5. Grundlagen der Datenabfragesprache SQL

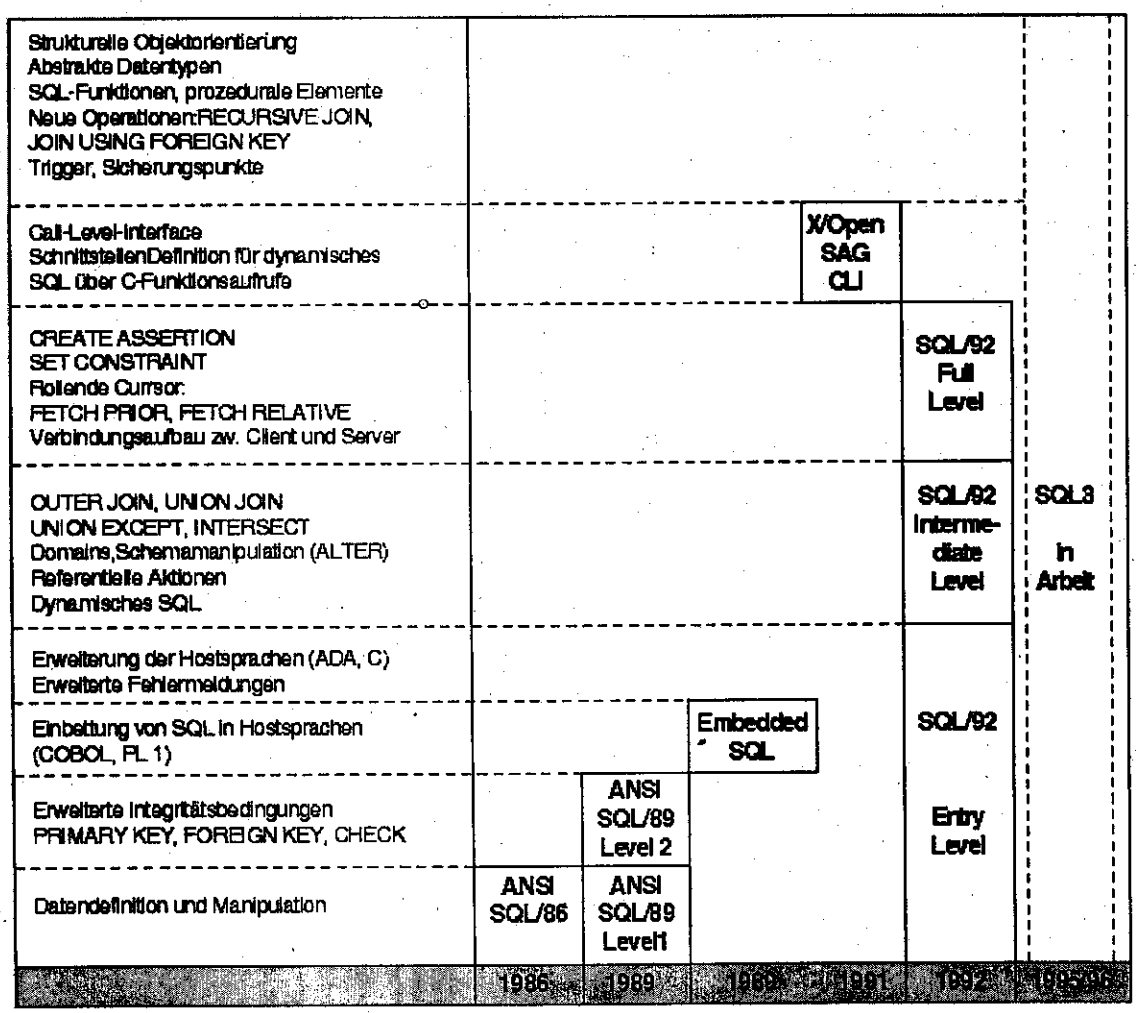
Seinen Ursprung hat SQL in der Implementierung der Abfragesprache SEQUEL (Structured English Query Language) für die erste relationale Datenbank von IBM, System/R. 1982 begann das American National Institut (ANSI) ausgehend von SEQUEL eine normierte Abfragesprache für relationale Datenbanken zu entwickeln. Diese Bemühungen führten 1986 zur ersten Veröffentlichung eines Standards (SQL/86), der weitgehend auf dem IBM-Dialekt beruht. Dem folgte 1989 eine Revision (SQL/89), die verschiedene Elemente zur Unterstützung der Datenbank-Integrität enthielt. Nachträglich wurde vom ANSI eine Spezifikation für Embedded SQL veröffentlicht, die aber kein Bestandteil von SQL/89 war.

