**Datenbanken Klausurvorbereitung**

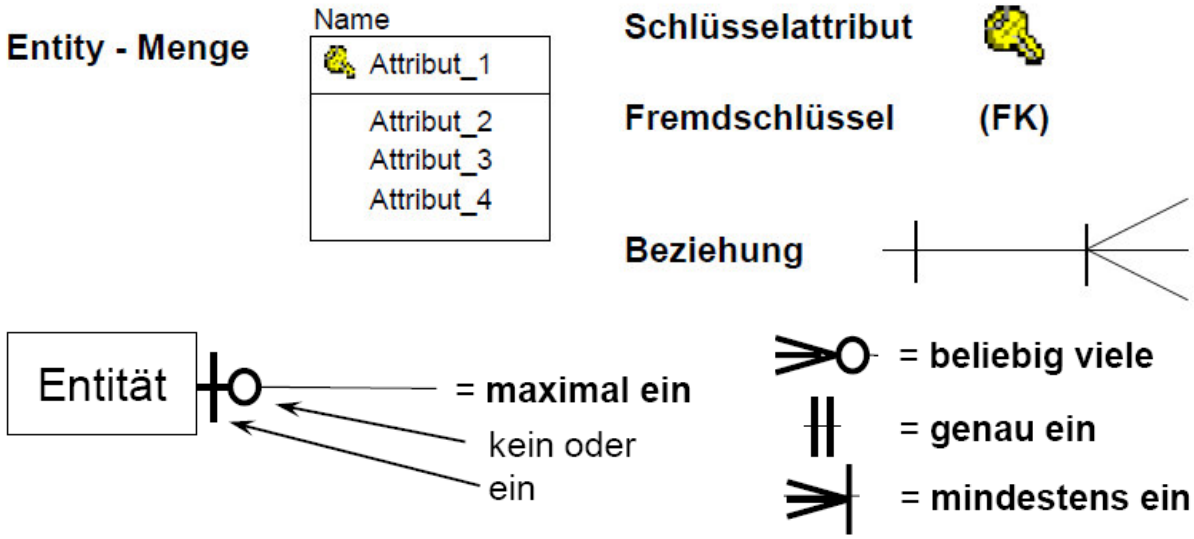
## Klassifizierung / Arten von Datenbanksystemen

- *Flache Dateien*
- *Hierarchische Datenbanken:* Datenmodell in der Art eines Baumes
- *Netzwerkdatenbanken:* Adressverweise wie im hierarchischen Datenmodell, Datenbank kann jedoch mit mehreren Sätzen in Beziehung stehen
- *Relationale Datenbanken:* Daten und Beziehungen werden in Tabellen abgebildet, **Tabellen heißen Relationen, Zeilen heißen Tupel (und entsprechen Datensätzen) und Spaltenüberschriften Attribute**
- *Objektorientierte Datenbanken:* Daten werden als Objekte mit ihren Methoden und Attributen gespeichert
- *Objektrelationale Datenbanken:* Aufbau auf relationalen Datenbanken, erlauben die Definition eigener Datentypen (in vielen großen Datenbanken heute verwendet)
- *Deduktive Datenbanken:* Erweiterung des relativen Datenbankmodells um eine Deduktionskomponente, die auf dem Prädikatenkalkül aufbaut

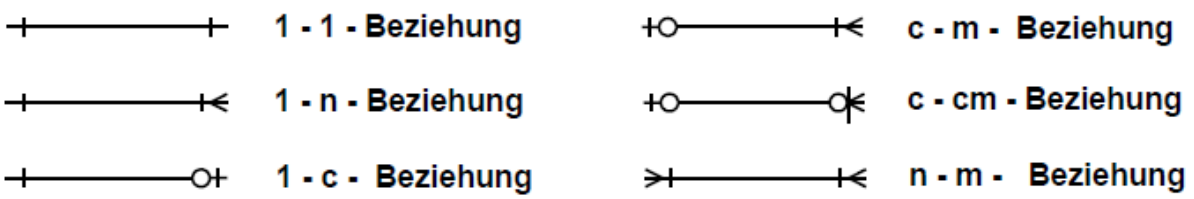
## ER-Schema

Zur Modellierung der Daten, die in einer Datenbank gespeichert werden sollen; enthält:

