MOBI-DBS-B: Datenbanksysteme Structured Query Language

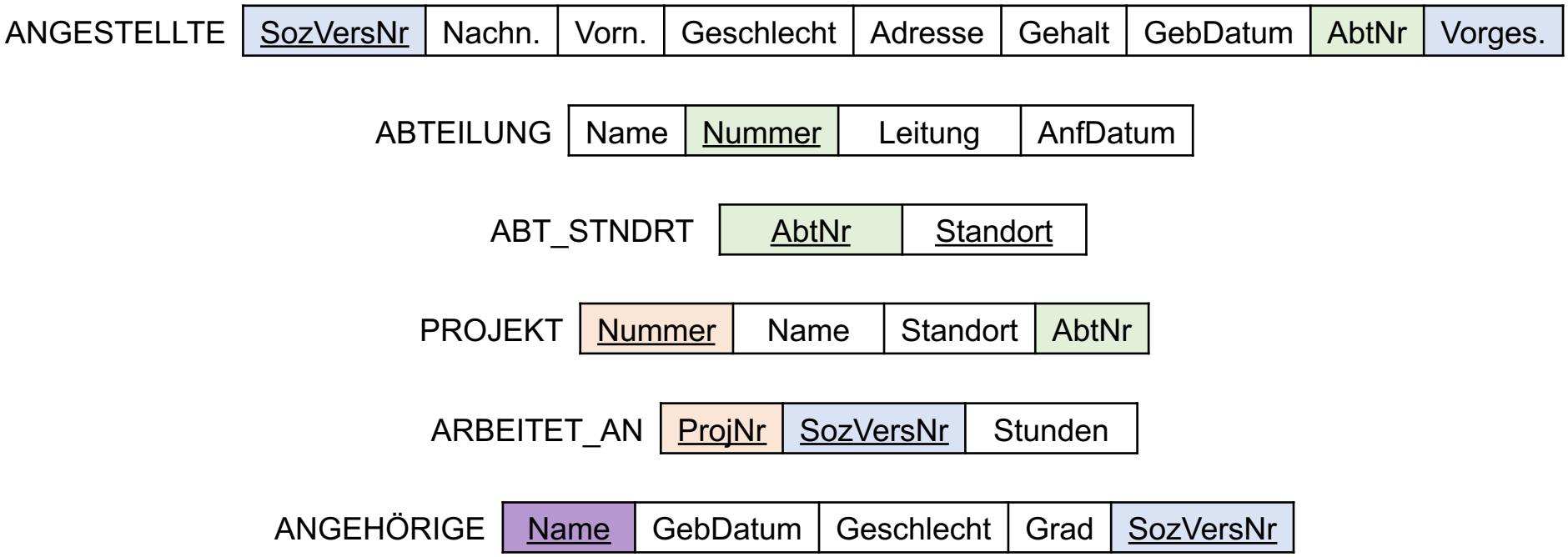
Vorlesung Sommersemester 2019

Tanya Braun, Universität zu Lübeck

Lehrauftrag SoSe 19, Universität Bamberg

Relationales Schema

- Relationales Schema (in BCNF): Firma



- Relationale Algebra: Datenmanipulation und Anfragen
  - Insert, delete, update
  - $\pi, \rho, \sigma, \cup, \cap, -, \times, \bowtie$

Structured Query Language (SQL)

Inhalte

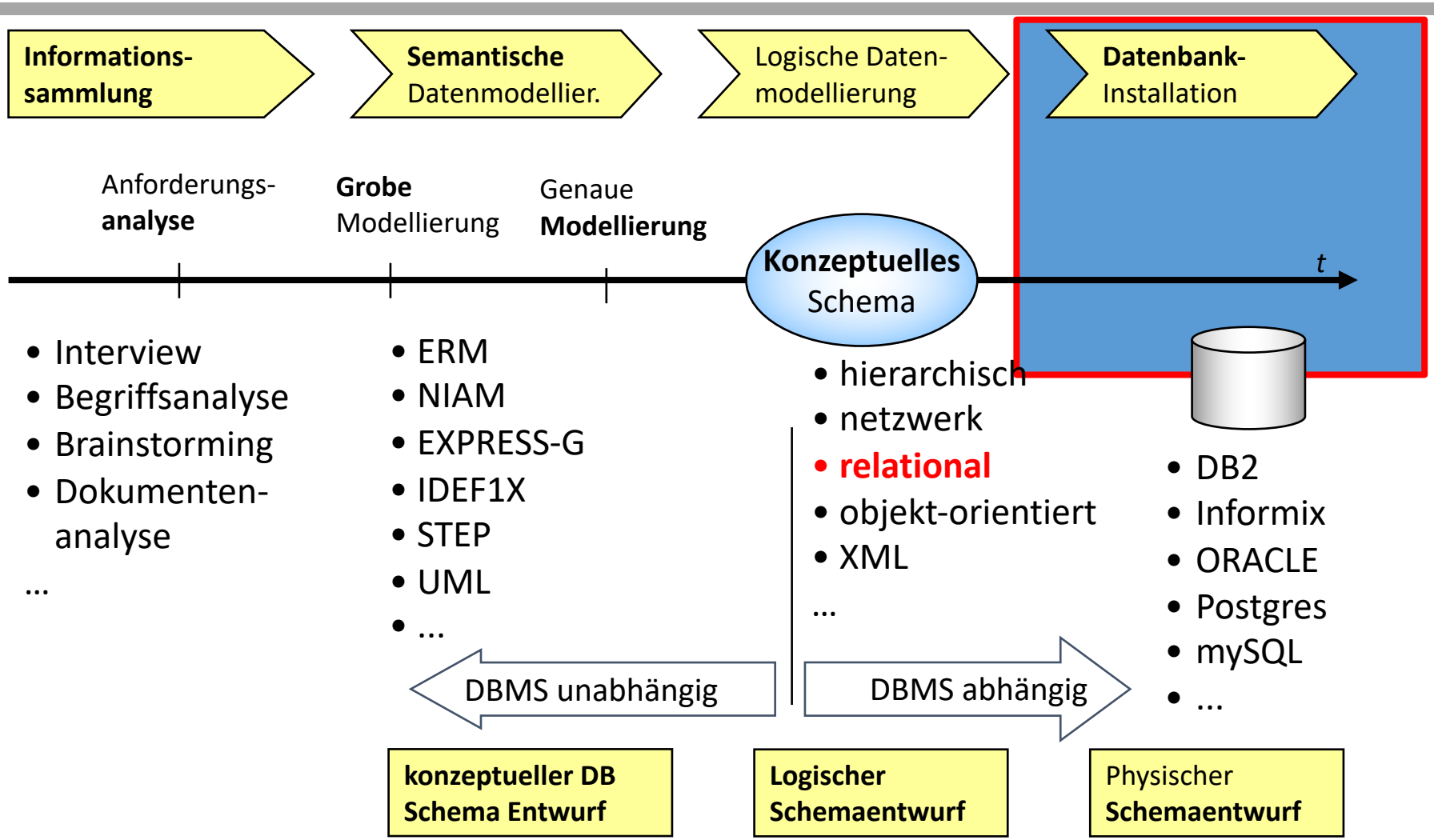
- DDL
  - Schema, table, Datentypen, Constraints
  - Drop, alter
- DCL zur Rechtevergabe
  - grant, revoke
- DML
  - Anfragen
  - Aggregationen, Gruppierungen
  - Datenmanipulationen
  - Sichten (Views)
- DB-Abstraction Layer/APIs (kurz)

Kompetenzen

- SQL-Anfragen verstehen
- SQL-Anfragen schreiben
- Nutzungsrechte verwalten
- Sichten definieren können
- SQL-Injektion verstehen (⇢PSI-IntroSP: verhindern davon)
- Die Rolle einer DB-Abstraction-Layer in einer Software verstehen
- Wiederholt:
  - Mengenorientierte Verarbeitung verstehen und anwenden

Bezug zu Phasen des DB-Entwurfs

Die Phasen des DB-Entwurfs



Übersicht

SQL

SQL - Historie

- 1974: SEQUEL
  - erster Vorschlag für die Sprache SQL
  - Entwicklung durch IBM
  - Implementierung für (experimentelles) relationales DBMS: System-R
- 1983: SQL ist de facto Standard
- 1986: SQL-86 / SQL 1
  - erster offizieller Standard durch American National Standards Institute (ANSI) und International Standards Organization (ISO)
- 1989: SQL-89
  - Revision des ersten Standards
- 1992: SQL-92 / SQL 2
  - zweite, deutlich erweiterte Revision
- 2000: SQL 3
  - mit OO-Konzepten, Multimedia, …
- Weitere Revisionen 2003, 2006 (mit XML), 2008, 2011, 2016 (aktuell)

Erinnerung: DB-Sprachen

- Definition von DBs:
  - View Definition Languages (VDLs): extern
  - Data Definition Languages (DDLs): logisch
  - Storage Definition Languages (SDLs): intern

- Zugriff auf DBs
  (Einfügen, Ändern, Löschen und Anfragen von Datensätzen):
  - Data Manipulation Languages (DMLs)
    - Einfüge-, Änderungs- und Löschoperationen: Updates
    - Reine Anfragen: „Queries“
    - Alle Zugriffsarten: „Manipulation“

- SQL : Structured Query Language
  - VDL, DDL, SDL und DML in einem

Datendefinition

SQL als DDL

Das SCHEMA Konstrukt

- SCHEMA: Namensraum in einer Datenbank
- Enthält:
  - Eindeutigen Namen
  - Autorisierungsbezeichner, um den Benutzer oder „Inhaber“ des Schemas zu identifizieren
  - Deskriptoren für jedes im Schema enthaltene Element:
    - Relationen
    - Wertebereiche
    - Einschränkungen
    - Sichten
    - Autorisierungsinformationen bzw. Zugriffsrechte
    - etc.

Definition eines SCHEMAs

- Syntax:
  - CREATE SCHEMA [SchemaName]
    [[AUTHORIZATION] Authorization]
    { SchemaElementDefinition };
- Beispiele:
  - CREATE SCHEMA Firma
    AUTHORIZATION JSmith;
  - CREATE SCHEMA Firma
    AUTHORIZATION JSmith
    CREATE TABLE Projekt;
- Kann mehrere Relationen(schemata) beinhalten
- Spätere Verwendung (und Erweiterung) über Punkt-Notation:
  - CREATE TABLE Firma.Angestellte

Katalog: INFORMATION_SCHEMA

- Erinnerung: Katalog eines DBMS
  - Beinhaltet Informationen über
    - Namen und Größe von Dateien
    - Namen und Datentypen von Datenelementen
    - Speicherdetails für jede Datei
    - Mappinginformation zwischen Schemata
    - Constraints

- SQL: Spezielles, vordefininiertes Schema INFORMATION_SCHEMA
  - enthält Metadaten zur Datenbank:
    - welche Schemata
    - welche Tabellen
    - welche Attribute
    - …
  - Können auch mit SQL abgefragt werden, z.B.:
    - SELECT * FROM INFORMATION_SCHEMA.tablename;

Definition von Relationen: CREATE TABLE

- CREATE TABLE spezifiziert eine neue Relation
- Legt fest:
  - Name der Relation
  - Name der Attribute
  - Name der Wertebereiche
    - DEFAULT-Werte zusätzlich angebbar
  - intrarelationale Einschränkungen:
    - NOT NULL, PRIMARY KEY, UNIQUE
  - interrelationale Einschränkungen:
    - FOREIGN KEY
    - CHECK (komplexe Bedingung über mehrere Relationen)
- Jede Relation ist einem Schema implizit (Defaultschema) oder explizit (Punktnotation) zugeordnet
  - Erinnerung: Verwendung (und Erweiterung) des Schemas über Punktnotation:
  - CREATE TABLE Firma.Angestellte

Beispiele

ANGESTELLTE	SozVersNr	Nachn.	Vorn.	Geschlecht	Adresse	Gehalt	GebDatum	AbtNr	Vorges.

```
CREATE TABLE Angestellte (
        SVN             CHAR(12)            NOT NULL,
        NName           VARCHAR(25)         NOT NULL,
        VName           VARCHAR(25)         NOT NULL,
        Geschlecht      CHAR,
        Adresse         VARCHAR(60),
        Gehalt          DECIMAL(10,2),
        Gdatum          DATE,
        Abt             INT                 NOT NULL    DEFAULT 1,
        VorgesSVN       CHAR(12),

        PRIMARY KEY (SVN),
        FOREIGN KEY (VorgesSVN) REFERENCES Angestellte(SVN)
);
```

Einrückung, erweiterter Leeraum dient nur der Übersicht und ist nicht erforderlich.

