WARUM DATENBANKEN?

Schlüsseltechnologie für die Realisierung komplexer Informationssysteme.

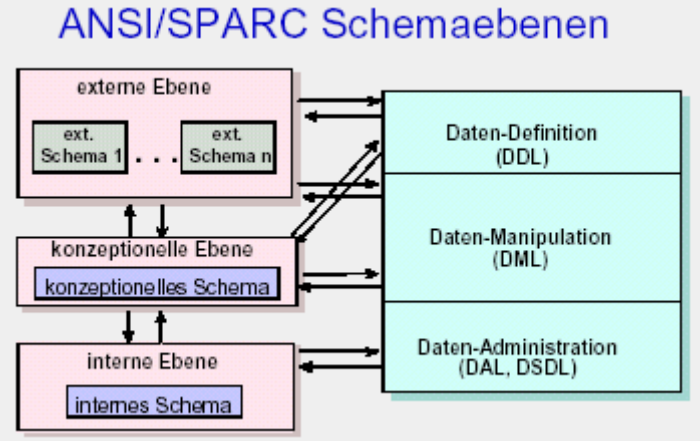
• KENNZEICHEN DER DATEN?

lange Lebensdauer, große Datenobjekte und -mengen, immer wiederkehrende Muster

WARUM DATENBANKSYSTEME?

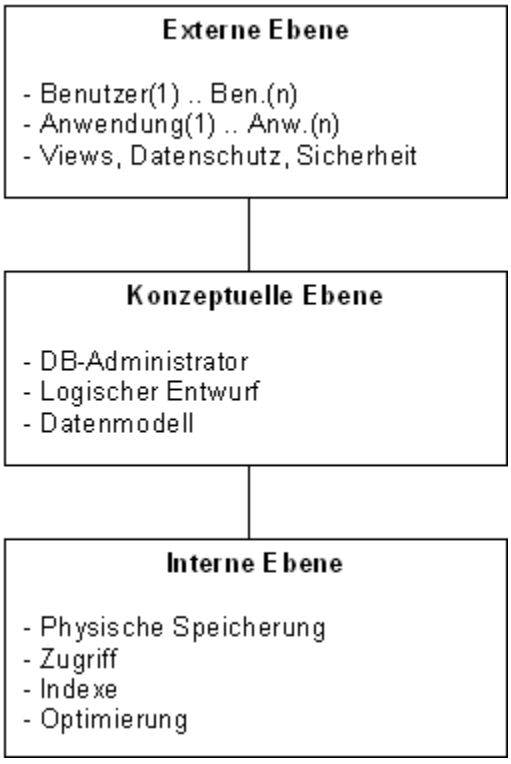
Probleme bei Dateisystemen (Redundanz, Inflexibilität)

ANSI-SPARC 3-SCHICHTEN-MODELL?



1. **Extern** (Sicht von Benutzer/Anwendungsprogrammen),
2. **Konzeptuell** (einheitliche Gesamtschau der Daten),
3. **Intern** (physische Speicherstrukturen)

3-Ebenen-Modell **Entwurf**



ERM

<=== Transformationsregeln

DATENMODELL

<=== Transformationsregeln

VORTEILE VON DATENBANKEN?

Physische Unabhängigkeit (internes Schema ändern, bei Abb. internes » konzept. Schema), **logische Datenunabhängigkeit** (konzept. Schema ändern, Abb. konzept. Schema » externes Schema), **integrierte zentrale Verwaltung** (Standards, Konsistenz, Redundanz)

EIGENSCHAFTEN VON DATENBANKEN

Persistenz, Sekundärspeichermmt., Mehrbenutzerfähigkeit, Zuverlässigkeit, Datensicherheit, Ad-hoc Abfragesprachen

- **PERSISTENZ**
  Daten überleben Sitzungs- und Transaktionsende, langlebig, "in situ" Aktualisierung

- **VERWALTUNG VON SEKUNDÄRSPEICHERN**
  große Datenmengen auf Platten, ein/ausgabeintensiv, Performanceerhöhung: Buffer (im Hauptspeicher), Indexierung, Cluster, Abfrageoptimierung

- **MEHRBENUTZERFÄHIGKEIT**
  DBMS verhindern gleichzeitige Datenmanipulation, Integrität bleibt erhalten

- **DATENZUVERLÄSSIGKEIT**
  Daten teuer - DBMS bestätigt Änderung, Systemfehler: wiederherstellen aller bestätigten Änderungen, **roll-backward**: alle unbestätigten Transaktionen eliminieren, **roll-forward**: alle bestätigten auf Sicherheitskopie nachziehen

- **DATENSICHERHEIT**
  Zugriffschutz, Berechtigungssystem (Subjekte = Benutzer, Objekte = Daten), Berechtigungen (lesen, schreiben) überprüfen

- **AD-HOC ABFRAGESPRACHEN**
  ohne prozedurales Programmieren, deklarativer Zugriff, SQL + QBE

WICHTIGE BEGRIFFE

DBMS (Datenbankmanagementsystem) := Software, die DB verwaltet und alle von den Anwendungsprogrammen verlangten Funktionen zentral durchführt
DBS (Datenbanksystem) := DBMS + DB
DB (Datenbank) := integriert vom DBMS verwaltete Dateien

- **PRODUKTE**
  Oracle, DB2, SQL-Server, Access, Informix, Sybase, Ingres, Progress, Adabas, u.u.u.

Modellierung

- **ZIELE BEIM DATENBANKEN-ENTWURF?**
  Gutes Abbild der Realität, Konsistenz, keine ungewollten Redundanzen, niedrige Antwortzeiten, niedriger Speicherplatzbedarf, niedriger Wartungs-/Pflegeaufwand, Einfachheit *(kurz: das ökonomische Prinzip soll hier auch gelten)*

- **FÜNF PHASEN DER DB ENTWICKLUNG?**
  (A) **Informationsanalyse** - wer braucht welche Daten wann in welcher Qualität? *(also eine Art Analysephase)*
  (B) **konzeptueller Entwurf** - formalisierte Beschreibung des Umweltausschnitts, oft mit graphischem semantischen Datenbankmodell *(entspricht einer*

Designphase bei normaler Software)
(C) **logischer Entwurf** - Abbildung des konzeptuellen Modells auf Datenmodell des DBS *(jetzt wird das allgemeine Modellierungszeug schon konkreter für unsere DB designed)*
(D) **physischer Entwurf** - Speicherstrukturen, Zugriffspfade festlegen *(also eine Art Implementierung)*
(E) **Verwendung, Wartung, Reorganisation** - Tuning, Adaptieren
(Wartungsphase, das Ding läuft schon und muss natürlich verbessert werden)

