Einteilung von Programmiersprachen (1)

1. Generation: Maschinensprachen

- interne Sprache eines bestimmten Rechnertyps
- Formulierung des Befehlscodes und Angabe von Speicheradressen in dualer, oktaler oder hexadezimaler Form

2. Generation: Maschinenorientierte Sprachen (Assemblersprachen)

- Sprache eines bestimmten Rechnertyps
- Abkürzung der Befehle durch mnemotechnische Bezeichnungen (z.B. ADD, SUB, MUL, DIV) und Symbolisierung von Speicheradressen

3. Generation: höhere Programmiersprachen

- Verwendung von Anweisungen oder Folgen von Anweisungen (an Stelle der Befehle von Maschinen- und Assemblerprachen)
- weitgehende Rechnerunabhängigkeit und Portierbarkeit
- breiter Einsatz bei der Anwendungsentwicklung

4. Generation: nichtprozedurale Programmiersprachen

Einteilung von Programmiersprachen (2)

4. Generation: nichtprozedurale Programmiersprachen

- Beschreibung der auszuführenden Aktionen und nicht, wie bei den prozeduralen Sprachen (=1. bis 3. Generation) , der Folge an auszuführenden Befehlen
- Formulierung, Was ist zu tun ist, nicht Wie
- weitgehende Rechnerunabhängigkeit und Portierbarkeit
- weitere Bezeichnungen: deskriptive Sprache, deklerative Sprache, 4 GL = Fourth Generation Language
- breiter Einsatz bei der Anwendungsentwicklung im Bereich von Datenbanken und deren Kopplung mit Programmiersprachen
- Bsp.:
  - Abfragesprache SQL (Structure Query Language)

Weitere Einteilung von Programmiersprachen

Objektorientierte Programmiersprachen

- Grundprinzip: Zusammenfassung von Daten und den darauf anzuwendenden Funktionen zu Objekten (Kapselung)

Wissensbasierte Sprachen

- weitere Bezeichnung: KI-Sprachen (KI … Künstliche Intelligenz)
- Entwicklung Wissensbasierter Systeme (insbesondere Expertensysteme)
- Bsp. LISP, Prolog

Scriptsprachen

- basieren auf bereits vorhandenen, in beliebiger Programmiersprachen, erstellten Programmkomponenten
- oft keine strenge Datentypisierung $\Rightarrow$ flexibel einsetzbar
- Bsp.: - Unterstützung von WEB-Technologien: jscript, php
  - Anwendungs- und Herstellerspezifische Scriptsprachen: ABAP (SAP), SQL*Forms(Oracle), Powerscript (Sybase), VBA (Mircosoft)

Beschreibungssprachen (engl.: Markup Language)

- im eigentlichen Sinne keine Programmiersprachen
- Beschreibung strukturierter Informationen (z.B. Druckseiten, Webseiten, Dokumenten)
- Bsp: - SGML (Standart Generalized Markup Language) für Druckerzeugnisse
  - HTML (Hypertext ML), XML (Extensible ML)

Chronologie des SQL-Standards (1)

SEQUEL

- 1974 durch D. Chamberlin von IBM San Joe Laboratory definiert Anfragesprache *Structured English Query Language* für den DBMS-Prototypen „System R“
- 1976 Nachfolgeversion SEQUEL/2 und aus rechtlichen Gründen Umbenennung in SQL

SQL86 (SQL1)

- „Data Base Language SQL“
- weitestgehend Übernahme der IBM-Spezifikationen (z.B. CREATE TABLE, SELECT, UPDATE, INSERT, DELETE, COMMIT, ROLLBACK, …)

SQL89

- „Data Base Language SQL with Integrity Enhancements“
- Einführung von Integrity Enhancement Features (z.B. DEFAULT, CHECK-Klauseln, PRIMARY KEY, REFERENCES, …)

SQL92 (SQL2)

- umfangreiche Erweiterungen der relationalen Ausdrucksmöglichkeiten
- Internationalisierung und benutzerdefinierte Zeichensätze
- drei Versionen: ENTRY SQL, INTERMEDIATE SQL, FULL SQL