Parallel zur Entwicklung von SQL/89 begannen schon 1987 erste Arbeiten für eine wesentlich erweiterte Version, die den Arbeitstitel SQL2 trugen und Ende 1992 als Standard veröffentlicht wurden. Dieser Standard heißt SQL2 oder SQL/92.

SQL/92 definiert drei Levels:

- das Entry Level entspricht SQL/89,
- im Intermediate Level finden sich schon eine Vielzahl von Erweiterungen,
- Full-Level-SQL realisiert den Standard vollständig.

Die verschiedenen Conformance Level sollen Datenbankherstellern den Weg zum Full Level SQL erleichtern.



Full-Level-SQL teilt sich auf in datendefinierende (DDL) und datenmanipulierende Sprachelemente (DML):

- Die DDL erlaubt neben der bloßen Strukturdefinition von Tabellen auf vielfältige Weise die Festlegung von Integritätsbedingungen, Schlüsselkandidaten, Primär- und Fremdschlüssel, Spalten- und Tabellenbedingungen, die sich auf mehrere Spalten der gleichen oder einer anderen Tabelle beziehen, etc.
  - Definition von logischen DB-Objekten
    - CREATE database
    - CREATE TABLE supplier (SNO CHAR (5) NOT NULL,
      SNAME CHAR (20),
      STATUS DECIMAL (3),
      CITY CHAR (15),
      PRIMARY KEY (SNO))
  - CREATE SYNONYM huhn FOR mitarbeiter
    => es kann auch über den Namen Huhn die Tabelle Mitarbeiter angesprochen werden
  - ALTER TABLE mitarbeiter ADD ...
    => Ändern einer Tabelle (Hinzufügen einer Spalte, etc.)
  - DROP TABLE mitarbeiter => Tabelle löschen
  - CREATE VIEW name [(spalte1, ...)] AS SELECT-Anweisung
    Es wird eine dauerhafte Teilmenge als Tabelle angelegt. Die Verbindung zur Basis-Tabelle geht nicht verloren => falls die Basis verändert würde ändert sich die Sicht automatisch auch (Vorteil im Vgl. zur temporären Tabelle)
    Sichten über mehrere Basistabellen können erzeugt werden: solche Tabellen dürfen aber nur zum Abfragen, nicht zum Ändern verwendet werden.
- Die DML enthält Operationen zur mengenorientierten Abfrage und Manipulation der Daten. Über den mächtigen Select-Befehl hinaus bietet SQL/92 mehrere neue Join-Operatoren sowie die Mengenoperationen Vereinigung, Schnittmenge und Differenz. Ferner unterstützt SQL eine Transaktionsverarbeitung. In Abhängigkeit vom gewählten „Isolation Level" bleibt eine Transaktion im Mehrbenutzerbetrieb von Änderungen anderer Benutzer unberührt.
  - Z.B. SELECT -Befehle:
    SELECT [ALL|DISTINCT] ausdruck FROM tab_liste
    [WHERE bedingung]
    [GROUP BY ausdruck]
    [HAVING bedingung]
    [ORDER BY ausdruck [ASC|DESC], ...];

Alle auf dem Markt befindlichen SQL-Server bieten als Datenbanksprache SQL an. Jedoch nicht alle SQL/92. Es gibt auch firmenspezifische Ausprägungen der SQL- Syntax.

Bedeutung der Basis-Statements:

- Aggregatsfunktionen: MIN, MAX, SUM, AVG, COUNT
  Bis auf COUNT werden bei allen Aggregatsfunktionen vor der Berechnung die Nullwerte aus der Spalte entfernt. Sind alle Werte NULL, so ist das Ergebnis NULL
- WHERE ... IN (Mengen-Spezifikation) bzw.
  - WHERE mittel BETWEEN 95000 AND 120000
  - Wildcards: LIKE 'k*'

- WHERE m_nr = ANY (SELECT ...) - Datensätze aus der Hauptabfrage, die den Vergleich mit mindestens einem der von der Unterabfrage zurückgegebenen Datensätze erfüllen
- WHERE m_nr <= ALL (SELECT ...)

