

Contents

ခက်သလားဟေ့ ဒေတာဘေ့ခ်

ခက်သလားဟေ့ ဒေတာဘေ့ခ် အပိုင်း (၂)

ခက်သလားဟေ့ ဒေတာဘေ့ခ် (ပေါ့ပေါ့ပါးပါး SQL များ)

ခက်သလားဟေ့ ဒေတာဘေ့ခ် (ပေါ့ပေါ့ပါးပါး SQL များ) (၂)

MySQL မှ SQL အခြေခံများ (၁)

MySQL မှ SQL အခြေခံများ (၂)

MySQL မှ SQL အခြေခံများ (3)

MySQL မှ SQL များ

MySQL မှ SQL များ (၂)

MySQL မှ SQL များ (၃)

MySQL မှ SQL များ (၄)

Normalization and Common Database Design Patterns (0)

Normalization and Common Database Design Patterns (1)

ခက်သလားဟေ့ ဒေတာဘေ့စ်

Database အကြောင်း လုံးဝ မသိသူများအတွက် Database မိတ်ဆက်

1. မိတ်ဆက်
2. ဇယား၊ အတိုင် နှင့် အတန်းများ
3. တန်ဖိုးနှစ်ခု မထပ်သော အတိုင်
4. အမှတ်စဉ် အလိုအလျောက် တိုးပွားခြင်း
5. တန်ဖိုးများ စီခြင်း
6. ရှာဖွေခြင်း (သို့မဟုတ် ရွေးချယ်ခြင်း) နှင့် မာတိကာ အသုံးဝင်ပုံ
7. တန်ဖိုးများ ပြုပြင်ပြောင်းလဲခြင်း
8. တန်ဖိုးများ ဖျက်ခြင်း
9. ဤမှဆက်၍

(၁) မိတ်ဆက်

Database အကြောင်းကို ကွန်ပျူတာနှင့် အိုင်တီ လိုက်စားသူတိုင်း နားစွန်နားဖျား သော်လည်းကောင်း၊ မကြာခဏ သော်လည်းကောင်း ကြားဖူးအံ့။ ယနေ့အခါ၊ ကွန်ပျူတာတွင် အလွန် ရှုပ်ထွေးများပြားသော အချက်အလက်များကို သိုမှီး ထိန်းသိမ်းရာ၌ Database ကား၊ မရှိမဖြစ် လိုအပ်ချက်တစ်ခု ဖြစ်လျက် ရှိလေသည်။

ဤဆောင်းပါးတွင် ထို Database ဆိုသည့် အကြောင်းကို နားလည် လွယ်စေရန် ကြိုးစား ရေးသား ထားပါသည်။ ရည်ရွယ်ချက်ကား၊ Database ကို IT သမားများ၊ Software Engineer များ၊ Programmer များ၏ ကိရိယာတစ်ခု၊ ရှုပ်ထွေး ပွေလီသော နားလည်ရ ခက်သည့် အကြောင်းတစ်ခု ဟူသော အမြင်မှ၊ ကွန်ပျူတာ အခြေခံရှိသူတိုင်း၊ မိမိ၏ လုပ်ငန်းဆိုင်ရာ ကွန်ပျူတာ အသုံးပြုမှုများနှင့် တွဲဖက်ကာ လိုအပ်သလို အသုံးပြုနိုင်သည့် ကိရိယာ တစ်ခုအဖြစ် မြင်စေရန် ဖြစ်သည်။

Database ကိုသုံးခြင်းဖြင့်

1. များစွာသော အချက်အလက် (Data) များကို စနစ်ကျစွာ ထိန်းသိမ်းနိုင်သည်။
2. မိမိ ရှာလိုသော အချက်အလက်ကို တိကျ မြန်ဆန်စွာရှာဖွေနိုင်သည်။
3. လက်ရှိ အချက်အလက်များမှ တိုးချဲ့မှုများ (တိုးပွားမှုများ) ကို လျှင်မြန် ထိရောက်စွာ ဆောင်ရွက်နိုင်သည်။
4. အချက်အလက်၏ ပြောင်းလဲမှု အခြေအနေများ၊ တန်ဖိုးများကို ပြန်လည် ကြည့်ရှုနိုင်သည်။
5. အမှားအယွင်းနည်းသည်။
6. အချက်အလက်များ၏ ဆက်သွယ်ချက်များကို ဖော်ညွှန်းပြီး ထိုဆက်သွယ်ချက် မှတဆင့် သက်ဆိုင်ရာ အချက်အလက်များကိုပါ ရှာဖွေ နိုင်သေးသည်။
7. မှားယွင်းစွာ ပြောင်းလဲ ရေးသားမိသော အချက်အလက်များကို မူလ တန်ဖိုးသို့ ပြန်လည် ပြောင်းလဲနိုင်သည်။
8. ထိုထို များပြားသော အချက်အလက်များထဲ စိတ်ဝင်စားဖွယ် ဆက်သွယ်ချက်များ၊ အရေးပါသော အချက်အလက် ပြောင်းလဲမှု ပုံစံများကို ရှာဖွေ ဖော်ထုတ်ကာ အသုံးပြုနိုင်သည်။

Database ၏ အသုံးဝင်ပုံကား ထိုဖော်ပြပါ အချက် ၈ ချက်ထက် ပိုမိုများပြားသဖြင့် မဆုံးနိုင်သော ၎င်းတို့ကိုမရေးတော့ဘဲ လိုရင်းကို ဆက်ကြပါစို့။

ယခု ဆောင်းပါးတွင် နည်းပညာ ပိုင်းဆိုင်ရာ (SQL များ၊ Database System များ၊ Server များကို) မဆွေးနွေးဘဲ၊ Database အကြောင်းကို လုံးဝ မသိသေး သူများအတွက် ရည်ရွယ်ကာ သဘောတရားမျှကို မိတ်ဆက်မည် ဖြစ်ရာ စာဖတ်သူသည် Database အခြေခံ ရှိပြီးသူပင် ဖြစ်က ဆက်ဖတ်ရန် မလိုသည်ကို အကြံပြု လိုက်ပါသည်။

(၂) ဇယား၊ အတိုင်နှင့် အတန်းများ

Database ၏ အခြေခံကား ဇယားများ ဖြစ်သည်။ (အဲ.. ဟို.. ဇယားလေးတွေ ပြောတာ မဟုတ်ဘူးနော်။) Database System ဆိုသည်ကား အချက်အလက်များကို ဇယားများထဲတွင် သိမ်းဆည်းကာ၊ ဇယား တစ်ခုချင်းစီ အကြား ဆက်သွယ်ချက် များကို ဖန်တီးခြင်းဖြင့်၊ များစွာသော အချက်အလက် များကို စနစ်တကျ သိုမှီးခြင်းဖြစ်သည်။

Database ကား အချက်အလက်များကို ထိန်းသိမ်းရန်ဖြစ်ရာ၊ အချက်အလက် ထိန်းသိမ်းမှုအား အသုံးချမည့် ဥပမာ တစ်ခုဖြင့် စကြပါစို့။ ကျွန်ုပ်တို့တွင် ကောင်မလေးများ အမည်ကို သိမ်းဆည်းသော စာရင်းတစ်ခုရှိသည် ဆိုကြပါစို့။ (ကဲ အဂလီပ် နာမည်တွေပဲ သုံးကြမယ်) စာရင်းတွင် မိန်းကလေး ငါးယောက်မှာ Alice, Betty, Cindy, Dolly, Emmy တို့ဖြစ်ကြလေသည်။

မိန်းကလေးများကို စာရင်းသွင်းရန် ဇယားတစ်ခု ဖန်တီးကြမည်။ ဇယားဖန်တီးခြင်းကို Database အခေါ်အားဖြင့် Create လုပ်သည်ဟု ခေါ်လေသည်။ ဇယားအမည်ကို Girls ဟုပေးကြမည်။ ထို့ကြောင့် Girls ဟူသော ဇယားကို Create လုပ်မည်။

ကောင်မလေး ငါးယောက်တို့အမည်များကို Girls ဇယားတွင် စာရင်းသွင်းသော်၊

NAME
Alice
Betty
Cindy
Dolly
Emmy

ဟု စာရင်းရ၏။

ဤနည်းဖြင့် ဇယားတစ်ခုကို Create လုပ်၏။

ထို့နောက်တွင် ထိုလူစုထဲသို့ နောက်တစ်ယောက် ရောက်လာလေသည်။ ထိုသူ၏ အမည်သည်လည်း Emmy ပင်ဖြစ်လေသည်။ ထိုအခါ စာရင်းအသစ်သည်ကား။

NAME
Alice
Betty
Cindy
Dolly
Emmy
Emmy

ဟုဖြစ်လာလေသည်။ Emmy နှစ်ယောက်ဖြစ်၏။ ဤတွင် ပြဿနာစလေပြီ။

ယခုအခါ Emmy ဟု ဆိုလိုက်သည်နှင့် မည်သည့် Emmy ကိုဆိုလိုမှန်းမသိဖြစ်ရလေသည်။ ထိုအခါ Emmy နှစ်ယောက်ကို ခွဲခြားရန်လိုလာသည်။

နောက်တစ်မျိုး တွေးကြဦးစို့။

နိုင်ငံသားတိုင်းတွင် မှတ်ပုံတင် နံပါတ်ရှိသည်။ လူနှစ်ဦးတွင် မှတ်ပုံတင် နံပါတ် မတူနိုင်။ အင်းစိန်က ကိုသိန်းနှင့် သာဓကတက ကိုသိန်းတို့သည် နာမည်သာ တူသော်လည်း မှန်ပုံတင်အမှတ် မတူနိုင်ပေ။ ထို့ကြောင့် အမည်တူ သူများကို စာရင်းထဲတွင် လွယ်ကူစွာခွဲခြားနိုင်ရန် ထိုသို့ကွဲပြားသော နံပါတ်ပေးရန် လိုလာလေသည်။ ထို့ကြောင့်စာရင်းကို အောက်ပါအတိုင်း အမှတ်စဉ်ထည့်ကြမည်။

SERIAL	NAME
1	Alice
2	Betty
3	Cindy
4	Dolly
5	Emmy
6	Emmy

Figure 3 တွင် Emmy နှစ်ယောက်ကို ခွဲခြားနိုင်လေပြီ။ နာမည်ကား နှစ်ခု ထပ်နိုင်သည်။ အမှတ်စဉ်ကား နှစ်ခု မထပ်နိုင်။ အမည် ခွဲခြားရခက်သော နှစ်ခုထပ်သော အခါတွင်၊ အမှတ်စဉ်ကို ကြည့်ကာလူကို ခွဲခြားနိုင်၏။

ဤသို့ဖြင့် ယခုဆွဲသားသော ဇယားတွင် အတိုင် (column) ၂ ခု (SERIAL နှင့် NAME) နှင့် အတန်း (row) ၆ ခုတို့ရှိပေပြီ။ ထိုသို့ row နှင့် column များရှိသော ဇယားကို table ဟုခေါ်ကြပါစို့။ DataBase ၏ အသုံးအနှုန်းများနှင့် အသားကျစေရန် ယခုမှစ၍ Table, Row, Column ဟုသာ ခေါ်ဆို ရေးသားတော့မည်။

(၃) တန်ဖိုးနှစ်ခု မထပ်သော အတိုင်

Table တစ်ခုတွင် နှစ်ခုမထပ်နိုင်သော (နှစ်ခုမထပ်စေရဟု သတ်မှတ်ချက်) သတ္တိကို Uniqueness ဟုခေါ်ဆို၏။ ထိုသို့ နှစ်ခုမထပ်သော တန်ဖိုးများသာ ပါဝင်သည့် Column ကိုသုံးကာ Table ထဲတွင်ရှိသော Row များကို တိကျစွာ ဖော်ညွှန်းနိုင်သည်။ တစ်နည်း၊ ထို Column မှ တန်ဖိုးများသည် Row တစ်ခုခြင်းစီကို ကိုယ်စားပြုသည် ဟုလည်း ဆိုနိုင်သည်။ ထို့ကြောင့် ထို column သည် အဓိကကျသော column၊ တစ်နည်း key column ဟု ခေါ်သည်။

ကဲ ဇယားတွင် အမှတ်စဉ်နှင့် အမည်တော့ရပြီ။ မှတ်ပုံတင် နံပါတ်များ ထည့်ကြပါစို့။ မှတ်ပုံတင်အမှတ်ကို လွယ်ကူစေရန် ဤနည်းဖြင့် သတ်မှတ်မည်။

မှတ်ပုံတင်အမှတ် = အမည် + အမှတ်စဉ် + အမည်၏ ပထမ စာလုံး

ထို့နောက် NRIC ဟူသော column အသစ်ကို ထည့်သော အခါ အောက်ပါအတိုင်းရမည်။

SERIAL	NAME	NRIC
1	Alice	Alice1A
2	Betty	Betty2B
3	Cindy	Cindy3C
4	Dolly	Dolly4D
5	Emmy	Emmy5E
6	Emmy	Emmy6E

ထိုအခါ NRIC အတိုင်သည်လည်း နှစ်ခုမထပ်သော Uniqueness ဂုဏ်ရှိလေရာ ၎င်းကိုလည်း Key Column ဟုခေါ်ကမမှားပေ။ Table တွင် SERIAL နှင့် NRIC key column နှစ်ခုရှိလာလေပြီ။ ထို့ကြောင့် Key Column နှစ်ခုကို ကွဲပြားစေရန် SERIAL column အား မူလဘူတ key column (Primary key column) ဟုခေါ်တွင်စေကာ။ NRIC အား ဒုတိယ key column (Secondary key column) ဟုခေါ်တွင်စေအံ့။

SERIAL ကား ဤဇယားအတွက် Primary Key ဖြစ်ကာ NRIC ကား Secondary Key ဖြစ်လေသည်။

(၄) အမှတ်စဉ် အလိုအလျောက်တိုးပွားခြင်း

ဆက်ကြည့်စို့။

လက်ရှိ Table ထဲသို့ နောက်ထပ် လူ ၄ ယောက်နာမည်များ ထပ်ထည့်ကြမည်။

SERIAL	NAME	NRIC
1	Alice	Alice1A
2	Betty	Betty2B
3	Cindy	Cindy3C
4	Dolly	Dolly4D
5	Emmy	Emmy5E
6	Emmy	Emmy6E
7	Amy	Amy7A
8	Elizabeth	Elizabeth8E

9	Bibo	Bibo9B
10	Ann	Ann10A

Table ထဲသို့ နာမည်များ ထည့်သွင်းသော အခါ၊ အမှတ်စဉ်များလည်း တိုးပွားလာရပေမည်။

နာမည် အသစ်များ စာရင်းသွင်းပုံကို အနည်းငယ် စဉ်းစားကြည့်ကြပါစို့။ ဥပမာ Amy ကိုစာရင်းသွင်းသော အခါ၊ Alice နှင့် Betty ကြားတွင် နေရာပေးပါက၊ Amy အတွက် အမှတ်စဉ်ပေးရန် ခက်ပေသည်။ Amy အား အမှတ်စဉ် 2 ဟုပေးမည်ဆိုပါကလည်း၊ Betty မှစ၍ အောက်မှ Row တို့၏ အမှတ်စဉ်များ အားလုံးကို ပြောင်းလဲရပေမည်။

ထို့ကြောင့် အသစ်တိုးသော အမည်များအား၊ အမည်၏ အစီအစဉ်ကို ဥပက္ခာပြုကာ ဇယား အောက်ခြေတွင် နေရာပေးထည့်သွင်းခြင်းဖြင့် လွယ်ကူစွာပင် ဖြေရှင်းနိုင်သည်။

ဤနည်းဖြင့် နာမည် အသစ်လေးခုကို ထည့်သွင်းလေရာ၊ အမှတ်စဉ်များလည်း တိုးပွားလာရလေသည်။ အမှတ်စဉ်သည် အထက်တွင် ဖော်ပြခဲ့သလို၊ Unique ဖြစ်ရလေရကား၊ နောက်ဆုံးရှိသော နံပါတ်မှ တစ်တိုးကာ အလိုအလျောက် ထည့်သွင်းလိုက်ရုံနှင့် ကိစ္စပြီး၏ ။

ထိုသို့ အလိုအလျောက် တိုးပွားခြင်းကို Auto Increment ဟုခေါ်သည်။ DataBase ၏အလေ့အထအားဖြင့် Primary Key column များကို Auto Increment အဖြစ် သတ်မှတ်လေ့ရှိသည်။

(၅) တန်ဖိုးများ စီခြင်း

လူ ၁၀ ယောက်ပါသော Table ကားရပြီ။

Table တွင် ရှိသော Column တစ်ခုခြင်းကို ကြည့်ကြည့်စို့။ SERIAL column တွင် ရှိသော အမှတ်စဉ်များမှာ ကြီးစဉ် ငယ်လိုက် ဖြစ်ကြသော်လည်း၊ NAME column ရှိအမည်များ မှာ ABCD အစဉ်အလိုက် မဖြစ်ကြပေ။ ၎င်း တန်ဖိုးများကို စီကြအံ့။

တန်ဖိုးများကို စီရာတွင် နှစ်မျိုးနှစ်စား ရှိ၏ ။

- 1. ကြီးစဉ် ငယ်လိုက်နှင့်
- 2. ငယ်စဉ် ကြီးလိုက်

တို့ ဖြစ်ကြသည်။

တန်ဖိုးများ စီခြင်းကို DataBase အခေါ် sorting ဟုခေါ်သည်။ ကြီးစဉ်ငယ်လိုက် စီခြင်းမှာ sorting in descending order ဖြစ်၍ ငယ်စဉ် ကြီးလိုက် စီခြင်းမှာ sorting in ascending order ဖြစ်သည်။

