SQL Data Manipulation Language (DML)

- **SELECT** – pozwala wydobyć dane z bazy danych
- **UPDATE** – uaktualnia/modyfikuje dane w bazie danych
- **DELETE** – usuwa dane z bazy danych
- **INSERT INTO** – wstawia nowe dane do tabeli bazy danych

SQL Data Definition Language (DDL)

- **CREATE TABLE** – tworzy nowe tabele
- **ALTER TABLE** – modyfikuje (zmienia) tabele
- **DROP TABLE** – usuwa tabele
- **CREATE INDEX** – tworzy indeks
- **DROP INDEX** – usuwa indeks

Zapytania są mechanizmem do uzyskiwania informacji z bazy danych. Wynik zapytania SQL zapisany jest jako zbiór wyników, tzw. result-set. Większość oprogramowania służącego obsłudze baz danych pozwala nawigować w zbiorze wyników za pomocą funkcji jak, np.: Move-To-First-Record, Get-Record-Content, Move-To-Next-Record, itd. (zobacz ADO.Net)

Aby uzyskać relację jako wynik zapytania (czyli zbiór wynikowy w postaci tabeli), samo zapytanie w minimalnej formie musi zawierać definicję schematu relacji odpowiedzi i definicję źródła danych z którego dane są pobierane. Jeżeli odpowiedzą jest relacja to do relacji odpowiedzi można sformułować kolejne zapytanie (z punktu widzenia pierwszego zapytania będzie to podzapytanie).

Bardzo często termin zapytanie jest utożsamiane z terminem kwerenda. Zapytania są podzbiorem kwerend. Kwerendy oprócz uzyskiwania informacji z bazy danych (zapytanie) mogą modyfikować, wstawiać lub usuwać dane, modyfikować schemat itp. W SQL rozkazy realizujące wymienione funkcje są oddzielnymi niezależnymi rozkazami (np. APPEND, INSERT, DELETE).

Średnik używany jest jako separator wyrażeń SQL, gdy wyrażenia te mają być wykonane jako jedno zgłoszenie do serwera bazy danych. Czasami średnik używany jest jako zakończenie każdego wyrażenia SQL (choć wcale nie musi być to konieczne, jak np. w MS Access lub SQL Server).

Wyrażenie SELECT

SELECT używane jest do selekcji danych z tabeli. Wynik zapisany jest w postaci tabeli (result-set).

Syntaktyka ANSI SQL 92

```
SELECT [ DISTINCT ]
  { { funkcja agregująca .. | wyrażenie [ AS nazwa kolumny ] } .,.. }
  | { kwalifikator.* }

FROM { { nazwa tabeli [ AS ][ nazwa korelacji ] [ ( nazwa kolumny.,.. ) ] }
  | { podzapytanie [ AS ] nazwa korelacji [ nazwa kolumny .,.. ] }
```

| **połączona tabela** } .,..

[WHERE **predykat**]
[GROUP BY { { [**nazwa tabeli | nazwa korelacji** .] } **nazwa kolumny** } .,..]
[HAVING **predykat**]
[{ UNION | INTERSECT | EXCEPT } [ALL]

[CORRESPONDING [BY (**nazwa kolumny** .,..)]]
instrukcja select | { TABLE **nazwa tabeli** } **| konstruktor tabeli** }
[ORDER BY { { **kolumna wyjściowa** [ASC | DESC] } .,...}
| { { **dodatnia liczba całkowita** [ASC | DESC] } .,...} ;

select-list:

table name
| **expression** [[AS] **alias**]
| *

search condition:
expression compare expression | **expression compare** { [ANY | SOME] | ALL }(**subquery**)
| **expression** IS [NOT] NULL| **expression** [NOT] BETWEEN **expression** AND **expression**
| **expression** [NOT] LIKE **pattern** [ESCAPE **expression**]
| **expression** [NOT] IN ({ **expression** | **subquery** | **value-expr1 , value-expr2 [, value-expr3**] ...})
| EXISTS (**subquery**)
| NOT **condition** | **condition** AND **condition** | **condition** OR **condition** | (**condition**)
| (**condition , estimate**)
| **condition** IS [NOT] { TRUE | FALSE | UNKNOWN }

compare:
= | > | < | >= | <= | <> | != | !< | !>

Przykłady

Selekcja wybranych kolumn

```
SELECT column_name(s)
FROM table_name
```

Aby wyselekcjonować kolumny "LastName" i "FirstName", należy użyć SELECT jak niżej:

```
SELECT LastName, FirstName FROM Persons
```

Jeśli mamy tabelę **"Persons"**

LastName	**FirstName**	**Address**	**City**
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

to wynikiem jest

LastName	**FirstName**
Hansen	Ola
Svendson	Tove

Pettersen	Kari

Selekcja wszystkich kolumn

Do selekcji wszystkich kolumn używa się symbolu * postawionego w miejsce nazw kolumn.

```
SELECT * FROM Persons
```

daje w wyniku:

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

Wyrażenie SELECT DISTINCT

Słowa kluczowego DISTINCT w wyrażeniach SQL służy do eliminacji powtarzające się wartości w zbiorze wynikowym:

Syntax

```
SELECT DISTINCT column_name(s) FROM table_name
```

Aby wyselekcjonować wszystkie wartości z kolumny nazwanej "Company", wystarczy użyć:
Dla tabeli "Orders"

Company	OrderNumber
Sega	3412
W3Schools	2312
Trio	4678
W3Schools	6798

polecenie

```
SELECT DISTINCT Company FROM Orders
```

da wynik:

