwendungsprogrammen und Daten die sogenannte **physische Datenunabhängigkeit**, d.h. Programme sind von den konkreten Speicher- und Zugriffsmethoden unabhängig. Andererseits stellen Datenbanksysteme ein Datenmodell, also eine Sprache zur Beschreibung von Datenstrukturen, zur Verfügung, die es ermöglicht einzelne Programme auf speziellen logischen Darstellungen der gespeicherten Datenbank arbeiten zu lassen (logische Unabhängigkeit).

2.1.3) Funktionalität von Datenbanksystemen

2.1.3.1) Persistente Datenhaltung

Ein DBMS muss Mechanismen zur Verfügung stellen, die eine **persistente Speicherung von Daten** garantieren, d.h. dass die Daten in der DB über die Ausführungszeit von Programmen hinaus erhalten bleiben. Die Daten werden üblicherweise auf einem Hintergrundspeicher (Sekundärspeicher - HDD) persistent gehalten. Nachdem Programme nur auf Daten im Hauptspeicher (Primärspeicher) direkt zugreifen können werden Ausschnitte der Datenbank zeitweise auch in einem Teil des Hauptspeichers, dem Datenbankpuffer, verwaltet.
Nachdem ein **Plattenzugriff sehr viel länger dauert als ein Hauptspeicherzugriff**, sind spezielle Techniken notwendig um unnötige Plattenzugriffe zu vermeiden.
Spezielle Puffersatzstrategien werden verwendet um bei Platzmangel im Datenbankpuffer zu entscheiden welche Blöcke wieder auf die Platte ausgelagert werden (z.B. **least recently used, least frequently used**). Aus Effizienzgründen gruppieren sogenannten Clustertechniken Datensätze so, dass jene Datensätze, auf die oft gemeinsam zugegriffen wird, physisch benachbart gespeichert werden. Weiters werden verschiedene Indextechniken verwendet um Daten auf einem Hintergrundspeicher rasch zu finden.

2.1.3.2) Recovery

DBMS unterstützen Änderungen in der Datenbank durch Transaktionen. Eine Transaktion ist eine Folge von Aktionen (Lese- und Schreibzugriffe auf Daten in der DB),die eine Datenbank von einem konsistenten Zustand in einen anderen konsistenten Zustand überführt. Die Recoveryeinheit eines **DBMS gewährleistet die Atomarität und die Dauerhaftigkeit (Persistenz) von**

Transaktionen trotz eventuell bei der Transaktion aufgetretenen Hard- oder Softwarefehlern.

Atomarität bedeutet, dass **entweder alle Aktionen** einer Transaktion ausgeführt werden **oder keine**.

Dauerhaftigkeit bedeutet dass **alle Effekte** einer einmal erfolgten Transaktion **trotz aufgetretenen Fehlern erhalten bleiben**.

Bsp. zu Atomarität: Überweisung eines Geldbetrages von einem Konto auf ein Sparbuch

```
- Kontostand lesen
- Kontostand schreiben
- Sparbuch lesen
- Sparbuch schreiben
```

Angenommen während der Überweisung tritt nach der Abbuchung aber noch vor der Aufbuchung ein Systemabsturz ein, so möchten die Kunden davon ausgehen können, dass sich nach einem Wideranlauf der gesamte Geldbetrag noch auf dem Konto befindet.

Bsp. zu Dauerhaftigkeit:
Angenommen wir zahlen einen Millionengewinn im Lotto auf unser Konto ein, dann stellt die Dauerhaftigkeit von Transaktionen sicher, dass der Gewinn auch nach einem Systemneustart immer noch auf dem Konto liegt. Für den Wiederanlauf verwenden die meisten DBS ein Log-Protokoll. In diesem Log-Protokoll werden der Start, das Ende und der Abbruch von Transaktionen verzeichnet, sowie die von Transaktionen durchgeführten Modifikationen von Datensätzen (Einfügen, Löschen und Ändern von Datensätzen). Zu jeder Änderung wird der alte Datensatz (before image) und der neue Datensatz (after image) im Log-Protokoll verzeichnet. Beim Wiederanlauf werden alle nicht beendeten Transaktionen unter Verwendung der before-images zurückgesetzt und alle bereits erfolgreich abgeschlossenen Transaktionen nachgeholt.

2.1.3.3) Concurrency Control

Die Concurrency Control - Einheit eines DBMS ermöglicht mehreren Benutzern eine Datenbank gemeinsam zur selben Zeit zu nutzen ohne ihre Konsistenz zu gefährden. Das traditionell verwendete **Korrektheitskriterium für parallele (oder verzahnte) Ausführung von Transaktionen im Mehrbenutzerbetrieb ist die Serialisierbarkeit**.
Die Serialisierbarkeit garantiert folgende Eigenschaft:
Das Ergebnis der beliebigen Parallelausführung mehrerer Transaktionen entspricht dem Ergebnis irgendeiner Hintereinander-Ausführung dieser Transaktion.

Bsp.: Angenommen wir wollen unsere Telefonrechnung (50€) bezahlen. Wir gehen zur Bank, wo die Abbuchung vom Konto durchgeführt wird. Diese Abbuchung wird nun gleichzeitig mit der Gehaltsbuchung (1000€) auf das Konto durchgeführt.

Angenommen der aktuelle Kontostand beträgt 2000€. Nachdem beide Transaktionen abgeschlossen wurden, ist der Kontostand auf 1950€. Eine Hintereinanderausführung hätte aber 2950€ ergeben, d.h. diese verzahnte Ausführung ist nicht serialisierbar und daher nicht korrekt.