Beispiele

ABTEILUNG	Name	Nummer	Leitung	AnfDatum

```
CREATE TABLE Abteilung (
        AbtName         VARCHAR(25)     NOT NULL,
        AbtNummer       INT             NOT NULL        AUTO_INCREMENT,
        MgrSVN          CHAR(12)        NOT NULL,
        MgrAnfDatum     DATE,

        PRIMARY KEY (AbtNummer),
        UNIQUE (AbtName),
        FOREIGN KEY (MgrSVN) REFERENCES Angestellte(SVN)
);

ALTER TABLE Angestellte
        ADD FOREIGN KEY (Abt) REFERENCES Abteilung(AbtNummer)
```

- Vorgriff auf später: ALTER um Relationen zu ändern

Beispiele

ABTEILUNG	Name	Nummer	Leitung	AnfDatum

```
CREATE TABLE Abteilung (
        ...,
        AbtNummer       INT             NOT NULL        AUTO_INCREMENT,
        ...
);
```

ABT_STNDRT	AbtNr	Standort

```
CREATE TABLE AbtStandort (
        AbtNummer       INT             NOT NULL,
        AStandort       VARCHAR(25)     NOT NULL,

        PRIMARY KEY (AbtNummer, AStandort),
        FOREIGN KEY (AbtNummer) REFERENCES Abteilung(AbtNummer)
);
```

Beispiele

PROJEKT	Nummer	Name	Standort	AbtNr

```
CREATE TABLE Projekt (
        PName           VARCHAR(25)     NOT NULL,
        PNummer         INT             NOT NULL,
        PStandort       VARCHAR(25),
        Abt             INT             NOT NULL,

        PRIMARY KEY (PNummer),
        UNIQUE (PName),
        FOREIGN KEY (Abt) REFERENCES Abteilung(AbtNummer)
);
```

Beispiele

ARBEITET_AN	ProjNr	SozVersNr	Stunden

```
CREATE TABLE ArbeitetAn (
        PNr             INT             NOT NULL,
        SVN             CHAR(12)        NOT NULL,
        Stunden         DECIMAL(3,1),

        PRIMARY KEY (SVN, PNr),
        FOREIGN KEY (SVN) REFERENCES Angestellte(SVN),
        FOREIGN KEY (PNr) REFERENCES Projekt(PNummer)
);
```

Beispiele

- Häufig gilt: keine Umlaute!

ANGEHÖRIGE	Name	GebDatum	Geschlecht	Grad	SozVersNr

```
CREATE TABLE Angehoerige (
        AngehoerigName    VARCHAR(25)        NOT NULL,
        Geschlecht        CHAR,
        GDatum            DATE,
        Grad              VARCHAR(8),
        SVN               CHAR(12)           NOT NULL,

        PRIMARY KEY (SVN, AngehoerigName),
        FOREIGN KEY (SVN) REFERENCES Angestellte(SVN)
);
```

Vordefinierte Datentypen in SQL

- Achtung: hier große Unterschiede zwischen DBMS!
- Vordefinierte Datentypen (Basisdatentypen):
  - Numerische Typen:
    - Ganze Zahlen: INTEGER / INT, SMALLINT
    - Reelle Zahlen: FLOAT, REAL, DOUBLE PRECISION (approximativ)
      - Achtung bei Test auf Gleichheit!
    - Formatierte Zahlen: DECIMAL(i,j), NUMERIC(i,j) (exakt)
      - i und j: Dezimalstellen und Nachkommastellen
      - Beispiel:
        ... STUNDEN DECIMAL(3,1)...
  - Text-Typen
    - Zeichenketten mit fester Länge: CHAR(n)
      - Bei Input mit weniger als n Zeichen wird aufgefüllt
    - Zeichenketten mit variabler Länge: VARCHAR(n)
      - n maximale Länge; kein Auffüllen
    - In manchen DBMS in der Zwischenzeit synonym

Vordefinierte Datentypen in SQL

- Achtung: hier große Unterschiede zwischen DBMS!
- Vordefinierte Datentypen (Basisdatentypen):
  - Datum, Zeit, Zeitstempel, Intervall
    - DATE ('YYYY-MM-TT'),
    - TIME ('hh:mm:ss[.nnnnnnn] ')
    - DATETIMEOFFSET ('YYYY-MM-DD hh:mm:ss[.nnnnnnn] [+|-]hh:mm‘)
    - INTERVAL
      - YEAR, DAY, SECOND...
  - Beispiele
    - SELECT to_number(to_char(SYSDATE, 'J’))
FROM dual;
-- Julianisches Datum für Daten vor 1841, Anzahl an Tagen seit 1. Januar 4712 v.Chr.
    - SELECT to_char(to_date(1,'J'),'DD-MM-YYYY’)
FROM dual;
    - SELECT '2020-01-01' + INTERVAL 1 DAY
FROM dual;
    - dual: so genannte Default Tabelle (1 Zeile, 1 Reihe) für Anfragen, die eigentlich keine Tab elle benötigten, wenn das DBMS das FROM Konstrukt zwingend vorsieht

Benutzerdefinierte Datentypen

- Benannte Definitionen (Domains)
- Können im Schema wie Basisdatentypen verwendet werden

- Warum verwenden?
  - Bessere Lesbarkeit
  - Bessere Wartbarkeit

- Beispiel:
  - CREATE DOMAIN svn_dom AS CHAR(12)

- Zusätzlich Default-Wert möglich:
  - CREATE DOMAIN abt_name AS CHAR(25) DEFAULT "invalid";

Intrarelationale Constraints

- NOT NULL verbietet die Speicherung von NULL-Werten
  - … SVN CHAR(9) NOT NULL, …
- UNIQUE – verlangt Eindeutigkeit von Attributen (NULL max. einmal):
  - UNIQUE kann sich auf ein Attribut beziehen
    - … SVN CHAR(12) UNIQUE, …
  - oder auch auf mehrere
    - … UNIQUE(VName, NName, SVN), …

Was ist der Zusammenhang zwischen PRIMARY KEY und UNIQUE?

- PRIMARY KEY – verlangt Eindeutigkeit von Attributen (NULL nicht erlaubt):
  - PRIMARY KEY kann sich auf ein Attribut beziehen
    - … SVN CHAR(12) PRIMARY KEY, …
  - oder auch auf mehrere Attribute beziehen
    - … PRIMARY KEY(VName, NName, SVN), …
- AUTO_INCREMENT – eindeutige ID
  - Beginnt bei 1, erhöht sich automatisch mit jedem INSERT
- CHECK – ermöglicht die Formulierung komplexer Einschränkungen

Interrelationale Constraints

- REFERENCES und FOREIGN KEY ermöglichen die Spezifikation von Bedingungen zur referenziellen Integrität:
  - Für einzelne Attribute (nach der Domain-Angabe):
    - REFERENCES Relation (Attrib)
  - Für mehrere Attribute:
    - FOREIGN KEY(Attrib1, Attrib2)
      REFERENCES Relation (Attrib1, Attrib2)

- Intra- und interrelationale Constraints können eigene Namen haben
  - Damit leichter änderbar oder löschbar
    - CONSTRAINT Angest_PK PRIMARY KEY (SVN),
    - CONSTRAINT Angest_Vorgesetzt_FK
      FOREIGN KEY (VorgesSVN) REFERENCES Angestellte(SVN)
      ON DELETE SET NULL
      ON UPDATE CASCADE

    → Fehlersituationen behandeln, um referentielle Integrität sicherzustellen

Reaktion auf Constraintverletzung

- CASCADE
  - Propagiert die durchgeführte Änderung in die referenzierende Relation
- SET NULL
  - Setzt die (wertebasierte) Referenz auf NULL
- SET DEFAULT
  - Setzt die (wertebasierte) Referenz auf den für das Attribut vorgesehenen Default-Wert.
- NO ACTION (manchmal auch: RESTRICT)
  - Verbietet Änderungen in einer referenzierten Relation – solange Abhängigkeiten bestehen

- Für DELETE und UPDATE getrennt spezifizierbar:
  - ON DELETE SET NULL
    ON UPDATE CASCADE

Beispiele für Constraints (1)

```
CREATE TABLE Angestellte (
        …,
        Abt             INT             NOT NULL        DEFAULT   1,
        …,

        CONSTRAINT PK__Angest
                PRIMARY KEY (SVN),

        CONSTRAINT FK__Angest_VorgesSVN
                FOREIGN KEY (VorgesSVN) REFERENCES Angestellte(SVN)
                ON DELETE SET NULL,

        CONSTRAINT FK__Angest_Abt
                FOREIGN KEY (Abt) REFERENCES Abteilung(AbtNummer)
                ON DELETE SET DEFAULT
);
```

Beispiele für Constraints (2)

```
CREATE TABLE Abteilung (
        …,
        MgrSVN      CHAR(12)      NOT NULL      DEFAULT '000000000000',
        …,

        CONSTRAINT PK__Abt
                    PRIMARY KEY (AbtNummer),

        CONSTRAINT UQ__AbtName
                    UNIQUE (AbtName),

        CONSTRAINT FK__Abt_Mgr
                    FOREIGN KEY (MgrSVN) REFERENCES Angestellte(SVN)
                    ON DELETE SET DEFAULT
);
```

Beispiele für Constraints (3)

```
CREATE TABLE AbtStandort (
    …,
    PRIMARY KEY (AbtNummer, AStandort),
    …,
    CONSTRAINT FK__Stand_Abt_Nummer
        FOREIGN KEY (AbtNummer) REFERENCES Abteilung(AbtNummer)
        ON DELETE CASCADE
);
```

- Oder nach der Definition der Tabelle(n)

```
ALTER TABLE Angestellte
ADD CONSTRAINT
            Angest_CK CHECK (Gehalt/168 > 11.50);  -- Mindestlohn
```

Löschen von DB-Konstrukten: DROP

- DROP SCHEMA name;
  - Löscht ein Schema
- DROP TABLE table_name;
  - Löscht eine Relation
- DROP VIEW view_name;
  - Löscht eine Sicht
- ALTER TABLE table_name
  DROP COLUMN attr_name;
  - Löscht eine Spalte einer Tabelle
- ALTER TABLE table_name
  DROP CONSTRAINT constraint_name;
  - Löscht eine Einschränkung
- Liste von Objekten möglich
- Kann durch CASCADE oder RESTRICT kontrolliert werden

Ändern einer Relation: ALTER

- Mögliche Änderungen:
  - Hinzufügen eines neuen Attributs
  - Entfernen eines Attributs
  - Ändern einer Attributdefinition
  - Hinzufügen von zusätzlichen Relationeneinschränkungen
  - Entfernen von Relationeneinschränkungen
- Beispiele:
  - ALTER TABLE Firma.Angestellte
    ADD Job VARCHAR(12)
  - ALTER TABLE Firma.Angestellte
    ALTER VorgesSVN DROP DEFAULT;
  - ALTER TABLE Firma.Angestellte
    RENAME Firma.Mitarbeiter;
- Werte für neue Attribute:
  - Default-Wert, manuelle Zuweisung, NULL
- ALTER schon auf vorherigen Folien benutzt (Constraints, Drop)