Company
Sega
W3Schools
Trio

Klauzula WHERE

Klauzula WHERE służy do określenia warunku (kryterium) selekcji. Warunkowa selekcja ma następującą składnię:

Syntax

```
SELECT column FROM table
WHERE column operator value
```

W klauzuli WHERE można używać następujących operatorów:

Operator	Opis
=	równe
<>	nierówne, czasami zapisywane jako !=
>	większe niż
<	mniejsze niż
>=	większe lub równe
<=	mniejsze lub równe
BETWEEN	z zadanego przedziału (zamkniętego)
LIKE	zgodny z szablonem

Dla przykładu, polecenie:

```
SELECT * FROM Persons WHERE City='Sandnes'
```

dla tabeli "Persons"

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951
Svendson	Tove	Borgvn 23	Sandnes	1978
Svendson	Stale	Kaivn 18	Sandnes	1980
Pettersen	Kari	Storgt 20	Stavanger	1960

da wynik:

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951
Svendson	Tove	Borgvn 23	Sandnes	1978
Svendson	Stale	Kaivn 18	Sandnes	1980

Użycie apostrofów

W powyższych wyrażeniach wartości występujące w warunkach otoczone zostały pojedynczym apostrofem. SQL używa pojedynczych apostrofów wokół wartości tekstowych (choć większość baz danych akceptuje również podwójne apostrofy). Wartości numeryczne nie powinny być ujmowane w apostrofy.Numeric values should not be enclosed in quotes. Przykład dla wartości tekstowych:

```
Dobrze:
SELECT * FROM Persons WHERE FirstName='Tove'
Źle:
SELECT * FROM Persons WHERE FirstName=Tove
```

dla wartości numerycznych:

```
Dobrze:
SELECT * FROM Persons WHERE Year>1965
Źle:
SELECT * FROM Persons WHERE Year>'1965'
```

Warunek LIKE

Warunku LIKE używa się, aby określić szablon dla poszukiwanych wartości w kolumnach. Dla operatora LIKE wzorzec (ang. *pattern*) wyszukiwania może zawierać dowolna liczbę znaków dzikiej karty. Interpretację znaków dzikiej karty zawiera poniższa tabela:

znak	odpowiadające znaki
_ (pokreślnik)	dowolny jeden znak
% (procent)	dowolny ciąg znaków nawet zerowy
[]	dowolny jeden znak w podanym zakresie lub zbiorze
[^]	dowolny jeden znak nie należący do podanego zakresu lub zbioru
inne	tak jak we wzorcu

Syntax

```
SELECT column FROM table WHERE column LIKE pattern
```

Osoby, których imię zaczyna się od litery 'O':

```
SELECT * FROM Persons WHERE FirstName LIKE 'O%'
```

Osoby, których imiona kończą sie na 'a':

```
SELECT * FROM Persons WHERE FirstName LIKE '%a'
```

Poszukiwanie osób zawierających w imieniu zlepek 'la':

```
SELECT * FROM Persons WHERE FirstName LIKE '%la%'
```

Słowo kluczowe ORDER BY

ORDER BY używane jest do sortowania wyników. Dla tabeli **"Orders"**

Company	OrderNumber
Sega	3412
ABC Shop	5678
W3Schools	2312
W3Schools	6798

alfabetyczne sortowanie wg nazw firm można osiągnąć przez:

```
SELECT Company, OrderNumber FROM Orders ORDER BY Company
```

Wynik:

Company	OrderNumber
ABC Shop	5678
Sega	3412
W3Schools	6798
W3Schools	2312

Aby dokonać posortowania wg nazw firm i wg numerów zamówień:

```
SELECT Company, OrderNumber FROM Orders ORDER BY Company,
OrderNumber
```

Wynik:

Company	OrderNumber
ABC Shop	5678
Sega	3412
W3Schools	2312

W3Schools	6798

Posortowanie w odwrotnym porządku alfabetycznym:

```
SELECT Company, OrderNumber FROM Orders ORDER BY Company DESC
```

Wynik:

Company	OrderNumber
W3Schools	6798
W3Schools	2312
Sega	3412
ABC Shop	5678

Posortowanie w odwrotnym porządku alfabetycznym i wg numerów zamówienia:

```
SELECT Company, OrderNumber FROM Orders ORDER BY Company DESC,
OrderNumber ASC
```

Wynik:

Company	OrderNumber
W3Schools	2312
W3Schools	6798
Sega	3412
ABC Shop	5678

Operatory AND i OR

AND i OR pozwalają połączyć dwa lub więcej warunków klauzuli WHERE. Operator AND wyświetla wiersz, jeśli wszystkie warunki na liście są prawdziwe, OR wyświetla wiersz, jeśli któryś z warunków na liście jest prawdziwy.