- *Entität:* Objekt der realen Welt
- *Entitäten-Mengen:* Zusammenfassung gleichartiger Entitäten
- *Attribute* (auf Relevanz, sprechende Namen und Datentypen achten)
- *Schlüssel:* Minimale Menge von Attributen, um eine Entität eindeutig zu identifizieren
  ➔ obligatorisch
- *Beziehungen:* Beschreiben einen Zusammenhang zwischen Entitäten
- *Beziehungsmengen:* Zusammenfassung gleichartiger Beziehungen




Beziehungen können durch 1-1, 1-n, n-m, etc. beschrieben sein:




Legende: 1 (genau ein), C (ein oder kein), M (ein oder mehrere), MC (kein, ein oder mehrere)

## Relationale Algebra

Beschreibt eine formale Sprache, mit der sich Abfragen über einem relationalen Schema formulieren lassen. Damit lassen sich Relationen verknüpfen, filtern oder umbenennen.

- **Kartesisches Produkt:** Menge aller Tupel, die aus den Einzelelementen gebildet werden können (eine Relation ist somit eine Teilmenge des kartesischen Produkts).
- **Selektion:** Eine Selektion selektiert zur Bedingung passende Tupel (Zeilen). Sie gibt somit für jedes Tupel wahr oder falsch zurück. Als Bedingung möglich sind Attribute, Konstanten, Vergleichsoperatoren, logische Operatoren und deren Kombination.
- **Projektion:** Eine Projektion filtert einzelne oder mehrere Attribute aus und zeigt nur diese an.
- **Theta-Join:** Bildet zuerst kartesisches Produkt von 2 Tabellen (an jedes Tupel der 1. Tabelle werden alle Tupel der 2. Tabelle angehängt, pro Kombination neues Tupel), danach wird nach einer Bedingung selektiert und übriggebliebene Tupel angezeigt.
- **Equi-Join:** Theta-Join, der auf den Vergleichsoperator (=) beschränkt ist.
- **Natural-Join:** Equi-Join, bei dem alle Attribute, die in beiden Tabellen gleich heißen, auf Gleichheit überprüft werden. Kein Selektionsprädikat nötig! Alle passenden Datensätze werden ausgegeben, wobei das doppelte Attribut nur einmal übernommen wird.
  - ➔ Verlustfreie Join Operationen: Equi-Join und Natural-Join sind verlustfrei, wenn alle Tupel beider Tabellen am Verbund teilnehmen.
  - ➔ Dangling Datensätze: Datensätze, denen bei Join-Operationen die entsprechenden Datensätze zur Verknüpfung in der anderen Tabelle fehlen.
- **Linker Outer Join:** Join mit Bedingung, bei dem die linke Seite (Tabelle 1) komplett übernommen wird und fehlende Werte aus Tabelle 2 mit NULL aufgefüllt werden.
- **Rechter Outer Join:** Join mit Bedingung, bei dem die rechte Seite (Tabelle 2) komplett übernommen wird und fehlende Tupel in Tabelle 1 mit NULL aufgefüllt werden.
- **Union (Vereinigung):** Gleiches Schema notwendig, zeigt die Spaltenwerte aus beiden Tabellen vereint an (wobei doppelte Werte nur einmalig angezeigt werden).
- **Durchschnitt:** Gleiches Schema notwendig, zeigt alle Tupel an, die in beiden Entitäten (Tabellen) enthalten sind.
- **Differenz:** Gleiches Schema notwendig, zieht 2. Tabelle von Erster ab und zeigt alle übriggebliebenen Datensätze an. (Mathematische, nicht logische Differenz! T1-T2)
- **Division:** Umkehrfunktion zum kartesischen Produkt. Wird Tabelle 1 durch Tabelle 2 dividiert, erhält man ein Ergebnis, falls ein oder mehrere Tupel in T1 auf alle Attributwerte in T2 passen. Dabei werden nur die Attribute angezeigt, die von der Division nicht betroffen sind. Gleiche Datensätze werden nicht doppelt angezeigt.

