

1. Beispiel DATENBANK	1
2. Der Index	1
2.1. Allgemeines	1
2.2. Syntax.....	2
2.2.1. index-name.....	2
2.2.2. table-name	2
2.2.3. UNIQUE	3
2.2.4. CLUSTERED HASHED.....	3
2.2.5. ASC,DESC	3
2.2.6. PCTFREE.....	3
2.2.7. SIZE integer-constatnt ROWS	3
2.3. Funktionen in Index	3
2.4. Beispiele	4
3. SELECT.....	5
3.1. Syntax.....	5
3.2. Verarbeitung einer SELECT ANWEISUNG.....	6
3.2.1. FROM.....	6
3.2.2. WHERE	7
3.2.3. GROUP BY	7
3.2.4. HAVING.....	7
3.2.5. Select	7
3.2.6. Order by.....	8

INDEX UND SELECT IN SQL

1. Beispiel DATENBANK

```
create table Ware
  (Artikel# SmallInt,
   Bezeichnung CHAR(20),
   Preis SmallInt,
   ME CHAR(3),
   Bestand SmallInt
  );
```

```
create table Kunde
  (Kunden# SmallInt,
   Name CHAR(20),
   Adresse CHAR(40),
   PLZ SmallInt,
   Ort CHAR(20)
  );
```

```
create table Auftrag
  (Re# SmallInt,
   Kunden# SmallInt,
   Datum date,
   status CHAR(3)
  );
```

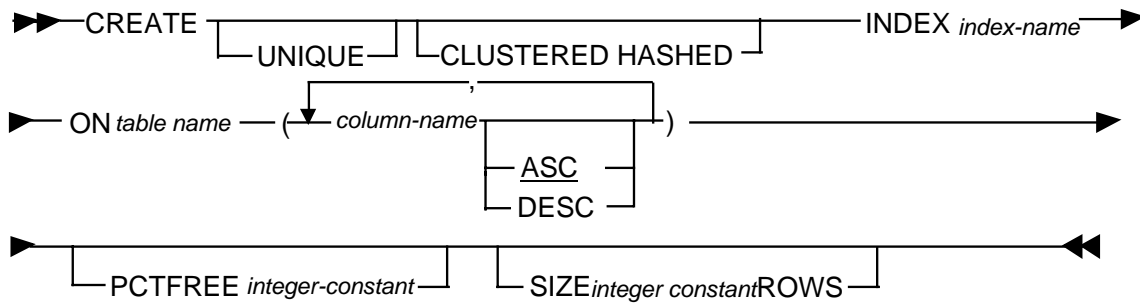
```
create table Pos
  (Re# SmallInt,
   Pos# SmallInt,
   Artikel SmallInt,
   Menge SmallInt
  );
```

2. Der Index

2.1. Allgemeines

Wird eine Zeile in einer realen Tabelle gesucht, die in einer Spalte bzw. Spaltenkombination vorgegebene Werte enthält, so muß die gesamte Tabelle durchsucht werden. Die realen Implementationen sind bemüht, die Zugriffe auf den externen Speicher möglichst gering zu halten, da diese Zugriffe langsam sind. Deshalb sind üblicherweise Methoden implementiert, die es gestatten, Zeilen direkt zu adressieren. Dies erreicht man durch Errichtung eines Indexes. Ist für die betrachtete Tabelle ein Index auf die Spalte bzw. Spaltenkombination errichtet worden, so muß nur im Index nachgeschaut werden, wo die betreffende Zeile steht (es können auch mehrere Zeilen sein, die die Bedingung erfüllen). Ein Index ist in der Regel wesentlich kleiner als die Tabelle und zudem so organisiert, daß er schnell durchsucht werden kann. Dadurch wird beim Suchen die Zahl der Zugriffe auf den externen Speicher erheblich verringert.

2.2. Syntax



$$6 + \text{Anzahl der Spalten} + \text{Summe der Länge der Spalten} \leq 255$$

Die Länge der einzelnen Spalten hängt von ihren Datentypen ab.

Datentyp	Länge
Character	pro Zeichen 1 Byte
Numeric	12 Bytes
Date/Time	12 Bytes

Ein Beispiel für die Berechnung von Indexes.