"Persons"

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

Wszystkie osoby o imieniu "Tove" i nazwisku "Svendson":

```
SELECT * FROM Persons WHERE FirstName='Tove' AND LastName='Svendson'
```

Wynik:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes

Wszystkie osoby, które mają na imię "Tove" lub na nazwisko "Svendson":

```
SELECT * FROM Persons WHERE firstname='Tove' OR lastname='Svendson'
```

Wynik:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

Kombinacja operatorów AND i OR (można używać nawiasów):

```
SELECT * FROM Persons WHERE (FirstName='Tove' OR FirstName='Stephen')
AND LastName='Svendson'
```

Wynik:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes

Svendson	Stephen	Kaivn 18	Sandnes

Operator IN

Operator IN jest używany, jeśli znana jest dokładnie wartość, która wystąpić ma przynajmniej w jednej kolumnie.

```
SELECT column_name FROM table_name WHERE column_name IN (value1,value2,..)
```

"Persons"

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Nordmann	Anna	Neset 18	Sandnes
Pettersen	Kari	Storgt 20	Stavanger
Svendson	Tove	Borgvn 23	Sandnes

Osoby, których nazwiska to "Hansen" or "Pettersen":

```
SELECT * FROM Persons WHERE LastName IN ('Hansen','Pettersen')
```

Wynik:

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

Konstrukcja BETWEEN...AND

Operator BETWEEN ... AND pozwala na selekcję danych z przedziału ograniczonego przez dwie wartości. Wartościami tymi mogą być liczby, teksty lub daty. Uwaga, operator ten ma różne znaczenie w różnych bazach danych. Tzn. wartości ograniczające przedział albo wchodzą w skład zbiór wynikowego (zawierając) albo nie (wykluczając).

```
SELECT column_name FROM table_name WHERE column_name
BETWEEN value1 AND value2
```

Osoby alfabetycznie pomiędzy "Hansen" (zawierając) a "Pettersen" (wykluczając):

```
SELECT * FROM Persons WHERE LastName BETWEEN 'Hansen' AND 'Pettersen'
```

Wynik:

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Nordmann	Anna	Neset 18	Sandnes

Można też, dzięki użyciu operatora NOT, wyświetlać wartości spoza przedziału:

```
SELECT * FROM Persons WHERE LastName NOT BETWEEN 'Hansen' AND 'Pettersen'
```

Wynik:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Svendson	Tove	Borgvn 23	Sandnes

Aliasy

Aliasy nazw kolumn

```
SELECT column AS column_alias FROM table
```

Przykład użycia aliasu do nazw kolumn:
"Persons"

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

Wyrażenie:

```
SELECT LastName AS Family, FirstName AS Name FROM Persons
```

Wynik:

Family	Name
Hansen	Ola
Svendson	Tove
Pettersen	Kari

Aliasy nazw tablic

```
SELECT column FROM table AS table_alias
```

Wyrażenie:

```
SELECT LastName, FirstName FROM Persons AS Employees
```

Wynik:

LastName	FirstName
Hansen	Ola
Svendson	Tove
Pettersen	Kari

Join

Czasami wynik wyszukiwania jest zbiorem, który powstaje przez połączenie dwóch lub więcej tabel. W łączeniu tabel korzysta się z kluczy (głównych i obcych), choć mogą to być też inne parametry.

SQL ANSI 92

tabela A **CROSS JOIN** tabela B	złączenia krzyżowe
tablica A **[NATURAL] [** ***typ złączenia*** **] JOIN** tablica B	złączenia naturalne
tablica A **[** ***typ złączenia*** **] JOIN** tablica B **ON** ***predykat***	złączenie przez predykat
tablica A **[** ***typ złączenia*** **] JOIN** tablica B **USING** (***nazwa kolumny.,..***)	złączenia po danej kolumnie

typy złączenia INNER | { { LEFT | RIGHT | FULL | } [OUTER] }

Tabele:

Employees:	**Orders:**

Employee_ID	Name
01	Hansen, Ola
02	Svendson, Tove
03	Svendson, Stephen
04	Pettersen, Kari

Prod_ID	Product	Employee_ID
234	Printer	01
657	Table	03
865	Chair	03

Wyniki z dwóch tabel

Kto zamówił produkt, i co zamówił?

```
SELECT Employees.Name, Orders.Product FROM Employees, Orders
WHERE Employees.Employee_ID=Orders.Employee_ID
```

Wynik:

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair

Kto zamówił drukarkę?

```
SELECT Employees.Name FROM Employees, Orders
WHERE Employees.Employee_ID=Orders.Employee_ID AND Orders.Product='Printer'
```

Wynik:

Name
Hansen, Ola

Podobny wynik można uzyskać stosując JOIN

Użycie połączeń (JOIN)

INNER JOIN

INNER JOIN zwraca wszystkie wiersze z obu tabel, dla których istnieje dopasowanie.
Wiersze niedopasowane (niepowiązane) nie wejdą do produkowanego zbioru wynikowego.

Syntax

```
SELECT field1, field2, field3 FROM first_table
INNER JOIN second_table
ON first_table.keyfield = second_table.foreign_keyfield
```

Kto zamówił produkt, i co zamówił?

```
SELECT Employees.Name, Orders.Product FROM Employees
INNER JOIN Orders ON Employees.Employee_ID=Orders.Employee_ID
```

Wynik:

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair

LEFT JOIN

Syntax

```
SELECT field1, field2, field3 FROM first_table
LEFT JOIN second_table ON first_table.keyfield = second_table.foreign_keyfield
```

LEFT JOIN zwraca wszystkie wiersze z pierwszej tabeli (Employees), nawet jeśli nie mają one dopasowań w tabeli drugiej (Orders).
Wyszukanie wszytkich pracowników i ich zamówień (jeśli istnieją)

```
SELECT Employees.Name, Orders.Product FROM Employees
LEFT JOIN Orders ON Employees.Employee_ID=Orders.Employee_ID
```

Wynik:

Name	Product
Hansen, Ola	Printer
Svendson, Tove	
Svendson, Stephen	Table
Svendson, Stephen	Chair
Pettersen, Kari	

RIGHT JOIN

RIGHT JOIN zwraca wszystkie wiersze z tabeli drugiej (Orders), nawet, jeśli nie mają ode dopasowań w tabeli pierwszej (Employees).