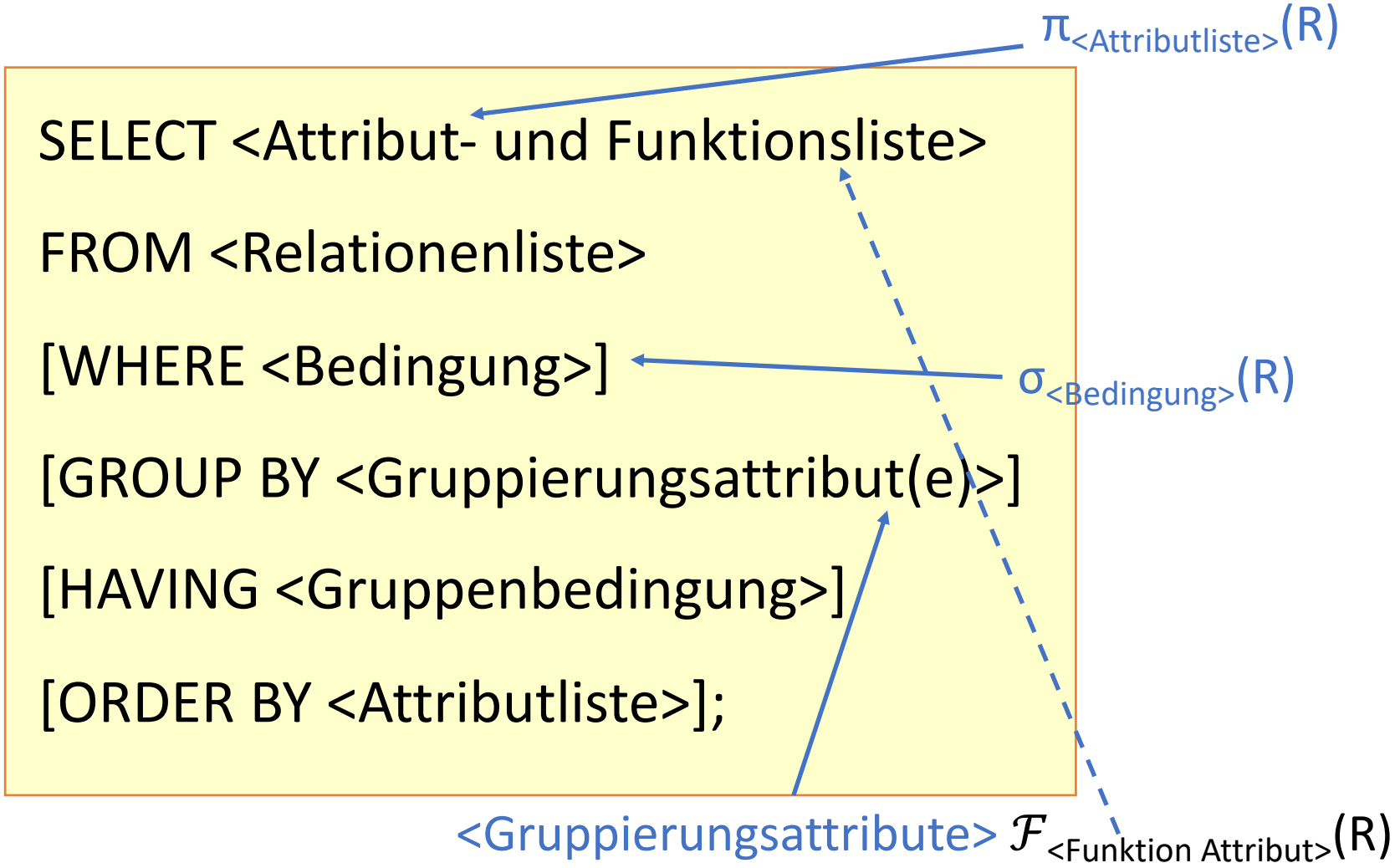
(Zwischen-) Rückblick

- Structured Query Language – SQL
  - Historie und Sprachumfang
  - DDL und DML
- DDL
  - Definition und Manipulation von DB-Schemata
  - Schema-Definition, Katalog, Definition von Relationen(-schemata)
  - Basisdatentypen und benutzerdefinierte Typen
  - Intra- und interrelationale Constraints
  - Referenzielle Integrität
  - Ändern von Schemata und Relationen(-schemata)

Anfragen inklusive Aggregation + Gruppierung

SQL als Data Manipulation Language (DML)

Anfragen: die SELECT-Anweisung



SELECT: Grundstruktur

- SELECT <Attribut- und Funktionsliste>
  FROM <Relationenliste>;

- * steht für alle Attribute einer Relation (keine Projektion)

- Beispiele:
  - Liefere alle Attributausprägungen der Relation ANGESTELLTE:
  - SELECT *
    FROM Angestellte;

  - Liefere Vornamen und Nachnamen aller Angestellter der Firma:
  - $\pi_{VNAME, NNAME}$ (ANGESTELLTE)
  - SELECT VName, NName
    FROM Angestellte;

SELECT: Grundstruktur

- SELECT <Attribut- und Funktionsliste>
  FROM <Relationenliste>;

- Relationenliste: Kartesisches Produkt der Tabellen

- Beispiele:
  - Liefere alle möglichen Kombinationen von Nachnamen und Abteilungsnamen der Firma:
  - $\pi_{\text{NNAME, ABT\_NAME}}$ (ANGESTELLTE x ABTEILUNG)
  - SELECT NName, AbtName
    FROM Angestellte, Abteilung;

- Achtung: In SQL keine automatische Duplikatseliminierung
  - Wenn gewünscht: DISTINCT
  - SELECT DISTINCT NName, AbtName …

WHERE-Klausel

- Ermöglicht Selektion von Tupeln:
  - SELECT <Attribut- und Funktionsliste>
    FROM <Relationenliste>
    [WHERE <Bedingung>];

- Beispiel:
  - Liefere Geburtsdatum und Adresse der Angestellten mit Namen John Smith:
  - $\pi_{\text{GDATUM, ADRESSE}}(\sigma_{\text{VNAME='John' AND NNAME='Smith'}}(\text{ANGESTELLTE}))$
  - SELECT Gdatum, Adresse
    FROM Angestellte
    WHERE VName='John' AND NName='Smith‘;

Uneindeutigkeit in Statements

- Tabellen/Relationen:
  - CREATE TABLE Mitglieder (
    ID INT NOT NULL,
    Name CHAR(20) NOT NULL,
    PRIMARY KEY (ID)
    );
  - CREATE TABLE Teilnahme (
    Kurs CHAR(25) NOT NULL,
    ID INT NOT NULL,
    PRIMARY KEY (Kurs, ID)
    );
- m, k: Aliases für lange Tabellennamen
  - Auch zur Übersicht bei komplexen Ausdrücken

- Anfrage:
  Liefere Namen der Mitglieder, die den Kurs 'Sport101' belegen:
  - π_{NAME} ($\sigma_{\text{COURSE='Sport101'}}$ (TEILNAHME) * MITGLIEDER)
  - SELECT Name
    FROM Mitglieder, Teilnahme
    ~~WHERE Kurs = 'Sport101' AND ID = ID;~~

```
SELECT Name
FROM Mitglieder, Teilnahme
WHERE Kurs = 'Sport101' AND
Mitglieder.ID = Teilnahme.ID;
```

```
SELECT NAME
FROM Mitglieder m, Teilnahme k
WHERE Kurs = 'Sport101' AND
m.ID = k.ID;
```

Natürlicher JOIN

JOIN in SQL

- Zwei Möglichkeiten für Join:
  - Über Kartesisches Produkt (,) und Join-Bedingung in WHERE-Klausel:
    - SELECT NName, AbtName
      FROM Angestellte, Abteilung
      WHERE AbtNummer=Abt
    - Auf vorheriger Folie schon getan: m.ID = k.ID
  - Über explizite JOIN-Klausel:
    - SELECT NName, AbtName
      FROM Angestellte JOIN Abteilung ON AbtNummer=Abt
    - auch NATURAL JOIN, RIGHT OUTER JOIN, LEFT OUTER JOIN, FULL OUTER JOIN, …
- Auch Mehrfachjoins möglich:
  - SELECT p.PNummer, p.Abt, ang.NName, ang.Adresse, ang.Gdatum
    FROM Projekt p, Abteilung abt, Angestellte ang
    WHERE p.Abt=abt.AbtNummer AND abt.MgrSVN=ang.SVN AND
    p.PStandort='Stafford';

JOIN in SQL (Forts.)

- Mehrfachjoins mit Kartesischem Produkt und Join-Bedingung in WHERE-Klausel:
  - SELECT p.PNummer, p.Abt, ang.NName, ang.Adresse, ang.Gdatum
    FROM Projekt p, Abteilung abt, Angestellte ang
    WHERE p.Abt=abt.AbtNummer AND abt.MgrSVN=ang.SVN AND
    p.PStandort='Stafford';
- Mehrfachjoins mit JOIN:
  - SELECT PNummer, Projekt.Abt, NName, Adresse, Gdatum
    FROM Projekt JOIN Abteilung ON Abt=AbtNummer
    JOIN Angestellte ON MgrSVN=SVN
    WHERE PStandort='Stafford';
  - Gleiches Ergebnis
    - (Projekt JOIN Abteilung) JOIN Angestellte
    - Projekt JOIN (Abteilung JOIN Angestellte)
    - (Projekt JOIN Angestellte) JOIN Abteilung
- “Food for thought“: Was bedeutet es für Zwischenergebnisse, wenn
  - man zuerst nach p.PStandort='Stafford' selektiert?
  - man zuerst die Joins durchführt?

Namen in SQL-Statement

- Schon eingeführt: Mehrdeutige Namen über Punkt-Notation auflösen
  - Relation Produkt(PNr, PName)
  - Relation Projekt (PNummer, PName, PStandort, Abt)
  - Relation Lieferung(PNummer, PNr, Datum)
  - SELECT Produkt.PName, Projekt.PName
    FROM Produkt JOIN Lieferung ON Produkt.PNR=Lieferung.PNr
    JOIN Projekt ON Lieferung.PNummer=Projekt.PNummer
- Problem?
  - Ergebnistabelle mit Spaltennamen PName und PName
- Umbenennung durch AS
  - SELECT Produkt.PName AS Produkt, Projekt.PName AS Projekt ...
- Auch bei Anfragen, die eine Relation mehrfach verwenden, nützlich:
  - Führe für jeden Angestellten seinen Nachnamen sowie den Nachnamen seines unmittelbaren Vorgesetzten auf:
  - SELECT a.NName AS UnterNName, v.NName AS VorgesNName,
    FROM Angestellte a, Angestellte v
    WHERE a.VorgesSVN=v.SVN;

Relationen als Mengen in SQL

- SQL behandelt Relationen als Multimengen (Mengen von Mengen)
  - Keine automatische Duplikatsentfernung
  - Wenn gewünscht: DISTINCT
- Gründe:
  - Teuer:
    - Die Entfernung von Duplikaten ist eine teure Operation. Eine Möglichkeit der Implementierung ist es, alle Tupel zunächst zu sortieren, um anschließend/dabei Duplikate zu entfernen.
  - Oft nicht nötig:
    - Viele praktische Anwendungen von Datenbanken sind dergestalt, dass Benutzer die Duplikate in der Ergebnismenge wünschen.
  - Manchmal falsch:
    - Wenn eine Aggregatfunktionen auf eine Menge von Tupel angewandt wird, soll in vielen Fällen zuvor keine Duplikatseliminierung erfolgen.

Mengenoperationen in SQL

- SQL bietet folgende Mengenoperationen an:
  - UNION realisiert die Vereinigung
  - INTERSECT realisiert den Schnitt
  - EXCEPT/MINUS realisiert die Differenz

- Ergebnis von Mengen-Operationen: Tupel-Mengen
  - d.h., Duplikate werden aus dem Ergebnis entfernt.