- **(A) INFORMATIONSBEDARFSANALYSE**
  Wer braucht wo wann was? **Relevante Informationen und Vorgänge** aus dem und über das Objektsystem. Zusammenhänge zwischen Informationen, Informationen und Vorgängen. **Vollständigkeit, Redundanz, Konsistenz**. » Was soll im zukünftigen Informationssystem vorhanden sein und wie wird es verwendet? *(in der Praxis fragst also Deinen Kunden, was er will. Die professionell-theoretischen Informatikstudenten scheitern hier häufig aufgrund Mangel an sozialer Kommunikationsfähigkeit.*

- **(B) KONZEPTUELLER ENTWURF**
  Formalisierte Beschreibung der ermittelten **Informationen und Funktionen**. Häufig mit graphischem Darstellungsmodell *(das dann der Kunde eh nicht mehr versteht)*. Entweder konzeptueller Entwurf des **gesamten Bereichs**, oder zuerst Formulierung der Modelle der einzelnen Benutzersichten und anschließend Integration (**View-Integration**). **Semantische Datenmodellierung**: Definition **aller zulässigen Zustände und Zustandsübergänge** der Datenbasis des geplanten Informationssystems. Ergebnis: konzeptuelles DB Modell *(Informatiker sind da am meisten stolz drauf, häufig geht dann aber in der Praxis nix weiter)*.

- **(C) LOGISCHER ENTWURF**
  Abbildung des konzeptuellen Modells auf das **Datenmodell eines konkreten DBS** *(d.h. jetzt haben wir das schöne allgemeine Modell, das wird jetzt aber z.B. erst mal in ein Modell von Access umgesetzt)*. Ergebnis: logisches DB-Schema *(wäre mE also bei normaler Software schon fast so ein Mischteil der Implementierungsphase)*.

- **(D) PHYSISCHER ENTWURF**
  Festlegen der **Einzelheiten der physischen Darstellung der Daten**. Abbildung auf Speichersrukturen / Datenstrukturen *(geht ja jetzt konkret z.B. in einen Windows PC, da muss man schon auf echte Speicherkriterien aufpassen, was kriegt wie viel KB, etc.)*. Bestimmen der Zugriffspfade. Verantwortlich für Antwortzeiten und Speicherplatzbedarf *(also wer darf was und wie schnell bekommt er was)*. Erforderlich: Mengengerüst, Transaktionsprofil, Nebenbedingungen (z.B. Antwortzeiten für bestimmte Transaktionen). Ergebnis: physisches DB-Schema.

- **(E) VERWENDUNG - WARTUNG - REORGANISATION**
  Reorganisation wegen
  - **veränderter Umweltbedingung** (dargestellte Realität hat sich gewandelt - *vergleiche mit juristischer Definition des Irrtums :-)*), z.B. weitere Anwendungen *(alles verwerfen, weil wir verwenden jetzt SAP)*, modifizierte Aufgabenstellung *(sehr oft: Kunde wollte eigentlich was ganz anderes, ha ha)*, veränderte gesetzliche Bestimmungen *(z.B. Steuerreform)*
  - **Revidierung früherer Entwurfsentscheidungen für Leistungsverbesserung** *(z.B. "so viele Access Tabellen wollte ich gar nicht!")*, Änderung der logischen Struktur (selten - *aber immer öfter*- **Änderung der physischen Struktur** *(z.B. "ein Megabyte pro Kundennummer ist mir zuviel!")*

• DATENMODELLE

System von Konzepten zur abstrakten Darstellung eines Ausschnitts der realen Welt mittels Daten.

1. GRAPHISCHE DATENMODELLIERUNG (HDM, NDM) UND MENGENMODELL (RDM) – LOGISCHE MODELLE (?)

HIERARCHISCHES DATENMODELL:

- Gliederung als typisches Beispiel für hierarchisches Datenmodell
- einfachste Form eines Datensystems
- in der "Informatikersprache" auch Bäume (Trees) genannt
- natürliche Hierarchien: Personaldatei (Firma/Abteilung/Mitarbeiter), künstliche Hierarchien: Lieferantenkartei (Artikel/Lieferant/Adresse)
- Arten von Hierarchien:
  einstufig: genau einem Vaterelement werden ein/mehrere Söhne zugeordnet
  mehrstufig: Zusammenfügen von mehreren Hierarchien
  aber: jedes Element darf nur in einer Gruppe Sohnelement sein
  das Wurzelelement ist selber nirgends Sohnelement (keine Zirkelbezüge)
- Beziehungen:
  1:1 - Vater wird Sohn zuordnet
  1 : n - Vater werden mehrere Söhne zugeordnet
  m : n - nicht darstellbar
- wichtig zum Finden, Ändern, Hinzufügen und Löschen von Daten ist das sequenzielle Auslesen:
- Söhne kommen nach ihrem Vater
- Söhne kommen vor den Brüdern
- programmiertechnisch erfolgt die Zuordnung nicht über Tabellen wie beim Relationenmodell, sondern direkt mit **Pointern**
- **Vorteile:** einfaches, effizientes Datenmodell (kommt daher in Directory Servern, Webservern und im Index von z.B. mySQL zur Anwendung)
- **Nachteile:** unflexibel und bei komplexen Umgebungen schwierig zu modellieren

NETZWERKMODELL:

- baut auf dem hierarchischen Modell auf
- Restriktion, dass Entität nur Mitglied in einer Gruppe sein kann, ist aufgehoben -> mehrere Wurzelelemente möglich
- Modellierung von m:n-Beziehungen möglich
- da es unpraktikabel bzw. nicht realisierbar ist, die m : n-Beziehung direkt zu modellieren, geht man einen Umweg über 2 1:m-Beziehungen (Darstellung: Autor/Beitrag/Buch)
- Zugriff auf die Entitäten erfolgt durch sequenzielles Auslesen der einzelnen Hierarchien
- Vorteile
  - komplexere Umgebungen lassen sich modellieren
  - vermaschte Strukturen sind möglich
  - m:n-Beziehung darstellbar

- Nachteile:
  - Verlust von Einfachheit und Übersichtlichkeit
  - sequenzielles Auslesen komplizierter
  - ineffizienter und demzufolge langsamer

RELATIONALES DATENMODELL:

- Die Tabelle ist die einzige Datenstruktur des relationalen Modells
- als Strukturelemente stehen ausschließlich Relationen, die sich durch Tabellen darstellen lassen, zur Verfügung

- einfache logische Schnittstelle zu den Datenbeständen
- Der Zugriff auf bestimmte Datensätze wird über die Feldinhalte ermöglicht. Dementsprechend arbeitet der Benutzer nur mit logischen, mengenorientierten Abfragen, wobei die physische Speicherung und der Datenzugriff für ihn im Hintergrund bleiben.
- das System bietet die Daten als Tupels an, zwischen denen der Benutzer später weitgehend beliebige Beziehungen - auch während der Anwendung - herstellen kann

Jede dieser Tabellen hat folgende Eigenschaften:

- Jede Tabelle ist autonom und wird durch einen eindeutigen Schlüssel identifiziert.
- Auf jeden elementaren Wert kann garantiert zugegriffen werden durch eine Kombination von Tabellenname, Primärschlüssel und Spaltenname
- NULL-Werte (unbelegte Attributfelder unabhängig von Datentyp) dürfen keine möglichen Primärschlüssel sein und müssen bei Bedarf auch in anderen Spalten ausgeschlossen werden können.
- Jede Spalte hat einen Bezeichner, der innerhalb der Tabelle eindeutig ist.
- Die Anordnung der Spalten ist für alle Tupels (Zeilen) in einer Tabelle einheitlich, aber ohne jede Bedeutung.
- Spalten sind elementar. Dies heißt, dass Tabellen "flache" Datenstrukturen sein müssen, in denen ein Datenfeld keine weitere Datenstruktur enthalten darf. (siehe NF)
- All Einträge innerhalb einer Spalte sind von demselben Typ (meist NUMERIC(n), AMOUNT(n), STRING(n), DATE(n); BLOB)
- Die Reihenfolge der Tupels ist in der Tabelle beliebig.