Name Column ကို Ascending အလိုက် (ငယ်စဉ်ကြီးလိုက်) စီသော်။

SERIAL	NAME	NRIC
1	Alice	Alice1A
7	Amy	Amy7A
10	Ann	Ann10A
2	Betty	Betty2B
9	Bibo	Bibo9B
3	Cindy	Cindy3C
4	Dolly	Dolly4D
8	Elizabeth	Elizabeth8E
5	Emmy	Emmy5E
6	Emmy	Emmy6E

Name Column ကို Descending အလိုက် (ကြီးစဉ်ငယ်လိုက်) စီသော်။

SERIAL	NAME	NRIC
5	Emmy	Emmy5E
6	Emmy	Emmy6E
8	Elizabeth	Elizabeth8E
4	Dolly	Dolly4D
3	Cindy	Cindy3C
9	Bibo	Bibo9B
2	Betty	Betty2B
10	Ann	Ann10A
7	Amy	Amy7A
1	Alice	Alice1

ထိုသို့ တန်ဖိုးများ စီရာတွင် Column တစ်ခုခြင်းစီသာမက၊ Column များကို အတွဲလိုက်လည်း တန်ဖိုးစီနိုင်လေသည်။

ဥပမာ Name column ကို Ascending စီ၍၊ No Column ကို Descending စီသော်၊

SERIAL	NAME	NRIC
1	Alice	Alice1A
7	Amy	Amy7A
10	Ann	Ann10A
2	Betty	Betty2B
9	Bibo	Bibo9B
3	Cindy	Cindy3C
4	Dolly	Dolly4D
8	Elizabeth	Elizabeth8E
6	Emmy	Emmy6E
5	Emmy	Emmy5E

အထက်ပါ တန်ဖိုး စီတန်းမှု ရလဒ် (Figure 8) တွင်၊ ပထမ Row ခုကိုကြည့်ပါက NAME column စီရာတွင် Alice, Amy, Ann ဟု စီတန်းထားသည်ကို တွေ့ရမည်ဖြစ်ပြီး၊ SERIAL column တွင်ကား 1, 7, 10 ဟုတွေ့ရမည် ဖြစ်သည်။ အဘယ်ကြောင့် 10, 7, 1 ဟု မစီရသနည်း မေးအံ့။ ထိုသို့ စီတန်းရာတွင်၊ စီတန်းမှု အစီအစဉ်၌ ပထမဦးစွာ ပြဌာန်းသည့် column သည် ဦးစားပေး အဆင့် အမြင့်ဆုံးဖြစ်ပြီး နောက်ဆုံးတွင် ပြဌာန်းသည့် column သည် ဦးစားပေး အဆင့် အနိမ့်ဆုံး ဖြစ်လေသည်။

အောက်ဆုံး Row များဖြစ်သော၊ Emmy နှစ်ခုကိုကြည့်သော် အမှတ်စဉ်မှာ 6, 5 ဟူ၍ တွေ့ရမည်။ SERIAL coloum ကို decending စီရန် ပေးခဲ့သည်ကို အမှတ်ရလေ။ Alice, Amy နှင့် Ann တို့တွင် SERIAL မှာ ငယ်စဉ်ကြီးလိုက်ဖြစ်ပြီး၊ Emmy နှစ်ခုတွင်အမှတ်စဉ်မှာ ကြီးစဉ်ငယ်လိုက် ဖြစ်ရခြင်းမှာ၊ Emmy နှစ်ယောက်၏ NAME column တန်ဖိုး မှာ တူညီနေ ကြသောကြောင့်၊ ဒုတိယ ဦးစားပေးအဆင့် column အတိုင်း စဉ်ခြင်း ဖြစ်လေသည်။

ခက်သလားဟေ့ ဒေတာဘေ့စ် အပိုင်း (၂)

(၆) ရှာဖွေခြင်း (ရွေးချယ်ခြင်း) နှင့် မာတိကာ အသုံးဝင်ပုံ

ဇယားထဲတွင် လူဆယ်ယောက် ရှိနေလေပြီ။ ထို ၁၀ ယောက်ထဲမှ နာမည် တစ်ခုခုကို ရှာဖွေလိုသော် နာမည်များကို တစ်ခုခြင်း စစ်ဆေးကာ ရှာရမည် ဖြစ်သည်။ ထိုဇယားထဲမှ အမှတ်စဉ် တစ်ခုခုကို ရှာဖွေလိုသော် အမှတ်စဉ်များကို တစ်ခုခြင်း စစ်ဆေးကာ ရှာရမည် ဖြစ်သည်။ ရှာဖွေခြင်း သို့မဟုတ် ရွေးချယ်ခြင်း ကို DataBase အခေါ် Select လုပ်သည် မည်၏။

လူဆယ်ယောက် သာရှိစဉ် ရှာရသည်မှာ အပမ်းမကြီး သော်လည်း ၁၀၀၊ ၁၀၀၀၊ ၁၀၀၀၀ ခန့်ရှိလာသော အခါ လက်တွေ့တွင် အမည်တစ်ခု၊ အမှတ်စဉ် တစ်ခု ရှာရသည်မှာ ခက်ခဲပေမည်။

အမှတ်စဉ် တစ်ခုကို ရှာမည်ဆိုလျှင်၊ အမှတ်စဉ်များ (SERIAL Column) ကို ကြီးစဉ်ငယ်လိုက် သော်လည်းကောင်း၊ ငယ်စဉ်ကြီးလိုက်သော် လည်းကောင်း စီထားပါက (Figure 5 တွင်ရှု) အမှတ်စဉ် တစ်ခုရှာရန် လွယ်ကူသော်လည်း၊ (Figure 8) တွင်ကဲ့သို့ ကဘောက်တိ ကဘောက်ချာ ဖြစ်နေပါက ရှာရခက်ပေမည်။ ထိုနည်းတူ အမည်တစ်ခုရှာလိုသော် (ဥပမာ Amy) Figure 8 တွင်ကဲ့သို့ အမည်များကို စီထားပါက ရှာရ ပိုလွယ်မည်ပင်။

လက်တွေ့တွင် လူကိုယ်တိုင် ရှာပါက တစ်ခုခြင်းစီ တန်းစီကာ တိုက်စစ်ရသည့် နည်းတူ၊ ကွန်ပျူတာကို ရှာခိုင်းသောအခါ၊ ကွန်ပျူတာသည်လည်း တစ်ခုခြင်းကို တိုက်စစ် ရလေသည်။ ထို့ပြင် တန်ဖိုး တစ်ခုကို ရှာရာ၌ (ဥပမာ Emmy) ရှာသော တန်ဖိုးအား တွေ့ရှိသော အခါ ရပ်တန့် လိုက်၍ မရပေ၊ ထိုတန်ဖိုးသည် တစ်ခုထက် ပို၍ ရှိနိုင်လေရာ ၎င်းတို့အားလုံးကို တွေ့ရှိစေရန်၊ ရှိသမျှ အတန်းအားလုံးကို ရှာရပေသည်။

ထို့ကြောင့် ဇယားတစ်ခုတွင် ရှာဖွေမှု ကြာမြင့်ချိန်မှာ၊ ၎င်းတွင်ပါသော အတန်း အရေအတွက်နှင့် တိုက်ရိုက် အချိုးကျသည့်ပြင်၊ မိမိတန်ဖိုး ရှာသော အတိုင်တွင် ရှိသော Data များကို ကြိုတင်ကာ စီတန်းထားပုံ ပေါ်တွင် များစွာ မူတည်လေသည်။

မကြာခဏ တန်ဖိုးရှာဖွေမည့် အတိုင် (column) များကို ကြိုတင် သိရှာပါက၊ ရှာဖွေမှုအချိန် တိုတောင်းစေရန် ၎င်းတို့ကို ကြိုတင် စီတန်း ထားနိုင်လေသည်။ ထိုသို့ပြင်ဆင်ခြင်းကို indexing လုပ်သည်ဟု ခေါ်သည်။ အလွယ်မှတ်ရန်မှာ မာတိကာ ထုတ်ထားသည်ဟု မှတ်လေ။

ယနေ့ခေတ် DataBase Server များသည် သန်းပေါင်း များစွာသော row များပါဝင်သည့် table များကို ကောင်းမွန်စွာ ထိန်းသိမ်းနိုင်သည့်ပြင်၊ indexing လုပ်ခြင်းကိုလည်း ခွင့်ပြုလေသည်။ Data တစ်ခုကိုရှာဖွေလျှင် မာတိကာ ရှိပါက DataBase Server သည်လျင်မြန်စွာ ရှာဖွေ ပေးနိုင်သည်။ သို့မဟုတ်ပါက row တစ်ခုခြင်းကို တိုက်စစ် နေရပေမည်။ အချိန်ကုန် လှပေ၏။ ဤကား index ထားခြင်း အကျိုးတည်း။

(ဤနေရာတွင် ဆရာ ဆရာများက index လုပ်ခြင်းဟူသည် sorting ချည်းသာ မဟုတ်ဟု ငြင်းချက် ထုတ်ကြပေမည်။ မှန်ပေ၏။ ယခု ဆောင်းပါးတွင် index အကြောင်း အသေးစိတ် မဖော်ပြလေ။)

(၇) တန်ဖိုးများ ပြုပြင်ပြောင်းလဲခြင်း

ဤသို့ အချက်အလက်များကို ဇယားများထဲတွင် စနစ်တကျ ထည့်သွင်းခြင်းမှာ၊ လွယ်ကူစွာ ပြုပြင်ပြောင်းလဲနိုင်ရန် သည်လည်း အကြောင်း တစ်ခု ဖြစ်လေရာ။ ဇယားများထဲမှ တန်ဖိုးများ ပြုပြင် ပြောင်းလဲပုံ ကိုလည်း သိရန် လိုအပ်ပေသည်။

တန်ဖိုးများ ပြုပြင် ပြောင်းလဲခြင်းကို DataBase အခေါ် update လုပ်သည်ဟု ခေါ်သည်။

Row တစ်ခုထဲမှာ တန်ဖိုးတစ်ခုကို ပြောင်းကြမည် (ဥပမာ အမှတ်စဉ် ၆ မှ Emmy ကို Elle သို့ပြောင်းမည်) ဆိုကြပါစို့။ ဇယားထဲတွင် Emmy နှစ်ယောက် ရှိလေရာ ၎င်းနှစ်ယောက်ကို ကွဲပြားစေသော တန်ဖိုးတစ်ခုကို ကြည့်ကာ ခွဲခြားပြီးမှ မိမိဆိုလိုသည့် Emmy ကိုပြောင်းနိုင်ပေမည်။

“Emmy ကို Elle သို့ပြောင်း” ဟုဆိုလိုက်လျှင် ဇယားထဲတွင် ရှိသမျှ Emmy နှစ်ယောက်လုံးသည် Elle ဖြစ်သွား ပေလိမ့်မည်။ ကျွန်ုပ်တို့ ဆောင်ရွက် လိုသည်မှာ အမှတ်စဉ် ၆ မှ Emmy ဖြစ်လေရာ “SERIAL တန်ဖိုး 6 မှ Emmy ကို Elle သို့ပြောင်း” ဟုဆိုပါမှ မှန်ပေမည်။ ရွေးချယ်မှု ဆိုင်ရာ ဖော်ပြချက် “SERIAL တန်ဖိုး 6” ဟု ဖော်ပြချက်သည် Emmy နှစ်ခုကို ကွဲပြားစေလေသည်။

ဤနေရာတွင် အဓိက မှတ်စေလိုသည်ကား၊ တန်ဖိုးများ ပြောင်းလဲလိုလျှင် မိမိဆောင်ရွက်လိုသည့် တန်ဖိုးကို တိကျစွာ ရွေးချယ်ဖော်ပြချက် လိုအပ်ခြင်းပင် ဖြစ်သည်။

(၈) တန်ဖိုးများ ဖျက်ခြင်း

ဇယားထဲတွင် မိမိသိမ်းဆည်းလိုသော အချက်အလက်များကို သိမ်းဆည်းသည် သာမက၊ မလိုအပ်သော အရာများကိုလည်း ဖျက်သိမ်းရန် လိုအပ် လာလေ့ ရှိသည်။ ဖျက်သိမ်းခြင်းကို Delete လုပ်သည် ဟု DataBase တွင် ခေါ်သည်။

ဥပမာ “အမည် Bibo ပါသော row ကိုဖျက်” ဟုဆိုပါက အမှတ်စဉ် ၉ မှာ Bibo ပျောက်သွား ပေလိမ့်မည်။ သို့သော် အမှတ်စဉ် 6 မှ Emmy ကို ဖျက်လိုပါက။ “အမည် Emmy ပါသော row ကိုဖျက်” ဟုသာ ဆိုလိုက်လျှင် အမှတ်စဉ် 5 တွင်ရှိသော Emmy ပါ ပျက်သွား ပေလိမ့်မည်။ ထို့ကြောင့် “SERIAL တန်ဖိုး 6 မှ Emmy ကိုဖျက်” ဟုဆိုမှ မှန်ပေမည်။

ရွေးချယ်မှုဆိုင်ရာ ဖော်ပြချက် “SERIAL တန်ဖိုး 6” ဟုဖော်ပြချက်သည် မိမိဖျက်လိုသည့် Row ရည်ညွှန်း စေလေသည်။

ထို့နည်းတူ row တစ်ခုတည်းကို မဟုတ်ဘဲ၊ တစ်ခုထက်ပိုသော Row များကို ဖျက်လိုပါက ဥပမာ Emmy အမည်ရှိသော Row အားလုံးကို ဖျက်လိုပါက၊ “အမည် Emmy ပါသော row ကိုဖျက်” ဟုဆိုလျှင် မှန်၏။ Emmy အမည်ရှိသော row အားလုံး ပျက်ချေမည်။

တန်ဖိုးများ ဖျက်ရာတွင်လည်း၊ ၇ တွင် ဖော်ပြခဲ့သော တန်ဖိုးများ ပြုပြင်ပြောင်းလဲသည့် နည်းတူ မိမိ ဆောင်ရွက်လိုသည့် တန်ဖိုးကို တိကျစွာ ရွေးချယ် ဖော်ပြချက် လိုအပ်သည်ကို မှတ်သားရာသည်။

(၉) ဤမှ ဆက်၍

ဤဆောင်းပါးတွင် DataBase ၏ သဘောကို အခြေခံကျသော ပုံစံဖြင့် ရှင်းလင်း တင်ပြကာ၊ DataBase နှင့်သက်ဆိုင်ရာ အသုံးအနှုံးများ ဖြစ်သည့် Table, Row, Column, Primary Key, Secondary Key, Select, Insert, Update, Delete တို့၏ အဓိပ္ပါယ်နှင့် ၎င်းတို့၏ ဆောင်ရွက် ချက်များကို အကျမ်းမျှ ဖော်ပြခဲ့ချေပြီ။

ဤမှဆက်၍ DataBase ၏အသက်ဖြစ်သော SQL (Structured Query Language) အကြောင်းကို ဆက်လက် လေ့လာသင့်သည်။ SQL သည်ကား စံအဖြစ် သတ်မှတ် ထားသည်ဖြစ်၍၊ မည်သည့် DataBase system နှင့်မဆိုတွဲဖက် အသုံးချနိုင်ရာ၊ SQL တတ်လျှင် မည်သည့် DataBase system ကိုမဆို အခြေခံအားဖြင့် အသုံးပြုနိုင် လေသည်။

ခက်သလားဟေ့ ဒေတာဘေ့စ် (ပေါ့ပေါ့ပါးပါး SQL များ)

1. အမှာ
2. SQL မရေးခင်
3. Table တစ်ခု တည်ဆောက်ခြင်း
4. Row တစ်ခု ထည့်သွင်းခြင်း
5. Data ရှာဖွေခြင်း (ရွေးချယ်ခြင်း)
6. Row ထဲမှ Data ကို ပြုပြင်ခြင်း
7. Row ကို ဖျက်သိမ်းခြင်း
8. Table ကို ဖျက်သိမ်းခြင်း
9. ဤမှ ဆက်၍

1. အမှာ

Database ၏ အခြေခံ ဆောင်ရွက်မှုများနှင့် ၎င်းနှင့် ဆိုင်ရာ ဝေါဟာရ အချို့ကို၊ အပိုင်း ၁ တွင် မိတ်ဆက် ပေးခဲ့ပြီ။ ဇယား တည်ဆောက်မှု ဥပမာ များကို သုံးကာ ရှင်းလင်း ဖော်ပြခဲ့ချေပြီ။

ထို ဆောင်ရွက်ချက် များကို Database System တစ်ခုအား ခိုင်းစေသော အခါတွင်၊ Database နားလည်သည့် စကားနှင့် ခိုင်းစေမှသာ အလုပ် ဖြစ်ပေမည်။ Database ကား ကွန်ပျူတာ ဖြစ်လေရာ၊ လူနှင့် စက်တို့အကြား Database ဆိုင်ရာ လုပ်ငန်း ဆောင်တာများ ဆောင်ရွက်နိုင် စေရန် စံအဖြစ် သုံးသော ဘာသာစကား သည် SQL (Structured Query Language) ပင် ဖြစ်လေသည်။

ယခု အပိုင်းတွင် SQL ပေါ်ပေါက်လာပုံများ၊ ခက်ခဲနက်နဲသော SQL သဘောတရားများကို မဆွေးနွေးဘဲ၊ ပေါ့ပေါ့ပါးပါး SQL စာကြောင်းများ အကြောင်းကို ဆွေးနွေး တင်ပြပါမည်။ ရည်ရွယ်ချက်မှာ SQL ဘာသာစကား အခြေခံကို သင်ပုန်းကြီးအဆင့် သိရှိစေရန် နှင့် SQL ကို ထမင်းစား ရေသောက်ခန့် တတ်မြောက်ပြီးမှ၊ ရှုပ်ထွေးသော သီဝရီများ နှင့် SQL statement များရေးသည့် အဆင့်ကို လွယ်ကူစွာ တတ်လှမ်း နိုင်စေရန်လည်း ဖြစ်ပါသည်။