**Beispiel:** SELECT (Projektion) * FROM (kartesisches Produkt) Tabelle JOIN Tabelle ON Bedingung (Join) […] WHERE Bedingung (Selektion);

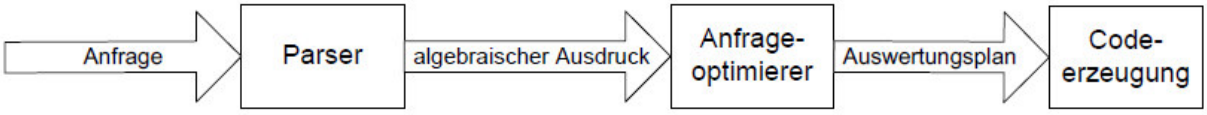
## Operatorbäume

Stellen notwendige Schritte zur Erstellung einer Abfrage für eine benutzerdefinierte Ausgabe grafisch dar.

## Funktionale Abhängigkeit

Konzept der relationalen Entwurfstheorie und Grundlage der Normalisierung. Eine Relation wird durch Attribute definiert. Funktionale Abhängigkeit besteht, wenn einige Attribute einer Relation eindeutig die Werte anderer Attribute bestimmen.
Sind X, Y Teilmengen von Attributen einer Relation, bedeutet X→Y: „X bestimmt Y“.

## Abfrageverarbeitung




## Anfrageoptimierung

- Bildet man zuerst das kartesische Produkt (FROM) aller Tabellen und schränkt dann das Produkt ein (WHERE), ergibt sich auch bei Projektion ausschließlich gewünschter Spalten (SELECT) eine enorme Menge an Blöcken (und bei 50 Blöcken pro Sekunde eine enorme Zugriffszeit bis zur Erfüllung der Anfrage).
- Selektiert man frühzeitig die nötigen Tupel und bildet danach das kartesische Produkt, verringert man die Anzahl der Tupel und beschleunigt die Anfrage deutlich.
- Der Join-Operator arbeitet am schnellsten und schränkt die Tupelanzahl bereits zu Beginn stark ein. Verwendet man ihn anstelle von Selektion und kartesischem Produkt ergibt sich eine Zugriffszeit, die auch bei einer großen Anzahl von Datensätzen nicht länger als wenige Sekunden dauert.

## Normalisierung

Schlechte Datenbankschemas enthalten Redundanzen, Unübersichtlichkeit, Inkonsistenz, Einfüge- und Löschanomalien. Normalisierung soll Redundanzen verringern und Anomalien verhindern, um die Wartung der Datenbank zu vereinfachen und die Konsistenz der Daten sicherzustellen.

- ER-Modell ist oft schon intuitiv in 3. Normalform
- Gut für Reorganisation
- Performanceverluste durch große Anzahl von Tabellen
- Widerspricht Objektorientierung

## 1. Normalform

Alle Attribute enthalten nur atomare (elementare) Werte, d.h. jedes Attribut enthält nur genau eine Information, wie z.B. Straße, PLZ, Ort. Ein Attribut Adresse würde gegen die 1. Normalform verstoßen, da hier neben der Straße sowohl die PLZ als auch der Ort innerhalb des gleichen Attributs stehen.

| **Name** | **Adresse** ✖ |
|---|---|
| Max | Musterstraße 1, 12345 Musterstadt |

| **Name** | **Straße** | **PLZ** | **Ort** ✚ |
|---|---|---|---|
| Max | Musterstraße 1 | 12345 | Musterstadt |

## 2. Normalform

Relation muss 1. Normalform erfüllen. Hier muss zudem jedes nicht-Schlüsselattribut abhängig vom Primärschlüssel sein. Es wird somit untersucht, ob ein Primärschlüssel (einzeln oder zusammengesetzt) eindeutig alle anderen Attribute eindeutig bestimmt. Hat man eine

Tabelle Hobby, bei der die zwei Attribute Schüler-Nr und Hobby-Nr den Primärschlüssel bilden, und ein nicht-Schlüsselattribut Hobby-Name, welches das Hobby beschreibt, ist die 2. Normalform nicht erfüllt. Denn Hobby-Name ist zwar abhängig von der Hobby-Nr, nicht jedoch vom gesamten Primärschlüssel, der sich aus der Hobby-Nr und der Schüler-Nr zusammensetzt. Hobby-Name müsste somit ausgelagert werden.