Ein Datenbanksystem kann dann relational genannt werden, wenn es die allgemeinen Anforderungen an ein DBS erfüllt und

1. die Daten dem Benutzer ausschließlich als Relationen (Tabellen) zugänglich macht,
2. die Primärschlüsselbedingung und die referentielle Integrität (Fremdschlüsselbedingungen) automatisch überwacht,
3. die Operationen der Relationenalgebra und die Änderungsoperationen auf Tabellen ausführen kann, ohne dass physische Zugriffe angegeben werden müssen.

- Vorteile:
  Hoher Handhabungskomfort
- Nachteile:
  Hohe Anforderungen an Rechengeschwindigkeit und Zugriffsgeschwindigkeit (evt. Durchsuchen sämtlicher Tabellen)

OBJEKTORIENTIERTES DATENMODELL:

Das objektorientierte Datenmodell beinhaltet eine Kombination von Ansätzen der klassischen Datenmodelle, der objektorientierten Programmierung und der Wissensrepräsentation.

2. Semantische Datenmodell

Beschreibung des **betrachteten Ausschnitts der realen Welt** (Miniwelt, Universe of Discourse - *was nehmen wir denn, wie legen wir's an - hintergründig?*). **Genaue, eindeutige, vollständige** Beschreibung aller für die Anwendung relevanten strukturellen Eigenschaften:
- in **semantischer Beschreibungssprache**
- **unabhängig** von Hardware und Software *(ca. so unabhängig und toll wie etwa in XML)*
Ergebnis: konzeptuelles Datenbankschema. Verständigungsbasis für Entwerfer, Entwickler und Anwender *(Vor allem Anwender, da er ja der zahlende Kunde ist! Ein Superschema, das der Kunde aber nicht versteht, ist nichts wert!)*.

ER-MODELL BZW. ER-DIAGRAMM *(AUSGESPROCHEN: EMERGENCY ROOM ;-))*

Ein konzeptuelles Modell: das **Entity-Relationship-Modell** *(alt, aber gut)*. Sprache zur Beschreibung der Datenanforderungen, leicht zu verstehen und zu kommunizieren, unabhängig von der tatsächlichen Realisierung in einem DBMS-Produkt. Graphische Sprache: **graphische** Repräsentation der Konstrukte durch **E-R-Diagramme**. Erweitert durch Unmengen von extended E-R-Modellen.

Das ER-Modell basiert auf den drei Grundkonzepten

Entity als zu modellierende Informationseinheit,
Relationsship zur Modellierung von Beziehungen zwischen Entities
und
Attribut als Eigenschaft von einem Entity oder einer Beziehung.

Die Basisbegriffe[1] können wie folgt charakterisiert werden:

Entity
Objekt der realen oder der Vorstellungswelt, über das Informationen zu speichern sind, z.B. eine Vorlesungsveranstaltung, ein Buch oder eine Lehrperson. Auch Informationen über Ereignisse wie Prüfungen können Objekte im Sinne des ER-Modells sein.

Entity-Typ *(Rechteck)*:
Bei der Modellierung betrachtet man nicht einzelne Entities, sondern Entity-Typen. Ein Entity-Typ umfaßt die Menge aller Entities, die die gleichen charakteristischen Eigenschaften besitzen. Also zum Beispiel BUCH, LEHRPERSON, etc.

Relationsship:
Beziehungen zwischen Entities.

Relationsship-Typ *(Rechteck-Raute-Rechteck)*:
Entities können untereinander in Beziehung stehen *(analog Beziehung-Typ)*.
Beispiel: Der Entity-Typ „MITARBEITER" steht in der Beziehung „IST-PROJEKTLEITER" zu dem Entity-Typ „PROJEKT".

Attribut: *(Kreis)*
Eigenschaften von Entities oder Beziehungen. Z.B. die ISBN.Nummer eines Buches, der Titel einer Vorlesung, oder das Semester, in dem eine Vorlesung gehalten wird.
Jedes Attribut kann Werte eines bestimmten Wertebereichs annehmen. Ein Entity wird durch die Kombination seiner Attribute beschrieben.
Beispiel: Beziehung mit Attribut: Die Beziehung „IST-MITARBEITER-AN" hat etwa das Attribut „PROZENT-DER-ARBEITSZEIT".

Schlüssel, Primärschlüssel: *(Unterstrichen)*
Da Entites voneinander unterscheidbare Einheiten sind, muss also ein Entity durch eine nichtleere Menge von Attributen eindeutig identifiziert werden können. Dies nennt man Schlüssel für diesen Entity-Typ. Um eindeutige Schlüssel zu erhalten, werden oft „künstliche" Attribute eingeführt, etwa die Matrikelnummer oder Personalnummer.

Gibt es für einen Entity-Typ mehrere Attribute, so wird im allgemeinem genau einer als Primärschlüssel bezeichnet.

---

[1] Streng genommen sind dies nur Entität, Attribut und Beziehung.

• IDENTIFIKATION

- Konzepte (Attribute, Entitäten), die eine Entität (Instanz) **eindeutig identifizieren**. *(Merk dir vielleicht: welche Informationen brauch ich mindestens als Angabe, um aus dem ganzen Haufen Daten das gesuchte Objekt hundertprozentig zu finden!)*
- bestehen häufig aus einem oder mehreren **Attributen** (**Schlüssel, interne Identifikatoren**)
- manchmal sind Attribute nicht ausreichend und Entitäten müssen über ihre **Beziehungen** zu anderen Entitäten identifiziert werden (**externe Identifikatoren**)
  Beispiele: BLZ ist der Schlüssel für eine Bank, Sozialversicherungsnummer ist der Schlüssel für den LehrerXYZ, Telefonnummer wird über Ländercode, Vorwahl und Rufnummer identifiziert.

DATENBANKENTWURF (von Ulf)

Bevor eine Datenbank entwickelt werden kann, muss analysiert werden, welche Informationen (informelle Analyse) und Funktionen (funktionelle Analyse) die Datenbank enthalten soll.