- Sollen Duplikate erhalten bleiben, so muss das Schlüsselwort ALL ergänzt werden:
  - UNION ALL
  - INTERSECT ALL
  - EXCEPT ALL

Beispiel für Mengenoperationen

- Erstelle eine Liste aller Projektnummern von Projekten, an denen ein Mitarbeiter mit Nachnamen 'Smith' als Mitarbeiter oder Leiter der Abteilung arbeitet, die das Projekt kontrolliert

```
(SELECT DISTINCT p.PNummer
 FROM Projekt p, ArbeitetAn arb, Angestellte ang
 WHERE
            p.PNummer=arb.PNr AND
            arb.SVN=ang.SVN AND
            ang.NName='Smith‘);
UNION
(SELECT DISTINCT p.PNummer
 FROM Projekt p, Abteilung abt, Angestellte ang
 WHERE
            p.Abt=abt.AbtNummer AND
            abt.MgrSVN=ang.SVN AND
            ang.NName='Smith')
```

Zeichenvergleiche

- Gleichheit von Substrings: LIKE und zwei Sonderzeichen/Platzhalter
  - % : beliebige Anzahl von Zeichen
  - _ : genau ein Zeichen

- Beispiele:
  - Liefere eine Liste der Vor- und Nachnamen aller Angestellten, die in Houston/Texas wohnen:
  - SELECT VName, NName
    FROM Angestellte
    WHERE Adresse LIKE '%Houston, TX%‘;

  - Liefere eine Liste der Vor- und Nachnamen aller Angestellten, deren SVN an der dritten Stelle die Ziffer 8 besitzt.
  - SELECT VName, NName
    FROM Angestellte
    WHERE SVN LIKE '_ _ 8 _ _ _ _ _ _ _ _ _';

Operatoren

- Für Zahlen: Arithmetische Operatoren (+, -, *, /)
- Für Zeichenketten: Verbindungsoperator (||)
- Für Datum, Zeit, Zeitstempel und Intervall: Plus und Minus (+, -)
- Vergleichsoperator BETWEEN für Intervall-Prüfung
- Beispiele:
  - 10% Lohnerhöhung testen:
  - SELECT ang.VName, ang.NName, 1.1*ang.Gehalt AS PlusGehalt
    FROM Angestellte ang, ArbeitetAn arb, Projekt p
    WHERE ang.SVN=arb.SVN AND arb.PNr=p.PNummer AND
    p.PName='ProductX‘;

  - E-Mail-Liste (Annahme: Angestellte haben ein Email-Attribut):
  - SELECT VName || ' ' || NName || ' <' || Email || '>'
    FROM Angestellte
    WHERE (Gehalt BETWEEN 30000 AND 40000) AND Abt=5;
  - Ausgabe bei VName='John', NName='Smith', Email='js@blub.de', Gehalt=35000, Abt=5: 'John Smith <js@blub.de>'

Disjunktiver Elementtest: IN

- Werte werden zunächst ausgelesen und dann in weiteren Vergleichsbedingungen verwendet
  - Beispiel: Formulierung der UNION-Query (vgl. Folie 43) ohne UNION.
  - SELECT DISTINCT PNummer
    FROM Projekt
    WHERE

```
PNummer IN (
        SELECT p.PNummer
        FROM Projekt p, Abteilung abt, Angestellte ang
        WHERE p.Abt=abt.AbtNummer AND
                abt.MgrSVN=ang.SVN AND ang.NName='Smith' )
        OR
PNummer IN (
        SELECT arb.PNr
        FROM ArbeitetAn arb, Angestellter ang
        WHERE arb.SVN=ang.SVN AND ang.NName='Smith' );
```

IN-Operator mit Tupeln

- Der IN-Operator kann auch mit ganzen Tupeln verwendet werden

- Beispiel:
  - Zeige die Sozialversicherungsnummern aller Angestellten, die in einer gleichen Kombination von Projekt und Stunden an einem Projekt arbeiten, an dem auch der Angestellte ‘John Smith‘ mit der SVN 01234567X890 beschäftigt ist.
  - SELECT DISTINCT SVN
    FROM ArbeitetAn
    WHERE
    (PNr, Stunden) IN (SELECT PNr, Stunden
    FROM ArbeitetAn
    WHERE SVN='01234567X890‘);

- Tupel-Ausdruck muss UNION-kompatibel zum Resultat des inneren Ausdrucks sein

Vergleichsoperatoren ANY und ALL

- Weitere Vergleichsoperatoren für ganze Mengen:
  - = ANY
    - liefert TRUE, falls v irgendeinem Wert aus V entspricht (= ANY [oder auch = SOME] ist somit äquivalent zu IN).
  - = ALL
    - liefert TRUE, falls v allen Werten aus V entspricht.
- Auch mit den Operatoren >, >=, <=, < und <> kombinierbar
  - v > ALL V liefert bspw. dann TRUE, wenn v größer ist als alle Werte der Menge V.
- Beispiel:
  - SELECT NName, VName
    FROM Angestellte
    WHERE Gehalt > ALL (SELECT Gehalt
    FROM Angestellte
    WHERE Abt=5);

Verschachtelte Anfragen

- Anfragen in Anfragen: Variablen-Sichtbarkeit
  - Liefere die Namen der Angestellten, die einen Angehörigen mit gleichem Vornamen und gleichem Geschlecht wie der Angestellte selbst haben.
  - SELECT a.NName, a.VName
    FROM Angestellte a
    WHERE a.SVN IN (SELECT fam.SVN
    FROM Angehoerige fam
    WHERE a.VName=fam.AngehoerigName AND
    a.Geschlecht=fam.Geschlecht
- Korrelierte Anfrage: innere Anfrage referenziert auf äußere Anfrage
- Semantik:
  Innere Anfrage wird einmal für jedes Tupel / jede Tupelkombination ausgewertet

EXISTS-Test

- Überprüft, ob das das Resultat einer korrelierten verschachtelten Anfrage leer (FALSE) ist – also kein Tupel enthält – oder nicht (TRUE).
- Kann auch mit NOT kombiniert werden
- Beispiel:
  - Liefere die Namen der Angestellten, die einen Angehörigen mit gleichem Vornamen und gleichem Geschlecht wie der Angestellte selbst haben.
  - SELECT a.NName, a.VName
    FROM Angestellte a
    WHERE EXISTS (SELECT fam.SVN
    FROM Angehoerige fam
    WHERE a.SVN=fam.SVN AND
    a.VName=fam.AngehoerigName AND
    a.Geschlecht=fam.Geschlecht);

UNIQUE zum Duplikattest

- Überprüft, ob eine Multimenge Duplikate enthält (FALSE) oder nicht (TRUE).

- Beispiel:
  - Liefere die Namen der Angestellten, die einen eindeutigen Vornamen haben.
  - SELECT a.NName, a.VName
    FROM Angestellte a
    WHERE UNIQUE (SELECT b.VName
    FROM Angestellte b
    WHERE a.VName=b.VName);

Junktoren bei Mengenvergleichen

- Mengenvergleiche können durch Junktoren AND und OR miteinander verknüpft werden (mehrere korrelierte Anfragen).

- Beispiel:
  - Erstelle eine Liste mit den Namen der Manager, die mindestens einen Angehörigen haben.
  - SELECT a.NName, a.VName
    FROM Angestellte a
    WHERE EXISTS (SELECT *
    FROM Angehoerige fam
    WHERE a.SVN=fam.SVN)
    AND
    EXISTS (SELECT *
    FROM Abteilung abt
    WHERE a.SVN=abt.MgrSVN);

Explizite Mengenangaben

- Menge explizit angeben und in der WHERE-Klausel verwenden

- Beispiel:
  - Gib die Sozialversicherungsnummern aller Angestellten aus, die an Projekten mit den Nummern 1, 2 oder 3 arbeiten.
  - SELECT DISTINCT SVN
    FROM ArbeitetAn
    WHERE PNr IN (1, 2, 3);

NULL-Test

- Prüft, ob der Wert eines Attributs NULL ist
- Hier kein = möglich! (IS NULL statt = NULL)

- Beispiel:
  - Liefere die Namen der Angestellten, die keinen Vorgesetzten haben.
  - SELECT NName, VName
    FROM Angestellte
    WHERE VorgesSVN IS NULL;

  - Liefere die Namen der Angestellten, die einen Vorgesetzten haben.
  - SELECT NName, VName
    FROM Angestellte
    WHERE VorgesSVN IS NOT NULL;

Aggregatfunktionen

- Funktionen zur Aggregation von Daten:
  - COUNT() … liefert die Anzahl der Tupel
    - COUNT(DISTINCT()) liefert die Anzahl der unterschiedlichen Tupel
  - SUM() … liefert die Summe der Werte der Tupelattribute
  - MIN() … liefert den Wert des minimalen Tupelattributs
  - MAX() … liefert den Wert des maximalen Tupelattributs
  - AVG() … liefert den durchschnittlichen Wert der Tupelattribute
    - AVG(DISTINCT()) liefert den Durchschnitt der unterschiedlichen Tupel
- Beispiel:
  - Liefere die Summe der Gehälter aller Angestellten, das maximale Gehalt, das durchschnittliche Gehalt und das minimale Gehalt.
    SELECT SUM(Gehalt), MAX(Gehalt), AVG(Gehalt), MIN(Gehalt)
    FROM Angestellte;
- Hier keine weiteren Attribute in SELECT
  - Nächste Folie: GROUP BY
  - Ermöglicht weitere, sinnvolle Attribute in SELECT

Gruppierung mit GROUP BY

- In der Praxis: Aggregatfunktionen häufig auf Gruppen von Tupeln
- Beispiel:
  - Liefere für jede Abteilung die Abteilungsnummer, die Anzahl der dort arbeitenden Mitarbeiter und deren Durchschnittsgehalt.
  - SELECT Abt, COUNT(*), AVG(Gehalt)
    FROM Angestellte
    GROUP BY Abt;

VName	NName	SVN	...	Gehalt	VorgesSVN	Abt
John	Smith	01234567X890		30000	12345678Y901	5
Franklin	Wong	12345678Y901		40000	78901234E567	5
Ramesh	Narayan	23456789Z012		38000	12345678Y901	5
Joyce	English	34567890A123		25000	12345678Y901	5
Alicia	Zelaya	45678901B234	...	25000	56789012C345	4
Jennifer	Wallace	56789012C345		43000	78901234E567	4
Ahmad	Jabbar	67890123D456		25000	56789012C345	4
James	Bong	78901234E567		55000	null	1

Abt	COUNT(*)	AVG(Gehalt)
5	4	33250
4	3	31000
1	1	55000

Gruppierung nach JOIN

- Aggregation und Gruppierung werden nach einem JOIN angewendet
- Beispiel:
  - Liefere für jedes Projekt die Projektnummer, den Projektnamen und die Anzahl der daran arbeitenden Angestellten.
  - SELECT PNummer, PName, COUNT(*)
    FROM Projekt, ArbeitetAn
    WHERE PNummer=PNr
    GROUP BY PNummer, PName;

  → Erst JOIN, dann Gruppierung, dann Aggregat (COUNT)

Bedingungen auf Gruppierungen: HAVING

- HAVING wählt Gruppen aus
  - Entspricht WHERE auf Einzel-Tupeln
- Beispiel:
  - Liefere für jedes Projekt, an dem mehr als zwei Angestellte arbeiten, die Projektnummer, den Projektnamen und die Anzahl der daran jeweils arbeitenden Angestellten.
  - SELECT PNummer, PName, COUNT(*)
    FROM Projekt, ArbeitetAn
    WHERE PNummer=PNr
    GROUP BY PNummer, PName
    HAVING COUNT(*) > 2;

Beispiel: HAVING

PName	PNummer	...	SVN	Stunden
ProduktX	1		01234567X890	32,5
ProduktX	1		34567890A123	20,0
ProduktY	2		01234567X890	7,5
ProduktY	2		34567890A123	20,0
ProduktY	2		12345678Y901	10,0
ProduktZ	3		89012345F678	40,0
ProduktZ	3		12345678Y901	10,0
Computerisation	10		12345678Y901	10,0
Computerisation	10	...	45678901B234	10,0
Computerisation	10		67890123D456	35,0
Reorganisation	20		12345678Y901	10,0
Reorganisation	20		56789012C345	15,0
Reorganisation	20		78901234E567	null
Newbenefits	30		67890123D456	5,0
Newbenefits	30		56789012C345	20,0
Newbenefits	30		45678901B234	30,0

```
SELECT PNummer, PName, COUNT(*)
FROM Projekt, ArbeitetAn
WHERE PNummer=PNr
GROUP BY PNummer, PName
HAVING COUNT(*) > 2;
```

Diese Gruppen fliegen raus!