| Schüler-Nr | Hobby-Nr | Hobby-Name ✖ |
|---|---|---|
| 1 | 1 | Badminton |
| 2 | 1 | Badminton |

→

| Schüler-Nr | H_Nr ✚ |
|---|---|
| 1 | 1 |
| 2 | 1 |

| H_Nr | H_Name |
|---|---|
| 1 | Badminton |
| 2 | Fußball |

**3. Normalform**

Relation muss 1. & 2. Normalform erfüllen. Hier müssen zudem alle Nichtschlüsselattribute unabhängig voneinander sein. So würde beispielsweise ein Attribut Alter vom Attribut Geburtsdatum abhängen. Das wäre ein Verstoß, der durch Löschen von Alter korrigiert werden könnte (da man ein Alter problemlos über das Geburtsdatum berechnen kann). Diese Unabhängigkeit voneinander nennt man auch **transitive Abhängigkeit**.

**Constraints**

Es gibt sowohl **TABLE CONSTRAINTS als auch COLUMN CONSTRAINTS**. Constraints sind Zwangsbedingungen, die zwingend vom Wert einer Variablen erfüllt werden müssen, damit der Wert ins System übernommen werden kann. Ein Constraint erzwingt Bedingungen wie *NULL* (auch leer)*, NOT NULL* (nicht leer)*, UNIQUE* (eindeutig, automatischer Index angelegt)*, PRIMARY KEY* (einmal pro Tabelle, nicht leer und automatischer Index)*, FOREIGN KEY* (realisiert 1:n-Beziehungen zu anderer Tabelle) und *CHECK* (benutzerdefinierte Bedingung). Erstellung und Löschung über CREATE- oder ALTER TABLE.

**Datendefinitionssprache**

- Tabelle: Eine Basisrelation
- Sinn eines INDEX: verbessert den Datenzugriff durch die Vereinbarung von Schlüsseln
- VIEW: Abfrage an die Datenbank, die als virtuelle Tabelle abgelegt ist
- SEQUENCE: erzeugt fortlaufende Nummer (ORACLE spezifisch)

**Datenbankschema und Modellierung**

Ein Datenbankschema legt die Daten fest, die in der Datenbank gespeichert werden, sowie deren Beziehungen untereinander. Es ist die Summe aller Objekte eines Benutzers. Datenmodellierung beschreibt den Vorgang der Erstellung eines Schemas. Zur Modellierung verwendet man oft das Entity-Relationship-Modell in Chen-Notation, das unter Verwendung eines Datenbankmanagementsystems (DBMS) implementiert werden kann.

**Vorteile von PL/SQL**

- **Wiederverwendbarkeit:** Häufig verwendete Abläufe vieler Befehle können durch einen einzigen Aufruf immer wieder ausgeführt werden
- **Verringerte Netzlast:** weniger Datenaustausch zwischen Client und Datenbanksystem
- **Performance:** Prozeduren/Funktionen sind in kompilierter Form in DB gespeichert
- **Sicherheit:** Client benötigt keine Rechte für DELETE, UPDATE, INSERT-Zugriffe mehr

## Unterschied zwischen Prozedur und Trigger

- Trigger hängen an einer Tabelle, sind keine eigenständigen Datenbankobjekte
- Trigger können nur implizit durch eine DML-Operation ausgeführt werden.

## Cursor

Cursor verarbeiten die SELECT-Anweisungen Tupel für Tupel. Es gibt implizite Cursor, die von PL/SQL für jede SELECT-Anweisung definiert werden und explizite Cursor, die vom Programmierer deklariert werden und zur Weiterverarbeitung von SELECT-Anweisungen dienen.