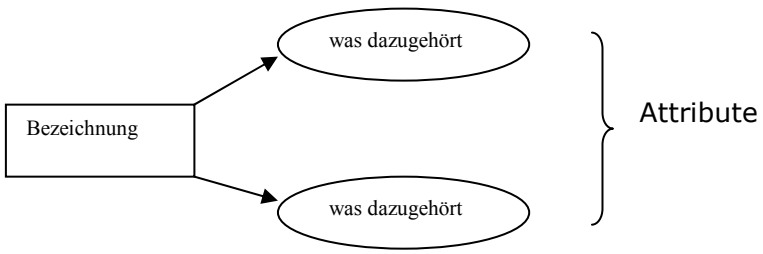
informelle Analyse: Ableitung von Datendiagrammen und von Entity-Relationship-Diagrammen (ERD)

funktionelle Analyse: Verarbeitungsbedingungen (Dialognutzung, Nutzeranzahl)
Integritätsbedingungen (Eingabesicherung)
Anforderungsbedingungen (Wer ist der Nutzer?)
Qualitative Bedingungen (Speicherart, -kapazität)
Datenorganisationsform

ERD: Die entstehende Datenbank wird auf Typebene analysiert.

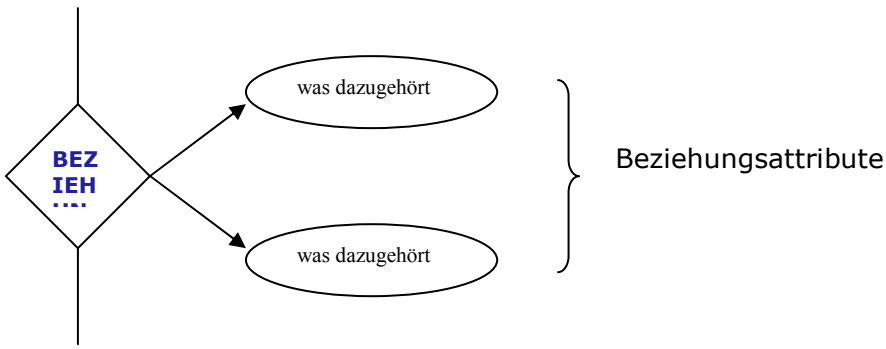
Symbolik:

1. Entity-Typ:



Beispiel: PROFESSOR → name, profnr

2. Beziehungs-Typen:



Beispiel: prüft (als Beziehung zwischen den Entity-Typen PROFESSOR und STUDENT) → fach, note

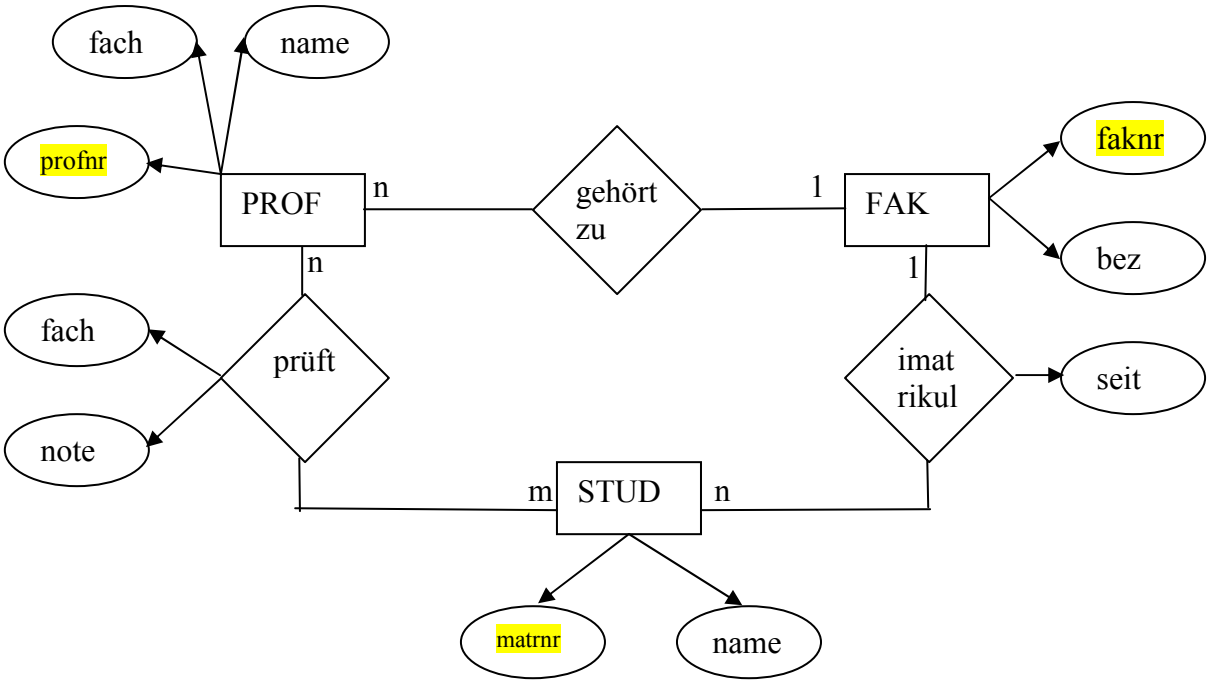
3. Kardinalitäten: erfassen die anzahlmäßige Beziehung zwischen den Entity-Typen

a) 1:1 (Ein STUDENT ist verheiratet mit einer STUDENTin.)
b) 1:n bzw. n:1 (Mehrere PROFESSORen gehören zu einer FAKULTÄT.)
c) n:m (Mehrere PROFESSORen prüfen mehrere STUDENTen)

Beispiel:

In den Fakultäten der Uni arbeiten mehrere Professoren. Dort sind auch viele Studenten imatrikuliert, die von den Professoren geprüft werden.

(Manchmal standen in den Beispielen von Frau Tischendorf auch die Attribute und die Beziehungsattribute schon mit drin, die man verwenden soll, ansonsten kann man die sich ausdenken. Beachten: Jeder Entity-Typ sollte ein eindeutiges Attribut haben, mit dem man diesen Entity-Typ identifizieren kann – wird später als Schlüssel verwendet!)



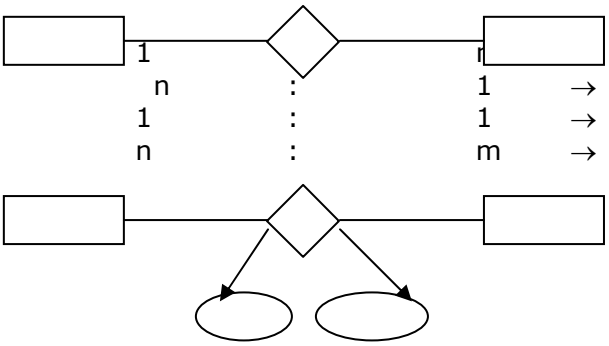
Um die Ableitung des relationalen Schemas zu vereinfachen, kann man schon im ERD die Schlüssel kennzeichnen.