- GROUP BY (pro Gruppe wird ein Wert ausgegeben). Die Feldliste nach „SELECT" muss identisch mit der nach „GROUP BY" sein !
  - Das folgende Beispiel ermittelt für jeden Lieferanten den durchschnittlichen Einzelpreis aller Artikel und gruppiert die Preisliste nach Lieferanten.
    SELECT [Lieferanten-Nr], Avg(Einzelpreis) AS DurchschnittEinzelpreis
    FROM Artikel
    GROUP BY [Lieferanten-Nr];
  - Das folgende Beispiel ermittelt in jeder Kategorie den höchsten Einzelpreis eines Artikels.
    SELECT [Kategorie-Nr], Max(Einzelpreis) AS MaxEinzelpreis
    FROM Artikel
    GROUP BY [Kategorie-Nr];
  - Das folgende Beispiel ermittelt, wie viele Bestellungen jedem Angestellten in der Datenbank zugeordnet sind.
    SELECT [Personal-Nr], Count([Bestell-Nr]) AS AnzahlBestellungen
    FROM Bestellungen
    GROUP BY [Personal-Nr];
- HAVING: Wähle Gruppe mit Bedingung aus. Nachdem GROUP BY Datensätze kombiniert, zeigt HAVING alle von dem GROUP BY-Abschnitt gruppierten Datensätze an, die die im HAVING-Abschnitt angegebenen Bedingungen erfüllen (wie WHERE bei einzelnen Datensätzen).
  - Das folgende Beispiel ermittelt alle Lieferanten, deren Artikel einen durchschnittlichen Einzelpreis über 40 DM aufweisen.
    SELECT [Lieferanten-Nr], Avg(Einzelpreis)
    FROM Artikel
    GROUP BY [Lieferanten-Nr]
    HAVING (Avg(Einzelpreis)>40);
  - Das folgende Beispiel wählt alle Angestellten aus, die mehr als 100 Bestellungen entgegengenommen haben.
    SELECT [Personal-Nr], Count([Bestell-Nr])
    FROM Bestellungen
    GROUP BY [Personal-Nr]
    HAVING Count([Bestell-Nr]) > 100;
- ORDER BY: Anordnen der Datensätze bei der Ausgabe
  - Das folgende Beispiel sortiert die Datensätze nach dem Feld Nachname in absteigender Reihenfolge (Z bis A).
    SELECT Nachname, Vorname FROM Personal ORDER BY Nachname DESC;
  - Das folgende Beispiel sortiert die Datensätze zuerst nach dem Feld Kategorie-Nr und dann nach dem Feld Artikelname.
    SELECT [Kategorie-Nr], Artikelname, Einzelpreis FROM Artikel
    ORDER BY [Kategorie-Nr], Artikelname;
- Einfügen und Verändern von Daten:
  INSERT INTO tab [(spalte1, ..,..)] VALUES (wert1, ..,..) bzw.
  INSERT INTO tab [(spalte1, ..,..)] SELECT-Anweisung
  UPDATE tab SET spalte1 = ausdr1 [WHERE-Bedingung];
  DELETE FROM tab [WHERE-Bedingung];

5.1. SQL und Relationale Algebra

Grundidee: Da eine relationale DB aus einer Menge von Datensätzen besteht, kann man zur Datenmanipulation die üblichen Operationen aus der Mengenlehre verwenden.

- R und S heißen vereinigungskompatibel, wenn die Wertebereiche der Attribute übereinstimmen (Attributname ist nicht wesentlich)
- Eine Datenbanksprache (genauer DML) heißt relational vollständig, wenn sich die fünf primitiven Operationen der relationalen Algebra in ihr darstellen lassen:
  - Vereinigung: $(R, S) \rightarrow R \cup S$
    (SELECT wohnort FROM r) UNION (SELECT stadt FROM s)
    d. h. Spaltenanzahl bleibt, Zeilen beider Tabellen füllen neue Tabelle
    SELECT DISTINCT ... bzw. UNION beseitigt auch Vielfachheiten (identische Zeilen)

-> Folie: 4/1

  - Differenz: $(R, S) \rightarrow R \setminus S$ (ohne)
    In Oracle mit MINUS bzw. in SQL mit: NOT EXIST
    d. h. Spaltenanzahl bleibt, die Zeilen von S werden weggelassen
  - kartesisches Produkt: $(R, S) \rightarrow R \times S$
    SELECT * FROM R, S (WHERE...)
    d.h. jeder Datensatz von R mit jedem Datensatz von S kombiniert

kartesisches Produkt

-> Folie: 4/2

  - Projektion: $R \rightarrow \pi_{Spalte}(R)$
    SELECT spalte FROM (WHERE ...)
    d. h. betrachte nur bestimmte Spalten aus der Tabelle