Chronologie des SQL-Standards (2)

SQL99 (SQL3)

- ISO/IEC-9075 – 1999, neue Struktur (Einführung von Teilen – parts) und Erweiterung
- Erweiterung des bisherigen relationalen Modells durch zusätzliche prozedurale und objektorientierte Konzepte
- Datentypen BOOLEAN, CLOB, BLOB
- User Defined Types (UDTs)
- Typ-Konstrukturen: strukturierte und Collection-Typs
- Typed Tables und Views
- Ereignissteuerung (trigger)
- Datenbankprozeduren (stored procedures)
- Einbettung von SQL in Java (SQLJ)

SQL2003

- Neue Konstrukte in SQL/Foundation
- Neuer Teil SQL/XML
- Call Level Interface

Von den Herstellern wurden bis heute der Standard SQL99 nur teilweise implementiert.

Tabellendefinition

SQL-Befehl CREATE TABLE

CREATE TABLE *tab_name*
(*sp_name1* <Datentyp> [<column_constraint>],
sp_name2 <Datentyp> [<column_constraint>],
…
sp_name3 <Datentyp> [<column_constraint>]
[, <table_constraint>]
)

Beispiele für die Definition sprachbeschreibender Variablen:

<Datentyp> ::= INT | SMALLINT | DECIMAL(m,n) | FLOAT(n) |
DOUBLE PRECISION | DATE | TIME | CHAR(n) |
VARCHAR(n)

<column_constraint> ::= { NULL | NOT NULL | UNIQUE |
PRIMARY KEY | CHECK(<condition>) }

Ändern der Tabellendefinition

SQL-Befehl ALTER TABLE

- **ALTER TABLE** verändert die Tabellenstruktur
- Mit ALTER TABLE muß sorgsam umgegangen werden, da nach ALTER TABLE eventuell VIEW's oder Embedded-SQL-Programme nicht mehr funktionieren.
- Es gibt Varianten von ALTER TABLE die,
  - nur mit leeren Tabellen funktionieren,
  - es zulassen, Spalten einzufügen, zu löschen oder Randbedingungen hinzuzufügen oder zu löschen.
- Angefügte Spalten werden mit NULL-Werten gefüllt.

Spalte anfügen:

ALTER TABLE *tab_name* ADD *sp_name* <Datentyp> [NULL] [, ...]

Datentypen ändern:

ALTER TABLE *tab_name* ALTER COLUMN *sp_name* <Datentyp> [NULL]

Spalte löschen:

ALTER TABLE *tab_name* DROP *sp_name*

Bemerkungen zu Views

- Abfragen, die zur Laufzeit abgearbeitet werden, keine physischen Tabellen
- aber Ergebnisse können Tabellen entsprechen und so verwendet werden (z.B. SELECT *view_name* …)
- fast alle SQL-Befehle benutztbar, Außnahmen:
  - kein UNION
  - kein ORDER BY
- gut für Realisierung des Zugriffsschutzes
- Änderungen in den Basistabellen => Änderung der Ergebnisse der View, aber Umkehrung gilt nur teilweise!
- Viewupdate => Änderung in den zugrundeliegenden Basistabellen (problematisch)
  Einschränkungen des Viewupdates
  - Bezug nur auf eine Tabelle
  - Einbeziehung des Primärschlüssels
  - keine statistischen Funktionen
  - keine DISTINCT, GROUP BY, HAVING-Klausel

Query Languages

Bei abbildungsorientierten Sprachen wie SQL werden die Daten
des Definitionsbereiches (from)
mit Hilfe eines Auswahlkriteriums (where)
auf einen Bildbereich (select)
abgebildet.