2. SQL မရေးခင်

SQL မရေးခင် ဆိုင်၏ မဆိုင်၏ မသိ၊ မှတ် သင့်သော အကြောင်းအရာ တစ်ချို့ကို ဖော်ပြလိုပါသည်။

2.1 Keywords

SQL သည် Database နှင့် လူတို့ အကြား ဆက်သွယ်ပေးသည့် ဘာသာစကား ဖြစ်လေရာ၊ Database ကို ခိုင်းစေရာတွင် (အသုံးချရာတွင်) Database နားလည်သော စကား (SQL) ကို သုံးကာ စေခိုင်း ရလေသည်။ ထို Database နားလည်သော စာလုံးများကို SQL language တွင် အထူး သတ်မှတ်ထား လေသည်။ ၎င်းတို့ကို keywords ဟု ခေါ်လေ၏။ အသုံးချသူ အနေဖြင့် ထို အထူးစာလုံးများ (keyword) များကို မှတ်သား ရာသည်။

2.2 ကွင်းစ ကွင်းပိတ်

SQL ရေးသား ရာတွင် ကွင်းစ ကွင်းပိတ်များကို အသုံးပြုရလေ့ ရှိသည်။ သတိပြုရန်မှာ ကွင်းစ တစ်ခုကို ရေးလျှင် နောက်တွင် ၎င်းနှင့် တွဲမည့် ကွင်းပိတ် တစ်ခု လိုသည်။ မရှိပါက ၎င်း SQL ကို Database က နားမလည်ပေ။ ကွင်းပိတ်သာရှိပြီး ကွင်းစ မရှိလျှင်လည်း ထို့နည်း တူစွာပင်။ အလေ့အကျင့် အားဖြင့် ကွင်းစ ရေးပြီးတိုင်း ကွင်းပိတ် ဆက်တိုက် ရေးခြင်းဖြင့် အမှားအယွင်း နည်းရာသည်။

2.3 Quotation Marks

ကွင်းစ ကွင်းပိတ် နည်းတူ Quotation mark များသည်လည်း အရေးကြီး လေသည်။ Quotation Mark တစ်ခုကို ဖွင့်လျှင် Quotation Mark နှင့် ပိတ်ရ လေသည်။ Double Quote နှင့် Single Quote တို့တွင် Single Quote သည် အထူး အဓိပ္ပါယ် ရှိလေသည်။

SQL သည် keyword များနှင့် ဖော်ပြသည့် ဖြစ်လေရာ၊ SQL ထဲတွင် စာကြောင်းများကို ဖော်ပြသော အခါ Single quote ကိုသုံးရလေသည်။ (နောက်ပိုင်းတွင် အသေးစိတ် ဖော်ပြပါမည်။)

စာကြောင်းများထဲတွင် single quote ပါလျှင် ဘယ်လို လုပ်မည်နည်းဟု စောဒက တက်ပါက၊ single quote ကိုနှစ်ခု ဆက်တိုက် ရေးခြင်းဖြင့် Database ကို နားလည် စေနိုင်ကြောင်း မှတ်ရာ၏။

2.4 SQL Command ကို အဆုံးသတ်လျှင်

အခြား ဘာသာစကား များနည်းတူ SQL တွင်လည်း command တစ်ကြောင်းကို အဆုံးသတ် ရာတွင် ပုဒ်မ ချရလေသည်။ SQL ၏ ပုဒ်မ အဖြစ်သုံးသော စာလုံးကား semi column “;” ဖြစ်လေသည်။ အဆင့်မြင့်သော Database system များသည် ပုဒ်မ မချသော်လည်း SQL command ကို နားလည် နိုင်လေသည်။ သို့သော် မူအားဖြင့်၎င်း၊ တစ်ခုထက် ပိုသော SQL command များကို တစ်ပြိုင်နက် ရေးသား အသုံးချလျှင် သော်လည်းကောင်း၊ SQL command တစ်ကြောင်း ဆုံးတိုင်း ပုဒ်မ ချရမည် ဖြစ်သည်။

3. Table တစ်ခု တည်ဆောက်ခြင်း

(မှတ်ချက်။ ။ ဤနေရာမှစကာ Database System အား DBS ဟု အတိုကောက် ရေးသားပါမည်။)

ဇယားတစ်ခု တည်ဆောက်ရန် DBS အား စေခိုင်းသောအခါ၊ ဖန်တီးလိုသည့် ဇယားအမည်အား ပေးရမည် ဖြစ်သည်။ Girls ဟုမည်သော Table ကို တည်ဆောက် စေလိုသော် “Girls ဇယားကို ဖန်တီးစေ” ဟုဆိုရသည်။ ဘုံလို “create a table called girls.” ဟုရေးရသည်။ SQL အားဖြင့်ကား “CREATE TABLE Girls” ဟု DBS အား အမိန့်ပေးရသည်။

ဤနေရာတွင် CREATE နှင့် TABLE တို့မှာ Keywords များဖြစ်ကြသည်။ ကဲ....ဇယား အမည်တော့ ဖော်ပြပြီ၊ သို့သော် ဇယား၏ အင်္ဂါ မပြည့်စုံသေး။ ဇယားတွင် ထည့်သွင်း သိမ်းဆည်းလိုသည့် အချက်အလက် များအတွက် အတိုင် (column) များကိုပါ DBS အား ထည့်သွင်း ဖော်ပြပေးရမည် ဖြစ်သည်။

Girls table တွင် အမှတ်စဉ် (No) ၊ အမည် (Name) နှင့် မှတ်ပုံတင် (NRIC) အတိုင် များကို ထည့်သွင်းကြမည်။ ထိုသို့ ထည့်သွင်း ဖော်ပြရန် DBS အား command ပေးရာတွင် အောက်ပါအတိုင်း ရေးသားနိုင်သည်။ ကွင်းစ ကွင်းပိတ်နှင့် အဆုံးသတ် ; အား သတိပြုလေ။

CREATE TABLE Girls (No, Name, Nric);

ထိုသို့ ဖော်ပြသော်လည်း မပြည့်စုံသေး။ DBS သည် ကွန်ပျူတာ ဖြစ်လေရာ၊ No နှင့် Name တို့ ကွဲပြားပုံကို မသိချေ။ ထို့ကြောင့် Data အမျိုးအစားအား ခွဲခြားပေး ရလေသည်။ No သည် အမှတ်စဉ် ဖြစ်လေရာ ကိန်းဂဏန်း အမျိုးအစား ဖြစ်လေသည်။ Name နှင့် NRIC တို့မှာကား စာအမျိုးအစား ဖြစ်လေသည်။ ကိန်းဂဏန်း အမျိုးအစားမှာ numeric ဖြစ်၍၊ စာလုံးများမှာ ကား text ဖြစ်လေသည်။ ၎င်းတို့အား Create table command တွင် အောက်ပါအတိုင်း ထည့်သွင်း ခွဲခြားပေးနိုင်သည်။

CREATE TABLE Girls (No Numeric, Name text, Nric text);

မလွယ်သလော။ ထိုဖော်ပြချက် မြင်သောအခါ DBS သည် တိုင် သုံးခု ပါသော Girls အမည်ရှိ ဇယားကို ဖန်တီး ပေးလေသည်။ ထိုသို့ Table တည်ဆောက်ခြင်း ဖော်ပြချက်ကို Database အခေါ်အားဖြင့် Create Statement (သို့) Create command ဟုခေါ်လေသည်။

4. Row တစ်ခု ထည့်သွင်းခြင်း

ဇယား ဖန်တီးပြီး သောအခါ ၎င်းထဲသို့ အချက်အလက်များ ထည့်သွင်း ကြမည်။ အချက်အလက် ထည့်သွင်းခြင်းကို insert လုပ်သည်ဟု ခေါ်သည်။ insert လုပ်ရန် DBS အား မည်သည့် ဇယားတွင် insert လုပ်ရမည်ဟု ဇယားအမည်ကို တိတိပပ ပေးရသည်။ ထို့နောက် insert လုပ်ချင်သည့် အချက်အလက်များ၊ Data များ နှင့် အတိုင်များကို ဖော်ပြပေးရသည်။

အချုပ်အားဖြင့်

1. ဇယားအမည်
2. အချက်အလက် ထည့်သွင်းလိုသည့် အတိုင်များ
3. အချက်အလက်များ

ကို Insert Command တစ်နည်း Insert Statement တွင် ဖော်ပြပေးရသည်။

ဆိုကြပါစို့၊ Database ထဲသို့ အမှတ်စဉ် ၁၊ အမည် Alice၊ မှတ်ပုံတင် Alice1A ကို ထည့်သွင်းလိုသော် ...

INSERT INTO Girls (No, Name, Nric)

VALUES (1, ‘Alica’, ‘Alice1A’);

ဟု ရေးသား ဖော်ပြ ပေးရသည်။

ဤ Insert statement တွင် INSERT, INTO နှင့် VALUES တို့မှာ keywords များဖြစ်ပြီး၊ Table အမည် Girls နောက်တွင် အချက်အလက် ထည့်သွင်း လိုသည့် အတိုင်(column)များ အမည်ကို၊ ကွင်းစ ကွင်းပိတ် ထဲတွင် အစဉ်အလိုက် ရေးသား ဖော်ပြပေး ရသည်။ ထို့နောက် တန်ဖိုး များကို၊ VALUES keyword နောက်တွင် ကွင်းစ ကွင်းပိတ် ထဲ၌ ရှေ့တွင် ဖော်ပြခဲ့သော အတိုင်များ၏ အစဉ်အလိုက် အတိုင်း ဖော်ပြပေးရသည်။

အစဉ်အလိုက် ဖြစ်ရန် အရေးကြီး လေသည်။ အစီအစဉ် မှားယွင်းပါက DBS နားမလည် ဖြစ်ကာ Error ပြလေမည်။

တန်ဖိုးများ ဖော်ပြရာတွင် 1 မှာ ကိန်းဂဏန်း ဖြစ်သောကြောင့် single quote မလိုချေ။ Name နှင့် Nric တို့မှာ စာလုံးများ ဖြစ်သောကြောင့် single quote ထဲတွင် ရေးသား ရလေသည်။ Insert Statement ကိုအထက်တွင် နှစ်ကြောင်း ခွဲရေးထား သော်လည်း အဆုံးသတ်တွင် ; ပါသည်ကို သတိပြုပါ။ DBS သည် ၎င်း ; ကို တွေ့မှသာ command တစ်ခု ပြီးဆုံးသည်ဟု မှတ်ယူလေသည်။

ထပ်မံ၍ အတန်း ၉ ခု insert လုပ်သော်။ အောက်ပါအတိုင်း insert statement များကို DBS သို့ တစ်ပြိုင်နက်တည်း ရေးသား ပေးနိုင်လေသည်။ တစ်ခုထက် ပိုသော insert statement များကို ဖော်ပြသောအခါ statement တစ်ခုဆုံးတိုင်း ; ထည့်ရသည်ကို သတိပြုပါ။

```
INSERT INTO Girls (No,Name,Nric)
VALUES(2,' Betty','Betty2B ');
INSERT INTO Girls (No,Name,Nric)
VALUES(3,' Cindy','Cindy3C ');
INSERT INTO Girls (No,Name,Nric)
VALUES(4,' Dolly','Dolly4D ');
INSERT INTO Girls (No,Name,Nric)
VALUES(5,' Emmy','Emmy5E ');
INSERT INTO Girls (No,Name,Nric)
VALUES(6,' Emmy','Emmy6E ');
INSERT INTO Girls (No,Name,Nric)
VALUES(7,' Amy','Amy7A ');
INSERT INTO Girls (No,Name,Nric)
VALUES(8,' Elizabeth','Elizabeth8E ');
INSERT INTO Girls (No,Name,Nric)
VALUES(9,' Bibo','Bibo9B ');
INSERT INTO Girls (No,Name,Nric)
VALUES(10,' Ann','Ann10A ');
```

5. Data ရှာဖွေခြင်း (ရွေးချယ်ခြင်း)

Database ထဲသို့ အတန်း ၁၀ခု သွင်းသောအခါ၊ အောက်ပါအတိုင်း Data များ ရှိနေပေပြီ။

Table **Girls**

NO	NAME	NRIC
1	Alice	Alice1A
2	Betty	Betty2B
3	Cindy	Cindy3C
4	Dolly	Dolly4D
5	Emmy	Emmy5E
6	Emmy	Emmy6E
7	Amy	Amy7A
8	Elizabeth	Elizabeth8E
9	Bibo	Bibo9B
10	Ann	Ann10A

Figure 1

Girls Table ထဲမှ Data များကို DBS အား ဖော်ပြပေးစေ လိုသော်။ DBS အား မည်သည့် Table မှ၊ မည်သည့် Column များကို ဖော်ပြပါဟု တိတိကျကျ command ပေးရလေသည်။ Data များ ရွေးချယ် ခြင်းကို၊ Database အခေါ်အားဖြင့် Select လုပ်သည်ဟု ခေါ်လေသည်။ Girls table မှ Data အားလုံးကို ရွေးချယ်သော select statement ကိုရေးသော်....

“Girls table မှ No, Name နှင့် Nric တို့ကို ရွေးချယ်ပါ”၊ တစ်နည်း

```
SELECT No, Name, Nric FROM Girls;
```

ဟု ဖော်ပြရလေသည်။

ဤ statement တွင် SELECT နှင့် FROM တို့မှာ keywords များဖြစ်ကြပြီး။ မိမိရွေးချယ်လိုသည့် အတိုင် (columns) များနှင့် table အမည်ကို ဖော်ပြ ထားလေသည်။ DBS အား ၎င်း select statement ကိုပေးသော် figure 1 တွင်ရှိသည့် အတိုင်း Data ကိုထုတ်ပေး လေသည်။

အကယ်၍ select statement တွင် column အားလုံးကို မဖော်ပြဘဲ တစ်ခုသာဖော်ပြသော်၊

```
SELECT Name FROM Girls;
```

DBS သည် အောက်ပါအတိုင်း အမည် အတိုင်ကိုသာ ထုတ်ပေး ပေလိမ့်မည်။

NAME
Alice
Betty
Cindy
Dolly
Emmy
Emmy
Amy
Elizabeth
Bibo
Ann

တစ်ဖန် Column များ၏ အစီအစဉ်ကို ပြောင်းလဲ ဖော်ပြသော် ..

```
SELECT Nric, No, Name FROM Girls;
```

ဖော်ပြပါ column အစီအစဉ်အတိုင်း Nric, No, Name ဟု ထုတ်ပေးပေလိမ့်မည်။

NRIC	NO	NAME
Alice1A	1	Alice
Betty2B	2	Betty
Cindy3C	3	Cindy
Dolly4D	4	Dolly
Emmy5E	5	Emmy
Emmy6E	6	Emmy
Amy7A	7	Amy
Elizabeth8E	8	Elizabeth
Bibo9B	9	Bibo
Ann10A	10	Ann

ဤသို့ Data အားလုံးကိုထုတ်ပေးသော Select statement ရေးနည်းကိုကား သိပြီ။

Data အားလုံး မဟုတ်ဘဲ မိမိလိုချင်သော Data ကိုသာ သီးခြား ရွေးချယ် လိုသော် DBS အား မည်သည့် အတိုင်မှ မည်သည့် တန်ဖိုးပါရှိသော အတန်းများကို သာဖော်ပြပါဟု ပြောရလေသည်။

ထို့ကြောင့် Girls Table ရှိ Data များထဲမှ တစ်ခုခုကိုရှာဖွေလိုသော်၊ အောက်ပါတို့ကို သိရှိရန်လိုပေသည်။

1. မိမိရှာဖွေလိုသည့် တန်ဖိုးနှင့်
2. မည်သည့်အတိုင်(column) တွင်ရှာဖွေလိုသည်

ရှာဖွေခြင်းမှာ နေရာကို ဖော်ပြခိုင်းခြင်း ဖြစ်ရာ ဘိုလိုတွင် ဘယ်နေရာ Where ကို သုံးကာ မေးရလေသည်။ ထို့နည်းတူ DBS အားမေးရာတွင်၊ ဆိုကြပါစို့ Emmy အမည်ရှိသော မိန်းကလေးကို Girls table တွင်ရှာလိုသော် “Name သည် Emmy ဖြစ်သော/ညီမျှသော Girls table မှ No, Name, Nric တို့ကိုရွေးပါ” တစ်နည်း

```
SELECT No, Name, Nric FROM Girls WHERE Name = ‘Emmy’ ;
```

ဟု ရေးရလေသည်။ ဤတွင် WHERE သည် keyword ဖြစ်လေသည်။ Emmy သည် text data ဖြစ်သဖြင့် single quote အတွင်းတွင် ရေးရသည်။ ထိုသို့ WHERE ပါသော statement ပိုင်းကို WHERE clause ဟုခေါ်လေသည်။ ၎င်း where clause ပါသော statement အား DBS သို့ပေးသော် အောက်ပါအတိုင်း ထုတ်ပေးလေသည်။

NO	NAME	NRIC
5	Emmy	Emmy5E
6	Emmy	Emmy6E

DBS သည် Name အတိုင်တွင် ရှိသော တန်ဖိုးများတွင် Emmy နှင့်တူသော (ညီမျှသော) အတန်းများကိုသာ ရွေးချယ် ဖော်ပြပေး လေသည်။ ဤသို့ Where clause ကိုသုံးကာ Select statement ဖြင့် မိမိရှာဖွေလိုသော စာကို တိကျစွာ ရှာဖွေနိုင်လေသည်။

