Piradon (Tien) Liengtiraphan
Database Systems
2016-02-01

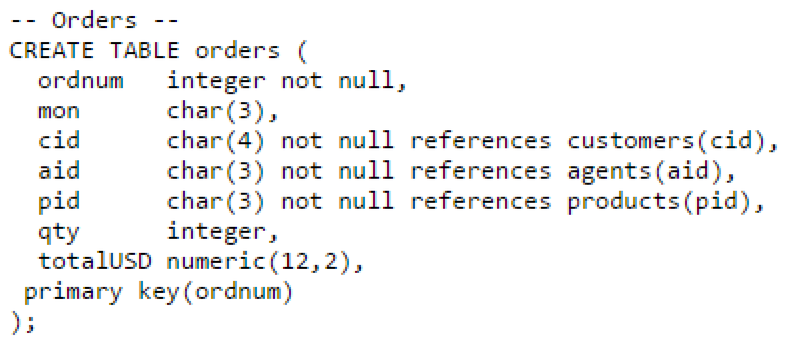
Definitions

Primary Key:	A singular column or combination of columns (though not recommended) serves to uniquely identify a record (row) in a specific table. While multiple Candidate Keys may exist, only one of they can be the Primary Key.
Candidate Key:	Is any column or combination of such that can be used to uniquely all the rows of a table in a database. Hence there can be multiple Candidate Keys in a singular table but only one Primary Key.
Super Key:	Simply put is a “non-minimal” Candidate Key. Meaning it has the potential to uniquely identify each row in a table but may also contain information which does not serve this purpose.

Data Types Short Essay

Data types exits in SQL Servers to serve as a way to minimize data entry error and ensure data consistency by limiting the type of value that the record of a particular field can hold. Some of the most common data types one might come across with are: Character Data, Date and Time Data, Integer Data, Binary String Data, Monetary Data, and etc. While extensive the list of Data Types that can be interpreted and inputted into a SQL Server are supplied by the Server itself, however with some Servers the user can define their own data types (Examples being Transact-SQL and Microsoft .NET Framework). The Data Type is assigned to the field during the creation of the table. (See example below)

For simplicity sake I will be using the example “Orders” table in our CAP3 database.



```
-- Orders --
CREATE TABLE orders (
  ordnum   integer not null,
  mon      char(3),
  cid      char(4) not null references customers(cid),
  aid      char(3) not null references agents(aid),
  pid      char(3) not null references products(pid),
  qty      integer,
  totalUSD numeric(12,2),
 primary key(ordnum)
);
```