Ableitung des relationalen Schemas:

1. Aus jedem Entity-Typ wird eine Tabelle (ein Relationstyp).

am Bsp.: PROF (profnr, name, fach)
FAK (faknr, bez)
STUD (matrnr, name)

2. Aus den Beziehungstypen können auch Tabellen (Relationstypen) entstehen, nur dann nicht, wenn die Kardinalität 1:1 bzw. 1:n (n:1) ist und es keine Beziehungsattribute an diesem Beziehungstyp gibt.



keine Tabelle
keine Tabelle
keine Tabelle
Tabelle

Fremdschlüsseleinschluss Von dem Entity-Typ der Kardinalität 1 in den Entity-Typ der Kard. n. Bei 1:1 ist es egal woher und worein.

es entsteht in jedem Kardinalitätsfall eine Tabelle, da es Beziehungsattribute gibt

Wenn eine Tabelle entsteht, kommen hinein:
1. Beide benachbarten Schlüssel der Entity-Typen (wieder als Schlüssel gekennzeichnet) **und**
2. alle (evtl.) vorhandenen Beziehungsattribute (bei n:m müssen sie nicht vorhanden sein).

am Bsp.: prüft (profnr, matrnr, fach, note)
imatrikul (faknr, matrnr, seit)

Keine Tabelle entsteht beim Beziehungstyp „gehört zu", da es keine Beziehungsattribute hat und die Kardinalität n:1 ist. Das bedeutet, dass wir in die Tabelle „PROF" noch den Fremdschlüssel aus der Tabelle „FAK" einschließen müssen (von Kard. 1 in n).
Es entsteht somit dieses relationale Schema:

PROF (profnr, faknr, name, fach)
FAK (faknr, bez)
STUD (matrnr, name)
prüft (profnr, matrnr, fach, note)
imatrikul (faknr, matrnr, seit)

- **Anforderungen an relationale Schemata**

  Normalformen: Regeln zum Aufbau von Relationen – bei Verstoß kann es zu Anomalien kommen

  1. NF:

  Alle Attributwerte haben atomaren Charakter. Relationen in der ersten Normalform sind also "flache Tabellen" ohne jede "Schachtelung"

  2. NF:

  Ist erfüllt, wenn 1. NF erfüllt ist und alle Nichtschlüsselattribute voll funktional abhängig von allen Schlüsselkandidaten sind.

Konsequenz: besteht der Schlüssel einer Relation in erster Normalform nur aus einem Attribut, so befindet sie sich automatisch auch bereits in zweiter Normalform.

3. NF:
Eine Relation befindet sich in dritter Normalform, wenn sie der ersten und zweiten Normalform genügt und kein Attribut transitiv (indirekt) von einem Schlüsselkandidaten abhängig ist.

Hilfreich ist das Aufspalten in zwei Tabellen (

• PRAKTISCHE SCHRITTE ZUR ERARBEITUNG EINES DATENMODELLS

(1) Ermitteln der Objekte („Find the Object")
(2) Zusammenfassung gleichartiger Objekte
(3) Bestimmung der Attribute
(4) Bestimmung der Wertebereiche der Attribute
(5) Ermittlung der Beziehungen zwischen den Klassen
(6) Abbildung der Ergebnisse in einem Modell

(1) Ermitteln der Objekte

• In Abhängigkeit von der darzustellenden Situation ist festzulegen, welche Gegenstände, Sachverhalte usw., also Objekte der realen Welt, mit dem zu entwerfenden System zu bearbeiten sind.
• Ein einzelnes Objekt wird als **Entität** bezeichnet.

(2) Zusammenfassung

gleichartiger Objekte
• Gleichartige Objekte werden zu Objektklassen zusammengefasst.
• Die Klassen werden auch als **Entitätstypen** bezeichnet.

(3) Bestimmung der Attribute

• Objekte eines Entitätstyps werden durch Merkmale beschrieben, deren Kombination der Ausprägungen jedes einzelne Objekt individuell charakterisiert.
• Diese Merkmale werden als Attribute bezeichnet.
• Besitzen Attribute zum Entitätstypen eine Beziehung, die eine eindeutige Identifikation der Entitäten zulässt, so handelt es sich Schlüsselattribute.

(4) Bestimmung der Wertebereiche der Attribute

• Jedes Attribut besitzt eine Menge von möglichen Werten.
• Dieser Wertebereich wird auch als **Domäne** bezeichnet.

(5) Ermittlung der Beziehungen zwischen den Klassen

• Zwischen einzelnen Entitätstypen bestehen Beziehungen.
• Die Beziehungen der Entitätstypen werden **Relationships** genannt.
– 1:1-Beziehung: Einer Entität ist stets genau eine andere zugeordnet.
– 1:n-Beziehung: Es existieren mehrere Entitäten, die sich auf eine gemeinsame andere beziehen.
– n:m-Beziehung: Auf jede Entität des einen Typs beziehen sich mehrere andere Entitäten des anderen Typs, die sich aber wiederum auf mehrere des einen Typs beziehen

(6) Abbildung der Ergebnisse in einem Modell

• Die Ergebnisse der Schritte 1. bis 5. müssen in einem Modell abgebildet werden.
• konzeptionelles Modell, von Datenbanksystemen unabhängig
– Daten (meist) in einer grafischen, für den Anwender verständlichen Form beschrieben.
• darauf aufbauend logische Datenbankmodelle mit ihren vom konkreten Datenbanksystem abhängigen Beschreibungsformen
• Datenbeschreibungssprache (Data Definition Language) zur konkreten Umsetzung Erweiterungen der ERM
• zusätzliche inhaltliche Bedeutung in der Grafik darstellbar

• Umdefinition von Beziehungen zu Entitätstypen: Beziehung zwischen zwei Entitäten hat den Charakter einer Entität, obwohl sie gleichzeitig auch eine Beziehung darstellt.
• grafisches Symbol: Rechteck und Raute

• DATENBANKSPRACHEN - SPRACHSCHNITTSTELLEN

Umsetzung des Datenbankmodells in die Sprache des Datenbanksystems

• Sprachen

Data Definition Language (DDL)
Data Manipulation Language (DML)
Data Storage Definition Language (DSDL)
Data Administrator Language (DAL)

• Verwendung der Sprachen durch:

(a) Einbettung in Programmiersprachen
- Precompiler behandelt Sprachkonstrukte
– Compilererweiterungen

(b) Application Programmers Interfaces (APIs):
Aufrufschnittstellen

Früher: unterschiedliche Sprachkonzepte
Heute: einheitliches Sprachkonzept
SQL (**S**TRUCTURED **Q**UERY **L**ANGUAGE)
QBE (Query by Example) } grafikorientierte Sprachen
QBF (Query by Form)