Nach JOIN und Anwendung der GROUP BY Klausel

Beispiel: HAVING - Ergebnisse

```
SELECT PNummer, PName, COUNT(*)
FROM Projekt, ArbeitetAn
WHERE PNummer=PNr
GROUP BY PNummer, PName
HAVING COUNT(*) > 2;
```

PName	PNummer	...	SVN	Stunden
ProduktY	2		01234567X890	7,5
ProduktY	2		34567890A123	20,0
ProduktY	2		12345678Y901	10,0
Computerisation	10		12345678Y901	10,0
Computerisation	10		45678901B234	10,0
Computerisation	10		67890123D456	35,0
Reorganisation	20	...	12345678Y901	10,0
Reorganisation	20		56789012C345	15,0
Reorganisation	20		78901234E567	null
Newbenefits	30		67890123D456	5,0
Newbenefits	30		56789012C345	20,0
Newbenefits	30		45678901B234	30,0

PNummer	COUNT(*)
2	3
10	3
20	3
30	3

Nach COUNT und SELECT
(PName nicht dargestellt)

Nach Anwendung der HAVING Klausel

Weitere Beispiele für Gruppierung

- Liefere für jedes Projekt die Nummer, den Namen und die Anzahl der Angestellten, die aus Abteilung 5 an dem betreffenden Projekt arbeiten
- SELECT PNummer, PName, COUNT(*)
  FROM Projekt p, ArbeitetAn arb, Angestellte ang
  WHERE p.PNummer=arb.PNr AND arb.SVN=ang.SVN AND ang.Abt=5
  GROUP BY PNummer, PName;

- Liefere für jede Abteilung mit mehr als fünf Angestellten die Nummer und die Anzahl der Angestellten, die mehr als 40.000 verdienen.
- SELECT Abt, COUNT(*)
  FROM Angestellte
  WHERE Gehalt>=40000 AND Abt IN (SELECT Abt
  FROM Angestellte
  GROUP BY Abt
  HAVING COUNT(*) >= 5)
  GROUP BY Abt;

Ordnen durch ORDER BY

- Resultatmenge kann sortiert werden
- Auch mehrere Sortierkriterien möglich
- ASC: aufsteigend, DESC: absteigend

- Beispiele:
  - SELECT AbtName, NName, VName, PName
    FROM Abteilung abt, Angestellte ang, ArbeitetAn arb, Projekt p
    WHERE abt.AbtNummer=ang.Abt AND ang.SVN=arb.SVN AND arb.PNr=p.PNummer
    ORDER BY AName, NName, VName;

  - SELECT AbtName, NName, VName, PName
    FROM Abteilung abt, Angestellte ang, ArbeitetAn arb, Projekt p
    WHERE abt.AbtNummer=ang.Abt AND ang.SVN=arb.SVN AND arb.PNr=p.PNummer
    ORDER BY AName DESC, NName ASC, VName ASC;

Datenmanipulation: INSERT, DELETE, UPDATE

SQL als DML (Forts.)

INSERT zum Tupel-Einfügen

- INSERT benötigt:
  - Ziel-Relation
  - Ggf. Attribute
    - Sonst Reihenfolge wie in Schemadefinition
  - Werteliste (oder Anfrage, die diese produziert)

- Beispiele:
  - INSERT INTO Angestellte
    VALUES ('90123456G789 ', 'Marini', 'Richard', 'M‘,
    '98 Oak Forest, Katy, TX', 37000, '30.12.1962', 4, '67890123D456');
  - INSERT INTO Angestellte (VName, NName, Abt, SVN)
    VALUES ('Richard', 'Marini', 4, '90123456G789');
  - INSERT INTO Abteilung(AbtName, MgrSVN, MgrAnfDatum)
    VALUES ('Facilities', '90123456G789', '2019-06-14‘);
    - AbtNummer AUTO_INCREMENT
      → Keinen Wert angeben, wird automatisch eingefügt

INSERT aus anderer Tabelle

- Neue Tabelle:
  - CREATE TABLE AbtInfo (
    AbtName VARCHAR(15),
    AnzahlAngest INTEGER,
    GehaltGesamt INTEGER
    );

- Einfügen von Daten:
  - INSERT INTO AbtInfo (AbtName, AnzahlAngest, GehaltGesamt)

```
SELECT AName, COUNT(*), SUM(Gehalt)
FROM (Abteilung JOIN Angestellte ON AbtNummer=Abt )
GROUP BY AName;
```

Abweisung von INSERT

- Wenn DB-Integrität verletzt wird
- Mögliche Fehlerquellen
  - Kein Primärschlüssel angegeben
  - Fremdschlüssel existiert nicht in Zieltabelle
  - Kein Wert angegeben trotz NOT NULL
  - CHECK-Constraint verletzt
  - Doppelter Wert trotz UNIQUE

Löschen von Tupeln: DELETE

- DELETE benötigt:
  - Name der Relation, aus der gelöscht werden soll
  - WHERE-Klausel, die bestimmt, welche Tupel gelöscht werden sollen
- Darf ebenfalls Integrität nicht verletzen
  - Aktionen zur Behandlung (CASCADE, NO ACTION, SET DEFAULT, SET NULL)
- Beispiele:
  - DELETE FROM Angehoerige;
  - DELETE FROM Angehoerige
    WHERE SVN = '34567890A123';
  - DELETE FROM Angehoerige
    WHERE SVN IN (SELECT SVN
    FROM Angestellte
    WHERE ABT = 4);
    - Test ob (noch) zu löschende Einträge vorhanden sind:
      SELECT * FROM Angehoerige
      WHERE SVN IN (SELECT SVN
      FROM Angestellte
      WHERE ABT = 4);

Aktualisieren von Tupeln: UPDATE

- UPDATE benötigt:
  - Name der Relation
  - SET – Anweisung für die Änderung
  - WHERE-Anweisung für die zu ändernden Tupel
- SET kann auch Berechnungen enthalten
- Reihenfolge mehrerer UPDATE-Anweisungen ist relevant!
- Beispiele:
  - UPDATE Projekt
    SET PStandort='Bellaire‘, AbtNr=5
    WHERE PNummer=5;
  - UPDATE Angestellte
    SET Gehalt=Gehalt * 1.1
    WHERE Abt IN (SELECT AbtNummer
    FROM Abteilung
    WHERE AName = 'Research');

Merge von Tabellen

- Eine Tabelle kann in eine andere eingefügt werden, wobei hier im WHEN MATCHED über das Einfügen bestimmt
- Beispiel:
  - MERGE INTO CopyAngestellte c
    USING Angestellte a
    ON (a.SVN = c.SVN)
    WHEN MATCHED THEN
        UPDATE
        SET
        c.VName = a.VName,
        c.NName = a.NName,
        c.Gdatum = a.Gdatum,
        ...
    WHEN NOT MATCHED THEN
        INSERT VALUES(a.SVN, ..., e.Abt);

Zwischenrückblick

- DML
  - Anfragen
    - SELECT
    - FROM
    - WHERE
    - ORDER BY
  - Konstrukte in / zwischen Anfragen
    - JOIN
    - Operatoren (+ - % / * IN LIKE…)
    - UNION
    - IN, ANY, ALL, EXIST, UNIQUE
    - DISTINCT
    - IS NULL / IS NOT NULL
  - Aggregationen und Gruppierungen
    - COUNT, SUM, AVG, MIN, MAX
    - GROUP BY
    - HAVING
  - Sortierung
    - ORDER BY
    - ASC, DESC
  - Datenmanipulationen
    - INSERT
    - UPDATE
    - DELETE

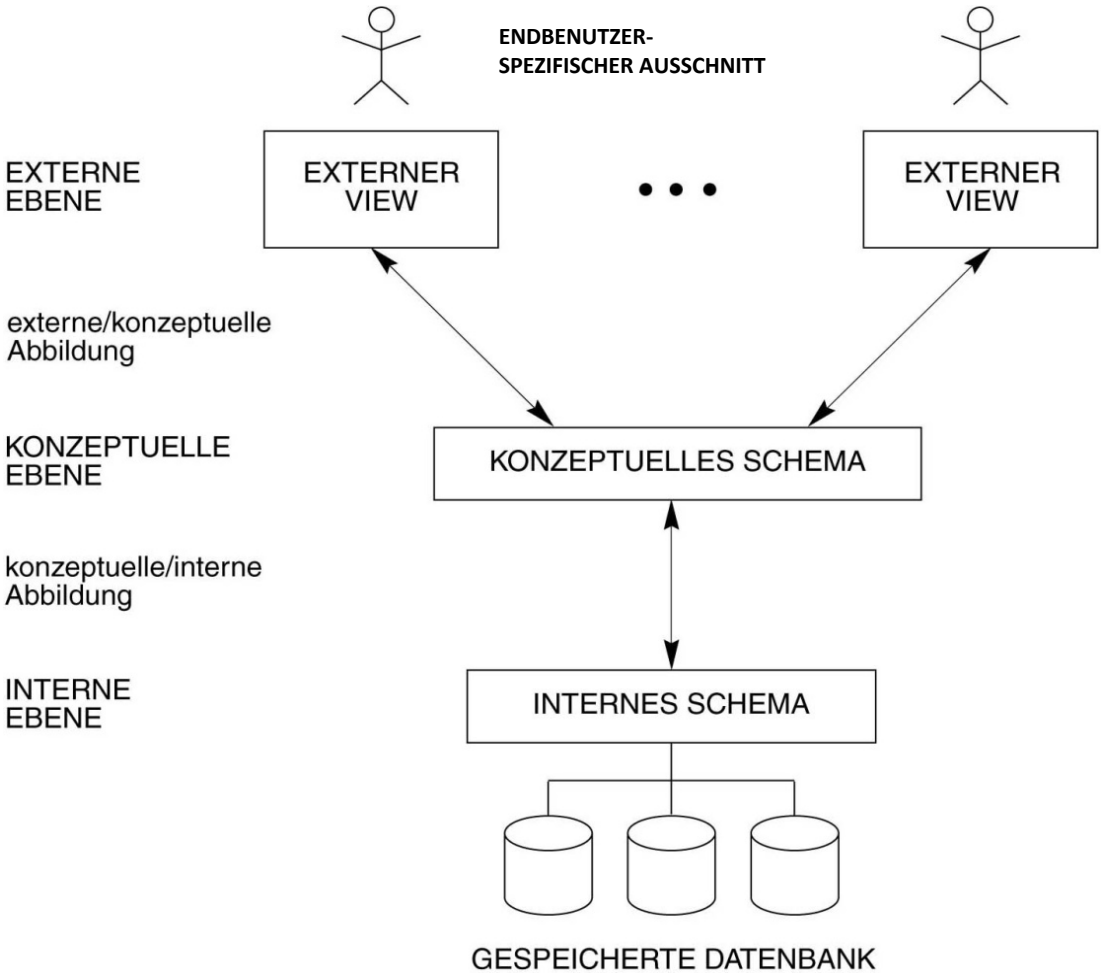
Sichten (Views)

SQL als View Definition Language (VDL)

(Machmal auch als Teil der DML aufgefasst)

Drei-Schichten-Architektur (DSA)

- Datenunabhängigkeit – Details vor anderen Schichten verbergen



VIEWS / virtuelle Relationen

- VIEW: aus anderen Relationen abgeleitete Relation
- Wird über eine SELECT-Anweisung spezifiziert
- Werden von DB aktuell gehalten
- Virtuelle Relation: kann, muss aber nicht in der Datenbank abgespeichert werden
- Beispiel:
  - CREATE VIEW ArbeitetAnMitNamen AS
    SELECT SVN, VName, NName, PName, Stunden
    FROM Angestellte ang, Projekt p, ArbeitetAn arb
    WHERE ang.SVN=arb.SVN AND arb.PNr=p.PNummer;
- VIEWs können für Abfragen wie normale Relationen genutzt werden
- Beispiel
  - SELECT VName, NName, Stunden
    FROM ArbeitetAnMitNamen
    WHERE SVN = '01234567X890';

Verwendung von VIEWs

- Aktualisierungen/Datenmanipulationen häufig nicht möglich
  - Non-updateable views
- Beispiel: Nutzung von Aggregationsfunktionen:
  - CREATE VIEW AbtInfo AS
    SELECT Abt, COUNT(*) AS Cnt, AVG(Gehalt) as AvgG
    FROM Angestellte
    GROUP BY Abt;
  - INSERT INTO AbtInfo VALUES (1,5,20000) ???
  - UPDATE AbtInfo SET Cnt = 5 WHERE Abt = 5 ???
- Beispiel: NOT NULL Attribut ohne DEFAULT nicht Teil des Views
  - Kein INSERT