Vergleichsoperatoren:

=	gleich
!=, <>	nicht gleich
<, >	größer, kleiner als
>=, <=	größer gleich, kleiner gleich
!>, !<	nicht größer, nicht kleiner als

Logische Operatoren:

NOT
AND
OR

abnehmende Priorität

Wildcards zur Patternsuche:

% …	beliebiges Zeichen	[ ] …	im Bereich enthalten
_ …	genau ein Zeichen	[^] …	nicht im Bereich enthalten

Sortieren/Gruppieren

Sortierung:

SELECT <attributsliste>
FROM *tab_name* [WHERE ...]
ORDER BY *sp_name1* [, *sp_name2* …] [DESC]

DESC … absteigend (Standard ist ASC aufsteigend)

Gruppierung

SELECT <attributsliste>
FROM *tab_name* [WHERE ...]
GROUP BY *sp_name1* [,*sp_name_2*, …]

- Gruppierung ist die Zusammenfassung von Tupeln mit gleichen Attributwerten in Verbindung mit statistischen Funktionen.

Ergebnisse dürfen nur einen Wert pro auszugebender Gruppe haben, oder anders ausgedrückt:

Attribute der SELECT-Komponente müssen auch in der GROUP BY Komponente stehen, außer sie kommen nur innerhalb von Statistikfunktionen vor!

Abfragen über mehrere Relationen (JOIN)

Syntaxvarianten

Angabe als implizite Bedingung in der WHERE-Klausel	Angabe der Bedingung mit dem Schlüsselwort JOIN

Arten:

- Equi-Join
- Natural Join
- Cross Join
- Teta Join
- Outer Join

Implizite Bedingungsangabe in der WHERE – Klausel für 2 Relationen

```
SELECT      tab_name1.sp_name1 [, tab_name2.spname2 …]
FROM        tab_name1, tabname2
WHERE ...]  tab_name1.sp_name1 = tab_name2.sp_name2
            [AND <bedingung>]
```

Verknüpfung von Relationen über JOIN

Equi-Join (Inner Join):

Es wird nur die Gleichheit der Attributswerte zweier Tupel überprüft und dabei ausschließlich die UND-Verknüpfung verwendet.
Die doppelten Attributwerte beider Tupel werden ausgegeben.

Natural Join (Natürlicher Join):

Analog Equijoin, zusätzlich werden doppelt auftretende Attribute eliminiert.

Cross Join (Kartesisches Produkt):

Alle Tupel der ersten Relation werden mit jedem einzelnen Tupel der zweiten Relation verknüpft.

Teta Join

Beim Theta Join werden die Join-Spalten in der WHERE-Klausel mit einem der Vergleichsoperatoren verglichen.

Outer Join

Auswahl auch vonTupeln, die die Bedingung mit den Join-Spalten nicht erfüllen.

Mengenoperationen

Voraussetzung:

Die Anwendung der Mengen-Operatoren fordert folgende Verträglichkeit der beteiligten Relationen:

- gleiche Attributanzahl
- paarweise syntaktisch verträgliche Wertebereiche

Operationen:

➢ Vereiniguung	UNION
➢ Durchschnitt	INTERSECT
➢ Differenz	EXCEPT

EXISTS - ANY - ALL

EXISTS

EXISTS liefert als Ergebnis den BOOLschen Wert wahr oder falsch.
Bei Verwendung in Subquery werden nur bei wahr Daten in Hauptfrage aufgenommen.

ANY

ANY liefert als Ergebnis den BOOLschen Wert wahr oder falsch.
Der **ANY-Operator** bildet einen zusammengesetzten Vergleichsausdruck mit einer nicht festgesetzten Anzahl von Einzelvergleichen, die dynamisch mit ODER untereinander verbunden sind. Wenn irgend einer (?? any) dieser Einzelvergleiche wahr ist, so ist das Ergebnis des Vergleiches wahr. Ist keiner der Einzelvergleiche wahr, ist das Ergebnis falsch.

ALL

ALL liefert als Ergebnis den BOOLschen Wert wahr oder falsch.
Der **ALL-Operator** bildet einen zusammengesetzten Vergleichsausdruck mit einer nicht festgesetzten Anzahl von Einzelvergleichen, die dynamisch mit UND untereinander verbunden werden. Wenn all diese Einzelvergleiche wahr sind, so ist das Ergebnis des ALL-Operators wahr. Ist nur einer der Einzelvergleiche falsch, ist das Ergebnis des Operators falsch.