• NUTZERKLASSEN ADMINISTRATIONSFUNKTIONEN PERSONEN UND ROLLEN

Datenbankadministrator (DBA):): externes, konzeptionelles und internes Schema

- Festlegung von Zugriffs- und Integritätskontrollen
- Planung und Zuweisung von Systemspeicher
- Erstellung primärer Speicherstrukturen
- Erstellen primärer Objekte (DB, Tabellen, Views, Indexe)
- Lizenzverträge, Kosten
- Sicherungs- und Recoverystrategien
- Performanceüberwachung und Tuning

Systemanalytiker, **Anwendungsprogrammierer**: externes und konzeptionelles Schema

- Anforderungserhebung, Softwareentwicklung;

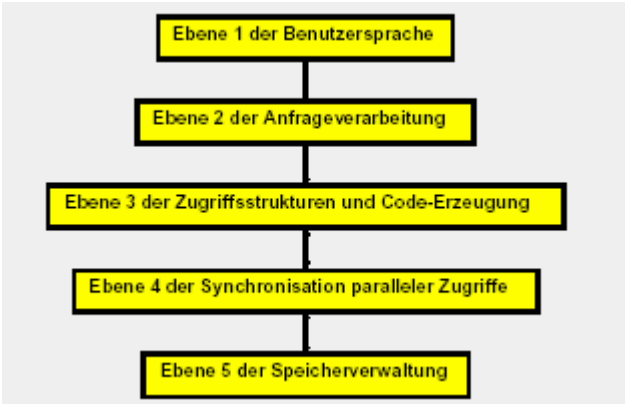
Enbenutzer: gelegentliche Benutzer/Manager, parametrische Benutzer/Sachbearbeiter (Anwendungsprogramme, canned transactions), Power-User/Analytiker (komplexe Anforderungen, gute Kenntnis DB u. Schnittstellen = *gescheite Informatiker*)

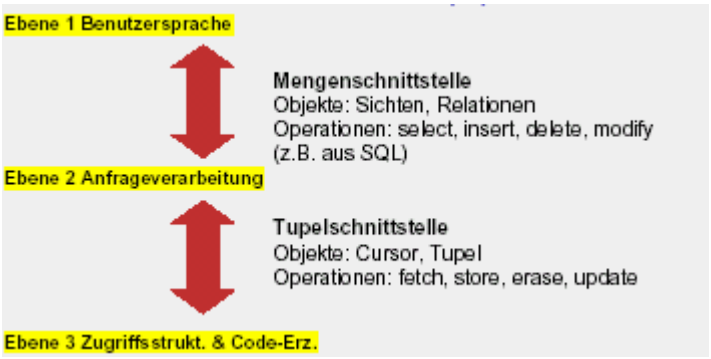
Relationenalgebra

Vereinigung	**R+S**	• *vereinigungskompatible Relationen*, d.h. Relationen mit gleichem Grad und Wertemenge der Attribute
Mengen-Differenz	**R—S**	• *vereinigungskompatible Relationen*, d.h. Relationen mit gleichem Grad und Wertemenge der Attribute
Kartesisches Produkt	**R x S**	• *neuer Relationen-Grad (g_R+g_S)*

Projektion	**R [L1, L2, ...] = { r[L] | r∈R }** **Proj R over L1, L2,. giving R[L1, L2, ..]**	• *Reduktion der Tabelle auf Spalten L1, L2*, • Streichen von Spalten der Tabelle
Selektion	**R [L=d] = { r∈R | r[L=d]}** **Select R with L=d giving R[L=d]**	• *Reduktion der Tabelle auf Zeilen*, welche die Spaltenbedingung L=d erfüllen • Streichen von Zeilen der Tabelle
Verbund (Join)	**R [A ⊗ B] S = { r∈R, s∈S | r[A] ⊗ s[B] }** **= RxS [R.A ⊗ B.S]** **Join (R,S) by (A ⊗ B) giving R[A ⊗ B]S**	• *Selektion des kartesischen Produkt bzgl. der Spaltenbedingung* A ⊗ B • Kombination von Zeilen aus A mit allen aus B, welche Bedingung A ⊗ B erfüllen • Wegfall von Zeilen, welche Bedingung A ⊗ B nicht erfüllen
Natürlicher Verbund (natural join)	**R** NATJOIN **S** **= (RxS) [R.a_i=S.a_i; i=1,2,..p]** **[a_p,....a_n]**	• *Verschmelzen bzgl gleicher Spalten*

• FÜNF-SCHICHTENMODELL FÜR DBMS





• DBMS FUNKTIONSKOMPLEXE FUNKTIONEN IMPLEMENTIERUNGSMETHODEN

Aufgaben eines DBMS

allgemein: - Sicherung der reibungslosen u. fehlerfreien Arbeit vieler Nutzer auf einer DB zur selben Zeit
- hohe Systemleistung
- keine gegenseitige Behinderung
- effiziente Lösungen für Wiederherstellung nach Ausfällen

1. Integration
   - einheitliche Verwaltung aller von Anwendung benötigter Daten
   - redundanzarme Speicherung der Daten
2. Operationen
   - Datenspeicherung (insert),
   - Suchen (select) und
   - Ändern (update) des Datenbestandes
3. Katalog – DataDictionary
   - ermöglicht Zugriffe auf die Datenbeschreibung der Datenbank
4. Benutzersichten (crest view)
   - unterschiedliche Sichten (nutzerspezifisch)auf den Datenbestand
5. Konsistenzüberwachung
   - Integritätssicherung, Gewährleistung der Korrektheit von DB – Inhalten u. der korrekten Ausfüllung von Änderungen so dass diese die Konsistenz nicht verletzen können
6. Datenschutz

7. Transaktionen (TA)

8. Synchronisation
   - konkurrierende TA mehrerer Benutzer müssen synchronisiert werden, um gegenseitige Beeinflussungen etwa versehentliche Schreibkonflikte auf gemeinsamen Datenbestand, zu vermeiden
9. Datensicherung
   - Wiederherstellung von Daten nach Systemfehlern

SOFTWARESCHICHTEN EINES DBMS:

- SYSTEMPUFFER-MANAGER
  - Seitenweiser Datentransport zwischen Arbeitsspeicher und externen Speichern
  - Seitenverwaltung und Seitenersetzung
    im Zusammenspiel mit der Recovery (pinned pages, forced output)
  - Seitenschnittstelle zum Zugriffsmanager
- ZUGRIFFSMANAGER
  - Tupel und Indexe als Objekte

- Operatoren auf Tupeln
  (Zugriff auf Tupel, Bereitstellen, Update-Operationen, Zugriffspfade)
- Operatoren für Relationsschemata (Anlegen, Lesen, Löschen)
- Operatoren für Transaktionsmanagement