Um Serialisierbarkeit von Transaktionen zu gewährleisten verwenden die meisten DBMS Sperrverfahren.

Dabei legt eine Transaktion auf Datenobjekte die sie schreiben oder lesen soll, eine **Sperre**. Besitzt eine Transaktion auf einem Datenobjekt eine Sperre und fordert eine andere Transaktion für dieses Datenobjekt ebenfalls eine Sperre an, so wird diese Sperre nur dann gewährt, wenn die neu angeforderte Sperre mit der bereits bestehenden Sperre verträglich ist. Ist sie es nicht, so muss die neue Transaktion **auf die Freigabe der bestehenden Sperre warten**.

Meist werden 2 Typen von Sperren verwendet:

- Geteilte Lese-Sperren
- Exklusive Schreib-Sperren

Lesesperren verschiedener Transaktionen für dasselbe Datenobjekt sind miteinander verträglich, eine Schreibsperre ist mit keiner Sperre anderer Transaktionen verträglich.

Das Sperren von Datenobjekten ist alleine jedoch nicht ausreichend um die Serialisierbarkeit paralleler Transaktionen zu gewährleisten. Es muss darüber hinaus ein **Sperrprotokoll** eingehalten werden. Das am meisten gebräuchliche Sperrprotokoll ist das **2-Phasen-Sperrverfahren**. Eine Transaktion erfüllt das 2-Phasen-Sperrverfahren, wenn es nach der 1. Freigabe einer Sperre keine neue Sperre mehr anfordert. Weiters garantiert die Concurrency Control-Einheit eines DBMS die Isolation von Transaktionen. **Isolation** bedeutet, dass **Effekte nach einer Transaktion erst nach ihrem erfolgreichem Abschluss für andere Transaktionen sichtbar** werden.

2.1.3.4) ACID - Atomicity Consistency Isolation Durability (zu d. Dt. AKID - Atomarität, Konstistenz, Isolation, Dauerhaftigkeit)

Die Eigenschaften des Transaktionskonzeptes werden unter der Abkürzung **ACID** zusammengefasst:

- **Atomicity:** Eine Transaktion stellt eine nicht weiter zerlegbare Einheit dar, mit dem Prinzip: **„ALLES oder NICHTS“**
- **Consistency:** Nach Abschluss der Transaktion liegt wieder ein konsistenter Zustand vor, während der Transaktion sind inkonsistente Zustände erlaubt
- **Isolation:** Nebenläufig ausgeführte Transaktionen dürfen sich nicht beeinflussen, d.h. jede Transaktion hat den Effekt, den sie verursacht hätte, als ob sie allein im System gewesen wäre.
- **Durability:** Die Wirkung einer erfolgreich abgeschlossenen Transaktion bleibt dauerhaft in der Datenbank, auch nach einem späteren Systemfehler.

2.1.3.5) Datenschutz

DBMS bieten die Möglichkeit für einzelne Benutzer oder Benutzergruppen den Zugriff auf Ausschnitte der Datenbank zu beschränken. Dabei kann hinsichtlich der Art des Zugriffs zwischen Lese-, Änderungs-, Einfüge- und Löschzugriffe unterschieden werden.

2.1.4) Architektur eines Datenbanksystems

Moderne Datenbanksysteme unterstützen alle die **ANSI-SPARC-Architektur**:

Die Architektur wurde 1975 vom Standards Planning and Requirements Committee (SPARC) des American National Standards Institute (ANSI) entwickelt und hat das Ziel, den Benutzer einer Datenbank vor nachteiligen Auswirkungen von Änderungen in der Datenbankstruktur zu schützen.

Die drei Ebenen sind:

- **Die externe Ebene**, die den Benutzern und Anwendungen individuelle Benutzersichten bereitstellt. Beispiele: Formulare, Masken-Layouts, Listen, Schnittstellen.
- **Die konzeptionelle Ebene**, in der beschrieben wird, welche Daten in der Datenbank gespeichert sind, sowie deren Beziehungen zueinander. Designziel ist hier eine vollständige und redundanzfreie Darstellung aller zu speichernden Informationen. Hier findet die Normalisierung des relationalen Datenbankschemas statt.
- **Die interne Ebene (auch physische Ebene)**, die die physische Sicht der Datenbank im Computer darstellt. In ihr wird beschrieben, wie und wo die Daten in der Datenbank gespeichert werden. Designziel ist hier ein effizienter Zugriff auf die gespeicherten Informationen. Das wird meistens nur durch eine bewusst in Kauf genommene Redundanz erreicht (z. B. im Index werden die gleichen Daten gespeichert, die auch schon in der Tabelle gespeichert sind).

Die Vorteile des Drei-Ebenen-Modells liegen in der

- **physischen Datenunabhängigkeit**, da die interne von der konzeptionellen und externen Ebene getrennt ist. Physische Änderungen, z. B. des Speichermediums oder des Datenbankprodukts, wirken sich nicht auf die konzeptionelle oder externe Ebene aus.
- **logischen Datenunabhängigkeit**, da die konzeptionelle und die externe Ebene getrennt sind. Dies bedeutet, dass Änderungen an der Datenbankstruktur (konzeptionelle Ebene) keine Auswirkungen auf die externe Ebene, also die Masken-Layouts, Listen und Schnittstellen haben.

Allgemein kann also von einer **höheren Robustheit gegenüber Änderungen** gesprochen werden.