Syntax

```
SELECT field1, field2, field3 FROM first_table
RIGHT JOIN second_table ON first_table.keyfield = second_table.foreign_keyfield
```

Wypisanie wszystkich zamówień oraz zamawiających – jeśli istnieją.

```
SELECT Employees.Name, Orders.Product FROM Employees
RIGHT JOIN Orders ON Employees.Employee_ID=Orders.Employee_ID
```

Wynik:

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair

Kto zamówił drukarkę?

```
SELECT Employees.Name FROM Employees
INNER JOIN Orders ON Employees.Employee_ID=Orders.Employee_ID
WHERE Orders.Product = 'Printer'
```

Wynik:

Name
Hansen, Ola

Wyrażenie INSERT INTO

INSERT INTO używane jest do wprowadzania nowych wierszy w tablicy.

Syntax

```
INSERT INTO table_name VALUES (value1, value2,....)
```

Można też wprowadzać wartości do kolumn, których nazwy pojawią się w poleceniu, jak np.:

```
INSERT INTO table_name (column1, column2,...) VALUES (value1, value2,....)
```

Wstawienie nowego wiersza

Niech będzie tabela "Persons":

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger

Wykonanie wyrażenia:

```
INSERT INTO Persons VALUES ('Hetland', 'Camilla', 'Hagabakka 24', 'Sandnes')
```

da wynik:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes

Wstawienie danych do konkretnej kolumny

Dla tabeli "Persons" jak wyżej wyrażenie:

```
INSERT INTO Persons (LastName, Address) VALUES ('Rasmussen', 'Storgt 67')
```

da wynik:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes
Rasmussen		Storgt 67	

Wyrażenie UPDATE

Wyrażenie UPDATE używane jest do modyfikacji danych w tabeli.

Syntax

```
UPDATE table_name
SET column_name = new_value
WHERE column_name = some_value
```

Dla tabeli "Person":

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen		Storgt 67	

Modyfikacja jednej kolumny w wierszu:

Aby dodać imię do osoby, której nazwisko brzmi "Rasmussen" należy:

```
UPDATE Person SET FirstName = ‘Nina’
WHERE LastName = 'Rasmussen'
```

Wynikiem jest:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Storgt 67	

Modyfikacja wielu kolumn w wierszu

Aby zmienić adres (ulicę) i dodać nazwę miasta można wykonać:

```
UPDATE Person
SET Address = 'Stien 12', City = 'Stavanger'
WHERE LastName = 'Rasmussen'
```

Wynikiem jest:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

Wyrażenie DELETE

DELETE używa się do usuwania wierszy w tabeli.

Syntax

```
DELETE FROM table_name WHERE column_name = some_value
```

Na przykład dla tabeli "Person":

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

aby usunąć "Nina Rasmussen" należy wykonać:

```
DELETE FROM Person WHERE LastName = 'Rasmussen'
```

Wynikiem jest:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger

Usuwanie wszystkich wierszy

Można usunąć wszytkie wiersze tabeli bez usuwania tabeli (tj. przy zachowaniu struktury tabeli, jej atrybutów i indeksów):

```
DELETE FROM table_name or DELETE * FROM table_name
```

UNION and UNION ALL

UNION

Komenda UNION jest używana do selekcji odpowiadających sobie informacji z dwóch tabel, podobnie jak komenda JOIN. Z tym, że dla UNION kolumny selekcjonowane muszą przechowywać dane jednakowego typu. UNION dokonuje selekcji bez powtórzeń (tj. tylko różnych wartości).

```
SQL Statement 1 UNION SQL Statement 2
```

UNION - zwraca wszystkie elementy wszystkich zapytań. Łączy wyniki jednego lub więcej wyrażenia SELECT, które muszą być zgodne w wybieranych kolumnach a wyrażenie SELECT nie powinno używać wyrażenia ORDER BY

```
... UNION [ALL] select-without-order-by
```

```
... [ UNION [ALL] select-without-order-by ] ...
... [ ORDER BY integer [ ASC | DESC ], ... ]
```

SELECT city AS Cities FROM contact
UNION SELECT city FROM customer
UNION SELECT city FROM employee

INTERSECT -wyświetla tylko wiersze mające odpowiedniki (wspólna część), wynik nie zawiera powtarzających się wierszy.

SELECT city AS Cities FROM contact
INTERSECT SELECT city FROM customer

MINUS -wybiera elementy znajdujące się w jednej z tabel, a nie należące do drugiej. Zamiast słowa kluczowego MINUS można użyć EXCEPT. Wynik nie zawiera powtarzających się wierszy.