Rechte

SQL als Data Control Language (DCL)

DCL

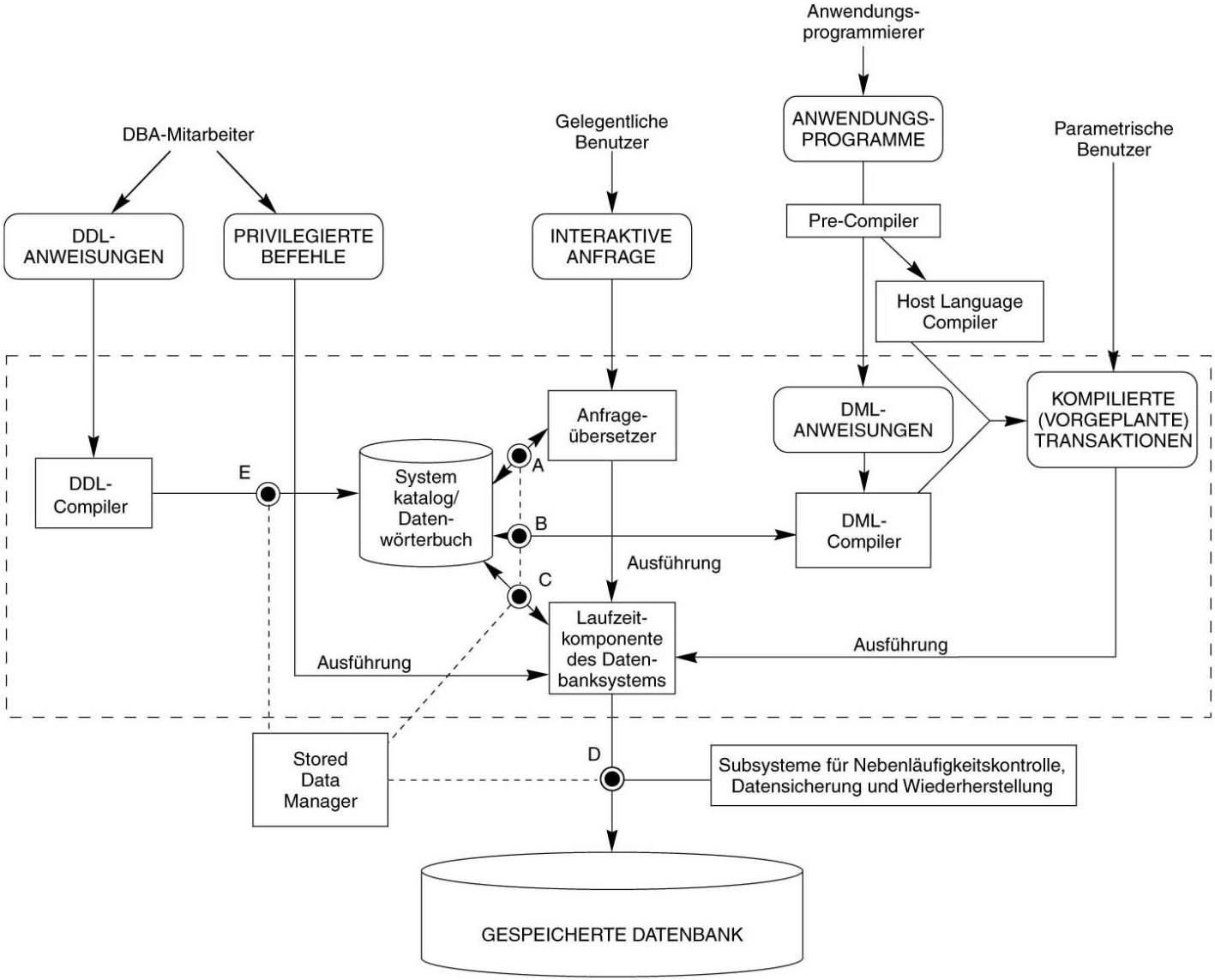
- Rechtevergabe an DB-Objekten
- Benutzerprivilegien:
  - Lesen oder Ändern von Relationen oder Spalten
  - Anlegen von Relationenschemata oder Datenbankschemata
  - Weitergabe von Privilegien
- Rechte mittels
  - GRANT (ein ausgewähltes Recht) geben
  - REVOKE (entsprechendes Recht) wieder entziehen
    - SELECT, UPDATE, DELETE, INSERT aber auch
    - EXECUTE, ALTER, CREATE, MANAGE,….., etc
- Beispiele
  - GRANT SELECT, UPDATE ON Nutzer.Tabelle
    TO AndererNutzer;
  - GRANT INSERT, SELECT, DELETE ON Angestellter
    TO joerg, sabine, harald WITH GRANT OPTION;
  - GRANT UPDATE(Gehalt) ON Angestellter
    TO chefe;

REVOKE UPDATE(Gehalt) ON Angestellte
FROM chefe;

Programmiermethoden

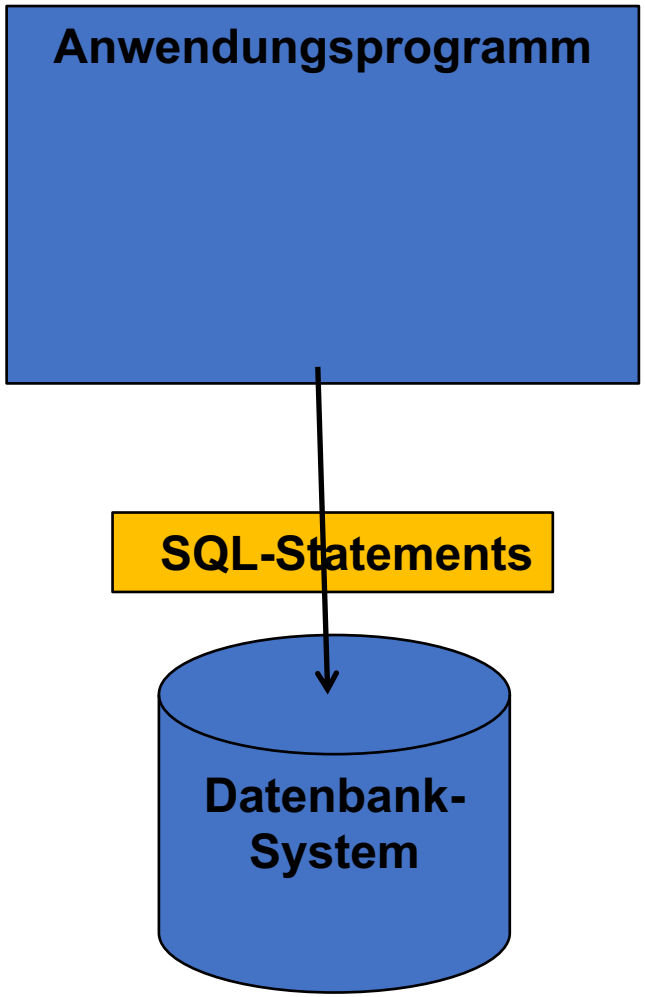
Benutzung von SQL

DBMS-Systemumgebung



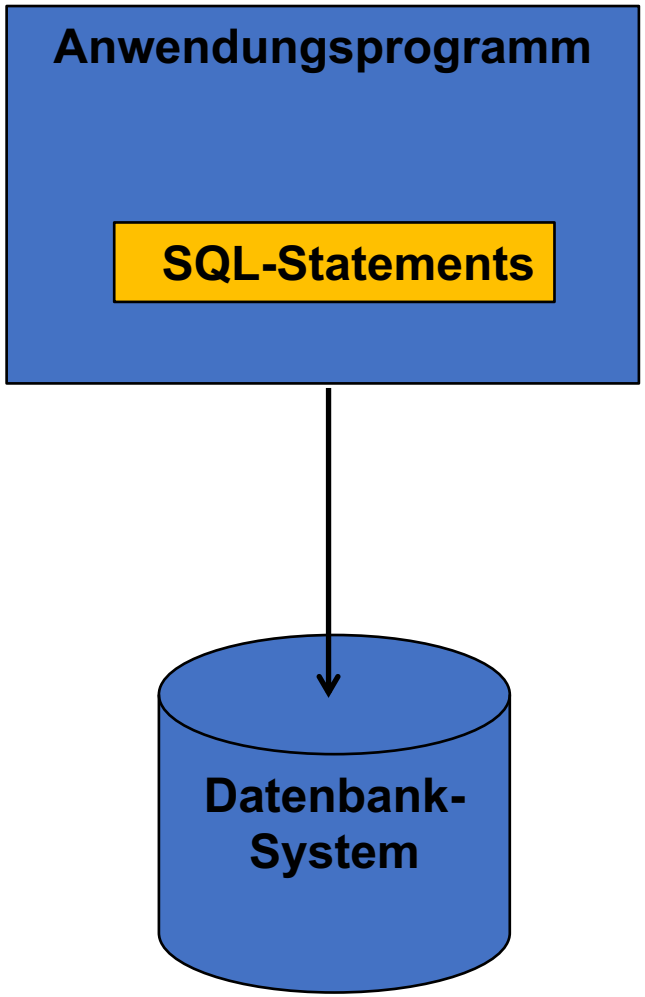
SQL Programmier-Methoden

- Direkter Aufruf
  - von SQL an die Datenbank



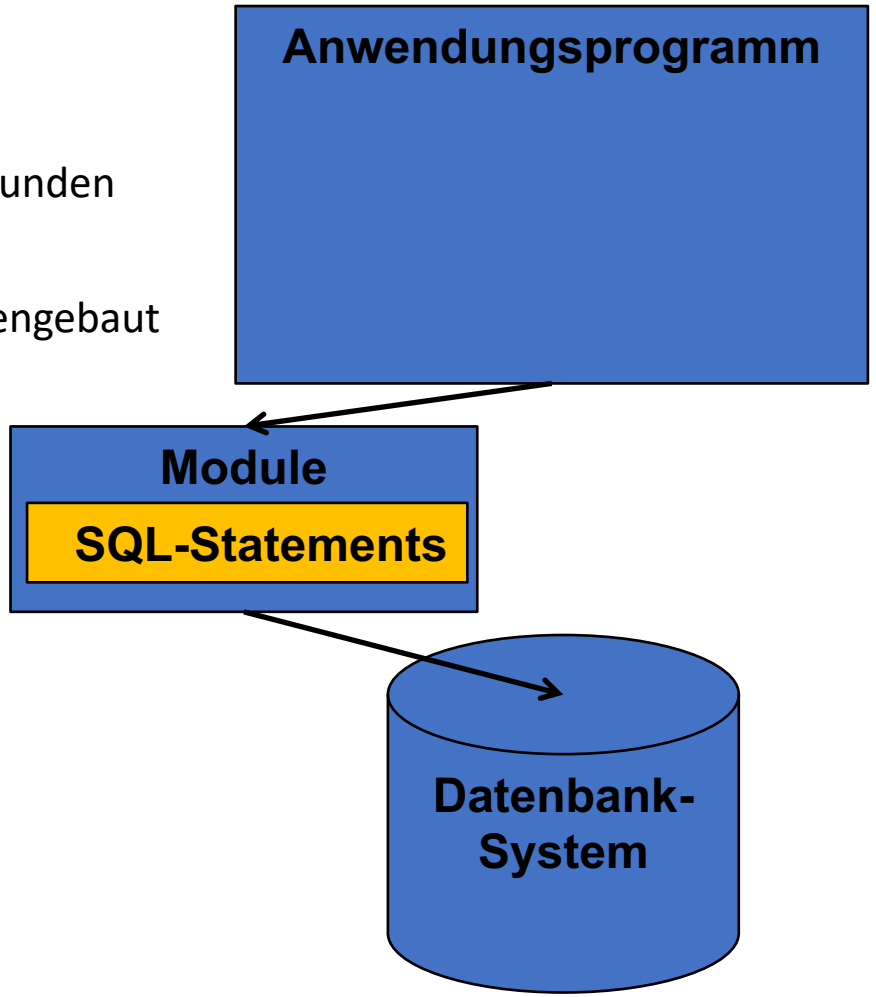
SQL Programmier-Methoden

- Direkter Aufruf
  - von SQL an die Datenbank
- Embedded SQL and SQLJ
  - SQL wird in die Host-Sprache eingebunden
- Dynamic SQL
  - wird zur Programmlaufzeit zusammengebaut
  - DB soll trotzdem vorbereitet sein!