## Transaktion

Reihe von Datenbank-Operationen (INSERT, UPDATE, DELETE, SELECT). Dabei wird die Datenbank konsistent gehalten, indem ein gleichzeitiges Lesen/Schreiben verhindert wird. Bei Erfolg entsteht ein Commit, bei Misserfolg ein Rollback. Eine Transaktion besitzt dabei:

## ACID-Eigenschaften

- Durability: Daten dauerhaft speichern
- Isolation: Ergebnisse wie im Single-User Betrieb
- Consistency: Nur konsistente Datenänderungen
- Atomarity: Ganz oder gar nicht ausgeführt

## Commit und Rollback

- COMMIT schreibt Datenzugriffe dauerhaft in die Datenbank
- ROLLBACK macht alle Änderungen bis zum Transaktionsanfang rückgängig

## Synonyme

Alternative Bezeichner für Datenbankobjekte. Synonyme können für Tabellen, Views, Sequenzen oder andere Synonyme erstellt werden:

```
CREATE OR REPLACE SYNONYM synonym_name FOR objektname;
```

Mit dem Schlüsselwort PUBLIC wird ein öffentliches Synonym erstellt, d.h. es wird dem Public-Benutzerkonto zugeordnet und ist für alle Benutzer zugänglich, wenn die entsprechenden Rechte des Objekts auch dem Public-Benutzerkonto zugeordnet wurden.

## Probleme bei Mehrbenutzerbetrieb

Entstehen, wenn mehrere Benutzer gleichzeitig auf den gleichen Datenbestand zugreifen.

- **Lost Update:** Eine Transaktion verändert den Wert einer anderen Transaktion, ohne zu erkennen, dass zwischen der eigenen Leseaktion und der anschließenden Änderung die Daten durch eine andere Transaktion verändert wurden.
- **Dirty Read:** Eine Transaktion liest den veränderten Wert einer anderen Transaktion, ohne zu erkennen, dass diese Daten noch nicht mit einem Commit bestätigt wurden.
- **Nicht wiederholbares Lesen:** Das Ausführen der gleichen Leseanweisung zu verschiedenen Zeitpunkten führt zu nicht wiederholbaren Ereignissen. Eine Transaktion liest Daten erneut und stellt fest, dass diese von einer anderen Transaktion verändert (UPDATE) wurden.
- **Phantom:** Ähnelt dem nicht-wiederholbaren Lesen, mit dem Unterschied, dass zwischen der wiederholten Ausführung der Leseanweisung eine andere Transaktion Daten hinzufügt (INSERT) oder löscht (DELETE). Es werden nun mehr oder weniger Datensätze zurückgeliefert.

**Lösung dieser Probleme**

Nebenläufigkeits-Kontrolle: gibt es mehrere Zugriffe auf einen Datensatz, von dem mindestens einer eine Schreiboperation ist, entsteht ein Konflikt, der über serielle Ausführungspläne (jede Transaktion vollständig hintereinander ausführen) behoben wird. Dabei gibt es sogenannte Nebenläufigkeitsprotokolle, die Regeln bilden, um serielle Transaktionen zu garantieren.

- **Zwei-Phasen Sperrprotokoll:** Hier gibt es einmal binäres Sperren mittels lock oder unlock. Das bedeutet aber auch, dass nur ein Nutzer lesen kann. Zusätzlich gibt es noch gemeinsames/exklusives Sperren, bei dem 3 Zustände (read_lock, write_lock, unlock) existieren, womit mehrere Lese-Transaktionen gleichzeitig erlaubt werden können. Realisiert werden die Sperren im Datenbanksystem mit LOCK-Tabellen. Es existieren genau 2 Phasen, in der Wachstumsphase werden nur Sperren vergeben, in der Schrumpfungsphase nur freigegeben.
- **Zeitstempelverfahren:** Neben den Sperrprotokollen gibt es noch die zeitstempelbasierte Synchronisation. Jede Transaktion bekommt einen eindeutigen Zeitstempel zugewiesen, den sie an jede Operation weitergibt. Ein Scheduler prüft nun anhand der Zeitstempel aller Transaktionen, ob eine bestimmte Transaktion in Konflikt mit einer anderen steht oder nicht, und führt so die Transaktion entweder aus oder bricht sie ab.

**Deadlock**

Belegen und Sperren 2 unterschiedliche Transaktionen unterschiedliche Datenbankobjekte, ist das in der Regel kein Problem. Benötigt nun aber Transaktion 1 Zugriff auf das Datenbankobjekt von Transaktion 2 und Transaktion 2 benötigt umgekehrt Zugriff auf Objekt 1, warten beide Transaktionen auf das Auflösen der Sperre des jeweils anderen, was jedoch von selbst nie passiert, da sie gegenseitig warten. Diese Deadlocks müssen von einem übergeordneten System aufgelöst werden.