(ဤအပိုင်းတွင် select statement နှင့် where clause ကို မိတ်ဆက်ခြင်းဖြစ်ရကား။ where clause ၌ AND, OR, LIKE, <, >, <> , <=, >= operator များ အကြောင်းကို မဖော်ပြတော့ပြီ။)

ခက်သလားဟေ့ ဒေတာဘေ့စ် (ပေါ့ပေါ့ပါးပါး SQL များ) (၂)

6. Row ထဲမှ Data များကို ပြုပြင်ခြင်း

လက်ရှိ Table ထဲမှ Row တွင်ပါဝင်သော အချက်အလက်များ ပြုပြင်ခြင်းကို DataBase အသုံးအားဖြင့် UPDATE လုပ်သည်ဟု ခေါ်လေသည်။ Row မှ data ကို update လုပ်ရန် DBS အား မည်သည့် Table, မည်သည့် Row မှ မည်သည့် အတိုင်ရှိ တန်ဖိုးကို ပြင်ပါဟု တန်ဖိုးအသစ် ပေးကာ command ပေးရလေသည်။

Update လုပ်ရန် အောက်ပါအချက်များကို ဖော်ပြရန်လိုလေသည်။

1. မည်သည့် Table
2. မည်သည့် Row
3. မည်သည့် အတိုင်
4. ပြင်ဆင်မည့် အချက်အလက်

ဥပမာ Girls Table မှ NRIC တန်ဖိုး Emmy6E နှင့် ညီမျှသော Row တွင် NAME ကို Elle ဟု ပြင်ပါဟု စေခိုင်းလိုသော် အောက်ပါအတိုင်း Command ပေးရသည်။

UPDATE Girls SET NAME = 'Elle' WHERE NRIC = 'Emmy6E' ;

ဤ statement တွင် Name သာမက Nric ကို တစ်ပါတည်း ပြုပြင်ထားလေသည်။ အထူးသတိပြုရန်မှာ UPDATE statement တွင် အကယ်၍ WHERE clause ကို မဖော်ပြသော် table တွင် ရှိသော Row အားလုံးအား အကျိုးသက်ရောက်ခြင်းပင် ဖြစ်လေသည်။

ဥပမာ

UPDATE Girls SET Name = 'Elle', Nric = 'Elle6E' ;

ဟု ဖော်ပြပါက ရလဒ်မှာ အောက်ပါအတိုင်း ဖြစ်ပေမည်။

No	NAME	NRIC
1	Elle	Elle6E
2	Elle	Elle6E
3	Elle	Elle6E
4	Elle	Elle6E
....	Elle	
10	Elle	Elle6e

7. Row ကို ဖျက်သိမ်းခြင်း

Table ထဲမှ Row တစ်ခုကို ဖျက်ခြင်းကို Database အသုံးအားဖြင့် DELETE လုပ်သည်ဟု ခေါ်လေသည်။ DELETE Command ပေးရာတွင် အောက်ပါ အချက်များကို ဖော်ပြရန် လိုလေသည်။

1. မည်သည့် Table
2. မည်သည့် Row

တို့ဖြစ်ကြသည်။

ဥပမာ Girls table ထဲမှ အမည် Alice ရှိသော Row ကို ဖျက်လိုသော်

DELETE FROM Girls WHERE Name = 'Alice' ;

ဟု Command ပေးရသည်။

ဤ statement တွင် DELETE သည် Keywork ဖြစ်လေသည်။ DELETE command တွင် SELECT command ကဲ့သို့ပင် FROM keyword ကို သုံးရလေသည်။

WHERE clause ကိုသုံးကာ မိမိ ဖျက်ချင်သည့် row ကို ရွေးချယ် ပေးရလေသည်။

အထူးသတိပြုရန်မှာ အကယ်၍ WHERE clause ကို မဖော်ပြခဲ့သော် Table မှ Row အားလုံးကို ဖျက်ပေလိမ့်မည်။

8. Table များကို ဖျက်သိမ်းခြင်း

အထက်တွင် Table ကို ဖန်တီးခြင်း၊ အချက်အလက်များကို ထည့်သွင်း၊ ရှာဖွေ၊ ပြုပြင်၊ ဖျက်ခြင်းများ ကို သိခဲ့ပေပြီ။ နောက်ဆုံးတွင် Girls Table တစ်ခုလုံးအား မလိုအပ်သဖြင့် Table ကို ဖျက်သိမ်းလိုသော် ...

DROP TABLE Girls;

ဟု ရေးသားရလေသည်။

ဤ statement တွင် DROP နှင့် TABLE တို့မှာ Keywords များ ဖြစ်ကြပြီး မိမိဖျက်လိုသည့် Table ၏ အမည်ကို ဖော်ပြရလေသည်။ Table ကို ဖျက်သိမ်းရာတွင် DELETE ကို မသုံးပဲ DROP ကို သုံးသည်ကို သတိပြုပါ။ Table တစ်ခုကို ဖျက်သိမ်းလိုက်သော အခါ၊ အမှီ Table ပင် မရှိတော့သဖြင့် ၎င်း Table ထဲရှိ အတိုင်များ၊ အတန်းများ အားလုံးပါ ပျက်ဆီးရလေသည်။

9. ဤမှ ဆက်၍

ဤအပိုင်းတွင် Database System တစ်ခုအား command ပေးသော အခြေခံ SQL statement များအကြောင်းကို ဖော်ပြခဲ့ချေပြီ။ စာတွေ့သာ ဖော်ပြခဲ့ခြင်း ဖြစ်သည်။

နောက်တစ်ပိုင်းတွင်၊ ယခုလေ့လာခဲ့သော SQL များအား လက်တွေ့ စမ်းသပ်ကြည့်မည် ဖြစ်သည်။ လက်တွေ့လုပ်ဆောင်ရန်၊ SQLite ခေါ် Database System တစ်ခုအား အသုံးပြုကြရမည်။

MySQL မှ SQL အခြေခံများ (၁)

Web Application တိုင်း SQL နဲ့ မကင်းကြပါဘူး။ Static Page တွေဆိုရင် SQL ကို လှည့်ကြည့်စရာ မလိုပေမယ့်၊ Dynamic Page တွေဆိုရင်တော့ SQL က သေချာပေါက် လိုလာပါပြီ။ Dynamic Page တွေအနေနဲ့ Server ထဲမှာ Database လို့ခေါ်တဲ့ အချက်အလက်တွေ သိမ်းထားတာ၊ ပြန်ထုတ်သုံးတာ၊ ဖျက်ပစ်တာ၊ ပြန်ပြင်တာ စသည်ဖြင့် လုပ်ဖို့ လိုတဲ့အခါ SQL(Structured Query Language) ကိုသုံးပြီး ဆက်သွယ်ဖို့ လိုလာပါတယ်။

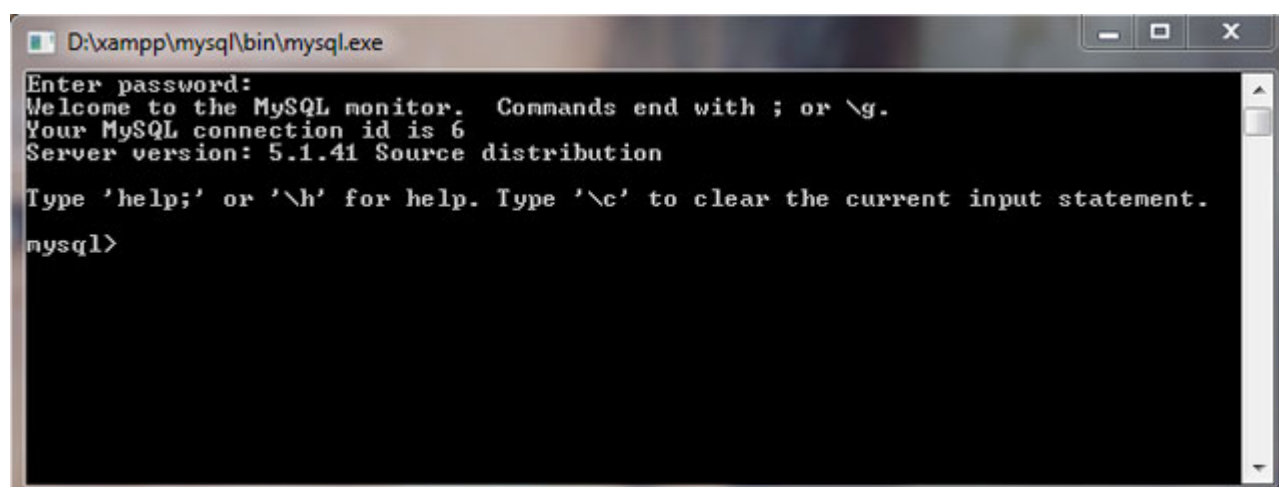
ဒီလို SQL ကိုသုံးကြတဲ့ နေရာမှာလည်း သုံးစွဲတဲ့ Server အပေါ် မူတည်ပြီး MS SQLServer, Sqlite, Oracle, MySQL ဆိုပြီး မျိုးစုံရှိပါတယ်။ ဒါပေမယ့် SQL အခြေခံ ရေးသားပုံတွေကတော့ တူညီကြတာ များပါတယ်။ တစ်ခုကို ကျွမ်းကျွမ်းကျင်ကျင် သိလိုရှိရင် အားလုံးကို လေ့လာလို့ ရတဲ့ အနေအထားမှာ ရှိပါတယ်။

SQL အခြေခံများမှာတော့ MySQL ကိုသုံးမှာ ဖြစ်ပြီး XAMPP သွင်းထားဖို့ လိုပါတယ်။ phpMyAdmin လည်း ရှိဖို့ လိုမှာ ဖြစ်ပါတယ်။ ဒါပေမယ့် XAMPP ကို သွင်းထားရင် phpmyadmin လည်း ပါပြီးသား ဖြစ်ပါတယ်။ XAMPP ကို ဘယ်လို install ရလဲဆိုတာ သိချင်ရင် [ဒီနေရာ](#) မှာ သွားကြည့် နိုင်ပါတယ်။

MySQL Console အတွက်ပြင်ဆင်ခြင်း

ပထမဦးဆုံး MySQL Database ကို တည်ဆောက်တဲ့ နေရာမှာ phpMyAdmin ကို မသုံးသေးပဲ MySQL Console ဆိုတဲ့ Command Prompt ကိုပဲ သုံးဦးမှာ ဖြစ်ပါတယ်။ အဲဒီလို သုံးနိုင်ဖို့အတွက်

- Start -> Run ကိုသွားပါ။
- ပြီးရင် XAMPP သွင်းထားတဲ့ Directory ကိုသွားပါ။ C:\ မှာ သွင်းထားရင်တော့ C:\>XAMPP ပေါ့။
- C:\>XAMPP\mysql\bin\mysql.exe -uroot လို့ ရိုက်ထည့်လိုက်ပါ။
- Password လာတောင်းပါလိမ့်မယ်။ ကိုယ်က Default Installation ဆိုရင် User Name – root ဖြစ်ပြီး password – <blank> ဖြစ်ပါတယ်။ Password မရှိဘူးပေါ့။ ဒီတော့ Enter ခေါက်လိုက်ပါ။



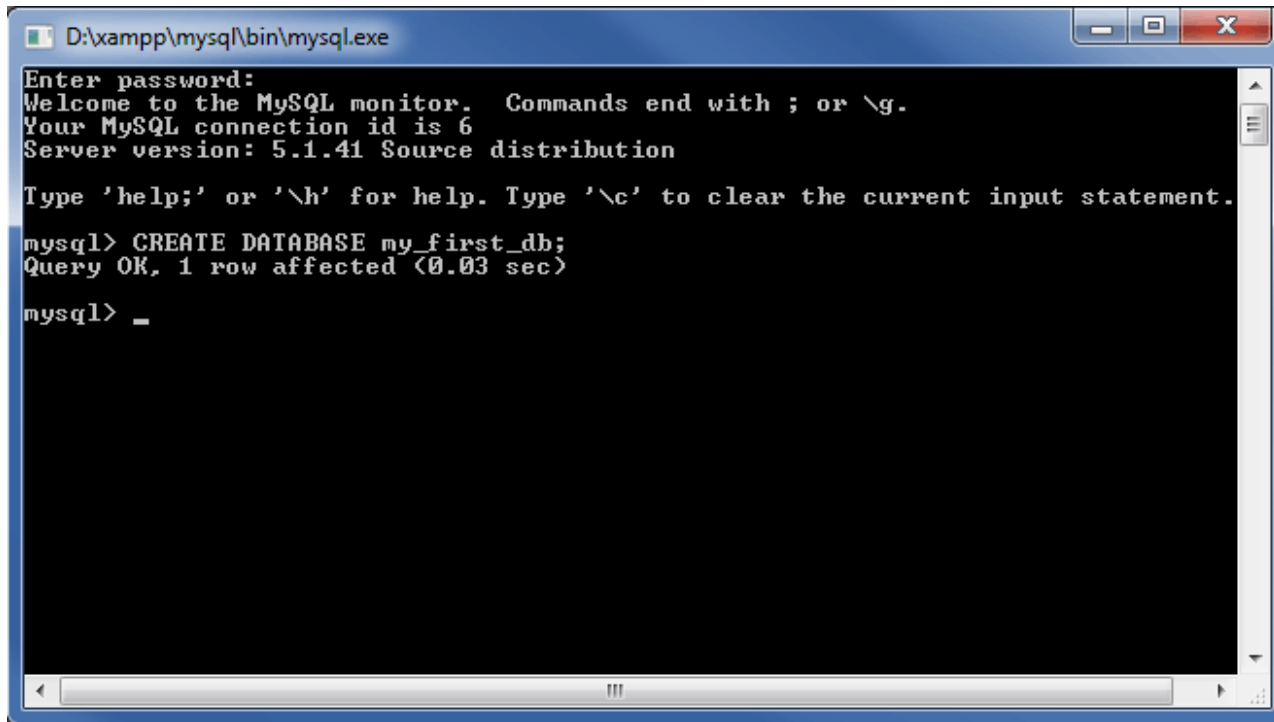
အဲဒီလို ပေါ်လာပြီဆိုရင်တော့ SQL Command တွေ စလို့ရပါပြီ။

Database တစ်ခု တည်ဆောက်ခြင်း (CREATE DATABASE)

အပေါ်မှာ ပြထားတဲ့အတိုင်း mysql> ဆိုတဲ့နေရာမှာ

1. CREATE DATABASE my_first_db;

ဆိုပြီး ရိုက်ထည့်လိုက်ပါ။



MySQL မှာ “;” ကို နောက်ဆုံးမှာ ထည့်ပေးဖို့ လိုပါတယ်။ CREATE DATABASE ကိုတော့ အကြီး အသေးကြိုက်တာ ရေးလို့ရပါတယ်။ create database ဝဲ ရေးရေး Create Database လို့ပဲရေးရေး အဆင်ပြေပါတယ်။ အဲဒီလို Database တည်ဆောက်တဲ့ နေရာမှာ CHARSET တို့ Collation တို့ကိုလည်း သတ်မှတ်ပေးလို့ ရပါတယ်။ CHARSET ကတော့ မြန်မာလိုသုံးမယ်ဆိုရင် UTF-8 ဆိုပြီး ရွေးဖို့လိုမှာ ဖြစ်ပါတယ်။ အများအားဖြင့်တော့ Database ဆိုရင် UTF-8 သုံးကြတာများပါတယ်။ Collation ဆိုတာကတော့ စာလုံးတွေကို ဘယ်လိုစုစည်းထားလဲ ဘယ်လို စီထားသလဲဆိုတာကို မှတ်ထားထုံစုံပါ။ UTF-8 ဆိုရင် utf8_general_ci ဆိုပြီး ရွေးကြပါတယ်။ ဒီတစ်ခါ Character set တွေ Collation တွေပါ ထည့်ပြီး Database ဆောက်ကြည့် ရအောင်

```
CREATE DATABASE my_first_db2 DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

MySQL အနေနဲ့ ဘယ်လို Character Set တွေ Collation တွေ အထောက်အပံ့ပေးတယ် ဆိုတာ ကြည့်ချင်ရင် [ဒီနေရာ](#) မှာ သွားကြည့်နိုင်ပါတယ်။

တည်ဆောက်ထားသော Database ဖိုင်များကို ပြန်ကြည့်ခြင်း (SHOW DATABASES)

MySQL ထဲမှာ ကိုယ်တည်ဆောက်ထားတဲ့ Database တွေကို ပြန်ကြည့်လို့ရပါတယ်။ Command ကတော့

1. SHOW DATABASES;

အဲဒီလို Command ရိုက်ထည့်လိုက်ရင် အောက်မှာ ပြထားတဲ့ ပုံအတိုင်း ပြပါလိမ့်မယ်။



Database ကို ပြန်ဖျက်ခြင်း (DROP DATABASE)

ကိုယ်တည်ဆောက်ထားတဲ့ Database ကို ပြန်ဖျက်ချင်တယ် ဆိုရင်

1. DROP DATABASE my_first_db;

ဆိုပြီး ပြန်ဖျက်လို့ရပါတယ်။


