19. UNION

The SQL **UNION** clause/operator is used to combine the results of two or more SELECT Statements without returning any duplicate rows.

To use UNION, each SELECT must have the same number of columns selected, the same number of column expressions, the same data type, and have them in the same order but they do not have to be the same length.

Syntax:
The basic syntax of **UNION** is as follows:

```
SELECT column1 [,
column2 ] FROM table1
[, table2 ] [WHERE
condition]

UNION

SELECT column1 [,
column2 ] FROM table1
[, table2 ] [WHERE
condition]
```

Here given condition could be any given expression based on your requirement.

Example:
Consider following two tables, (a) CUSTOMERS table is as follows:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

(b) Another table is ORDERS as follows:

```
+-----+---------------------+-------------+--------+
OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```

|Now let us join these two tables in our SELECT statement as follows:

```
SQL> SELECT  ID, NAME,
   AMOUNT, DATE FROM
   CUSTOMERS
   LEFT JOIN ORDERS
   ON CUSTOMERS.ID =
ORDERS.CUSTOMER_ID UNION
   SELECT  ID, NAME,
   AMOUNT, DATE FROM
   CUSTOMERS
   RIGHT JOIN ORDERS
   ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce following result:

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
|    1 | Ramesh   |   NULL | NULL                |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|    5 | Hardik   |   NULL | NULL                |
|    6 | Komal    |   NULL | NULL                |
|    7 | Muffy    |   NULL | NULL                |
+------+----------+--------+---------------------+
```

20. NULL

The SQL **NULL** is the term used to represent a missing value. A NULL value in a table is a value in a field that appears to be blank.

A field with a NULL value is a field with no value. It is very important to understand that a NULL Value is different than a zero value or a field that contains spaces.

Here **NOT NULL** signifies that column should always accept an explicit value of the given data type. There are two columns where we did not use NOT NULL which means these column could be NULL.

A field with a NULL value is one that has been left blank during record creation.

Example:

The NULL value can cause problems when selecting data, however, because when comparing an unknown value to any other value, the result is always unknown and not included in the final results.
You must use the **IS NULL** or **IS NOT NULL** operators in order to check for a NULL value. Consider following table, CUSTOMERS having following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |          |
|  7 | Muffy    |  24 | Indore    |          |
+----+----------+-----+-----------+----------+
```

Now following is the usage of **IS NOT NULL** operator:

```
SQL> SELECT  ID, NAME, AGE,
   ADDRESS, SALARY FROM
   CUSTOMERS

    WHERE SALARY IS NOT
                NULL;
```

This would produce following result:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
+----+----------+-----+-----------+----------+
```

21. ALIAS

You can rename a table or a column temporarily by giving another name known as alias.
The use of table aliases means to rename a table in a particular SQL statement. The renaming is a temporary change and the actual table name does not change in the database.
The column aliases are used to rename a table's columns for the purpose of a particular SQL query.

Syntax:
The basic syntax of **table** alias is as follows:

```
SELECT column1,
column2.... FROM
table_name AS alias_name
WHERE [condition];
```

The basic syntax of **column** alias is as follows:

```
SELECT column_name AS alias_name
FROM table_name
WHERE [condition];
```

Example:
Consider following two tables, (a) CUSTOMERS table is as follows

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

(b) Another table is ORDERS as follows:

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```

Now following is the usage of **table alias**:

```
SQL> SELECT C.ID, C.NAME, C.AGE,
     O.AMOUNT FROM CUSTOMERS AS
     C, ORDERS AS O WHERE  C.ID =
     O.CUSTOMER_ID;
```

This would produce following result:

```
+----+----------+-----+--------+
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
|  3 | kaushik  |  23 |   3000 |
|  3 | kaushik  |  23 |   1500 |
|  2 | Khilan   |  25 |   1560 |
|  4 | Chaitali |  25 |   2060 |
+----+----------+-----+--------+
```

Following is the usage of **column alias**:

```
SQL> SELECT  ID AS CUSTOMER_ID, NAME AS
   CUSTOMER_NAME FROM CUSTOMERS
    WHERE SALARY IS NOT
                NULL;
```

This would produce following result:

```
+-------------+---------------+
| CUSTOMER_ID | CUSTOMER_NAME |
+-------------+---------------+
|           1 | Ramesh        |
|           2 | Khilan        |
|           3 | kaushik       |
|           4 | Chaitali      |
|           5 | Hardik        |
|           6 | Komal         |
|           7 | Muffy         |
+-------------+---------------+
```

22. **ALTER TABLE**

The SQL **ALTER TABLE** command is used to add, delete, or modify columns in an existing table. You would also use ALTER TABLE command to add and drop various constraints on a an existing
table.

Syntax:

The basic syntax of **ALTER TABLE** to add a new column in an existing table is as follows:

ALTER TABLE table_name ADD column_name datatype;

The basic syntax of ALTER TABLE to **DROP COLUMN** in an existing table is as follows:

ALTER TABLE table_name DROP COLUMN column_name;

The basic syntax of ALTER TABLE to change the **DATA TYPE** of a column in a table is as follows:

ALTER TABLE table_name MODIFY COLUMN column_name datatype;

The basic syntax of ALTER TABLE to add a **NOT NULL** constraint to a column in a table is as follows:

ALTER TABLE table_name MODIFY column_name datatype NOT NULL;

The basic syntax of ALTER TABLE to **ADD UNIQUE CONSTRAINT** to a table is as follows:

ALTER TABLE table_name
ADD CONSTRAINT MyUniqueConstraint UNIQUE (column1, column2...);

The basic syntax of ALTER TABLE to **ADD CHECK CONSTRAINT** to a table is as follows:

ALTER TABLE table_name
ADD CONSTRAINT MyUniqueConstraint CHECK (CONDITION);

The basic syntax of ALTER TABLE to **ADD PRIMARY KEY** constraint to a table is as follows:

ALTER TABLE table_name
ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);

The basic syntax of ALTER TABLE to **DROP CONSTRAINT** from a table is as follows:

```
ALTER TABLE table_name
DROP CONSTRAINT MyUniqueConstraint;
```

If you're using MySQL, the code is as follows:

```
ALTER TABLE table_name
DROP INDEX MyUniqueConstraint;
```

The basic syntax of ALTER TABLE to **DROP PRIMARY KEY** constraint from a table is as follows:

```
ALTER TABLE table_name
DROP CONSTRAINT MyPrimaryKey;
```

If you're using MySQL, the code is as follows:

```
ALTER TABLE table_name
DROP PRIMARY KEY;
```

Example:
Consider CUSTOMERS table is having following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is the example to ADD a new column in an existing table:

```
ALTER TABLE CUSTOMERS ADD SEX char(1);
```

Now CUSTOMERS table is changed and following would be output from SELECT statement:

```
+----+---------+-----+-----------+----------+------+
| ID | NAME    | AGE | ADDRESS   | SALARY   | SEX  |
+----+---------+-----+-----------+----------+------+
|  1 | Ramesh  |  32 | Ahmedabad |  2000.00 | NULL |
|  2 | Ramesh  |  25 | Delhi     |  1500.00 | NULL |
|  3 | kaushik |  23 | Kota      |  2000.00 | NULL |
|  4 | kaushik |  25 | Mumbai    |  6500.00 | NULL |
|  5 | Hardik  |  27 | Bhopal    |  8500.00 | NULL |
|  6 | Komal   |  22 | MP        |  4500.00 | NULL |
|  7 | Muffy   |  24 | Indore    | 10000.00 | NULL |
+----+---------+-----+-----------+----------+------+
```

Following is the example to DROP sex column from existing table:

```
ALTER TABLE CUSTOMERS DROP SEX;
```

Now CUSTOMERS table is changed and following would be output from SELECT statement:

```
+----+---------+-----+-----------+----------+
| ID | NAME    | AGE | ADDRESS   | SALARY   |
+----+---------+-----+-----------+----------+
|  1 | Ramesh  |  32 | Ahmedabad |  2000.00 |
|  2 | Ramesh  |  25 | Delhi     |  1500.00 |
|  3 | kaushik |  23 | Kota      |  2000.00 |
|  4 | kaushik |  25 | Mumbai    |  6500.00 |
|  5 | Hardik  |  27 | Bhopal    |  8500.00 |
|  6 | Komal   |  22 | MP        |  4500.00 |
|  7 | Muffy   |  24 | Indore    | 10000.00 |
+----+---------+-----+-----------+----------+
```

23. **TRUNCATE TABLE**

The SQL **TRUNCATE TABLE** command is used to delete complete data from an existing table. You can also use DROP TABLE command to delete complete table but it would remove complete table structure form the database and you would need to re-create this table once again if you wish you store some data.

Syntax:
The basic syntax of **TRUNCATE TABLE** is as follows:

```
TRUNCATE TABLE  table_name;
```

Example:

Consider CUSTOMERS table is having following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is the example to turncate:

SQL > TRUNCATE TABLE CUSTOMERS;
Now CUSTOMERS table is truncated and following would be output from SELECT statement:
SQL> SELECT * FROM
CUSTOMERS; Empty set (0.00
sec)

The HAVING clause enables you to specify conditions that filter which group results appear in the final results.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

Syntax:

The following is the position of the HAVING clause in a query:

```
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
```

The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used. The following is the syntax of the SELECT statement, including the HAVING clause:

```
SELECT column1, column2
FROM table1, table2
WHERE [ conditions ]
GROUP BY column1, column2
HAVING [ conditions ]
ORDER BY column1, column2
```

Example:

Consider CUSTOMERS table is having following records:

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is the example which would display record for which similar age count would be more than or equal to 2:

```
SQL > SELECT *
FROM
CUSTOMERS
GROUP BY age
HAVING COUNT(age) >= 2;
```

This would produce following result:

```
+----+--------+-----+---------+---------+
| ID | NAME   | AGE | ADDRESS |
SALARY  |
+----+--------+-----+---------+---------+
|  2 | Khilan |  25 | Delhi   | 1500.00 |
+----+--------+-----+---------+---------+
```

24. DATE FUNCTIONS

Following is the list of all important Date and Time related functions available through SQL. There are various other functions supported by your RDBMS. Given list is based on MySQL RDBMS.

Name	Description
ADDDATE()	Add dates
ADDTIME()	Add time
CONVERT_TZ()	Convert from one timezone to another
CURDATE()	Return the current date
CURRENT_DATE(), CURRENT_DATE	Synonyms for CURDATE()
CURRENT_TIME(), CURRENT_TIME	Synonyms for CURTIME()
CURRENT_TIMESTAP(), CURRENT_TIMESTAP	Synonyms for NOW()
CURTIME ()	Return the current time
DATE_ADD()	Add two dates
DATE_FORMAT()	Format date as specified
DATE_SUB()	Subtract two dates
DATE()	Extract the date part of a date or datetime expression
DATEDIFF ()	Subtract two dates
DAY ()	Synonym for DAYOFMONTH()
DAYNAME()	Return the name of the weekday
DAYOFMONTH()	Return the day of the month (1-31)
DAYOFWEEK()	Return the weekday index of the argument
DAYOFYEAR()	Return the day of the year (1-366)
EXTRACT	Extract part of a date
FROM_DAYS()	Convert a day number to a date
FROM_UNIXTIME()	Format date as a UNIX timestamp
HOUR()	Extract the hour
LAST_DAY	Return the last day of the month for the argument

LOCALTIME(), LOCALTIME NOW() LOCALTIMESTAMP, LOCALTIMESTAMP()	Synonym for Synonym for NOW()
MAKEDATE()	Create a date from the year and day of year
MAKETIME	MAKETIME()
MICROSECOND()	Return the microseconds from argument
MINUTE()	Return the minute from the argument
MONTH()	Return the month from the date passed
MONTHNAME()	Return the name of the month
NOW()	Return the current date and time
PERIOD_ADD()	Add a period to a year-month
PERIOD_DIFF()	Return the number of months between periods
QUARTER()	Return the quarter from a date argument
SEC_TO_TIME()	Converts seconds to 'HH:MM:SS' format
SECOND()	Return the second (0-59)
STR_TO_DATE()	Convert a string to a date
SUBDATE() DATE_SUB()	When invoked with three arguments a synonym for Subtract times
SUBTIME()	
SYSDATE()	Return the time at which the function executes
TIME_FORMAT()	Format as time
TIME_TO_SEC()	Return the argument converted to seconds
TIME()	Extract the time portion of the expression passed
TIMEDIFF()	Subtract time
TIMESTAMP()	With a single argument, this function returns the date or date time expression . With two arguments, the sum of the arguments
TIMESTAMPADD()	Add an interval to a datetime expression
TIMESTAMPDIFF()	Subtract an interval from a datetime expression
TO_DAYS()	Return the date argument converted to days
UNIX_TIMESTAMP()	Return a UNIX timestamp
UTC_DATE()	Return the current UTC date
UTC_TIME()	Return the current UTC time
UTC_TIMESTAMP()	Return the current UTC date and time
WEEK()	Return the week number
WEEKDAY()	Return the weekday index
WEEKOFYEAR()	Return the calendar week of the date (1-53)
YEAR()	Return the year
YEARWEEK()	Return the year and week

Examples of SQL Commands in Different Tables:

ITEMS table

This table stores information about all the items that are offered by company. The structure of the table is as follows:

Column	Datatype	Meaning
Itemno	Number(5)	A unique number assigned to each item.
ItemName	Varchar2(20)	Name of the item.
Rate	Number(8,2)	Rate of the item.
taxrate	Number(4,2)	Sales tax rate for this item.

The following are the constraints related to ITEMS table:

ITEMNO is primary key
RATE and TAXRATE must be >= 0
Default value for TAXRATE is 0