Nachname CHAR(20)
 Vorname CHAR(20)
 Geschl CHAR(1)
 41

daraus ergibt sich:

$$6 + 3 + 41 = 50$$

SEHR_GROSS CHAR(249)

$$6 + 1 + 249 = 256$$

$$256 > 255$$

Es kommt zur Fehlermeldung:

Index key is too large

2.2.1.index-name

Der Indexname muß in der gesamten Datenbank eindeutig sein.

2.2.2.table-name

Auf Views können keine Indexe gelegt werden.

2.2.3.UNIQUE

Erzeugt einen Schlüssel - Index. Die Wertekombination der einzelnen Elemente darf in der Tabelle nur ein einziges mal vorkommen. Existieren bereits zur Indexerstellung doppelte Werte kommt es zu einer Fehler Meldung. Ebenso bei Update oder Insert.

2.2.4.CLUSTERED HASHED

Ein Clustered Hashed Index erlaubt einen schnellern Zugriff auf die Daten. Besteht ein Index aus unique und clustered hashed kann mit einem Plattenzugriff eine Zeile ausgelesen werden.

Ist kein clustered hashed definiert wird der Index als B-Baum erzeugt.

2.2.5.ASC,DESC

Definiert ob der index aufsteigend oder Absteigend sortiert wird. Diese Klausel ist nur für B-Bäume interessant. Wird die Klausel nicht angegeben wird default mäßig ASC genommen.

2.2.6.PCTFREE

Die PCTFREE (prozent frei) klausel definiert, wieviel Platz in jeder Indexseite sein muß wenn der Index erstellt wird. Wird diese klausel nicht angegeben ist der default Wert 10%

2.2.7.SIZE *integer-constant* ROWS

Diese Angabe kontrolliert die gröÙe des Indexes und wird in Anzahl der Reihen angegeben. Ist diese gröÙe zu klein angeben kann es overflow pages kommen. Ist die angebene gröÙe zu groß, wird das overflow page nicht verwendet, aber der Platz auf der Platte ist verbraucht. Diese Klausel wird nur bei clustered hashed Indexe benötigt. und ist dort unbedingt erforderlich.

2.3. Funktionen in Index

Ein Index kann für mehrere Spalten gebildet werden. Diese Spalten können durch Funktionen verbunden werden. Es können jedoch nicht alle Funktionen in Index eingebaut werden.

@CHAR
@CODE
@DATEVALUE
@DAY
@HOUR
@LEFT
@LENGTH
@LICS
@LOWER
@MICROSECOND
@MID
@MINUTE
@MONTH
@MONTHBEG
@PROPER
@QUARTER
@QUARTERBERG
@RIGHT
@SECOND
@STRING
@SUBSTRING
@TIMEVALUE
@TRIM
@UPPER
@VALUE
@WEEKBEG
@WEEKDAY
@YEAR
@YEARBEG
@YEARNUM