```

D:\xampp\mysql\bin\mysql.exe
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.1.41 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement

mysql> DROP DATABASE my_first_db;
Query OK, 0 rows affected (0.12 sec)

mysql> _

```

Database ကို ဖျက်တဲ့နေရာမှာတော့ သတိကြီးကြီးထားဖို့ လိုပါမယ်။ Windows မှာ file ဖျက်မယ်ဆိုရင် ဖျက်မှာ သေချာလားလို့ လာမေးပေမယ့် ဒီမှာတော့ မမေးပါဘူး။ ချက်ချင်း ဖျက်ပစ်မှာပါ။ အကယ်၍ ကိုယ်သုံးနေလက်စ Database ကို မှားဖျက်မိလို့ကတော့ သွားပြီးသာမှတ်။ Hosting မှာဆိုရင် ပိုပြီး ဂရုစိုက်ဖို့ လိုပါတယ်။ ခုလို Testing Environment မှာ ဆိုရင်တော့ သိပ်ပြဿနာ မရှိဘူးပေါ့။

Database ဖိုင်ကို ရွေးချယ်ခြင်း (USE DATABASE)

ဒါကတော့ SQL Query မဟုတ်ပါဘူး။ statement တစ်ခုသာ ဖြစ်ပါတယ်။ MySQL ထဲမှာ ဆောက်ထားတဲ့ Database တွေ များတဲ့အခါ ကိုယ် ဘယ် Database ကို သုံးမယ်ဆိုတာ ရွေးပေးဖို့ လိုပါတယ်။ အဲဒီအတွက် သုံးတာပါ။ statement ဖြစ်တဲ့အတွက် နောက်ဆုံးမှာ ‘;’ ထည့်စရာ မလိုပါဘူး။ Command အနေနဲ့က

1. USE my_first_db

```

D:\xampp\mysql\bin\mysql.exe
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.1.41 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement

mysql> DROP DATABASE my_first_db;
Query OK, 0 rows affected (0.12 sec)

mysql> USE my_first_db
ERROR 1049 (42000): Unknown database 'my_first_db'
mysql> CREATE DATABASE my_first_db;
Query OK, 1 row affected (0.03 sec)

mysql> USE my_first_db
Database changed
mysql> _

```

USE my_first_db လို့ ပထမ Command ရိုက်ထည့်တော့ File ကို ဖျက်ထားတဲ့အတွက် ERROR 1049 <42000>: Unknown database ‘my_first_db’ ဆိုပြီး Error လာပြပါလိမ့်မယ်။ ဒါကြောင့် CREATE DATABASE ထပ်လုပ်ပါတယ်။ ပြီးမှ Use my_first_db ဆိုပြီး သုံးပါတယ်။ Database changed ဆိုပြီး လာပြပါလိမ့်မယ်။ ဒါဆိုရင်တော့ Database Table တွေ ဆောက်ဖို့ အသင့် ဖြစ်ပြီပေါ့။

Database Table ဆိုတာဘာလဲ

Database Table ဆိုတာ သက်ဆိုင်ရာ အချက်အလက်တွေကို စနစ်တကျ မှတ်ထားတဲ့ မှတ်တမ်းလို့ ဆိုရမယ် ထင်ပါတယ်။ Student Database ဆိုရင် Student နဲ့ သက်ဆိုင်တဲ့ Data တွေ မှတ်ထားပါလိမ့်မယ်။

	A5			
	A	B	C	D
1	student_id	student_name	email	date
2		1 Aung Aung	aungaung@gmail.com	12/12/2009
3		2 Maung Maung	maungmaug@hotmail.com	5/14/2010
4		3 Tu Tu	tutu@mmtut.com	8/14/2009
5				
6				
7				
8				

ဒါကတော့ Database Table ရဲ့ နမူနာ ပုံစံပါ။ Database Table တစ်ခုမှာ Column Name တွေပါမယ်။ Row of Data တွေ ပါပါလိမ့်မယ်။ Database Table ကို တည်ဆောက်လို့ရမယ်။ ဖတ်လို့ရမယ်။ ပြင်လို့ရမယ်။ ဖျက်လို့ရပါလိမ့်မယ်။ CRUD လို့ အတိုကောက် ခေါ်ပါတယ်။ Create, Read, Update, Delete ပေါ့။

Database Table တစ်ခု တည်ဆောက်ခြင်း (CREATE TABLE)

SQL Query သုံးပြီး Database Table တစ်ခု တည်ဆောက်ကြည့်ရအောင်။ Create Table အကြောင်း အသေးစိတ် သိချင်ရင် [MySQL Documentation](#) မှာ သွားကြည့်နိုင်ပါတယ်။ ဒါပေမယ့် အခုမှ စလေ့လာမယ့် သူတွေအတွက် သိပ်ပြီး လေ့လာလို့ မကောင်းပါဘူး။

Database Field နှစ်ခုနဲ့ Table တစ်ခု ဆောက်ကြည့်ရအောင်၊ အရင် ဆောက်ခဲ့တဲ့ my_first_db မှာပေါ့။ Command အနေနဲ့

1. **CREATE TABLE** students (
2. **student_name** VARCHAR(20),
3. **date** DATE
4.);

အရင် Command တွေ ရေးလာတုန်းကတော့ တစ်ကြောင်းတည်းပါပဲ။ ဒီတစ်ခါတော့ Command တွေကို Multiple Lines အနေနဲ့ သုံးလို့ ရိုက်လို့ရပါတယ်။

အဲဒီ Command အသေးစိတ်ကို လေ့လာကြည့်မယ်ဆိုရင် ပထမဦးဆုံး စာကြောင်းက CREATE TABLE students ဆိုတာ students ဆိုတဲ့ Table တစ်ခုကို တည်ဆောက်မယ် ဆိုတဲ့ အဓိပ္ပါယ်ပါပဲ။ အဲဒီနောက်မှာ ‘(‘ ’) ကြားထဲမှာ Data Column တွေ ထည့်ပါတယ်။ Data Column တစ်ခုနဲ့ တစ်ခု ကြားမှာ , လေးတွေ ခြားပေးဖို့ လိုပါတယ်။ Column Name တစ်ခုခြင်းစီကို သူ့ Data Type သတ်မှတ်ပေးဖို့ လိုပါတယ်။ အဲဒီလို Data Type ဆိုတာ Numeric လို့ခေါ်တဲ့ ၁၊ ၂၊ ၃၊ ၄ တွေသုံးမှာလား၊ Text လို့ခေါ်တဲ့ စာတွေ သုံးမှာလား၊ Date လို့ခေါ်တဲ့ နေ့စွဲတွေ သုံးမှာလား စသည်ဖြင့် သတ်မှတ်ပေးရပါတယ်။ အဲဒီလို သတ်မှတ်ပေးတဲ့အပြင် တစ်ချို့ Data Column တွေ အတွက် အများဆုံးအသုံးပြုမယ့် စာလုံးအရေ အတွက်ကိုပါ သတ်မှတ်ပေးရပါတယ်။ VARCHAR(20) ဆိုရင် VARCHAR က Data Type ဖြစ်ပြီး (20) ကတော့ အများဆုံး ထည့်နိုင်တဲ့ စာလုံး အရေအတွက်ပါ။ ဘယ်လို Data Type တွေ သုံးနိုင်လဲ သိချင်ရင် [Data Type Overview](#) မှာသွားဖတ်နိုင်ပါတယ်။

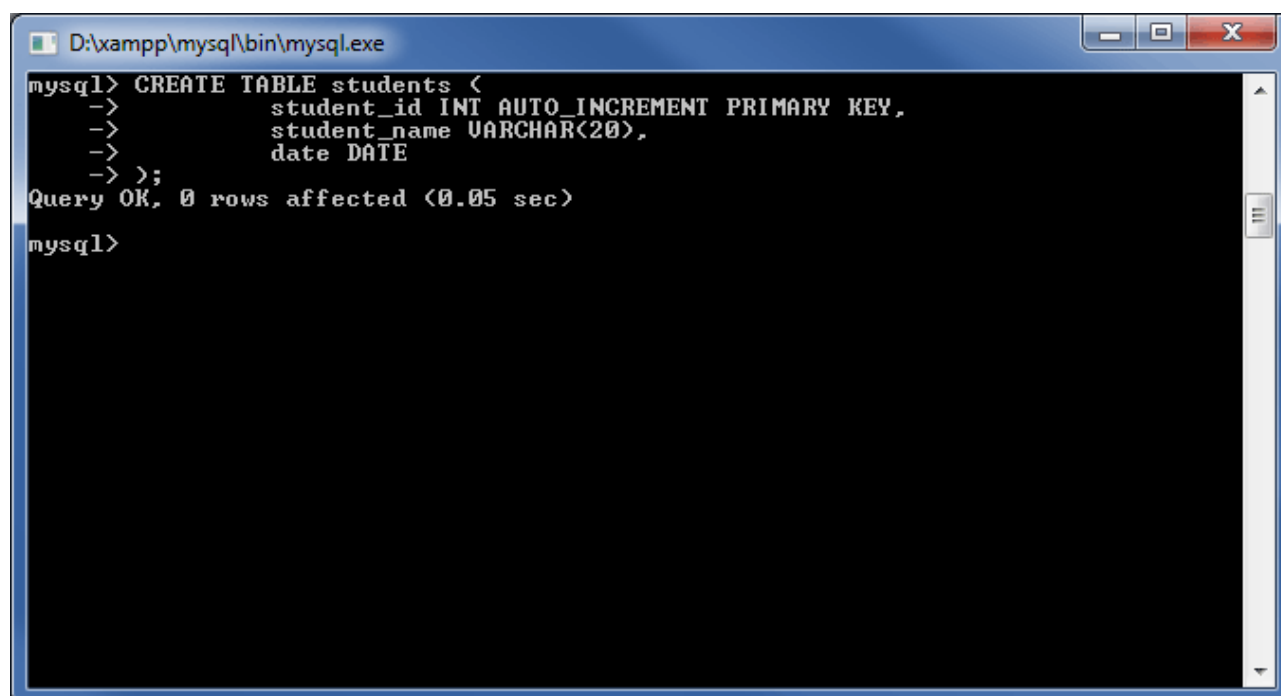
VARCHAR က String value ကို လက်ခံပြီး Date ကတော့ Date Format ‘YYYY-MM-DD’ ကို လက်ခံပေးမှာ ဖြစ်ပါတယ်။

PRIMARY KEY

Primary Key ဆိုတာကတော့ Data Row တစ်ခုကို ကိုယ်စားပြုနိုင်တဲ့ Column Name ကို သတ်မှတ်ပေးထားတာ ဖြစ်ပါတယ်။ student_id နဲ့ student_name မှာ ဘယ်ဟာကို Primary Key သတ်မှတ် သင့်သလဲ စဉ်းစားကြည့်ရအောင် student_name မှတော့ အောင်အောင် တွေထပ်နိုင်ပါတယ်။ ဒါပေမယ့် student_id ကတော့ ထပ်စရာ အကြောင်းမရှိပါဘူး။ ကျောင်းသား တစ်ယောက်ကို id တစ်ခုပဲ ရှိမှာမို့ပါ။ ဒီတော့ PRIMARY KEY ကို student_id ကို သတ်မှတ်မှ အဆင်ပြေပါလိမ့်မယ်။ ဒီတော့ ကျွန်တော်တို့ အပေါ်က Query ကိုပြင်ရေး ကြည့်ရအောင်

1. **CREATE TABLE** students (
2. **student_id** INT AUTO_INCREMENT PRIMARY KEY,
3. **student_name** VARCHAR(20),
4. **date** DATE
5.);

ဒီ Table မှာ student_id အတွက် INT ဆိုတာ 32bit integer type ကို ဆိုလိုပါတယ်။ AUTO_INCREMENT ဆိုတာကတော့ Data Row တစ်ခုထည့်တိုင်း သူ့အလိုလို တန်ဖိုး တစ်ခုတိုးတိုးသွားဖို့ပါ။ ကိုယ့်ဘာသာ ထည့်လည်းရပေမယ့် သူ့ဘာသာ ထည့်လိုက်တော့ အလုပ် သက်သာ သွားတာပေါ့။ နောက်တစ်ခု PRIMARY KEY ဆိုပြီး သတ်မှတ် ပေးပါတယ်။ PRIMARY KEY မသတ်မှတ်ပေးလည်း ဘာမှတော့ မဖြစ်ပါဘူး။ ဒါပေမယ့် Relational Database ဖြစ်ဖို့အတွက် တစ်ခုနဲ့ တစ်ခု ချိတ်ဆက် နိုင်ဖို့အတွက် PRIMARY KEY က မရှိမဖြစ် လိုအပ်ပါတယ်။ Query ကို Run ကြည့်ရအောင်



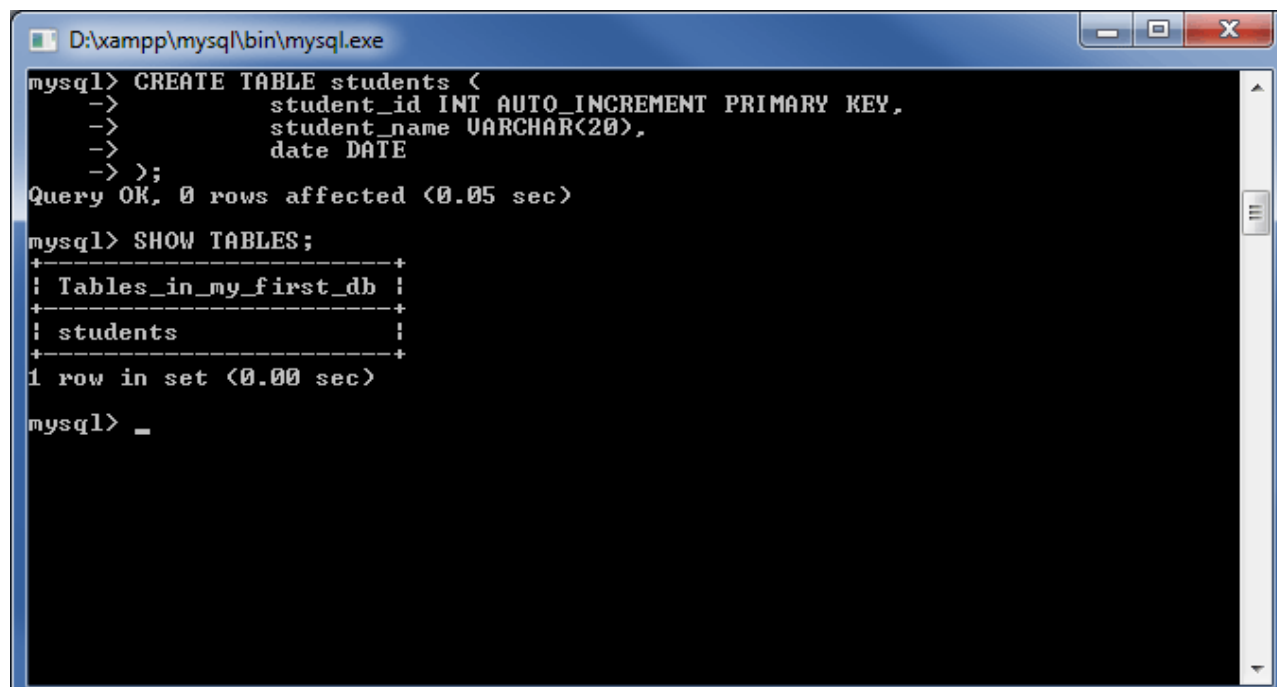
```
D:\xampp\mysql\bin\mysql.exe
mysql> CREATE TABLE students (
->     student_id INT AUTO_INCREMENT PRIMARY KEY,
->     student_name VARCHAR(20),
->     date DATE
-> );
Query OK, 0 rows affected (0.05 sec)
mysql>
```

ဒါဆိုရင်တော့ Database Table တစ်ခု ဆောက်ပြီးသွားပါပြီ။

Table များကို ကြည့်ခြင်း (SHOW TABLES)

ကိုယ့်ရဲ့ Database ထဲမှာ Database Table တွေ ဘယ်နှစ်ခု ရှိလဲ၊ ဆောက်ထားလဲ ကြည့်ချင်တဲ့အခါ

1. `SHOW TABLES;`



```
D:\xampp\mysql\bin\mysql.exe
mysql> CREATE TABLE students (
->     student_id INT AUTO_INCREMENT PRIMARY KEY,
->     student_name VARCHAR(20),
->     date DATE
-> );
Query OK, 0 rows affected (0.05 sec)