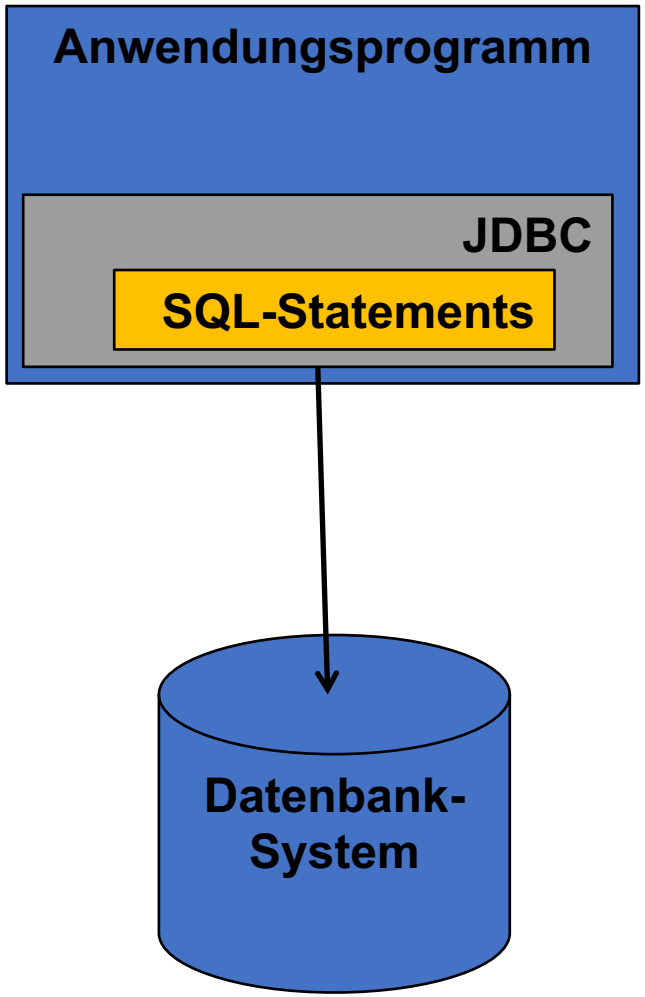
SQL Programmier-Methoden

- Direkter Aufruf
  - von SQL an die Datenbank
- Embedded SQL and SQLJ
  - SQL wird in die Host-Sprache eingebunden
- Dynamic SQL
  - wird zur Programmlaufzeit zusammengebaut
  - DB soll trotzdem vorbereitet sein!
- Module Language
  - SQL wird in Module ausgelagert, die von Host-Sprache aus angefragt werden



SQL Programmier-Methoden

- Direkter Aufruf
  - von SQL an die Datenbank
- Embedded SQL and SQLJ
  - SQL wird in die Host-Sprache eingebunden
- Dynamic SQL
  - wird zur Programmlaufzeit zusammengebaut
  - DB soll trotzdem vorbereitet sein!
- Module Language
  - SQL wird in Module ausgelagert, die von Host-Sprache aus angefragt werden
- Call-Level APIs:
  - Schnittstellen, um aus der Host-Sprache Datenbanken anzusprechen
    - SQL/CLI, ODBC, JDBC
- Mappings
  - Programmierer sieht kein SQL mehr
  - Datenbankschema wird erzeugt

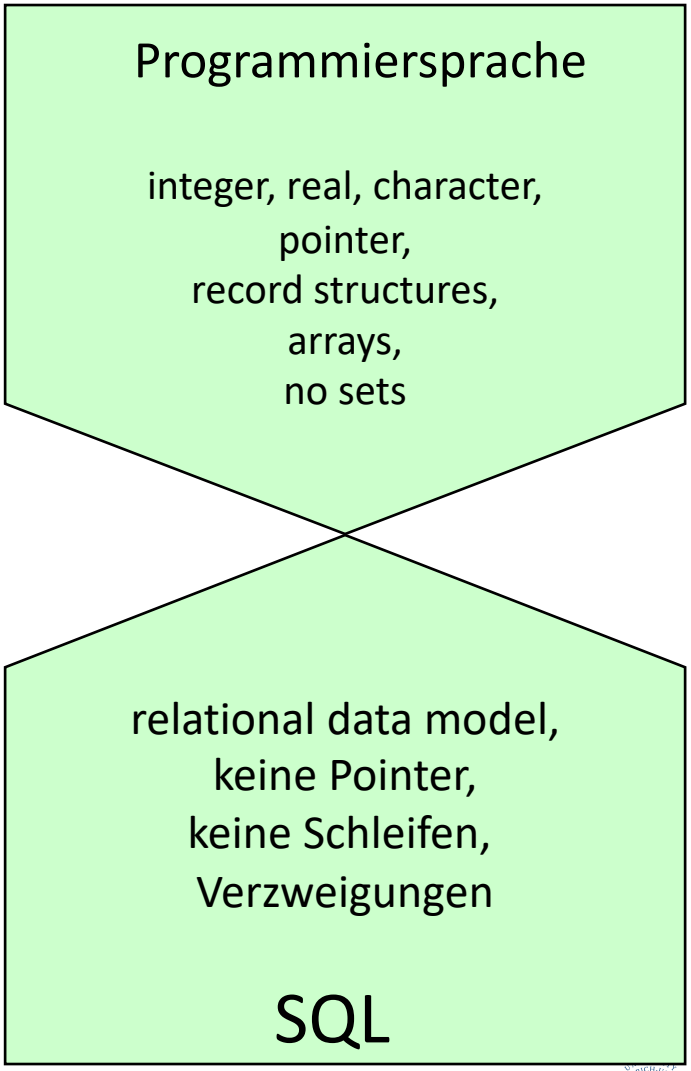


Impedance Mismatch

- Problem: Datenzugriff unterscheidet sich zwischen SQL und anderen Programmiersprachen
  - SQL: mengenorientiert
  - Andere: verarbeiten einzelne Werte

→ sog. "Impedance Mismatch"

- Relationales Datenmodell wird von den meisten Programmiersprachen nicht unterstützt



Embedded SQL: "Shared" Variablen

- Transferieren Informationen zwischen Datenbank und Anwendungsprogramm
- Werden in einem DECLARE Abschnitt deklaiert:

```
EXEC SQL BEGIN DECLARE SECTION;
...
EXEC SQL END DECLARE SECTION;
```

  - "…" hängt von der Programmiersprache ab
- Können im SQL-Statement statt einer Konstante verwendet werden
- Variablen-Name wird mit Doppelpunkt gekennzeichnet
- Spezielle Variable SQLSTATE enthält Fehlercodes
  - '00000': no error condition, '02000': no tuple found

Beispiel: Shared Variable mit INSERT

```
void setParts() {
   EXEC SQL BEGIN DECLARE SECTION;
      char part[4], project[4],
      version[10], description[50];
      char SQLSTATE[6];
   EXEC SQL END DECLARE SECTION;

   /* request part, project, version, description */

   EXEC SQL INSERT INTO parts(partno, version,
                        projectno, part_description)
         VALUES (:part, :version, :project,
:description);
}
```

- Um Daten aus der DB in die Variablen zu bekommen:
  - EXEC SQL SELECT INTO :part, :version, :project, :description FROM parts;

Beispiel: Single-Row SELECT Statements

```
void getNumProjects(int minBudget) {
   EXEC SQL BEGIN DECLARE SECTION;
      int num, budget;
      char SQLSTATE[6];
   EXEC SQL END DECLARE SECTION;

   budget := minBudget;

   EXEC SQL    SELECT COUNT(*)
               INTO :num
               FROM projects
               WHERE budget >= :budget;

   /* check that SQLSTATE has all 0's    */
   /* and if so print the value of num   */
   if SQLSTATE == '00000':
      return num;
}
```

Cursor

- Konzept, um durch Ergebnismenge zu navigieren
- 4 Schritte, um einen Cursor zu nutzen:
  1. Cursor Deklaration:
     EXEC SQL DECLARE <cursor> CURSOR FOR <query>
     - <cursor> : Name des Cursor; <query> : SQL-Ausdruck
  2. Cursor Initialisierung:
     EXEC SQL OPEN <cursor>
     - Initialisiert den Cursor vor dem ersten Tupel, Anfrage wird ausgeführt
  3. Tupel holen:
     EXEC SQL FETCH FROM <cursor> INTO <variables>
     - Holt das nächste Tupel und schreibt es in die <variables>
     - kann mehrfach ausgeführt werden
     - wenn keine Tupel mehr da sind: SQLSTATE = '02000'.
  4. Cursor schließen:
     EXEC SQL CLOSE <cursor>

Beispiel: Cursor

```
void getAllProjects() {
   EXEC SQL BEGIN DECLARE SECTION;
      char project[4], description[50];
      char SQLSTATE[6];
   EXEC SQL END DECLARE SECTION;
   EXEC SQL DECLARE execCursor CURSOR FOR
      SELECT projectno, description
      FROM projects;

   EXEC SQL OPEN CURSOR execCursor;
   while (1) {
      EXEC SQL FETCH FROM execCursor
         INTO :project, :description;
      if (!(strcmp(SQLSTATE, "02000")) break;
      printf("projectno: %s, description: %s",
               project, description);
   }
   EXEC SQL CLOSE execCursor;
}
```

SQLJ (deprecated)

- Standard für statische SQL-Statements in Javaprogrammen
- Wurde von einer informellen Gruppe von Firmen definiert (IBM, Informix, Microsoft, Oracle, Sun, Sybase, ...).
- Besteht aus drei Teilen:
  - Part 0: Embedded SQL in Java
  - Part 1: SQL Routines using Java
  - Part 2: SQL Types using Java
- Wurde teilweise in den SQL-Standard übernommen
  - Part 0: SQL - Part 10: Object Language Bindings (SQL/OLB)

(J. Melton, A. Eisenberg: Understanding SQL and Java Together, 2000)

(ISI/IEC 9075:1999, Information Technology - Database languages - SQL - Part 10: Object Language Bindings (SQL/OLB))

Module Language

- Anwendungsprogramm und SQL-Statements werden getrennt
  - Modul enthält Methoden und Deklarationen von Cursorn und temporären Tabellen, wird in einer Datenbank gespeichert
  - Anwendung kann die Methoden des Moduls aufrufen
  - Sog. Linker kombiniert SQL Statements und Anwendungsprogramm
- Beispiel:

```
MODULE projects_module
   NAMES ARE ascii     LANGUAGE C
   SCHEMA user_schema AUTHORIZATION user

PROCEDURE num_projects
   (  :budget     INTEGER,
      :num        INTEGER, SQLSTATE )
   SELECT   COUNT(*)
   INTO     :num
   FROM     projects
   WHERE    budget >= :budget;
…
```

(J. Melton, A. R. Simon: SQL:1999 Understanding Relational Language Components, 2002)

Dynamic SQL

- Standard für Anwendungsprogramme, die SQL-Statements zur Laufzeit erstellen und absenden
- Es ist also der DB vorab unbekannt …
  - ob ein Statement Daten holen oder speichern will
  - wie viele Variablen benutzt werden, und welchen Typ sie haben
- Eigenschaften der SQL-Statements durch Deskriptor beschreibbar
- Zwei Möglichkeiten:
  - Execute Immediate:
    - Statement wird direkt ausgeführt
  - Prepare and Execute:
    - Das gleiche Statement wird mehrfach ausgeführt (mit verschiedenen Parametern)
    - Zwischenergebnisse der Vorbereitung werden behalten (z.B. Ausführungsplan)

(J. Melton, A. R. Simon: SQL:1999 Understanding Relational Language Components, 2002)

Example: Dynamic SQL

```
/* execute statement only once */
EXEC SQL EXECUTE IMMEDIATE "UPDATE projects
                            SET budget = 10 000 000
                            WHERE projectno = 'PJ47'";

/* prepare and execute statement */
dynstmt = "DYN1";
temp = "UPDATE projects
        SET budget = 1 000 000
        WHERE projectno = ?";
EXEC SQL PREPARE :dynstmt FROM :temp;

prjno = "PJ47";
EXEC SQL EXECUTE :dynstmt USING :prjno;
```