2.4. Beispiele

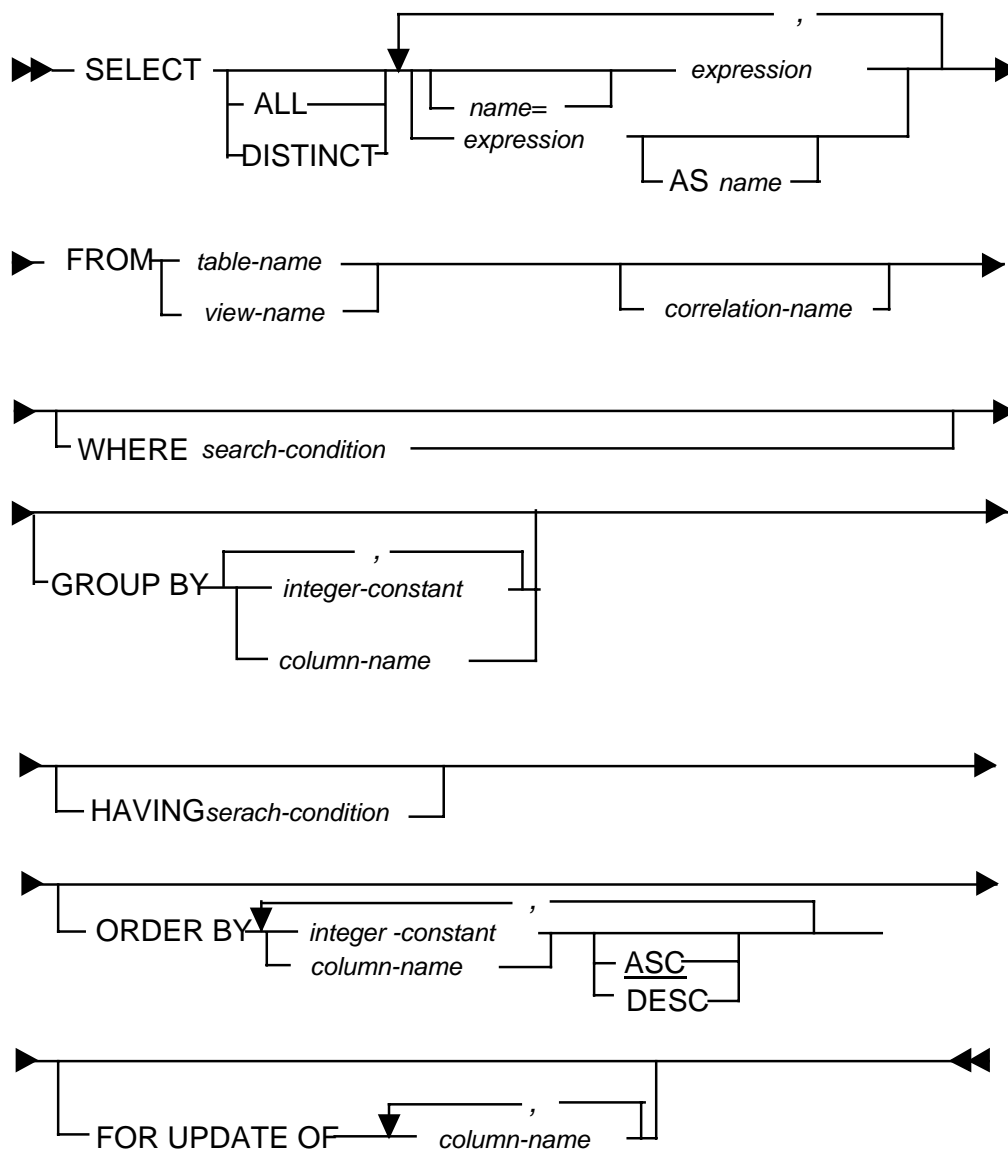
```
CREATE UNIQUE INDEX sWare ON Ware (Artikel#)
CREATE UNIQUE INDEX sKunde ON Kunde (Kundenl#)
CREATE UNIQUE INDEX sAuftrag ON Auftrag(RE#)
CREATE UNIQUE INDEX sPos ON Pos (RE#,POS#)

CREATE INDEX KUINDEX ON KUNDE (@UPPER(name))

SELECT name FROM KUNDE
WHERE @UPPER(name) = 'HUBER'
```

3. SELECT

3.1. Syntax



Im Prinzip besteht das SELECT Statement aus sechs Komponenten

SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY

Zwei Punkte sind äußerst wichtig und sollten immer vor Augen gehalten werden:

- Die Reihenfolge der Komponenten ist fix vorgegeben; eine GROUP BY Komponente darf nicht vor einer WHERE oder vor einer FROM Klausel stehen.

- Eine HAVING-Komponente darf nur dann verwendet werden wenn eine GROUP BY Komponente verwendet wird.