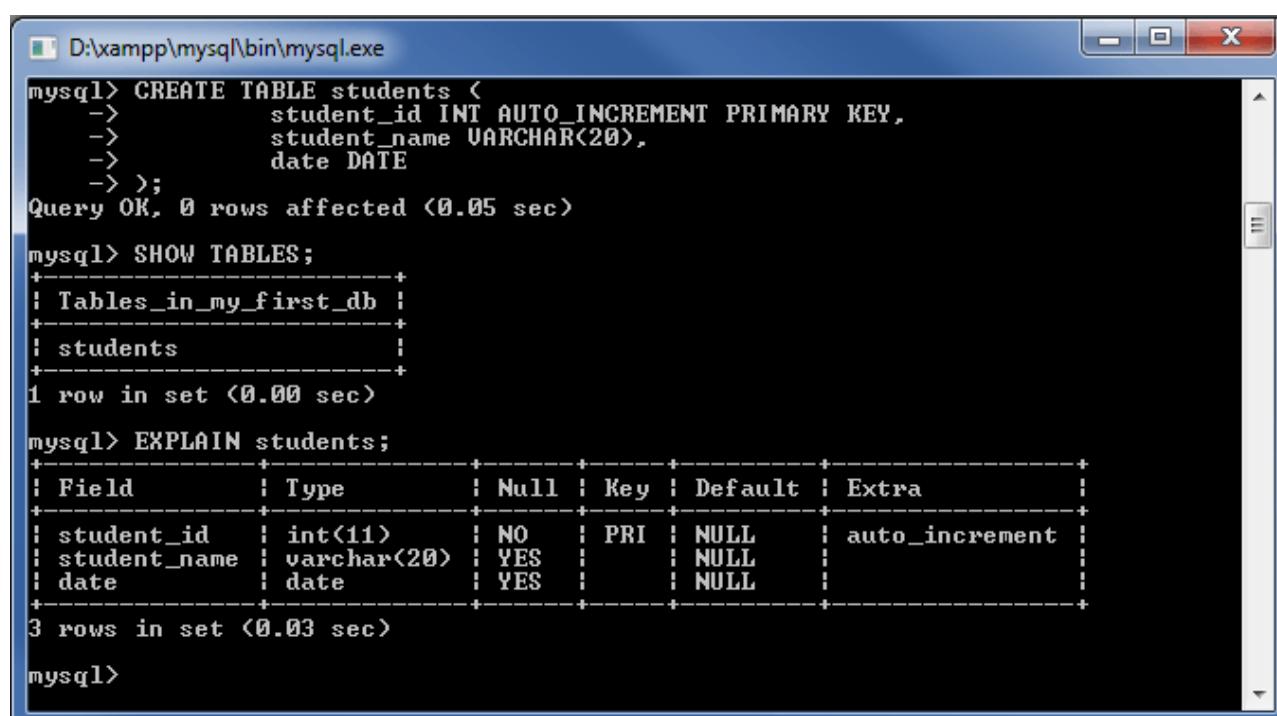
mysql> SHOW TABLES;
+-----+
| Tables_in_my_first_db |
+-----+
| students               |
+-----+
1 row in set (0.00 sec)

mysql> _
```

Database Table ၏ တည်ဆောက်ပုံကို ကြည့်ခြင်း (EXPLAIN)

ကိုယ်တည်ဆောက်ထားတဲ့ Database Table ရဲ့ Table Structure ကို ကြည့်ချင်တယ်ဆိုရင်

1. `EXPLAIN students;`



```
D:\xampp\mysql\bin\mysql.exe
mysql> CREATE TABLE students (
->     student_id INT AUTO_INCREMENT PRIMARY KEY,
->     student_name VARCHAR(20),
->     date DATE
-> );
Query OK, 0 rows affected (0.05 sec)

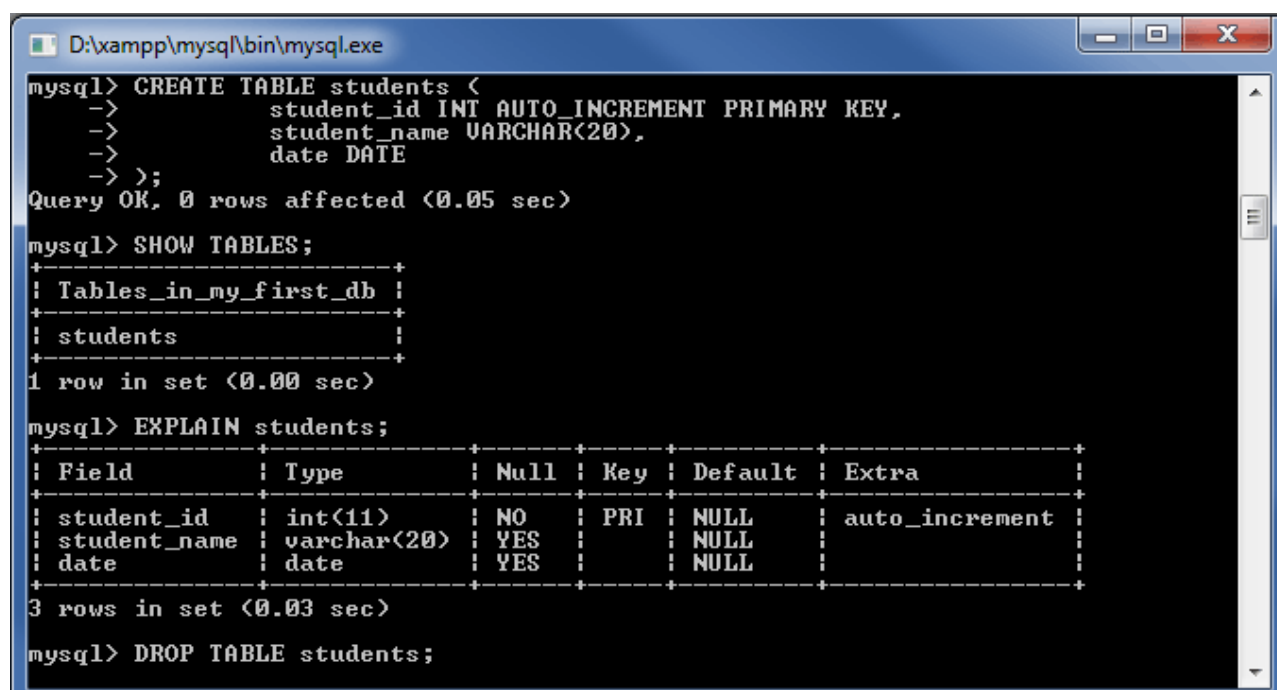
mysql> SHOW TABLES;
+-----+
| Tables_in_my_first_db |
+-----+
| students               |
+-----+
1 row in set (0.00 sec)

mysql> EXPLAIN students;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra      |
+-----+-----+-----+-----+-----+-----+
| student_id | int(11)   | NO   | PRI | NULL    | auto_increment |
| student_name | varchar(20) | YES  |     | NULL    |              |
| date       | date      | YES  |     | NULL    |              |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.03 sec)

mysql>
```

Database Table ကို ဖျက်ခြင်း (DROP TABLE)

ဒီနေရာမှာ Database ကို ဖျက်တဲ့ Command နဲ့ အတူတူပါပဲ။



```
D:\xampp\mysql\bin\mysql.exe
mysql> CREATE TABLE students (
->     student_id INT AUTO_INCREMENT PRIMARY KEY,
->     student_name VARCHAR(20),
->     date DATE
-> );
Query OK, 0 rows affected (0.05 sec)

mysql> SHOW TABLES;
+-----+
| Tables_in_my_first_db |
+-----+
| students               |
+-----+
1 row in set (0.00 sec)

mysql> EXPLAIN students;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra      |
+-----+-----+-----+-----+-----+-----+
| student_id | int(11)   | NO   | PRI | NULL    | auto_increment |
| student_name | varchar(20) | YES  |     | NULL    |              |
| date       | date      | YES  |     | NULL    |              |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.03 sec)

mysql> DROP TABLE students;
```

MySQL မှ SQL အခြေခံများ (၂)

Table ကို ပြင်ဆင်ခြင်း (Modifying)

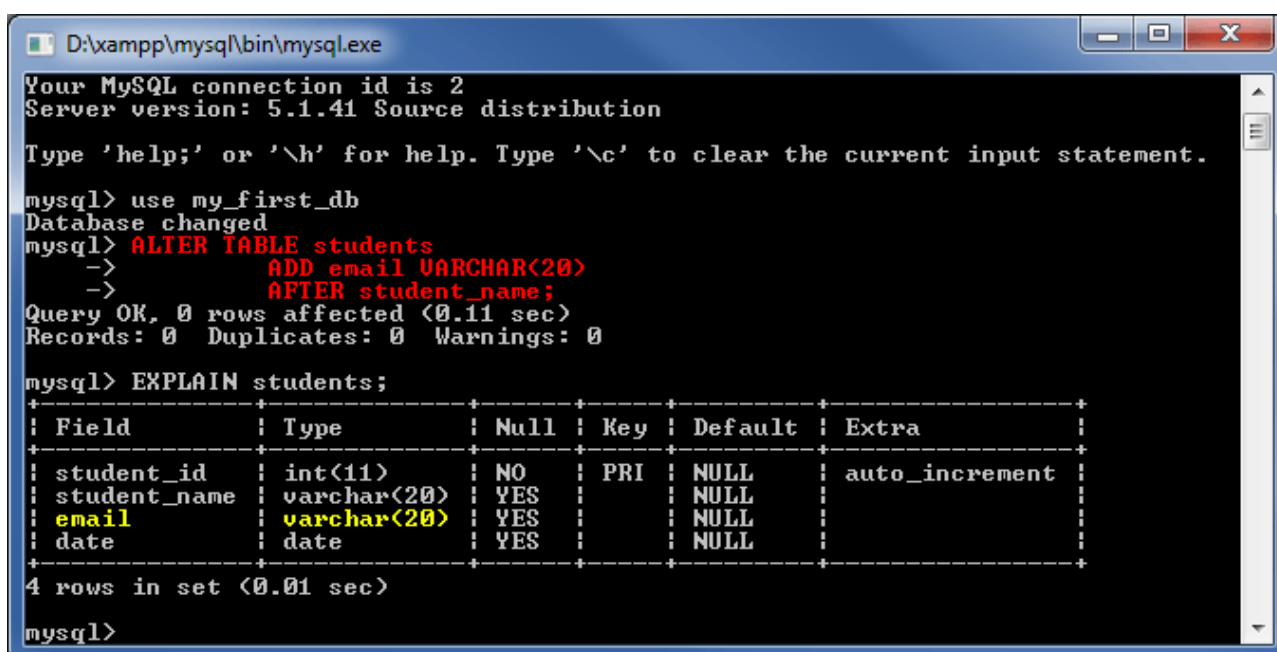
Table ကို ပြင်ဆင်ခြင်း ဆိုတဲ့နေရာမှာ အထဲမှာ ရှိတဲ့ Data တွေပြင်ဖို့ မဟုတ်ပဲ၊ Database Column တွေကို ပြုပြင်တာ၊ ထပ်ပေါင်း ထည့်တာ စတာတွေကို ဆိုလိုပါတယ်။ Data Field တွေကို ပြင်တဲ့အပိုင်း၊ ပေါင်းထည့်တဲ့အပိုင်း ဖျက်တဲ့ အပိုင်း မဟုတ်သေးပါဘူး။ အဲဒီလိုလုပ်တဲ့ Query တွေက နည်းနည်းတော့ ရှုပ်ပါတယ်။ အားလုံးသိချင်ရင်တော့ [ဒီမှာ](#) သွားဖတ်နိုင်ပါတယ်။

(အပေါ်က DROP TABLE မှာတုန်းက Table ကို ဖျက်ထားမိရင် ပြန်ဆောက်ဖို့ မမေ့နဲ့နော်။)

SQL Query အနေနဲ့

Column တစ်ခု ထပ်ပေါင်းထည့်ချင်တယ် ဆိုရင်

1. [ALTER TABLE students](#)
2. [ADD email VARCHAR\(100\)](#)
3. [AFTER student_name;](#)



```
D:\xampp\mysql\bin\mysql.exe
Your MySQL connection id is 2
Server version: 5.1.41 Source distribution

Type 'help;' or 'h' for help. Type 'c' to clear the current input statement.

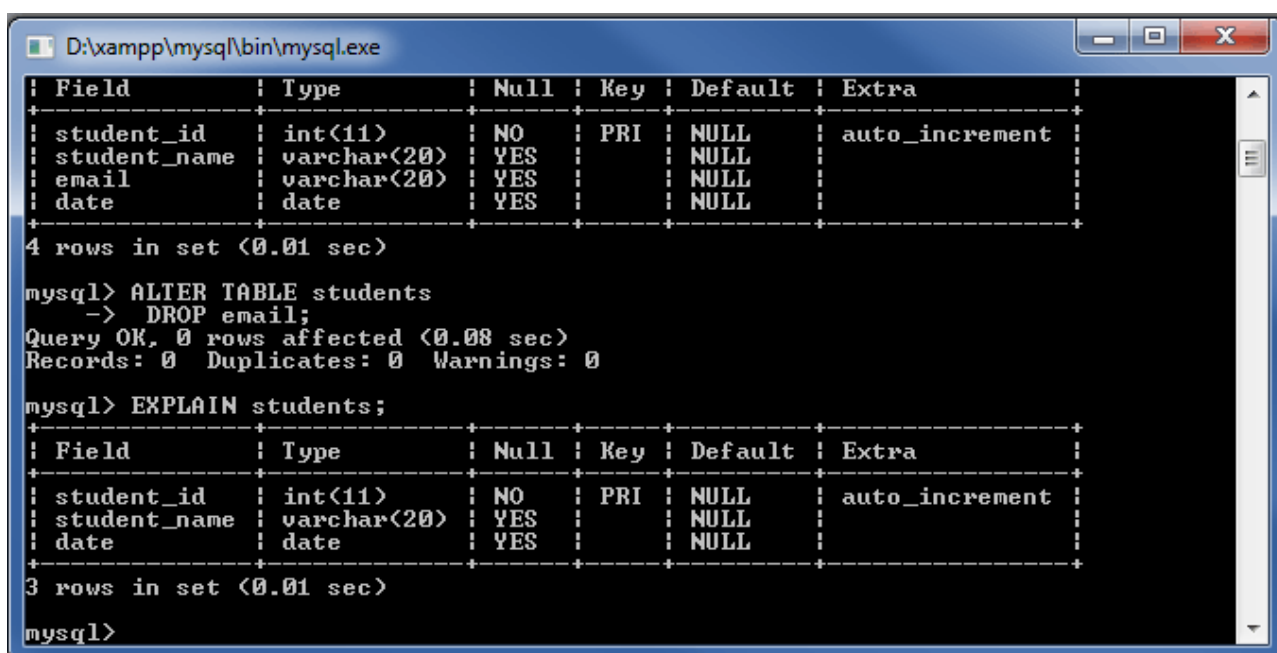
mysql> use my_first_db
Database changed
mysql> ALTER TABLE students
-> ADD email VARCHAR(20)
-> AFTER student_name;
Query OK, 0 rows affected (0.11 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> EXPLAIN students;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| student_id | int(11) | NO | PRI | NULL | auto_increment |
| student_name | varchar(20) | YES | | NULL | |
| email | varchar(20) | YES | | NULL | |
| date | date | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.01 sec)

mysql>
```

Column တစ်ခု ဖယ်ထုတ်ချင်တယ် ဆိုရင်

1. [ALTER TABLE students](#)
2. [DROP email;](#)



```
D:\xampp\mysql\bin\mysql.exe

+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| student_id | int(11) | NO | PRI | NULL | auto_increment |
| student_name | varchar(20) | YES | | NULL | |
| email | varchar(20) | YES | | NULL | |
| date | date | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.01 sec)

mysql> ALTER TABLE students
-> DROP email;
Query OK, 0 rows affected (0.08 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> EXPLAIN students;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| student_id | int(11) | NO | PRI | NULL | auto_increment |
| student_name | varchar(20) | YES | | NULL | |
| date | date | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)

mysql>
```

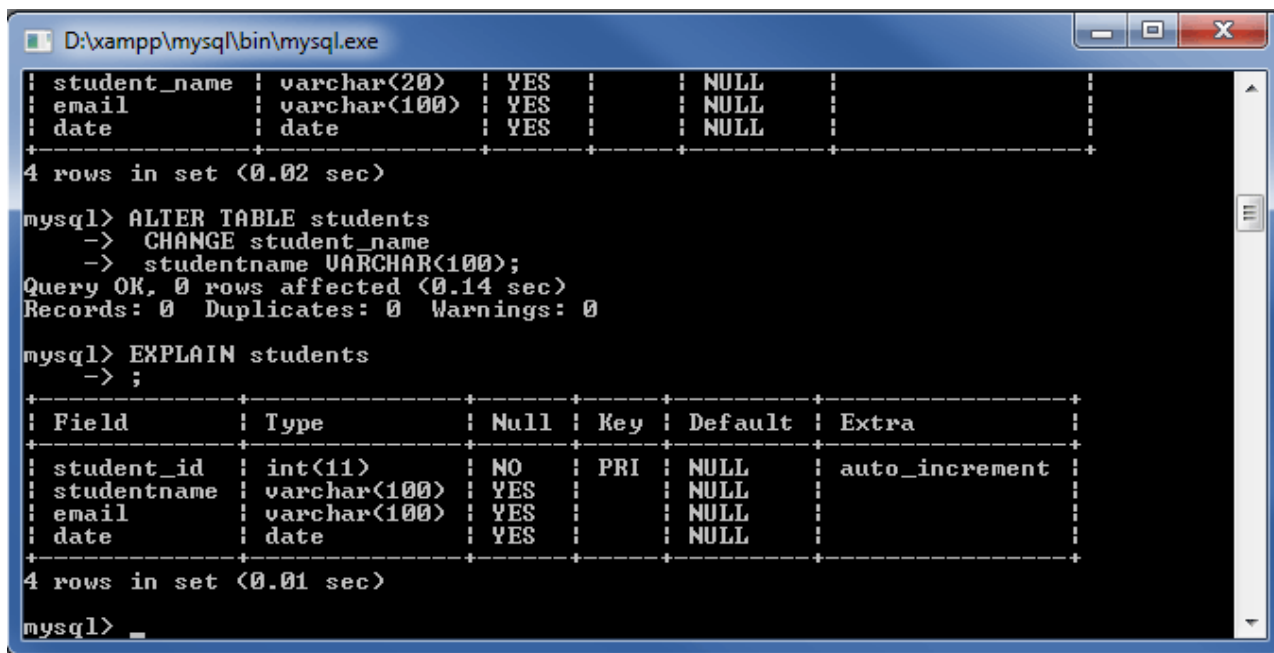
ဖျက်ပြီဆိုရင်တော့ နေရာတကာ သတိထားဖို့ လိုမယ်ဆိုတာ သိလောက်ပြီ ထင်ပါတယ်။ အထဲမှာ Data တွေ ရှိရင် ပိုသတိထားဖို့ လိုပါတယ်။

email ကို ဖျက်လိုက်တော့ နောက်တစ်ခါ EXPLAIN မှာ မပါလာတော့ပါဘူး။ email ကို ပြန် Add လိုက်ပါဦး။

1. [ALTER TABLE students](#)
2. [ADD email VARCHAR\(100\)](#)
3. [AFTER student_name;](#)

Column တစ်ခုကို ပြန်ပြင်ချင်တယ် ဆိုရင်

1. `ALTER TABLE students`
2. `CHANGE student_name`
3. `studentname VARCHAR(100);`