-> Folie: 4/2

  - Selektion: $R \rightarrow \delta_p(R)$
    (SELECT name FROM) **WHERE** p
    bzw. SELECT * FROM (reine Selektion, keine Projektion)
    d.h. betrachte nur bestimmte Zeilen aus der Tabelle

-> Folie: 4/3

-> Übung: Datenmodellierung mit MS-Access

5.2. Verknüpfung von Tabellen mit Hilfe von Joins

Für relationale Datenstrukturen ist bezeichnend, daß der Prozeß der Normalisierung ursprünglich zusammengehörige Informationen auf mehrere Tabellen verteilt. Referenzen zwischen den Tabellen stellen Primär- und Fremdschlüsselfelder her. Um die Informationen bei Datenbankabfragen wieder zusammenzuführen, lassen sich Tabellen mit Hilfe des Join-Befehls miteinander verknüpfen.

Der SQL/89 Standard kannte keinen expliziten Join-Befehl. Der Join wurde implizit im Select-Statement durch die Angabe mehrerer Tabellen und einer Verknüpfungsbedingung in der Where-Klausel hergestellt.

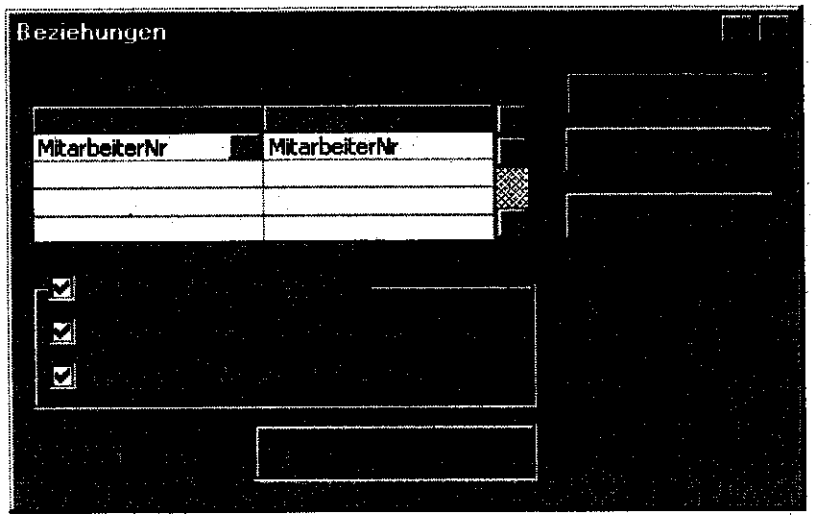
- Seien R, S Relationen, sei p ein Verbundprädikat (Werte zweier Spalten mit Vergleichsoperator) für R und S: $(R, S) \rightarrow R \,|\times|_p\, S$
- SELECT mitarbeiter.*, abteilung.* FROM mitarbeiter, abteilung
  WHERE mitarbeiter.abt_nr=abteilung.abt_nr

3. Datenmodellierung in MS-Access

Es erfolgt eine Einführung in Access:

- Erklärung der Register des Datenbankfensters (Aufbau einer Datenbank-Datei)
- Erstellen von Tabellen mit:
  - Schaltfläche „Neu", Entwurfsansicht
  - Schlüssel festlegen (rechte Maustaste auf vorher markierte Felder - Primärschlüssel)
- Beziehungseditor:
  - Extras/Beziehungen bzw. Symbol (wenn Datenbankfenster den Fokus hat)
  - Tabellen hinzufügen
  - Beziehungen festlegen
    - Ziehen eines Feld aus einer Tabelle in das entsprechende Feld der anderen Tabelle.
    - Die Symbole oberhalb der Verknüpfungslinie geben die Art der Verknüpfung (1:n, etc.) an. Diese Symbole erscheinen nur, wenn referentielle Integrität durchgesetzt ist.

- Alle angebotenen referentiellen Integritätsbedingungen auswählen



Aufgabe ist:

1. Tabellen und Felder in Access anlegen
2. Beziehungen und Kardinalitäten in Access festlegen
3. Auswertungen überlegen (Implementierung in MS-Access erst in der nächsten Übung):

- Welcher Mitarbeiter besucht welche Seminare?
- Alle Seminare mit Quoten für eine bestimmte Dienststelle.
- In welcher Lehrgangsstätte werden welche Seminare abgehalten?
- Welche Mitarbeiter haben noch keine Seminare besucht?
- Für jeden Organisationsbereich soll ermittelt werden, wieviele Teilnehmer insgesamt wieviele Stunden in einem bestimmten Zeitraum auf Seminar gewesen sind.