FROM
definiert die Ausgangstabellen

WHERE
selektiert die Reihen, die der Bedingung genügen

GROUP BY
gruppiert Reihen auf Basis gleicher Werte in Spalten

HAVING
selektiert Gruppen, die der Bedingung genügen

SELECT
selektiert Spalten

ORDER BY
sortiert Reihen auf der Basis von Spalten

3.2. Verarbeitung einer SELECT ANWEISUNG

Ausgabe aller Kundennummer die 1993 mehr als einen Einkauf getätigt haben.

```
SELECT Kunden#
FROM Auftrag
WHERE DATUM > 01011993
GROUP BY Kunden#
HAVING COUNT(RE#) >1
ORDER BY Kunden#
```

3.2.1.FROM

In der FROM-Komponente wird nur die Tabelle Auftrag genannt. Die bedeutet für SQL, daß mit der Tabelle AUFTRAG gearbeitet werden muß. Das Zwischenergebnis dieser Operation ist eine exakte Kopie der Tabelle AUFTRAG

Zwischenergebnis:

RE#	Kunden#	Datum	status
1	1	1990-01-01	fa
2	1	1993-05-19	fa
3	3	1993-05-19	fa
4	2	1993-05.17	fa
5	3	1993-07-19	fa

3.2.2.WHERE

In der WHERE Komponente ist die Bedingung 'DATUM > 01011993' angegeben. Alle Reihen, in denen der Wert der Spalte DATUM größer als 01011993 ist, genügen dieser Bedingung. alle diese Reihen zusammen bilden das Zwischenergebnis der WHERE Komponente.

RE#	Kunden#	Datum	status
2	1	1993-05-19	fa
3	3	1993-05-19	fa
4	2	1993-05.17	fa
5	3	1993-07-19	fa

3.2.3.GROUP BY

Die GROUP BY Komponente gruppiert alle Reihen im Zwischenergebnis. Das Gruppieren erfolgt auf der Basis von Werten in der Spalte KUNDEN#. Reihen werden gruppiert wenn sie in der betreffenden Spalte den gleichen Wert haben

Zwischenergebnis

RE#	Kunden#	Datum	status
2	1	1993-05-19	fa
3,5	3	1993-05-19,1993-07-19	fa,fa
4	2	1993-05-17	fa

3.2.4.HAVING

Die vierte Komponente kann in gewissem Sinne mit der WHERE Komponente verglichen werden. Der Unterschied liegt darin, daß WHERE sich auf das Zwischenergebnis der FROM Komponente bezieht, während HAVING auf das gruppierte Zwischenergebnis der GROUP BY Komponente Bezug nimmt. Der Effekt ist jedoch der gleiche, auch in der HAVING-Komponente werden Reihen mit Hilfe einer Bedingung selektiert. In diesem Fall handelt es sich dabei um die Bedingung:

COUNT(RE#)>1

Das bedeutet: Alle Reihen, die mehr als eine Rechnungsnummer enthalten, genügen der Bedingung.

Zwischenergebnis:

RE#	Kunden#	Datum	status
3,5	3	1993-05-19,1993-07-19	fa,fa

3.2.5. Select

In der SELECT Komponente wird angegeben welche Spalten im endgütigen Ergebnis wiedergeben werden müssen. Die Select-Komponente wählt also die Spalten aus.

Kunden#
3

3.2.6. Order by

Hier wird am Inhalt des Zwischenergebnisses nichts mehr verändert. Die Ergebniszeilen werden lediglich noch in die angegebenen Reihenfolge sortiert. Da aber aufgrund der Beispieldaten nur eine Zeile übrig bleibt, ist die Order by Komponente in diesem Fall belanglos

Die Where - Komponente

Die Where Komponente kann folgende Formen haben

- der einfache Vergleich
- Bedingungen mit AND, OR, oder NOT aneinanderkoppeln
- der BETWEEN - Operator
- der IN-Operator
- der LIKE Operator
- der NULL Operator
- der IN-Operator Unterabfrage (Subquery)
- der Vergleichsoperator mit Unterabfrage

Der einfache Vergleich

Ein einfacher Vergleich wird durch einen Ausdruck gebildet, auf den ein Operator und wiederum ein Ausdruck folgt. Der Wert auf der linken Seite des Operators wird mit dem Ausdruck auf der rechten Seite verglichen. Vom Operator hängt es ab, ob die Bedingung WAHR, UNWAHR oder NULL ist

SQL kennt folgende Vergleichsoperatoren:

Vergleichsoperator	Bedeutung
=	gleich
<	kleiner als
>	größer als
<=	kleiner als oder gleich
>=	größer als oder gleich
<>	nicht gleich

(Der Vergleichsoperator <> wird in einigen SQL-Implementierungen auch als != oder != wiedergegeben.)