SELECT city AS Cities FROM contact
MINUS SELECT city FROM customer

Przykład:

Employees_Norway	
Employee_ID	**E_Name**
01	Hansen, Ola
02	Svendson, Tove
03	Svendson, Stephen

Employees_USA	
Employee_ID	**E_Name**
01	Turner, Sally
02	Kent, Clark
03	Svendson, Stephen

Lista wszystkich różnych (powtórki nie zostaną uwzględnione) nazwisk pracowników w Norway i USA:

```
SELECT E_Name FROM Employees_Norway
UNION
SELECT E_Name FROM Employees_USA
```

Wynik:

Name
Hansen, Ola
Svendson, Tove
Svendson, Stephen
Pettersen, Kari
Turner, Sally
Kent, Clark
Scott, Stephen

UNION ALL

Działa podobnie jak UNION, tylko że selekcjonuje wszystkie wartości.

```
SQL Statement 1 UNION ALL SQL Statement 2
```

Lista wszystkich pracowników w Norway i USA:

```
SELECT E_Name FROM Employees_Norway
UNION ALL
SELECT E_Name FROM Employees_USA
```

Wynik

Name

Hansen, Ola
Svendson, Tove
Svendson, Stephen
Pettersen, Kari
Turner, Sally
Kent, Clark
Svendson, Stephen
Scott, Stephen

Create Database, Table, and Index

Create Database

SQL ANSI 92:

```
CREATE [ {GLOBAL | LOCAL} TEMPORARY ] TABLE nazwa-tablicy
( { definicja-kolumny
| [ ograniczenie tablicy ] } .,..

  [ ON COMMIT { DELETE | PRESERVE } ROW ] ) ;

definicja kolumny ::=
  nazwa kolumny { nazwa domeny | typ danych [rozmiar ] }
  [ ograniczenie kolumny ... ]

[ DEFAULT domyślna wartość ]
[ COLLATE nazwa porównania ]
```

Utworzenie bazy danych:

```
CREATE DATABASE database_name
```

Create Table

Utworzenie tabeli:

```
CREATE TABLE table_name
(
column_name1 data_type,
column_name2 data_type,
.......
)
```

Utworzenie tablicy "Person", z czterema kolumnami: "LastName", "FirstName", "Address", "Age":

```
CREATE TABLE Person
(
LastName varchar,
FirstName varchar,
Address varchar,
Age int
)
```

Ograniczenia na typ:

```
CREATE TABLE Person
(
```

```
LastName varchar(30),
FirstName varchar,
Address varchar,
Age int(3)
)
```

Utworzenie indeksu

Index

Indeksy tworzone są w celu przyspieszenia operacji wyszukiwania. Możliwe jest indeksowanie na jednej lub wiecej kolumnach tabeli (wtedy każdy indeks ma inną nazwę). Indeksów nie można zobaczyć (są one ukryte w bazie danych).
Modyfikacja tabel z indeksami zajmuje więcej czasu, gdyż indeksy muszą zostać na nowo przeliczone.

SQL Sybase AnyWHERE

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX nazwa-indeksu
ON [[ baza.] właściciel.] nazwa-tabeli (nazwa-kolumny
[,nazwa-kolumny]...)
[WITH {{FILLFACTOR | MAX_ROWS_PER_PAGE} = x,
CONSUMERS = x, IGNORE_DUP_KEY, SORTED_DATA,
[IGNORE_DUP_ROW | ALLOW_DUP_ROW]}]
[ON SEGMENT_NAME]
```

Unique Index

Unikalny indeks tabeli znaczy tyle, że dwa wiersze w tabeli nie mogą mieć takich samych indeksów.

```
CREATE UNIQUE INDEX index_name ON table_name (column_name)
```

gdzie "column_name" to nazwa kolumny indeksowanej.

Simple Index

Tworzy prosty indeks na tabeli (gdy nie ma słowa kluczowego UNIQUE dopuszczalne są duplikaty).

```
CREATE INDEX index_name ON table_name (column_name)
```

gdzie "column_name" to nazwa kolumny indeksowanej.

Utworzenie indeksu o nazwie "PersonIndex", dla kolumny LastName tabeli Person:

```
CREATE INDEX PersonIndex ON Person (LastName)
```

Indeksowanie może być malejące, wtedy używa się słowa kluczowego **DESC** po nazwie kolumny:

```
CREATE INDEX PersonIndex ON Person (LastName DESC)
```

Indeksy na większej ilości kolumn deklaruje się pisząc nazwy kolumn oddzielone przecinkami, zamknięte w nawias:

```
CREATE INDEX PersonIndex ON Person (LastName, FirstName)
```

Drop

Drop Index

Usuwanie indeksu:

```
DROP INDEX table_name.index_name
```

Drop Table

Usuwanie tabeli (struktury, atrybutów, indeksów)

```
DROP TABLE table_name
```

Drop Database

Usuwanie bazy danych:

```
DROP DATABASE database_name
```

Truncate Table

Aby usunąć część dane z tabeli (ale nie samą tabelę) używa się polecenia TRUNCATE TABLE (deletes only the data inside the table):

```
TRUNCATE TABLE table_name
```

ALTER TABLE

Wyrażenie ALTER TABLE używane jest do dodawania lub usuwania kolumn do/z istniejącej tabeli (choć niektóre systemy baz danych nie dopuszczają do usuwania kolumn z tabel).