Nachteile der Dynamik: SQL Code Injection

```
CREATE PROCEDURE search_orders
                @custid   nchar(5)  = NULL,
                @shipname nvarchar(40) = NULL AS
DECLARE @sql nvarchar(4000)

SELECT @sql =
' SELECT * ' +
' FROM dbo.Orders WHERE 1 = 1 ‘

IF @custid IS NOT NULL
                SELECT @sql = @sql + ' AND CustomerID LIKE ''' + @custid + '''‘

IF @shipname IS NOT NULL
                SELECT @sql = @sql + ' AND ShipName LIKE ''' + @shipname + '''‘

EXEC(@sql)
```

Enter shipname: Sea Lion

➔

```
EXEC(
SELECT * FROM dbo.Orders
WHERE 1 = 1 AND ShipName LIKE ‘Sea Lion’
)
```

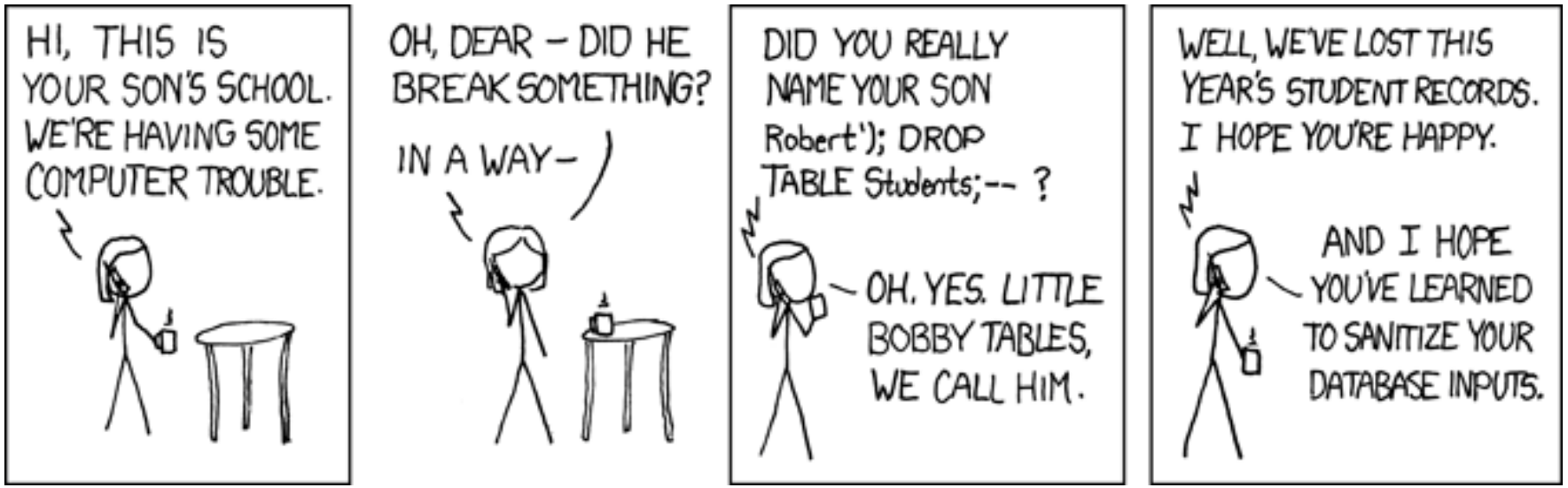
Enter shipname: `;DROP TABLE ORDERS;

➔

```
EXEC(
SELECT * FROM dbo.Orders
WHERE 1 = 1 AND ShipName LIKE ‘’;
DROP TABLE ORDERS;
)
```

Nachteile der Dynamik: SQL Code Injection

- Vorlesung PSI-IntroSP: Verhindern von Code Injections



Call-Level APIs

- ODBC (Open Database Connectivity)
  - basiert auf informellem DBMS-Hersteller-Standard aus 1992
  - Microsoft adaptiert die Schnittstelle und nennt es ODBC
- SQL/CLI
  - Formales Konsortium (SQL Access Group) übernimmt die Entwicklung, nennt es CLI (Call-Level Interface).
  - Ergebnis wurde als Teil des SQL-92 Standards 1995 veröffentlicht, ist heute Teil 3 von SQL:1999.
  - ODBC ist SQL/CLI sehr ähnlich
- JDBC (Java Database Connectivity)
  - Schnittstelle speziell für Java Anwendungen
  - JDBC wurde durch allgemeine APIs wie ODBC und SQL/CLI stark beeinflusst

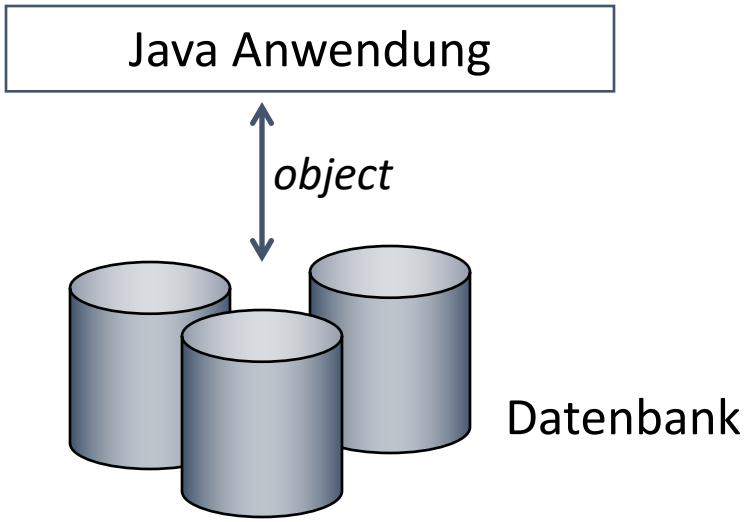
(J. Melton, A. R. Simon: SQL:1999 Understanding Relational Language Components, 2002)

Beispiel: JDBC

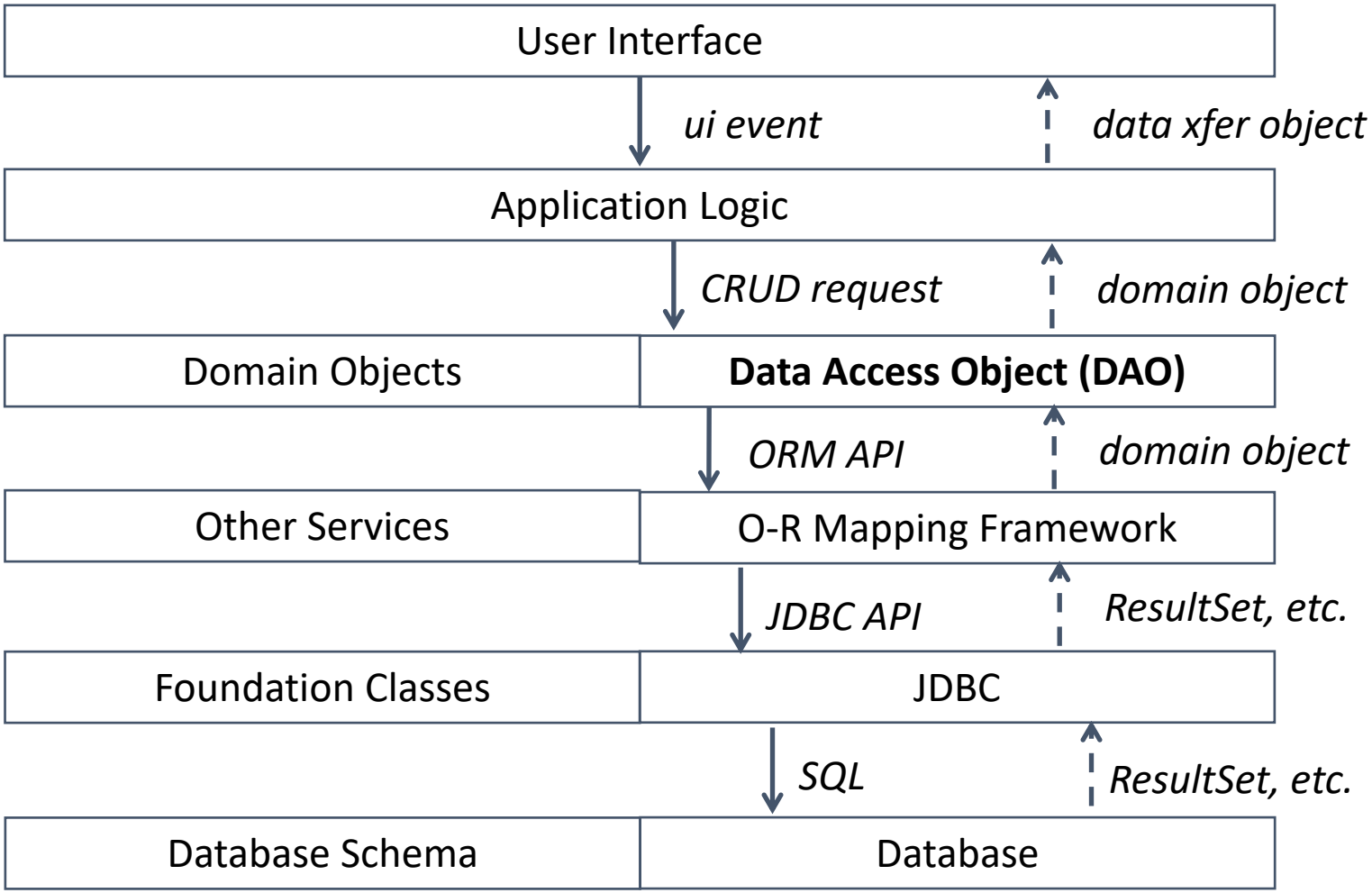
```
class Employee {
   public static void main (String args []) throws SQLException {
      // Load the Oracle JDBC driver
      DriverManager.registerDriver(new
         oracle.jdbc.driver.OracleDriver());
      Connection conn = DriverManager.getConnection
         ("jdbc:oracle:thin:" + "@cip-s.kbs.uni-
            hannover.de:1521:dbs1", "scott", "tiger9i");
      Statement stmt = conn.createStatement ();
      // Select the ENAME column from the EMP table
      ResultSet rset = stmt.executeQuery ("select ENAME from EMP");
      // Iterate through the result and print the employee names
      while (rset.next ())
         System.out.println(rset.getString (1));
      rset.close();
      stmt.close();
      conn.close();
   }
}
```

Object-relational Mappings (ORM)

- Ziel:
  - Persistierung in objektorientierten Anwendung
  - Framework verbirgt SQL vor Entwickler
  - Mapping: OO-Datenobjekte → Datenbankschemata und passende SQL-Statements



OO-Mappings: Typische Schichtenarchitektur



Zwischenrückblick: SQL Programmierung

- Direkter Aufruf
  - von SQL an die Datenbank
- Embedded SQL and SQLJ
  - SQL wird in die Host-Sprache eingebunden
- Dynamic SQL
  - wird zur Programmlaufzeit zusammengebaut
  - DB soll trotzdem vorbereitet sein!
- Module Language
  - SQL wird in Module ausgelagert, die von Host-Sprache aus angefragt werden
- Call-Level APIs:
  - Schnittstellen, um aus der Host-Sprache Datenbanken anzusprechen
    - SQL/CLI, ODBC, JDBC
- Mappings
  - Verbergen SQL vor Programmierer
  - Beispiele: SQLAlchemy, Hibernate, Enterprise JavaBeans, …

Rückblick: SQL

- DDL
  - Datenbankkomponenten erstellen:
    CREATE ...
- DML
  - Anfragen, Aggregation, Gruppierung, Sortierung:
    SELECT ... FROM ... [WHERE ...] [GROUP BY ...] [HAVING ...] [ORDER BY ...]
  - Datenmanipulation:
    INSERT, DELETE, UPDATE
- VDL
  - Virtuelle Relationen:
    VIEW
- DCL
  - Rechte/Privilegien kontrollieren:
    GRANT, REVOKE
- Programmiermethoden
  - Direkt, embedded, dynamisch
  - APIs, Mappings