```
create table
ITEMS (
 itemno   number(5)   constraint items_pk  primary
 key, itemname varchar2(20),
 rate     number(8,2) constraint items_rate_chk check( rate >= 0),
 taxrate  number(4,2) default 0 constraint items_rate_chk check( rate >= 0)
);

insert into items values(1,'Samsung 14"
monitor',7000,10.5); insert into items values(2,'TVS Gold
Keyboard',1000,10); insert into items values(3,'Segate
HDD 20GB',6500,12.5); insert into items values(4,'PIII
processor',8000,8);
insert into items values(5,'Logitech Mouse',500,5);
insert into items values(6,'Creative MMK',4500,11.5);
```

CUSTOMERS Table

This table contains information about customers who have placed one or more orders. The following is the structure of the table.

Column	Datatype	Meaning
Custno	Number(5)	A unique number assigned to each customer.
CustName	Varchar2(20)	Complete name of the customer.
Address1	varchar2(50)	First line of address.
Address2	varchar2(50)	Second line of address.
City	varchar2(30)	Name of the city where customer lives.
state	varchar2(30)	Name of the state where customer lives.
PinCode	varchar2(10)	Pincode of the city.
Phone	varchar2(30)	One or more phone numbers separated using comma(,).

The following are the constraint related to CUSTOMERS table.

CUSTNO is primary key
CUSTNAME is not null column

```
create table
CUSTOMERS (
 custno    number(5)    constraint customers_pk  primary key,
 custname  varchar2(20) constraint customers_custname_nn not
 null, address1   varchar2(50),
 address2
 varchar2(50), city
 varchar2(30), state
 varchar2(30), pin
 varchar2(10), phone
 varchar2(30)
);
```

```
insert into customers values(101,'Raul','12-22-29','Dwarakanagar',
        'Vizag','AP','530016','453343,634333');
  insert into customers values(102,'Denilson','43-22-22','CBM Compound',
        'Vizag','AP','530012','744545');
  insert into customers values(103,'Mendiator','45-45-52','Abid Nagar',
        'Vizag','AP','530016','567434');
  insert into customers values(104,'Figo','33-34-56','Muralinagar',
        'Vizag','AP','530021','875655,876563,872222');
  insert into customers values(105,'Zidane','23-22-56','LB Colony',
        'Vizag','AP','530013','765533');
```

ORDERS Table

Contains information about all orders placed by customers. Contains one row for each order. The details of items ordered in an order will be found in LINEITEMS table. The following is the structure of the table.

Column	Datatype	Meaning
OrdNo	Number(5)	A unique number assigned to each order.
OrdDate	Date	Date on which order is placed.
ShipDate	Date	Date on which goods are to be shipped to customer.
Address1	varchar2(50)	First line of shipping address.
Address2	varchar2(50)	Second line of shipping address.
City	varchar2(30)	City name in shipping address.
state	varchar2(30)	State name in shipping address.
PinCode	varchar2(10)	Pincode of the city in shipping address.
Phone	varchar2(30)	One or more phone numbers separated using comma(,) of shipping place.

The following are the constraint related to ORDERS table.

ORDNO is primary key
CUSTNO is foreign key referencing CUSTNO of CUSTOMERS table. SHIPDATE must be >= ORDDATE.