```
D:\xampp\mysql\bin\mysql.exe
+-----+-----+-----+-----+-----+-----+
| student_name | varchar(20) | YES | | NULL |
| email         | varchar(100) | YES | | NULL |
| date          | date       | YES | | NULL |
+-----+-----+-----+-----+-----+
4 rows in set (0.02 sec)

mysql> ALTER TABLE students
-> CHANGE student_name
-> studentname VARCHAR(100);
Query OK, 0 rows affected (0.14 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> EXPLAIN students
-> ;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| student_id | int(11) | NO | PRI | NULL | auto_increment |
| studentname | varchar(100) | YES | | NULL |
| email | varchar(100) | YES | | NULL |
| date | date | YES | | NULL |
+-----+-----+-----+-----+-----+
4 rows in set (0.01 sec)

mysql> _
```

နဂို student_name ကနေ studentname ဆိုပြီး ပြောင်းသွားပါလိမ့်မယ်။ ဒီအထဲက Table တစ်ခုလုံးနဲ့ ဆိုင်တဲ့ Column Name စတာတွေနဲ့ ဆိုင်တဲ့ အပိုင်းတွေပါ။

Data Entry အပိုင်း:

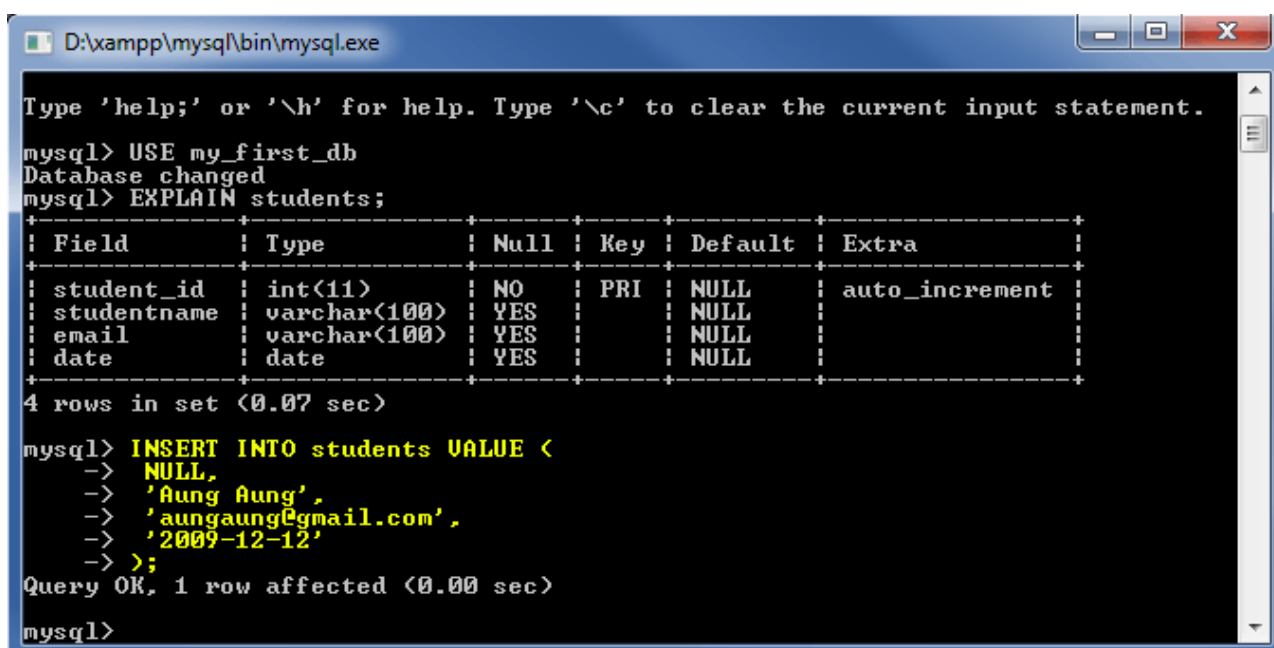
Table တစ်ခုအတွင်းသို့ Data ထည့်ခြင်း

ကျွန်တော် ရှေ့ပို့စ်မှာတုန်းက Excel ဇယားနဲ့ ပြထားခဲ့တဲ့ Data တွေကို Table ထဲ ထည့်ကြည့်ရအောင်

A5				
	A	B	C	D
1	student_id	student_name	email	date
2	1	Aung Aung	aungaung@gmail.com	12/12/2009
3	2	Maung Maung	maungmaug@hotmail.com	5/14/2010
4	3	Tu Tu	tutu@mmtut.com	8/14/2009
5				
6				
7				
8				

ပထမဦးဆုံး တစ်ကြောင်းကို Data Table ထဲထည့်မယ်ဆိုရင် (Myanmar Tutorials မှာ mail hide လုပ်ထားတော့ @ ရေးလို့ မရပါဘူး။ ဒါကြောင့် (at) ဆိုပြီး ပြောင်းထားပါတယ်။ @ လို့ ပြောင်းလိုက်ပါ။

1. `INSERT INTO students VALUES (`
2. `NULL,`
3. `'Aung Aung',`
4. `'aungaung(at)gmail.com',`
5. `'2009-12-12'`
6. `);`



```
D:\xampp\mysql\bin\mysql.exe
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE my_first_db
Database changed
mysql> EXPLAIN students;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| student_id | int(11) | NO | PRI | NULL | auto_increment |
| studentname | varchar(100) | YES | | NULL |
| email | varchar(100) | YES | | NULL |
| date | date | YES | | NULL |
+-----+-----+-----+-----+-----+
4 rows in set (0.07 sec)

mysql> INSERT INTO students VALUE <
-> NULL,
-> 'Aung Aung',
-> 'aungaung(at)gmail.com',
-> '2009-12-12'
-> >;
Query OK, 1 row affected (0.00 sec)

mysql>
```


VALUES() ဆိုတဲ့ နေရာမှာ ကိုယ်ထည့်ချင်တဲ့ Value တွေကို ‘,’ လေးတွေ ခံပြီး ထည့်ပေးရပါတယ်။ အဲဒီလို ထည့်တဲ့နေရာမှာ String Value ဆိုရင် ‘ ’ ကြားထဲထည့် ပေးပေးရပါတယ်။ ရိုးရိုး Numeric Value ဆိုရင်တော့ ဒီအတိုင်း ထည့်နိုင်ပါတယ်။ အဲဒီတန်ဖိုးတွေကို ကိုယ်အရင် တည်ဆောက်ခဲ့တဲ့ Table ရဲ့ Column Name တွေရဲ့ Data Type တွေအပေါ်မှာ မူတည်ပြီး ထည့်ပေးရမှာပါ။ ကိုယ်က VARCHAR ဆိုရင် String Value ထည့်ပေးရပါမယ်။ INT ဆိုရင် Numeric Value ထည့်ပေးရမှာပေါ့။

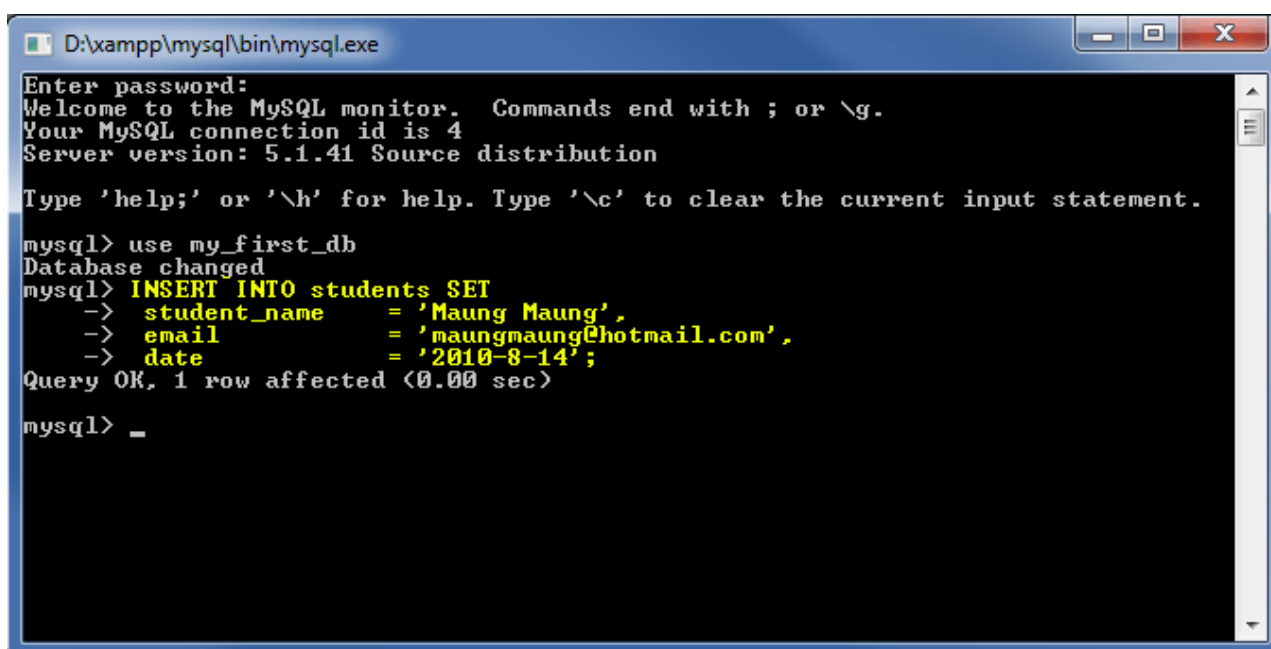
PRIMARY KEY ဖြစ်တဲ့ student_id ကို NULL ဆိုပြီး ထည့်ထားတာ သတိပြုမိပါလိမ့်မယ်။ ဘာကြောင့် အဲဒီလို ထည့်ရတာလဲဆိုရင် AUTO_INCREMENT ပေးထားတဲ့အတွက် အလိုလို Key Value တစ်ခု သူ့ဘာသာ ထည့်လို့ရအောင်ပါ။ ဥပမာ – ကိုယ်ထည့်လိုက်တဲ့ Data Row တစ်ခုမှာ 1 ဆိုရင် နောက်တစ်ခုမှာ 2 လို့ ထည့်ပေးပါလိမ့်မယ်။

Table တစ်ခုအတွင်းသို့ Data ထည့်ခြင်း (နောက်တစ်နည်း)

Data Entry အတွက် နောက်ထပ် နည်းတစ်နည်းလည်း ရှိပါသေးတယ်။

1. **INSERT INTO students SET**
2. **student_name = 'Maung Maung',**
3. **email = 'maungmaung(at)hotmail.com',**
4. **date = '2010-8-14';**

ဒီနေရာမှာကျွန်တော် student_name ကို ပြန်ပြောင်းထားပါတယ်။ အရင်တစ်ခါ student_name ကို studentname ဆိုပြီး ပြောင်းထားတာ မှတ်မိဦးမယ် ထင်ပါတယ်။ အဲဒီအတိုင်း ပြန်ပြောင်းလိုက်ပါ။



```
D:\xampp\mysql\bin\mysql.exe
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.1.41 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

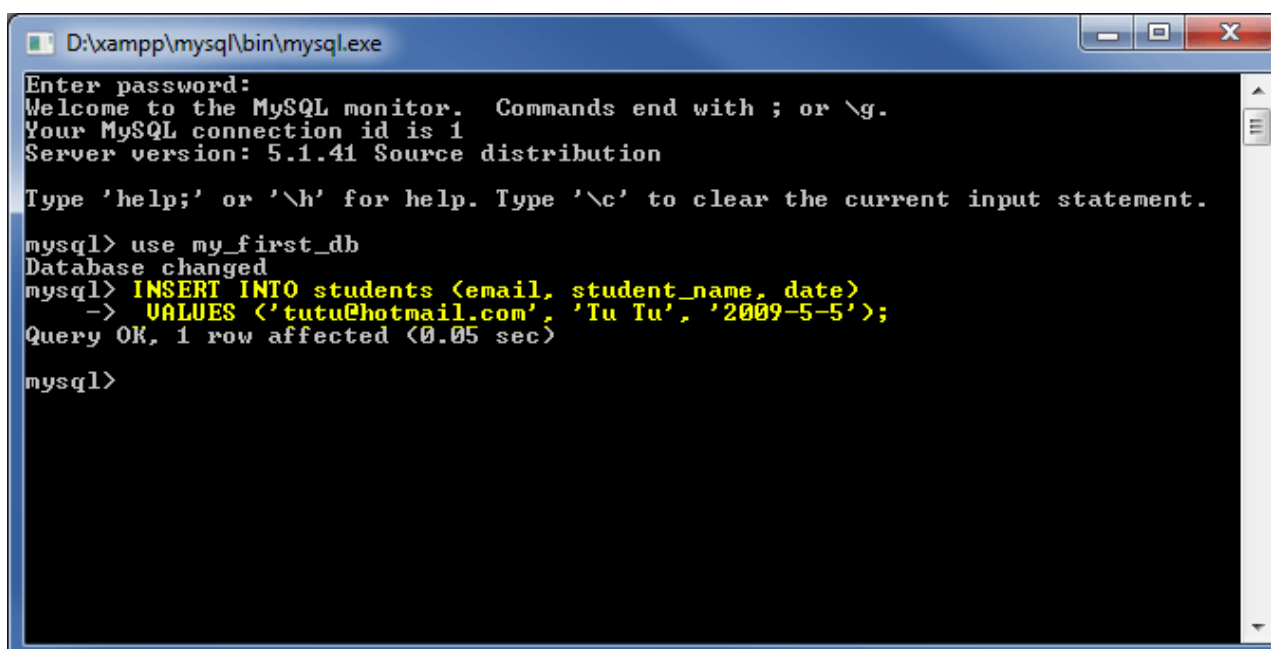
mysql> use my_first_db
Database changed
mysql> INSERT INTO students SET
-> student_name = 'Maung Maung',
-> email = 'maungmaung@hotmail.com',
-> date = '2010-8-14';
Query OK, 1 row affected (0.00 sec)

mysql> _
```

ဒီတစ်ခါ ရေးတဲ့ ပုံစံက VALUES အစား SET ဆိုပြီး သုံးသွားပါတယ်။ နောက်ပြီး () ကွင်းစ ကွင်းပိတ်လည်း မလိုပါဘူး။ အဲဒီနှစ်ခု ဘာကွာလဲ ဆိုရင် သူနေရာနဲ့သူ အားနည်းချက် အားသာချက်တွေ ရှိပါတယ်။ အပေါ်က တစ်ခုက Data Column Name တွေ ရေးစရာ မလိုပါဘူး။ အစဉ်လိုက်ထည့် ပေးရုံပါပဲ။ ဒါပေမယ့် ကျော်လို့ မရပါဘူး။ နောက်တစ်ခုကတော့ Column Name တွေ ထည့်ပေးဖို့ လိုပေမယ့် student_id လို AUTO_INCREMENT Value တွေကို ကျော်သွားလို့ရပါတယ်။ VARCHAR တွေကို ကျော်သွားမယ်ဆိုရင် blank ထည့်ထားပါလိမ့်မယ်။ (အကယ်၍ ကိုယ်က Default Value မပေးထားဘူး ဆိုရင်ပေါ့) column name တွေနဲ့ ကိုက်ပြီး ထားတဲ့အတွက် အစီအစဉ်တကျ ဖြစ်စရာ မလိုတော့ပါဘူး။

Table တစ်ခုအတွင်းသို့ Data ထည့်ခြင်း (နောက်တစ်နည်း)

1. **INSERT INTO students (email, student_name, date)**
2. **VALUES ('tutu(at)hotmail.com', 'Tu Tu', '2009-5-5');**



```
D:\xampp\mysql\bin\mysql.exe
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.1.41 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use my_first_db
Database changed
mysql> INSERT INTO students (email, student_name, date)
-> VALUES ('tutu@hotmail.com', 'Tu Tu', '2009-5-5');
Query OK, 1 row affected (0.05 sec)

mysql>
```

LAST_INSERT_ID()

ဒါကတော့ AUTO_INCREMENT id ကိုသုံးပြီး Data ထည့်ထားတဲ့ အခါမှာ နောက်ဆုံး ထည့်ခဲ့တဲ့ ID ကို ပြန်ကြည့်တဲ့နေရာမှာ သုံးပါတယ်။


```
D:\xampp\mysql\bin\mysql.exe
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.1.41 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use my_first_db
Database changed
mysql> INSERT INTO students (email, student_name, date)
-> VALUES ('tutu@hotmail.com', 'Tu Tu', '2009-5-5');
Query OK, 1 row affected (0.05 sec)

mysql> SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
| 3 |
+-----+
1 row in set (0.03 sec)

mysql>
```

NOW()

NOW() ဆိုတာကို သုံးရင် ယခုလက်ရှိ Date ကို return ပြန်ပေးမှာ ဖြစ်ပါတယ်။ အကယ်၍ ကိုယ်ထည့်တဲ့ Date Entry အခု လက်ရှိ Date ကိုထည့်ချင်တဲ့အခါမှာ သုံးပါတယ်။