- DATENMANAGER
  - Relationen und Tupel als Objekte
  - Relationale Sprachen, z.B. SQL, als Operatoren
  - Mengenorientierte relationale Schnittstelle
    zwischen Anwendung und Datensystem:
  - Übersetzung und Optimierung der Benutzerabfragen
  - Zugriffskontrolle
  - Integritätskontrolle

---

1. DATENSPEICHERUNG /DATENZUGRIFF

Ziel:Sicherung der physischen Datenunabhängigkeit
Nutzerebene -> logische Strukturen
Hauptspeicher ->Speicherstrukturen, Speicherzuordnungsstrukturen
Externspeicher ->Betriebssystemfunktionen

Hauptspeicherverwaltung
Abbildung von Entitys auf (logische) Sätze, dabei werden Attribute auf Datenfelder gesetzt
Sätze haben feste Länge -> evt. Auffüllen mit Leerzeichen
Sätze haben variable Länge (Längenangabe pro Attribut oder Begrenzer zwischen den Attributen oder Zeiger)

Pufferverwaltung
-> über AP erfolgt die Ein- und Ausgabe
–> Zwischenspeicherung im Datenbank-Puffer (Hauptspeicherbereich zum Datenaustausch zwischen Anwendung und DB)
-> DB-Puffer ist unterteilt in Seiten (Blöcke)
-> es wird seitenweise/blockweise gelesen, nicht satzweise
-> nach Veränderung wird auf externen Speicher geschrieben

DATENAUSTAUSCH ANWENDUNG-PUFFER

- Ablage variabler Datensätze
  Löschen von Datensätzen -> unproblematisch
  Einfügen von Datensätzen -> Freiplatzverwaltung in Seite
  Erweitern Datensätze -> Freiseitenverwaltung im Puffer

- Adressierung der Sätze
  absolute Byte-Adressierung von Pufferanfang (Problem bei Satzverschiebung)
  relative Byte-Adresse in Seite (TID Tupel Identifer) -> Adresse rutscht mit

DATENAUSTAUSCH EXTERNSPEICHER-PUFFER

- erst wird geprüft, ob Datenblock schon/ noch im Puffer
  ja -> keine Datenblockübertragung
  nein -> Blockübertragung
- Bereitstellen leerer Seite für Block
- Freigeben der Seite:
  Überschreiben (Seite ohne Änderung)
  Rausschreiben (falls Seite durch DB-Operator geändert)
- Datenzugriff auf Tupel über Attribute (Schlüssel- und Nicht-schlüsselattribut)

Paging
(Seitenwechsel)
Freigabestrategien:
die am längsten nicht angesrochene Seite (LRU)
.die am wenigsten angesprochene Seite (NFU)
die nicht geänderte Seite

Zugriffspfad

- schell über: Primärschlüssel (Primärzugriffspfad)
, Sekundärschlüssel (Sekundärzugriffspfad)
- sequentiell: Listen (linear, verkettet)
- indexsequentiell: Indexlisten
- baumstrukturiert: Suchbäume
- direkt: Hashtabellen (statisch, dynamisch) Berechnung der Adresse aus Primärschlüsselwert über Hash-Funktion

2. DATENSICHERHEIT

Maßnahmen zur Sicherung der DB-Konsistenz gegenüber:
Eingabe-, AP- und Bedienungsfehlern
Datenverfälschung und Datenzerstörung durch Hard- und Softwarefehler

- Konsistenz (Intergität)
  - Semantische Integrität äußere Konsistenz
  - Operationale Integrität innere oder physische Konstistenz

Transaktionen

Transaktion:

- Zusammengehörige Menge elementarer Operationen zu einer logischen Arbeitseinheit
- Spezifikation von Transaktionen im Anwendungsprogramm durch
  BEGIN_TRANSACTION
  END_TRANSACTION bzw. **COMMIT**
  ABORT bzw**. ROLLBACK**
- Synchronisation und Recovery sind alleinige Aufgabe des DBMS.

ACID-Prinzip**:**

- **Atomarität:**
  Jede Transaktion ist eine unteilbare Einheit, sie wird entweder ganz oder gar nicht ausgeführt
  (=> Rückgängigmachen bzw. Annulierung der Transaktion bei Recovery).
- **CONSISTENZ:**
  Eine Transaktion garantiert den Übergang von einem konsistenten DB-Zustand zu einem anderen konsistenten DB-Zustand (Konsistenz).
- **Isolation**
  Die Ausführung einer Transaktion ist isoliert von der Ausführung parallel ablaufender anderer Transaktionen.
- **Dauerhaftigkeit:**
  Wenn eine Transaktion Commit macht, ist ihre Wirkung persistent (Dauerhaftigkeit)

Recovery

Wiederherstellen eines korrekten DA-Zustandes nach Fehlern, Crash o. Arbeitsunterbrechnungen, TA-Abruch.

Maßnahmen:

- USV
- Spiegelplatten
- Logfiles-> Before-Image und After-Image
- Mehrprozessorsysteme

Synchronisationsproblem Fehler im Mehrnutzersystem

Parallele Ausführung von Datenbankmanipulationen können zu folgenden Inkonsistenzen führen:

- **lost update** (verlorene Änderung)
  Änderungen W(A) einer Transaktionen A gehen verloren, da die folgende Transaktion B auf denselben Ausgangsdaten R(B) wie A operiert. D.h. Transaktion B liest, bevor Transaktion A geänderte Daten zurückgeschrieben hat.
- **Inkonsistente Sicht:**
  Sicht auf inkonsistenten Zwischenzustands einer noch nicht beendeten Transaktion
- **Inkonsistenz** der Datenbank:
  Eine Transaktion arbeitet auf einer inkonsistente Sicht weiter und hinterlässt daher die Datenbank nach ihrer Beendigung in einem inkonsistenten Zustand.
- **Phantome:**
  Inkonsistente Sicht auf Daten vor und nach einer Änderung

Synchronisation

2-Pasensperrkonzept:

1. Vor jedem Zugriff wir Objekt gesperrt
2. nach Freigabe, darf kein weiteres gesperrt werden

Man unterscheidet zwei Klassen von Synchronisationsverfahren:

- Optimistische (Verifizierende) Verfahren
  - Rücksetzen (Abbruch und Neustart) von Transaktionen, wenn bisher entstandene Schedule nicht serialisierbar ist.
  - Voraussetzung ist das seltenes Auftreten nicht-serialisierbarer Schedules (optimistische V).
- Pessimistische (Präventive) Verfahren
  - Es wird verhindert, dass nicht serialisierbare Schedules entstehen.
  - Voraussetzung für diesen hohen Aufwand ist häufiges Auftreten (pessimistische Verfahren).
  - Sperrverfahren: (Locks)
    Setzen von temporären Sperren auf Objekte- keine andere TA kann zugreifen
  - Zeitstempelverfahren:
    Zugriff auf Objekte in der Reihenfolge des Alters der Transaktionen
  -

Serialisierbarkeit

- Prinzip der Serialisierbarkeit:
  Ein paralleles System von Transaktionen ist korrekt synchronisiert, wenn es serialisierbar ist, d.h. wenn es eine mindestens irgendeine serielle Ausführung der Transaktionen gibt, welche
  1. denselben Datenbankzustand und
  2. dieselben Ausgabedaten der Transaktionen liefert

3. DATENSCHUTZ

Datenverfälschung, Computerkriminalität -> gesetztl., organ. und techn. Regelungen ==Zugriffskontrolle, Zugriffsrechte, Sichtkonzept, Kryptografie

• SQL

SQL *(Structured Query Language)* ist die **Standardabfragesprache bei relationalen Datenbanken**.