```
create table
ORDERS (
 ordno     number(5)  constraint orders_pk  primary
 key, orddate   date,
 shipdate  date,
 custno    number(5) constraint orders_custno_pk references
 customers, address1   varchar2(50),
 address2
 varchar2(50), city
 varchar2(30), state
 varchar2(30), pin
 varchar2(10),phone
 varchar2(30),
 constraint order_dates_chk  check( orddate <= shipdate)
);

insert into orders values(1001,'15-May-2001','10-jun-2001',102,
            '43-22-22','CBM Compound','Vizag','AP','530012','744545');
insert into orders values(1002,'15-May-2001','5-jun-2001',101,
  '12-22-29','Dwarakanagar','Vizag','AP','530016','453343,634333');

insert into orders values(1003,'17-May-2001','7-jun-2001',101,
  '12-22-29','Dwarakanagar','Vizag','AP','530016','453343,634333');

insert into orders values(1004,'18-May-2001','17-jun-2001',103,
                    '45-45-52','Abid Nagar',
                'Vizag','AP','530016','567434');

insert into orders values(1005,'20-May-2001','3-jun-2001',104,
'33-34-56','Muralinagar','Vizag','AP','530021','875655,876563,872222');

insert into orders values(1006,'23-May-2001','11-jun-2001',104,
'54-22-12','MVP Colony','Vizag','AP','530024',null);
```

LINEITEMS Table

Contains details of items ordered in each order. For each item in each order this table contains one row. The following is the structure of the table.

Column	Datatype	Meaning
OrdNo	Number(5)	Refers to the order number of the order.
Itemno	Number(5)	Refers to the item number of the item.
qty	number(3)	Howmany units of this item arerequired in this order.
price	Number(8,2)	Selling price of the item for this order.
DisRate	Number(4,2)	Discount Rate for this item in this order.

The following are the constraint related to ORDERS table.

Primary key is ORDNO and ITEMNO.
ORDNO is a foreign key referencing ORDNO of ORDERS table. ITEMNO is a foreign key referencing ITEMNO of ITEMS table.

Default DISRATE is 0
QTY must be >= 1
DISRATE must be >= 0

```
create table
LINEITEMS (
 ordno   number(5)   constraint LINEITEMS_ORDNO_FK references ORDERS,
 itemno  number(5)   constraint LINEITEMS_itemno_FK references ITEMS,
 qty     number(3)   constraint LINEITEMS_qty_CHK CHECK( qty >= 1),
 price   number(8,2),
 disrate number(4,2) default 0
                     constraint LINEITEMS_DISRATE_CHK CHECK( disrate >= 0),
 constraint lineitems_pk primary key (ordno,itemno)
)
;
```

```
insert into lineitems values(1001,2,3,1000,10.0);
insert into lineitems values(1001,1,3,7000,15.0);
insert into lineitems values(1001,4,2,8000,10.0);
insert into lineitems values(1001,6,1,4500,10.0);

insert into lineitems values(1002,6,4,4500,20.0);
insert into lineitems values(1002,4,2,8000,15.0);
insert into lineitems values(1002,5,2,600,10.0);

insert into lineitems values(1003,5,10,500,0.0);
insert into lineitems values(1003,6,2,4750,5.0);

insert into lineitems values(1004,1,1,7000,10.0);
insert into lineitems values(1004,3,2,6500,10.0);
insert into lineitems values(1004,4,1,8000,20.0);

insert into lineitems values(1005,6,1,4600,10.0);
insert into lineitems values(1005,2,2,900,10.0);

insert into lineitems values(1006,2,10,950,20.0);
insert into lineitems values(1006,4,5,7800,10.0);
insert into lineitems values(1006,3,5,6600,15.0);
```