Beispiele

Gib die Kundennummer aller Kunden an, die in Perchtoldsdorf wohnen.

```
SELECT      Kunden#
FROM        Kunde
WHERE       PLZ = 2380
```

Gib die Bezeichnung, den Preis, die Menge der einzelnen Artikel an wo der Lagerwert 1.000 S übersteigt

```
SELECT      Bezeichnung, Preis, Bestand
FROM        Artikel
WHERE       Preis * Bestand > 1000
```

Bedingungen mit AND, OR, oder NOT aneinanderkoppeln

Eine WHERE - Komponente kann mehrere Bedingungen enthalten, wenn die Operatoren AND, OR und NOT verwendet werden.

Beispiele

Gib die Kundennummer aller Kunden an die in Wien wohnen oder deren Kundennummer kleiner 100 ist.

```
SELECT      Kunden#
FROM Kunde
WHERE       PLZ>=1010 AND PLZ<=1239
OR          Kunden#<100
```

Gib die Kundennummer und den Namen aller Kunden an die nicht in der Ortschaft "Groß Enzersdorf " wohnen.

```
SELECT      Kunden#, Name
FROM Kunde
WHERE       NOT Ort = 'Groß Enzersdorf'
```

oder

```
SELECT      Kunden#, Name
FROM Kunde
WHERE       Ort <> 'Groß Enzersdorf'
```

Der BETWEEN - Operator

Der Between Operator lässt sich am Besten anhand eines Beispiels erklären.

Gib die Kunden# aller Kunden an, die zwischen März 1993 und Juni 1993 einen Einkauf getätigt haben.

```
SELECT      Kunden#
FROM Auftrag
WHERE       Datum Between 01031993 AND 30061993
```

Der IN-Operator

Auch hier sagt ein Beispiel mehr als 1000 Worte

Gib den Namen aller Kunden an, die in Wien, St.Pölten, Eisenstadt, Graz, Klagenfurt, Salzburg, Linz, Innsbruck oder Bregenz ihren Frimensitz haben.

```
SELECT      Name
FROM Kunde
WHERE       ORT IN ('Wien','St. Pölten', 'Eisenstadt', 'Graz', 'Klagenfurt', 'Salzburg',
                  'Linz', 'Innsbruck','Bregenz')
```

Der LIKE Operator

Gib den Namen aller Kunden an, die mit H beginnen

```
SELECT      Name
FROM Kunde
WHERE       Name LIKE 'H%'
```

Hinter dem LIKE Operator können zur Bildung von sogenannten Masken 2 Zeichen als Jokerzeichen stehen

% für kein, ein oder mehrere Zeichen (Vergleiche DOS *)
_ für genau ein Zeichen (Vergleiche DOS ?)

Der NULL Operator

Der Null operator gibt alle jene Zeilen an, die keinen Wert besitzen.

Gib alle Artikel# an, die derzeit nicht lagernd sind

```
SELECT Artikel#
FROM Ware
WHERE Bestand is NULL
```

Der IN-Operator Unterabfrage (Subquery)

Der IN-Operator mit Unterabfrage wird dann verwendet, wenn die Menge von einer anderen Tabelle abhängt.

Beispiel

Gib die Bezeichnung aller Artikel an, die schon mindestens einmal verkauft wurden.

```
SELECT Bezeichnung
FROM WARE
WHERE Artikel# IN
      (SELECT Artikel
      FROM POS)
```

Gib die Bezeichnung aller Artikel an, die noch nie verkauft wurden.

```
SELECT Bezeichnung
FROM WARE
WHERE Artikel# NOT IN
      (SELECT Artikel
      FROM POS)
```

Der Vergleichsoperator mit Unterabfrage

Unterabfragen können nicht nur nach dem IN-Operator verwendet werden, sondern auch nach Vergleichsoperatoren.