```
ALTER TABLE table_name ADD column_name datatype
ALTER TABLE table_name DROP COLUMN column_name
```

Person:

LastName	FirstName	Address
Pettersen	Kari	Storgt 20

Aby dodać kolumnę "City" do tabeli "Person":

```
ALTER TABLE Person ADD City varchar(30)
```

Wynik:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	

Aby usunąć kolumnę "Address" z tabeli "Person"

```
ALTER TABLE Person DROP COLUMN Address
```

Wynik:

LastName	FirstName	City

Pettersen	Kari	

Funkcje SQL

Syntax

Wbudowane funkcje SQL wywołuje się w następujący sposób:

```
SELECT function(column) FROM table
```

Typy funkcji

Podstawowe typy funkcji to:

- funkcje agregujące
- funkcje skalarne

Funkcje agregujące

Działają na kolekcjach wartości, zwracają jednak pojedynczą wartość. Jeśli pojawiają się na liście parametrów instrukcji SELECT, to SELECT musi mieć klauzulę GROUP BY.

SQL ANSI 92:

```
funkcja agregująca ::=
        { COUNT (*) } |
            { { AVG
            | SUM
            | MAX
            | MIN

            COUNT }
        ( [ DISTINCT | ALL ] wyrażenie ) }
```

```
 GROUP BY { { [ nazwa tablicy | nazwa korelacji } . ] nazwa kolumny } .,..]
[ HAVING predykat ]
```

"Persons"

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	19

Funkcje agregujące MS Access

Function	Description

AVG(column)	Returns the average value of a column
COUNT(column)	Returns the number of rows (without a NULL value) of a column
COUNT(*)	Returns the number of selected rows
FIRST(column)	Returns the value of the first record in a specified field
LAST(column)	Returns the value of the last record in a specified field
MAX(column)	Returns the highest value of a column
MIN(column)	Returns the lowest value of a column
STDEV(column)	
STDEVP(column)	
SUM(column)	Returns the total sum of a column
VAR(column)	
VARP(column)	

Funkcje agregujące SQL Server

Function	Description
AVG(column)	Returns the average value of a column
BINARY_CHECKSUM	
CHECKSUM	
CHECKSUM_AGG	
COUNT(column)	Returns the number of rows (without a NULL value) of a column
COUNT(*)	Returns the number of selected rows
COUNT(DISTINCT column)	Returns the number of distinct results
FIRST(column)	Returns the value of the first record in a specified field (not supported in SQLServer2K)
LAST(column)	Returns the value of the last record in a specified field (not supported in SQLServer2K)
MAX(column)	Returns the highest value of a column
MIN(column)	Returns the lowest value of a column
STDEV(column)	
STDEVP(column)	
SUM(column)	Returns the total sum of a column
VAR(column)	
VARP(column)	

Funkcje skalarne

Funkcje skalarne działają na pojedynczych wartościach i zwracają pojedyncze wartości.

Funkcje skalarne MS Access

Function	Description

UCASE(c)	Converts a field to upper case
LCASE(c)	Converts a field to lower case
MID(c,start[,end])	Extract characters from a text field
LEN(c)	Returns the length of a text field
INSTR(c)	Returns the numeric position of a named character within a text field
LEFT(c,number_of_char)	Return the left part of a text field requested
RIGHT(c,number_of_char)	Return the right part of a text field requested
ROUND(c,decimals)	Rounds a numeric field to the number of decimals specified
MOD(x,y)	Returns the remainder of a division operation
NOW()	Returns the current system date
FORMAT(c,format)	Changes the way a field is displayed
DATEDIFF(d,date1,date2)	Used to perform date calculations

GROUP BY oraz HAVING

Funkcje agregujące często wymagają grupowania przez GROUP BY.

GROUP BY... było dodane do SQL ponieważ funkcje agregujące (jak SUM) zwracają agregację wszystkich wartości w kolumnie. Bez GROUP BY niemożliwe było obliczenie wartości przez funkcję agregującą dla indywidualnych grup danych.

Syntax

```
SELECT column,SUM(column) FROM table GROUP BY column
```

"Sales"

Company	**Amount**
W3Schools	5500
IBM	4500
W3Schools	7100

Wyrażenie SQL:

```
SELECT Company, SUM(Amount) FROM Sales
```

Wynik (błędny, bo bez grupowania):

Company	**SUM(Amount)**
W3Schools	17100
IBM	17100
W3Schools	17100

Wyrażenie SQL:

```
SELECT Company,SUM(Amount) FROM Sales GROUP BY Company
```

Wynik:

Company	**SUM(Amount)**
W3Schools	12600
IBM	4500

HAVING...

HAVING... dodano do SQL, ponieważ WHERE nie można używać do funkcji agregujących (jak SUM). Bez HAVING... nie było możliwe testowanie warunków dla wyników. Syntax:

```
SELECT column,SUM(column) FROM table
GROUP BY column
HAVING SUM(column) condition value
```

Wyrażenie SQL:

```
SELECT Company,SUM(Amount) FROM Sales
GROUP BY Company
HAVING SUM(Amount)>10000
```

Wynik:

Company	**SUM(Amount)**
W3Schools	12600

SELECT INTO

Wyrażenie SELECT INTO używane jest najczęściej do robienia kopii tabeli lub do archiwizowania rekordów.

Syntax

```
SELECT column_name(s) INTO newtable [IN externaldatabase] FROM source
```

Utworzenie kopii tabeli "Persons":

```
SELECT * INTO Persons_backup
FROM Persons
```

Klazula IN używana jest do kopiowania tabeli do innych baz danych:

```
SELECT Persons.* INTO Persons IN 'Backup.mdb' FROM Persons
```

Jeśli kopiowanych ma być kilko pól, wtedy można je wymienić:

```
SELECT LastName,FirstName INTO Persons_backup FROM Persons
```

Można również dodać klauzulę WHERE.

Utworzenie tabeli "Persons_backup" z dwoma kolumnami (FirstName i LastName) przez wyekstrachowanie z tabeli "Persons" tych osób, które mieszkają w "Sandnes"

```
SELECT LastName,Firstname INTO Persons_backup FROM Persons
WHERE City='Sandnes'
```

Selekcja danych z więcej niż jednej tabeli:
"Empl_Ord_backup" tabela zawierająca dane z Employees and Orders:

```
SELECT Employees.Name,Orders.Product
INTO Empl_Ord_backup
FROM Employees
INNER JOIN Orders
```

```
ON Employees.Employee_ID=Orders.Employee_ID
```

CREATE VIEW

View jest wirtulaną tabelą bazującą na wynikach instrukcji SELECT (z użyciem WHERE oraz JOIN). Jest ona tworzona przez SZBD w chwilach, gdy użytkownik użyje view.

Syntax

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

View może być użyte z wnętrza zapytania, z procedury zapisanej, z wmętrza innego view. Przez dodanie funkcji, połączeń (join) to view można dostarczać użytkownikowi wybrabe fragmenty danych.

Przykład z bazy danych Northwind:
View "Current Product List" przedstawia listę wszystkich aktywnych produktów z tabeli Products.

```
CREATE VIEW [Current Product List] AS
SELECT ProductID,ProductName
FROM Products
WHERE Discontinued=No
```

Na view można wykonywać zapytania:

```
SELECT * FROM [Current Product List]
```

View zawierające każdy produkt tabeli Products, który ma cenę jednostkową wyższą niż zadana wartość:

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName,UnitPrice
FROM Products
WHERE UnitPrice>(SELECT AVG(UnitPrice) FROM Products)
```

Na view można wykonać zapytanie:

```
SELECT * FROM [Products Above Average Price]
```

View z obliczoną całkowitą sprzedażą dla każdej kategorii w 1997 roku (wykorzystane jest inne view "Product Sales for 1997"):

```
CREATE VIEW [Category Sales For 1997] AS
SELECT DISTINCT CategoryName,Sum(ProductSales) AS CategorySales
FROM [Product Sales for 1997]
GROUP BY CategoryName
```

Na view można wykonać zapytanie:

```
SELECT * FROM [Category Sales For 1997]
```

Do zapytania można również wstawić warunek (np. pozwalający wypisać całkowitą sprzedaż dla kategorii "Beverages":

```
SELECT * FROM [Category Sales For 1997] WHERE CategoryName='Beverages'
```

Podzapytania

Kiedy tworzymy zapytanie używając klauzul WHERE i HAVING do wybrania wierszy, często zdarza się, że warunek wyboru jest informacją przechowywaną w więcej niż jednej tabeli. Podzapytania w klauzulach WHERE lub HAVING powalają na wybór wierszy z tabeli zgodnie z wymogami uzyskanymi z innej tabeli czyli podzapytania. Podzapytania to zapytania zwracające dane, które będą wykorzystane w dalszej części instrukcji. Podzapytań używa się w predykatach innych zapytań, instrukcji DELETE, UPDATE oraz w ograniczeniach. Można ich również używać w klauzuli FROM zapytania, konstruktorach wierszy i wyrażeniach.
Podzapytania dzielą się na trzy rodzaje:

- podzapytania zwracające tylko jedną wartość - skalarne,
- podzapytania zwracające tylko jeden wiersz, który może zawierać więcej niż jedną wartość wierszowe,
- podzapytania zwracające dowolną ilość wierszy -tablicowe.

Podzapytania skalarne używa się w predykatach porównania bez kwalifikatorów. Poniższy przykład wyświetla nazwiska, imiona i pensje wszystkich pracowników zarabiających więcej niż średnia pensja w firmie.

```
SELECT emp_lname , emp_fname , salary
FROM employee
WHERE salary >( SELECT AVG(salary)FROM employee ) ORDER BY salary DESC;
```

Podzapytania wierszowe
Podzapytania tablicowe

```
SELECT order_date, sales_rep
FROM sales_order
WHERE cust_id IN (

  SELECT id
      FROM customer
      WHERE lname = 'Clarke' OR lname = 'Boyle');
```

Podzapytania zagnieżdżone

```
SELECT id, line_id
FROM sales_order_items
WHERE ship_date = ANY (

  SELECT order_date
  FROM sales_order
  WHERE fin_code_id IN (
```

SELECT code
FROM fin_code
WHERE (description = 'Fees')))

Warunki szukania:

wyrażenie porównanie wyrażenie
| wyrażenie porównanie { [**ANY** | **SOME**] | **ALL** }(podzapytanie)
| wyrażenie **IS** [**NOT**] **NULL**
| wyrażenie [**NOT**] **BETWEEN** wyrażenie **AND** wyrażenie
| wyrażenie [**NOT**] **LIKE** wyrażenie [**ESCAPE** wyrażenie]
| wyrażenie [**NOT**] **IN** ({ wyrażenie | podzapytanie | wyrażenie_zwracające_wartość1,

wyrażenie_zwracające_wartość2 [, wyrażenie_zwracające_wartość3] ...})

| **EXISTS** (podzapytanie)
| **NOT** warunek
| warunek **AND** warunek
| warunek **OR** warunek
| (warunek)
| (warunek, obliczenie)
| warunek **IS** [**NOT**] { **TRUE** | **FALSE** | **UNKNOWN** }

Operator ANY jest używany w połączeniu z operatorami porównania (=, <>, <, <=, >, >=) do porównania pojedynczej wartości kolumny z danymi otrzymanymi jak wynik działania podzapytania. SQL porównuje wartość sprawdzaną z każdą wartością kolumny zwracanej przez podzapytanie, jeżeli dowolne porównanie zwróci wartość PRAWDA to operator ANY zwróci wartość prawda. Synonimem operatora ANY jest SOME.
Podobnie jak ANY, ALL jest używane z operatorami porównania (=, <>, <, <=, >, >=). SQL porównuje wartość sprawdzaną z każdą wartością kolumny zwracanej przez podzapytanie, jeżeli wszystkie porównania zwrócą wartość PRAWDA to operator ALL zwróci wartość prawda.
Sprawdzanie EXISTS sprawdza czy podzapytanie zwraca jakiekolwiek wiersze, jeżeli podzapytanie zwróci jeden lub więcej wierszy to EXISTS zwróci wartość PRAWDA w przeciwnym wypadku FAŁSZ.