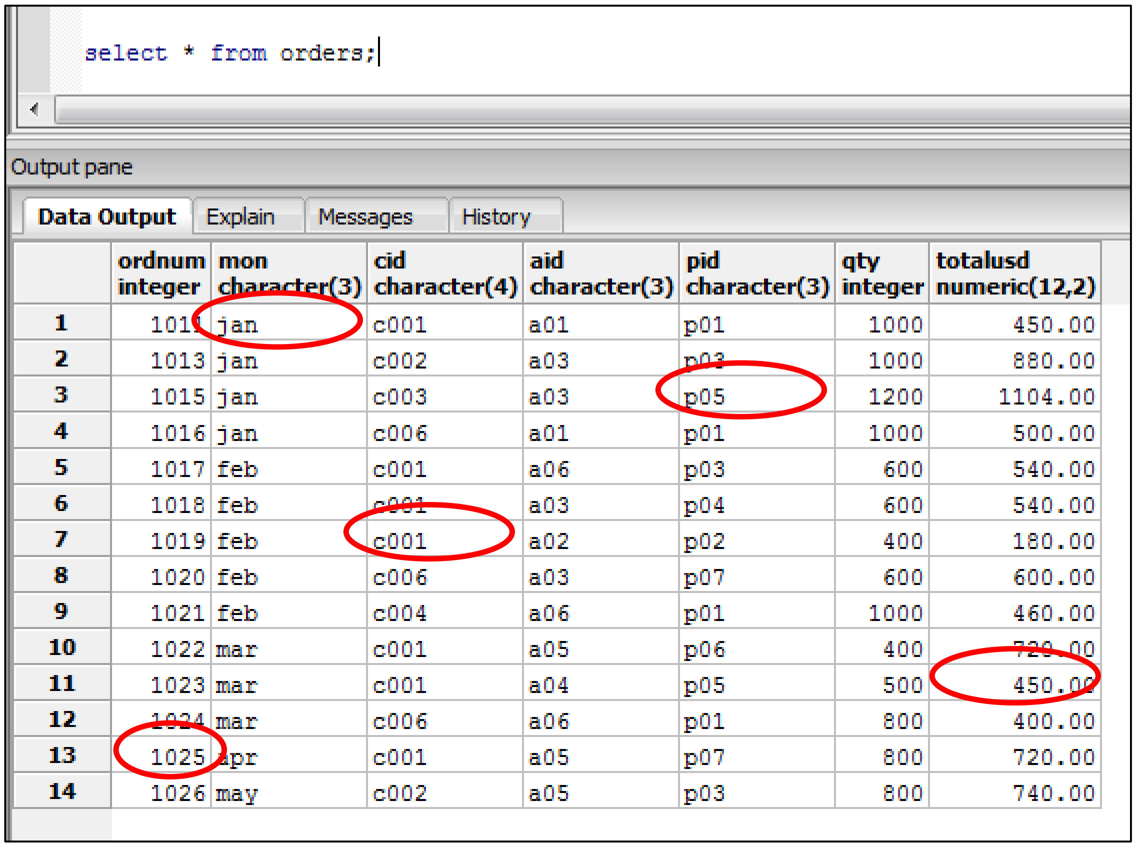
NN = Not Null

Orders							
Ordnum	Mon	CID	AID	PID	QTY	TotalUSD	Primary Key
Int, NN	Char(3)	Char(3), NN	Char(3), NN	Char(3), NN	Int	Numeric	Key

Rules of Relational Databases

1. First Normal Form
    a. Is the concept/rule that each attribute is atomic (i.e. indivisible). In other words this means that each attribute only contains a single value of data for that domain. In the cases of tables, each record only contains on value of data. This is important because it

helps reduces inconsistencies in the data. In tandem with Data Types First Normal Form directs the user to input only a specific type of data and keep it atomic in nature, meaning the chances of user's inputting data on a whim in whatever format they want is limited.



```
select * from orders;
```