Gib alle Kundennummern an, die vor der Rechnung 5 eingekauft haben.

```
SELECT kunden#
FROM Auftrag
WHERE datum <
      (SELECT datum
      FROM Auftrag
      WHERE re = 5)
```

Die GROUP BY Komponente

Mit Hilfe der GROUP-BY Komponente werden die Zwischenergebnisse gruppiert.

Gib die Verkaufte Artikel an.

```
SELECT Artikel
FROM pos
GROUP BY Artikel
```

```

SELECT    re#,pos#,artikel,menge
FROM      pos
GROUP BY  re#,pos#,artikel,menge

```

ANMK: Alle spalten der Select Anweisung müssen in der Group by Anweisung enthalten sein.

Die Having - Klausel

Die Having Klausel entspricht im Prinzip der WHERE klausel, nur das diese auf Gruppen die mittels GROUP BY erstellt wurden angewendet wird.

In der HAVING Komponente sind alle bei WHERE besprochenen Funktionen erlaubt sowie fünf weitere Funktionen

COUNT	Ermittelt die Anzahl der Werte einer Spalte oder die Anzahl der Reihen einer Tabelle
MIN	Ermittelt den kleinsten Wert der Spalte
MAX	Ermittelt den größten Wert der Spalte
SUMME	Ermittelt die Summe der Werte in einer Spalte
AVG	Ermittelt das arithmetische Mittel der Werte in einer Spalte

Beispiele

Gib alle Rechnungsnummern an die mehr als eine Position haben

```

SELECT    re#
FROM      pos
GROUP BY  re#
HAVING    count(re#)>1

```

Gib alle Kundennummern an, die an mindestens zwei Tagen eingekauft haben.

```

SELECT    Kunden#
FROM      Auftrag
GROUP BY  Kunden#
HAVING    count(distinct datum)>1

```

Gib die Rechnungsnummern an, bei welchen die größte menge die eingekauft wurde 20 überstiegen hat.

```

SELECT    re#
FROM      pos
GROUP BY  re#
HAVING    max(Menge)>20;

```

Gib die Rechnungsnummern an, bei welchen die Summe der Menge die Gekauft wurde größer 100 war

```

SELECT    re#
FROM      pos
GROUP BY  re#
HAVING    SUM(MENGE)>100

```

Gib die Rechnungsnummern an, wo der durchschnitt der eingekauften ware kleiner 10 war

```

SELECT    re#
FROM      pos

```

```
GROUP BY re#
HAVING AVG(MENGE)<10
```

Select mit UNION verbinden

Bei Union werden die Ergebnisse der einzelnen SELECT anweisungen aneinander gefügt.

Bei der Arbeit mit dem UNION Operator gibt es vier wichtige Regeln

- Die SELECT anweisungen müssen alle die Gleiche Anzahl an Spalten aufweisen.
- Die Spalten, die aneinandergefügt werden, müssen denselben Datentyp aufweisen.
- Nur die letzte SELECT anweisung darf eine Order By Komponente enthalten, Sortiert wird erst auf der Grundlage es Endergebnisses, also erst nachden alle Zwischenergebnisse aneinandergefügt worden sind.
- DISTINCT dar in den SELECT-Komponenten nicht vorkommen. Wenn UNION verwendet wird, löscht SQL automatisch alle doppelten Werte

Beispiel

Angenommen es gäbe zwei Kunden dateien. Eine mit Stammkunden und eine mit "Laufkunden" und wir wollen nun alle Kunden wissen die Huber heißen.

```
SELECT name
FROM Stammkunde
WHERE name='Huber'
UNION
SELECT name
FROM Laufkunde
WHERE name='Huber'
```

Unterabfragen

Bei Unterabfragen wie sie schon verwendet wurden sind noch folgende Punkte zu beachten:

1. Das Unterselect darf nur einen Spaltenausdruck enthalten
2. Die SELECT Komponente darf nicht DISTINCT enthalten
3. Eine ORDER BY Komponente ist nicht erlaubt.

JOINEN von Tabellen

Das Joinen von Tabellen ist ja bereits aus der Relationenalgebra bekannt

Beispiel

```
SELECT
```