Podzapytania nieskorelowane

SELECT name, description
FROM product
*WHERE quantity < 2 * (*

SELECT avg (quantity)
FROM sales_order_items)

W tym przykładzie, podzapytanie wylicza dokładnie jedna wartość: średnią ilość z tabeli sales_order_items. W wykonywanym zapytaniu SQL wylicza wartość raz i porównuje z każdą wartością pola quantity tabeli product w celu wybrania spełniających warunek wierszy.

Podzapytania skorelowane

Kiedy zapytanie zawiera odwołanie zewnętrzne jak w przykładzie poniżej

```
SELECT name, description
FROM product
WHERE quantity < 2 * (

  SELECT avg (quantity)
      FROM sales_order_items
      WHERE product.id=sales_order_items.prod_id)
```

zwraca wartość zależną od aktywnych wierszy w tabeli product. Takie podzapytania nazywamy pozapytaniami skorelowanymi. W tym przypadku podzapytanie może zwrócić różne wartości dla każdego wiersza zewnętrznego zapytania.

Podzapytania równoległe

```
SELECT emp_lname, emp_fname, YEARS(birth_date, CURRENT DATE) as wiek,
salary
FROM employee
WHERE wiek > ( SELECT AVG(YEARS(birth_date, CURRENT DATE)) FROM
employee )
AND salary > (SELECT AVG( salary ) FROM employee )
Order by wiek DESC;
```

Zapytania krzyżowe

```
SELECT DISTINCT 'Podsumowanie',
        ( SELECT COUNT(*) FROM employee
        WHERE sex='M' ) AS 'Chłop',
        ( SELECT COUNT(*) FROM employee
        WHERE sex='F' ) AS 'Baba'
FROM employee

SELECT DISTINCT region,
    ( SELECT COUNT(*) AS '1993 '
    FROM sales_order AS s
    WHERE ( ( so.region=s.region ) AND ( YEAR(order_date ) = '1993' ) ) ) ,
    ( SELECT COUNT(*) AS '1994'
    FROM sales_order AS s
    WHERE ( ( so.region=s.region ) AND ( YEAR(order_date ) = '1994' ) ) )
FROM sales_order AS so
ORDER BY 1;
```

Quick Reference

SQL Syntax

Statement	Syntax
AND / OR	SELECT column_name(s) FROM table_name WHERE condition AND|OR condition
ALTER TABLE (add column)	ALTER TABLE table_name ADD column_name datatype
ALTER TABLE (drop column)	ALTER TABLE table_name DROP COLUMN column_name
AS (alias for column)	SELECT column_name AS column_alias FROM table_name
AS (alias for table)	SELECT column_name FROM table_name AS table_alias
BETWEEN	SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value1 AND value2
CREATE DATABASE	CREATE DATABASE database_name
CREATE INDEX	CREATE INDEX index_name ON table_name (column_name)
CREATE TABLE	CREATE TABLE table_name (column_name1 data_type, column_name2 data_type, ...)
CREATE UNIQUE INDEX	CREATE UNIQUE INDEX index_name ON table_name (column_name)
CREATE VIEW	CREATE VIEW view_name AS SELECT column_name(s) FROM table_name WHERE condition
DELETE FROM	DELETE FROM table_name (**Note:** Deletes the entire table!!) *or* DELETE FROM table_name WHERE condition
DROP DATABASE	DROP DATABASE database_name
DROP INDEX	DROP INDEX table_name.index_name
DROP TABLE	DROP TABLE table_name
GROUP BY	SELECT column_name1,SUM(column_name2) FROM table_name GROUP BY column_name1
HAVING	SELECT column_name1,SUM(column_name2) FROM table_name GROUP BY column_name1

	HAVING SUM(column_name2) condition value
IN	SELECT column_name(s) FROM table_name WHERE column_name IN (value1,value2,..)
INSERT INTO	INSERT INTO table_name VALUES (value1, value2,....) *or* INSERT INTO table_name (column_name1, column_name2,...) VALUES (value1, value2,....)
LIKE	SELECT column_name(s) FROM table_name WHERE column_name LIKE pattern
ORDER BY	SELECT column_name(s) FROM table_name ORDER BY column_name [ASC|DESC]
SELECT	SELECT column_name(s) FROM table_name
SELECT *	SELECT * FROM table_name
SELECT DISTINCT	SELECT DISTINCT column_name(s) FROM table_name
SELECT INTO (used to create backup copies of tables)	SELECT * INTO new_table_name FROM original_table_name *or* SELECT column_name(s) INTO new_table_name FROM original_table_name
TRUNCATE TABLE (deletes only the data inside the table)	TRUNCATE TABLE table_name
UPDATE	UPDATE table_name SET column_name=new_value [, column_name=new_value] WHERE column_name=some_value
WHERE	SELECT column_name(s) FROM table_name WHERE condition