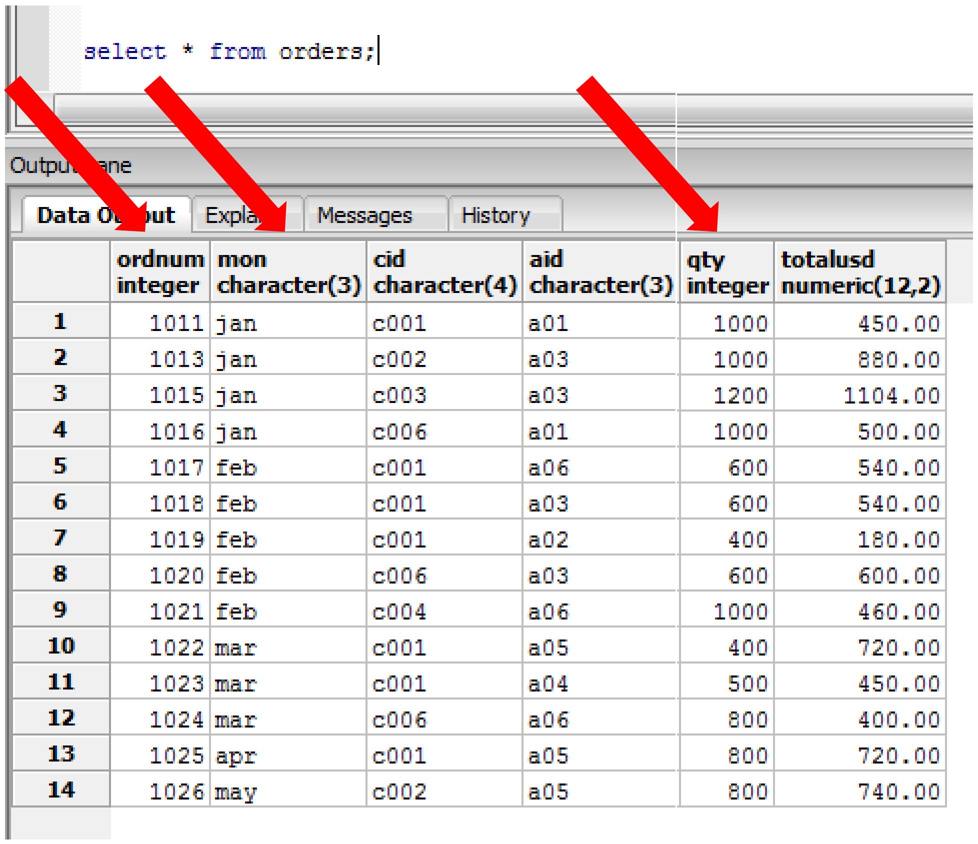
Output pane

Data Output | Explain | Messages | History

	ordnum integer	mon character(3)	cid character(4)	aid character(3)	pid character(3)	qty integer	totalusd numeric(12,2)
1	1011	jan	c001	a01	p01	1000	450.00
2	1013	jan	c002	a03	p03	1000	880.00
3	1015	jan	c003	a03	p05	1200	1104.00
4	1016	jan	c006	a01	p01	1000	500.00
5	1017	feb	c001	a06	p03	600	540.00
6	1018	feb	c001	a03	p04	600	540.00
7	1019	feb	c001	a02	p02	400	180.00
8	1020	feb	c006	a03	p07	600	600.00
9	1021	feb	c004	a06	p01	1000	460.00
10	1022	mar	c001	a05	p06	400	720.00
11	1023	mar	c001	a04	p05	500	450.00
12	1024	mar	c006	a06	p01	800	400.00
13	1025	apr	c001	a05	p07	800	720.00
14	1026	may	c002	a05	p03	800	740.00

b.

2. Access Rows By Content Only
    a. While the name implies exactly what the rule wants us to do it still bares the need for a little explanation. Accessing Content Rows, and by extension Columns, only by content allows for a more modular database. This means that the database is more flexible to change and modification. If in the backend of a program a row or column is accessed by its position, which is entirely possible, this setup might work so long as the format of the database does not change. But imagine if an extra row was added or a whole new column. The pointer to fetch the data does not adapt to that and will continue to point towards the same spot, where now a different piece of data resides.
    b. SELECT Column_name From user_tab_columns WHERE table_name = "My Table" and Column_id in (1,2,5)