Befehle lasse sich in 3 verschiedene Sprachen aufteilen

DDL-Befehle (Data-Description-Language)
- Erzeugung, Löschen, Veränderung von Tabellen, Views, Idizes

create view ...;
after nachträgliche Änderungen
drop Tabelle entfernen

DCL-Befehle (Data-Control-Language)
- Vergabe von Zugriffsrechten

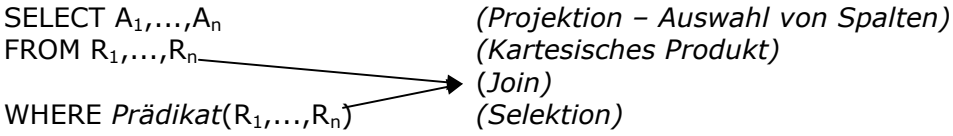
grant zur Festlegung
revoke zum Entfernen

DML-Befehle (Data-Manipulation-Language)
- Abfrage und Veränderung der Daten

insert fügt eine Zeile in eine Tabelle ein
delete löscht eine oder mehrere Zeilen einer Tabelle
update ändert eine oder mehrere Zeilen einer Tabelle
select holt Daten aus einer oder mehreren Tabellen

Der **SFW-Block** beschreibt eine **Standardform** einer SQL-Anfrage. Er ist nach den ersten drei Klauseln einer solchen Anfrage *(select, from, where)* benannt.

Prinzipieller Aufbau:



Beispiel Join:
{Spalten}SELECT ArtNr, Bezeichnung, Name, Ort
{Tabellen}FROM Artikel, Lieferanten
{Bedingung}WHERE Artikel.StammLief = Lieferanten.LiefNr

Data Definition Language (DDL)

- Tabellen anlegen:
  - create table prof
    (
    vorname char(15),
    name char(20),
    fak char(15),
    profnr integer,
    alter integer
    );
- alter table: nachträglich Attribute einfügen, löschen oder im Datentyp ändern
  alter table prof
  add Universitaet varchar(15)

- Tabellenspalte ändern:
  alter table prof
  modify (Universitaet varchar(20)

- Tabelle um eine Spalte verkürzen:

```
alter table prof
drop  column Universitaet
```

Data Manipulation Language (DML)

- Die Grundbausteine der Datenabfrage: SELECT ,FROM und Where

select <Spaltennamen>
from <Tabelle>
where <Suchbedingung>

select * from bikes; alle Datenzeilen der Tabelle Bikes

Ergebnis:

NAME	RAHMEN	MATERIAL	KM_STAND	TYP
Diamant	2800 28	KOHLEFASER	3500	CROSS
MIFA	24	STAHL	12000	CITY
ARROW	28	ALU	2000	MTB

select * from bikes where name = 'Diamant';

Ergebnis:

NAME	RAHMEN	MATERIAL	KM_STAND	TYP
Diamant	2800 28	KOHLEFASER	3500	CROSS

select * from prof where vorname = 'AL';

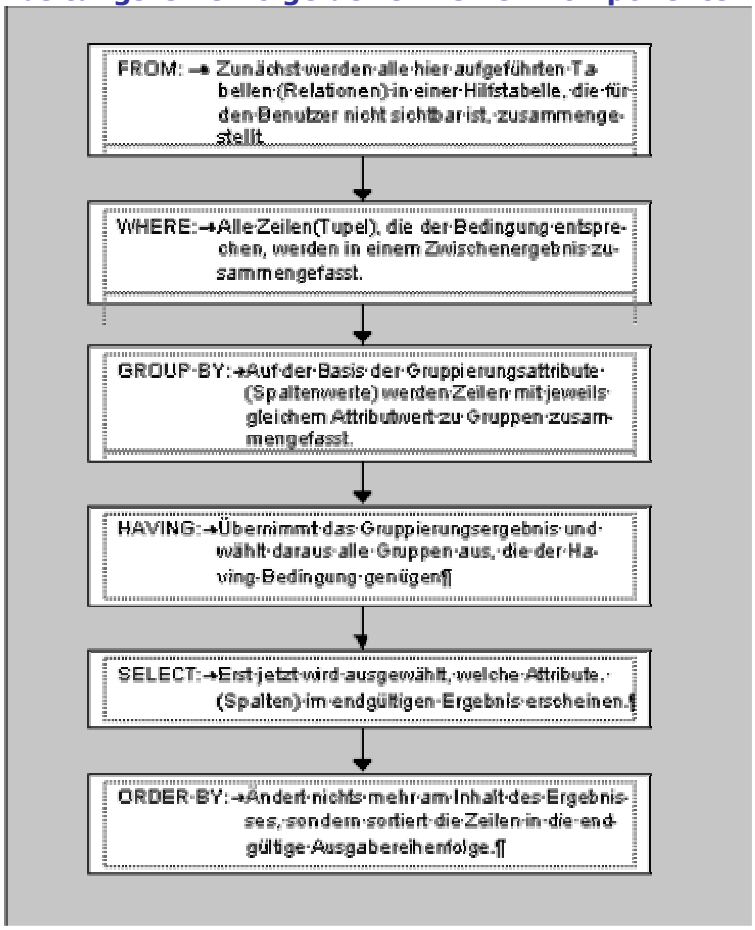
Ergebnis:

NACHNAME	VORNAME	ALTER	FAK	PROFNR
BUNDY	AL	50	IF2MATIK	647700

count (liefert die Anzahl der Zeilen, die der Bedingung in der WHERE-Klausel entsprechen)

select count(*)from prof where alter < 60;

Prinzipielle Abarbeitungsreihenfolge der einzelnen Komponenten



Was ist ein Index?
Was sind die Nachteile von Indexen?
Was sind die Vorteile von Indexen?

Die Vorteile bei der Nutzung von Indexen ist ein schnelles Auffinden bestimmter Datensätze (auch bei Verbundbildung). Als Beispiel kann man ein Telefonbuch einmal sortiert und einmal unsortiert anbringen. Indexe können primär durch die physikalische Sortierung der Datensätze angelegt werden oder sekundär in einer eigenen (Baum-)Struktur (Verzeichnis).

Nachteile von Indexen sind ein höherer Speicherbedarf (im Sekundärspeicher) und eine schlechtere Performance beim Einfügen von Datensätzen.