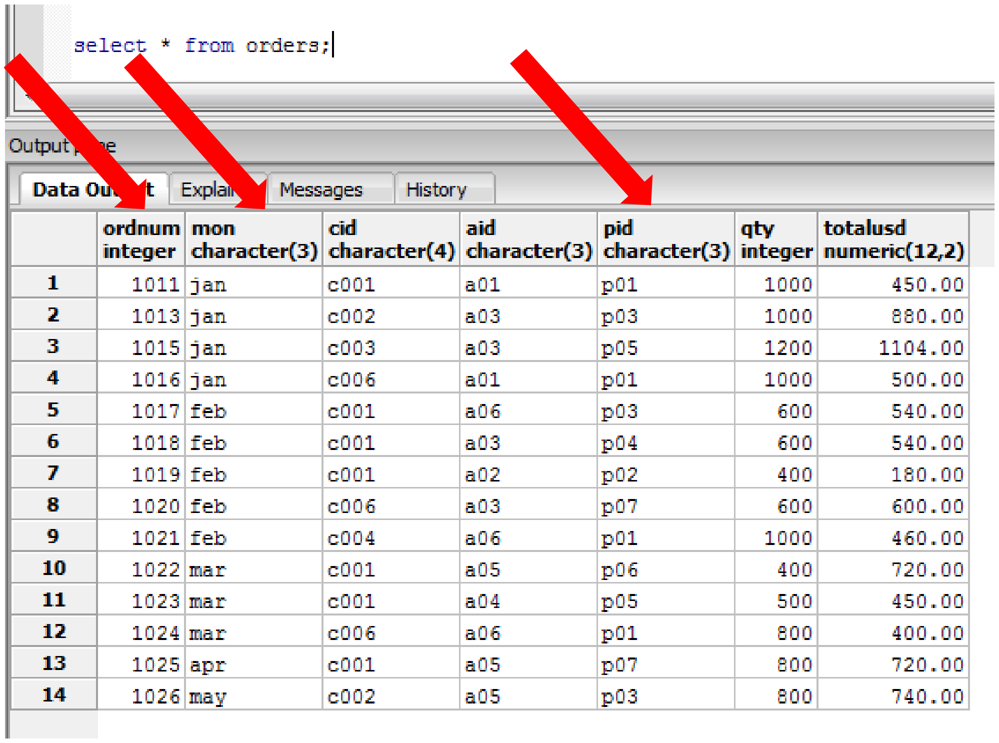
```
select * from orders;
```

Output pane

Data Output | Explain | Messages | History

	ordnum integer	mon character(3)	cid character(4)	aid character(3)	qty integer	totalusd numeric(12,2)
1	1011	jan	c001	a01	1000	450.00
2	1013	jan	c002	a03	1000	880.00
3	1015	jan	c003	a03	1200	1104.00
4	1016	jan	c006	a01	1000	500.00
5	1017	feb	c001	a06	600	540.00
6	1018	feb	c001	a03	600	540.00
7	1019	feb	c001	a02	400	180.00
8	1020	feb	c006	a03	600	600.00
9	1021	feb	c004	a06	1000	460.00
10	1022	mar	c001	a05	400	720.00
11	1023	mar	c001	a04	500	450.00
12	1024	mar	c006	a06	800	400.00
13	1025	apr	c001	a05	800	720.00
14	1026	may	c002	a05	800	740.00

i.
ii. Returns columns 1, 2, and 5
iii. But if the table is modified…



```
select * from orders;
```

Output pane

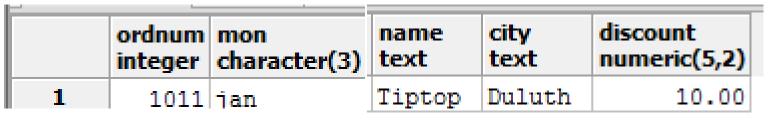
Data Output | Explain | Messages | History

	ordnum integer	mon character(3)	cid character(4)	aid character(3)	pid character(3)	qty integer	totalusd numeric(12,2)
1	1011	jan	c001	a01	p01	1000	450.00
2	1013	jan	c002	a03	p03	1000	880.00
3	1015	jan	c003	a03	p05	1200	1104.00
4	1016	jan	c006	a01	p01	1000	500.00
5	1017	feb	c001	a06	p03	600	540.00
6	1018	feb	c001	a03	p04	600	540.00
7	1019	feb	c001	a02	p02	400	180.00
8	1020	feb	c006	a03	p07	600	600.00
9	1021	feb	c004	a06	p01	1000	460.00
10	1022	mar	c001	a05	p06	400	720.00
11	1023	mar	c001	a04	p05	500	450.00
12	1024	mar	c006	a06	p01	800	400.00
13	1025	apr	c001	a05	p07	800	720.00
14	1026	may	c002	a05	p03	800	740.00

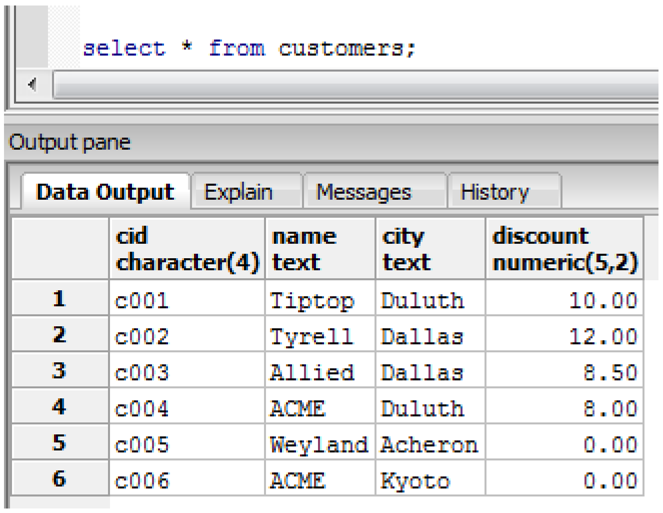
iv.

3. All Rows Must Be Unique

a. The anathema of good database design is data inconsistency, its close brother is data redundancy. All rows in a table must be unique to prevent this frugal waste of storage space. Also if data is entered multiple times then that increases the chance for data inconsistency, therefore by eliminating that possibility and enforcing this rule we can avoid the two major pitfalls of database design. To avoid this issue we can use foreign keys to instead of inputting data again.

b. 

	ordnum integer	mon character(3)	name text	city text	discount numeric(5,2)
1	1011	jan	Tiptop	Duluth	10.00

c. 

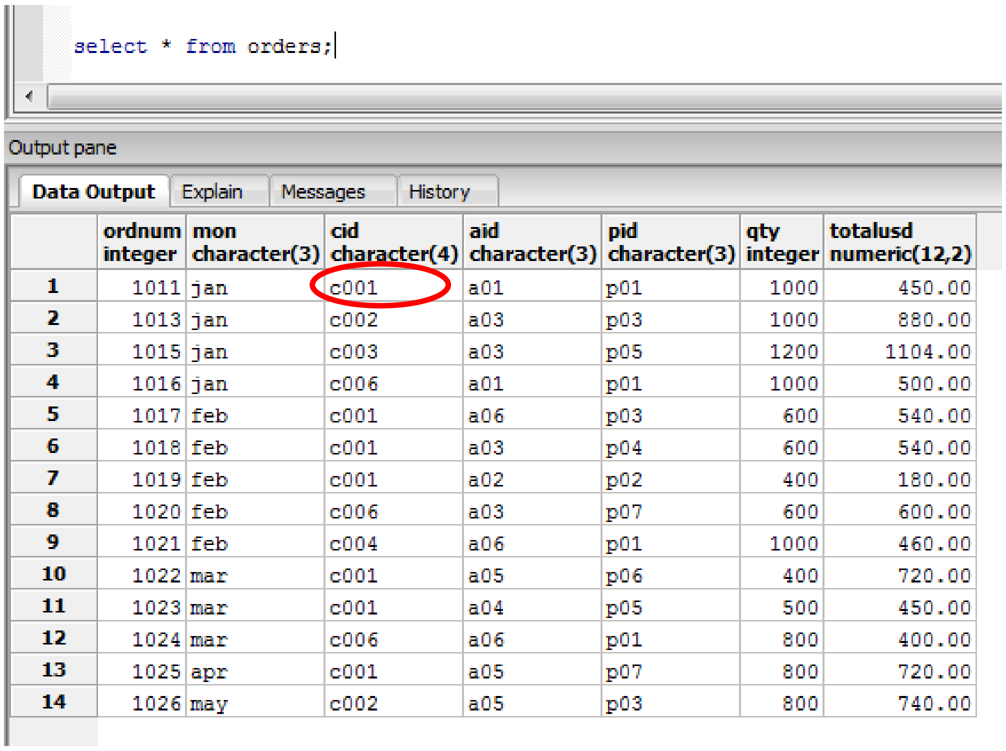
```
select * from customers;
```

Output pane

Data Output | Explain | Messages | History

	cid character(4)	name text	city text	discount numeric(5,2)
1	c001	Tiptop	Duluth	10.00
2	c002	Tyrell	Dallas	12.00
3	c003	Allied	Dallas	8.50
4	c004	ACME	Duluth	8.00
5	c005	Weyland	Acheron	0.00
6	c006	ACME	Kyoto	0.00

d. The data inside the first row already exists elsewhere in another table. To fix this we could simply use foreign keys.

e. 

```
select * from orders;
```

Output pane

Data Output | Explain | Messages | History

	ordnum integer	mon character(3)	cid character(4)	aid character(3)	pid character(3)	qty integer	totalusd numeric(12,2)
1	1011	jan	c001	a01	p01	1000	450.00
2	1013	jan	c002	a03	p03	1000	880.00
3	1015	jan	c003	a03	p05	1200	1104.00
4	1016	jan	c006	a01	p01	1000	500.00
5	1017	feb	c001	a06	p03	600	540.00
6	1018	feb	c001	a03	p04	600	540.00
7	1019	feb	c001	a02	p02	400	180.00
8	1020	feb	c006	a03	p07	600	600.00
9	1021	feb	c004	a06	p01	1000	460.00
10	1022	mar	c001	a05	p06	400	720.00
11	1023	mar	c001	a04	p05	500	450.00
12	1024	mar	c006	a06	p01	800	400.00
13	1025	apr	c001	a05	p07	800	720.00
14	1026	may	c002	a05	p03	800	740.00