```
D:\xampp\mysql\bin\mysql.exe
Server version: 5.1.41 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use my_first_db
Database changed
mysql> INSERT INTO students (email, student_name, date)
-> VALUES ('tutu@hotmail.com', 'Tu Tu', '2009-5-5');
Query OK, 1 row affected (0.05 sec)

mysql> SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
| 3 |
+-----+
1 row in set (0.03 sec)

mysql> INSERT INTO students SET
-> date = NOW(),
-> student_name = 'Goo Goo',
-> email = 'googoo@gmail.com';
Query OK, 1 row affected, 1 warning (0.03 sec)

mysql>
```

အဲဒီလို NOW() ကို သုံးလိုက်တဲ့ အခါမှာ Warning တစ်ခု ထွက်လာတာ တွေ့ပါလိမ့်မယ်။ ဘာကြောင့် အဲဒီလို Warning တစ်ခု တက်လာသလဲဆိုရင် NOW() ဆိုတဲ့ function က Date တင်မကပဲ Time ကိုသာ return ပြန်ပေးတာကြောင့်ပါ။

```
D:\xampp\mysql\bin\mysql.exe
Query OK, 1 row affected (0.05 sec)

mysql> SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
| 3 |
+-----+
1 row in set (0.03 sec)

mysql> INSERT INTO students SET
-> date = NOW(),
-> student_name = 'Goo Goo',
-> email = 'googoo@gmail.com';
Query OK, 1 row affected, 1 warning (0.03 sec)

mysql> SELECT NOW();
+-----+
| NOW() |
+-----+
| 2010-06-10 12:58:40 |
+-----+
1 row in set (0.00 sec)

mysql>
```

ဒီနေရာမှာ NOW() ကိုသုံးတာထက် CURDATE() ကို ပိုပြီး သုံးသင့်ပါတယ်။ CURDATE() ကတော့ လုံးဝ Date ကိုပဲ Return ပြန်ပေးမှာ ဖြစ်ပါတယ်။

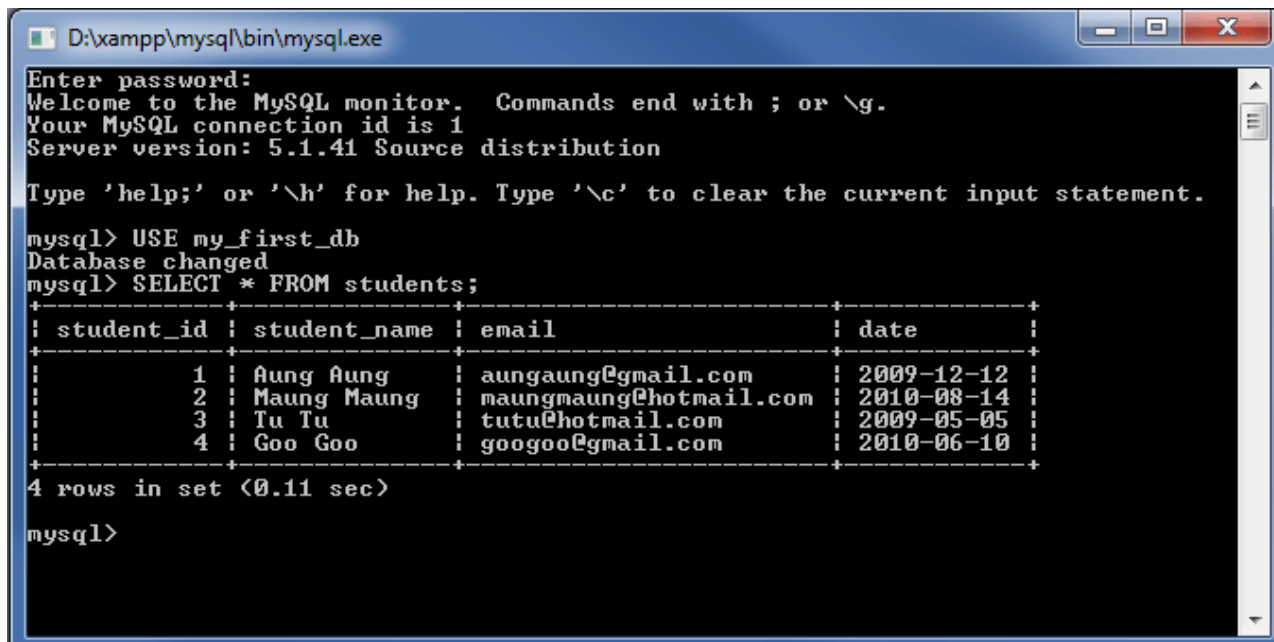
MySQL မှ SQL အခြေခံများ (3)

အခုဆက်ရေးမယ့် အပိုင်းကတော့ Retrieving Database ဆိုတဲ့ အပိုင်း ဖြစ်ပါတယ်။

Database ဖိုင်အတွင်းမှာ Data များကို ဖတ်ခြင်း

Data တွေကို Entry လုပ်တယ်ဆိုတာ ပြန်ဖတ်ဖို့ပါ။ အဲဒီလိုမှ ပြန်မဖတ်နိုင်ရင် Database ဆိုတာ အလကားပါပဲ။ SELECT ဆိုတဲ့ Query ပဲ သုံးပါတယ်။

1. **SELECT * FROM students;**



```
D:\xampp\mysql\bin\mysql.exe
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.1.41 Source distribution

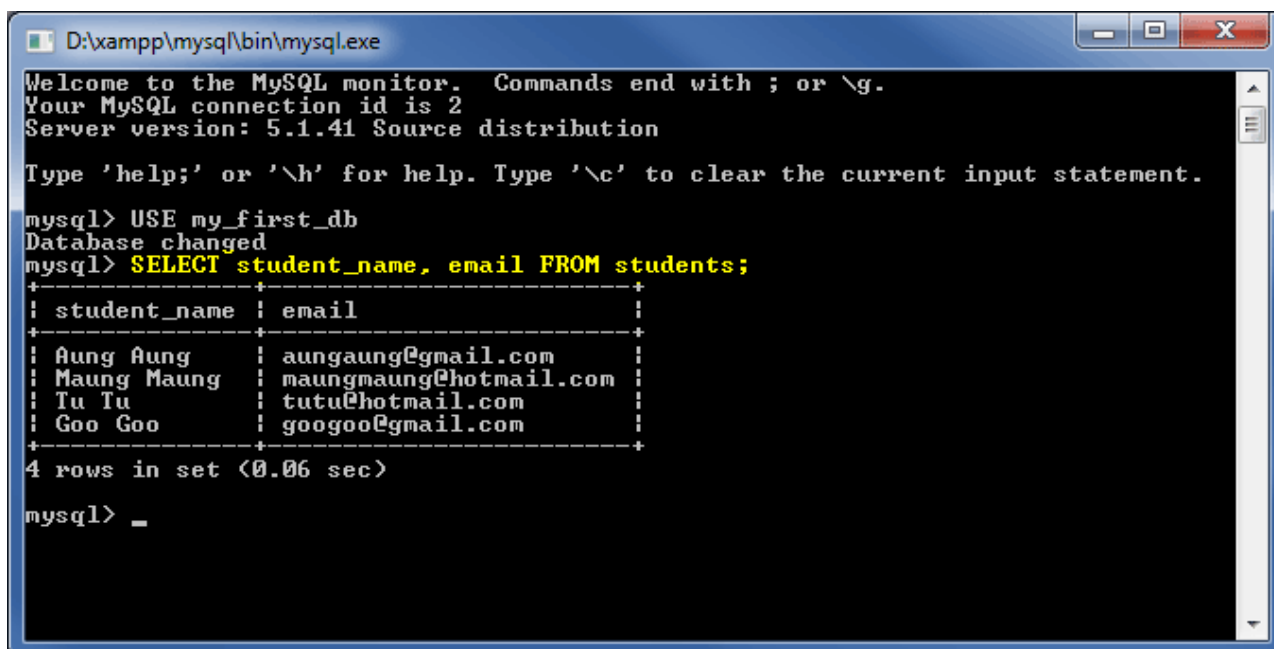
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE my_first_db
Database changed
mysql> SELECT * FROM students;
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 1 | Aung Aung | aungaung@gmail.com | 2009-12-12 |
| 2 | Maung Maung | maungmaung@hotmail.com | 2010-08-14 |
| 3 | Tu Tu | tutu@hotmail.com | 2009-05-05 |
| 4 | Goo Goo | googoo@gmail.com | 2010-06-10 |
+-----+-----+-----+-----+
4 rows in set (0.11 sec)

mysql>
```

SELECT * ဆိုတဲ့ နေရာမှာ * ဆိုတဲ့ အဓိပ္ပါယ်က Table ထဲမှာရှိတဲ့ Column တွေ အားလုံးထဲမှာ ရှိတဲ့ Data တွေကို ပြပါဆိုတဲ့ အဓိပ္ပါယ်ပါ။
သီးခြားစီ ရွေးပြီး ပြစေချင်တယ် ဆိုရင်တော့

1. **SELECT student_name, email FROM students;**



```
D:\xampp\mysql\bin\mysql.exe
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.1.41 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE my_first_db
Database changed
mysql> SELECT student_name, email FROM students;
+-----+-----+
| student_name | email |
+-----+-----+
| Aung Aung | aungaung@gmail.com |
| Maung Maung | maungmaung@hotmail.com |
| Tu Tu | tutu@hotmail.com |
| Goo Goo | googoo@gmail.com |
+-----+-----+
4 rows in set (0.06 sec)

mysql> _
```

SELECT အတွက် မရှိမဖြစ် WHERE

ကိုယ်က Data အချက်အလက်တွေကို ရွေးချယ်တဲ့ နေရာမှာ ဒီထက်ပိုပြီး Specific ဖြစ်စေဖို့ WHERE ဆိုတဲ့ စာလုံးကို မသုံးလို့ မဖြစ်ပါဘူး။ ဒီတော့ WHERE ဆိုတာကို သုံးကြည့်ရအောင်

1. **SELECT email FROM students**
2. **WHERE student_name = "Aung Aung";**

```

D:\xampp\mysql\bin\mysql.exe
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> USE my_first_db
Database changed
mysql> SELECT student_name, email FROM students;
+-----+-----+
| student_name | email |
+-----+-----+
| Aung Aung    | aungaung@gmail.com |
| Maung Maung  | maungmaung@hotmail.com |
| Tu Tu       | tutu@hotmail.com |
| Goo Goo     | googoo@gmail.com |
+-----+-----+
4 rows in set (0.06 sec)

mysql> SELECT email FROM students
-> WHERE student_name = 'Aung Aung';
+-----+
| email |
+-----+
| aungaung@gmail.com |
+-----+
1 row in set (0.00 sec)

mysql>

```

ဒီတစ်ခါတော့ WHERE နဲ့ စစ်ထားတာကြောင့် Aung Aung နဲ့ ဆိုင်တဲ့ email ကိုပဲ ပြပေးပါတော့တယ်။ Programming မှာ ဆိုရင်တော့ IF နဲ့ တူပါလိမ့်မယ်။ အခြား နှိုင်းယှဉ်တဲ့ Condition တွေကိုလည်း သုံးလို့ရပါတယ်။ ဆက်လေ့လာကြည့်ရအောင်

1. `SELECT * FROM students WHERE student_id <= 2;`
2. `SELECT * FROM students WHERE date != '2009-12-12';`

```

D:\xampp\mysql\bin\mysql.exe
Your MySQL connection id is 3
Server version: 5.1.41 Source distribution
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> USE my_first_db
Database changed
mysql> SELECT * FROM students WHERE student_id <= 2;
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 1 | Aung Aung | aungaung@gmail.com | 2009-12-12 |
| 2 | Maung Maung | maungmaung@hotmail.com | 2010-08-14 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM students WHERE date = '2009-12-12';
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 1 | Aung Aung | aungaung@gmail.com | 2009-12-12 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>

```

အဲဒီလို WERE ကို သုံးလို့ရသလို AND တို့ OR တို့နဲ့လည်း တွဲသုံးလို့ရပါတယ်။ AND ဆိုလိုရင် Condition နှစ်ခုလုံးနဲ့ တူဖို့ လိုမှာ ဖြစ်ပြီး၊ OR ဆိုရင်တော့ တစ်ခုမဟုတ် တစ်ခုနဲ့ တူဖို့ လိုမှာ ဖြစ်ပါတယ်။

```

D:\xampp\mysql\bin\mysql.exe
+-----+-----+-----+-----+
| 1 | Aung Aung | aungaung@gmail.com | 2009-12-12 |
| 2 | Maung Maung | maungmaung@hotmail.com | 2010-08-14 |
+-----+-----+-----+-----+
2 rows in set (0.03 sec)

mysql> SELECT * FROM students WHERE date != '2009-12-12';
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 2 | Maung Maung | maungmaung@hotmail.com | 2010-08-14 |
| 3 | Tu Tu | tutu@hotmail.com | 2009-05-05 |
| 4 | Goo Goo | googoo@gmail.com | 2010-06-10 |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> SELECT * FROM students WHERE
-> student_id = 1 OR student_name = 'Aung Aung';
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 1 | Aung Aung | aungaung@gmail.com | 2009-12-12 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> _

```

WHERE ရဲ့ နောက်မှာ Value တွေကို ရွေးဖို့ပေးတဲ့နေရာမှာ (ဥပမာ - student_name = 'Aung Aung') လိုမျိုးဆိုရင် ' ' လေးတွေ ကြားမှာ ရေးပေးရမှာ ဖြစ်ပေမယ့် Numeric Value တွေမှာတော့ လိုမှာမဟုတ်ပါဘူး။

IN()

Multiple Values တွေကို ရွေးထုတ်တဲ့နေရာမှာ အင်မတန် အသုံးဝင်ပါတယ်။ ဥပမာ - ၁၂-၁၂-၂၀၀၉ ကနေ ၁၂-၁၂-၂၀၁၀ အတွင်းက အချက်အလက်တွေ ပြပါဆိုတာမျိုးတွေမှာ သုံးလို့ရပါတယ်။

```
D:\xampp\mysql\bin\mysql.exe

+-----+-----+-----+-----+
| 1 | Aung Aung | aungaung@gmail.com | 2009-12-12 |
| 2 | Maung Maung | maungmaung@hotmail.com | 2010-08-14 |
+-----+-----+-----+-----+
2 rows in set (0.03 sec)

mysql> SELECT * FROM students WHERE date != '2009-12-12';
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 2 | Maung Maung | maungmaung@hotmail.com | 2010-08-14 |
| 3 | Tu Tu | tutu@hotmail.com | 2009-05-05 |
| 4 | Goo Goo | googoo@gmail.com | 2010-06-10 |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> SELECT * FROM students WHERE
-> student_id = 1 OR student_name = 'Aung Aung';
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 1 | Aung Aung | aungaung@gmail.com | 2009-12-12 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> _
```

LIKE()

သူကတော့ Windows မှာ ဖိုင်တွေ ရှာသလိုပါပဲ။ ဘယ်စာလုံးပါတဲ့ဟာတွေ ရှာပေး စသည်ဖြင့် သုံးလို့ရပါတယ်။ ကျွန်တော်တို့ Database ထဲမှာ ဆို ရင် hostmail သုံးတာ ဘယ်သူတွေလဲ ကြည့်ချင်တာမျိုးတွေမှာ သုံးလို့ရပါတယ်။

1. `SELECT * FROM students WHERE`
2. `email LIKE '%hotmail%';`

```
D:\xampp\mysql\bin\mysql.exe

+-----+-----+-----+-----+
| 4 | Goo Goo | googoo@gmail.com | 2010-06-10 |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> SELECT * FROM students WHERE
-> date IN ('2009-12-12', '2010-6-10');
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 1 | Aung Aung | aungaung@gmail.com | 2009-12-12 |
| 4 | Goo Goo | googoo@gmail.com | 2010-06-10 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM students WHERE
-> email LIKE '%hotmail%';
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 2 | Maung Maung | maungmaung@hotmail.com | 2010-08-14 |
| 3 | Tu Tu | tutu@hotmail.com | 2009-05-05 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

ORDER BY

ORDER BY ဆိုတဲ့ အသုံးကို ငယ်စဉ်ကြီးလိုက် ဖြစ်ဖြစ်၊ ကြီးစဉ် ငယ်လိုက် ဖြစ်ဖြစ် စီချင်တဲ့အခါမှာ သုံးပါတယ်။

```
D:\xampp\mysql\bin\mysql.exe

mysql> USE my_first_db
Database changed
mysql> SELECT * FROM students
-> ORDER BY date;
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 3 | Tu Tu | tutu@hotmail.com | 2009-05-05 |
| 1 | Aung Aung | aungaung@gmail.com | 2009-12-12 |
| 4 | Goo Goo | googoo@gmail.com | 2010-06-10 |
| 2 | Maung Maung | maungmaung@hotmail.com | 2010-08-14 |
+-----+-----+-----+-----+
4 rows in set (0.07 sec)

mysql> SELECT * FROM students
-> ORDER BY student_name DESC;
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 3 | Tu Tu | tutu@hotmail.com | 2009-05-05 |
| 2 | Maung Maung | maungmaung@hotmail.com | 2010-08-14 |
| 4 | Goo Goo | googoo@gmail.com | 2010-06-10 |
| 1 | Aung Aung | aungaung@gmail.com | 2009-12-12 |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

ဒီနေရာမှာ မှတ်စရာ ရှိတာက ASC နဲ့ DESC ပါ။ Default Order အနေနဲ့ ASC (Ascending Order) ကို သုံးပြီး DESC (Descending Order) ပါ။

LIMIT ... OFFSET ...

ကိုယ်လိုချင်တဲ့ Results ကို နှစ်ခုတည်းပြပါ၊ သုံးခုပြပါ၊ နှစ်ခုကျော်က ပြပါ ဆိုတာမျိုးတွေလည်း လုပ်လို့ရပါတယ်။ အဲဒီလို လုပ်နိုင်ဖို့အတွက် LIMIT ... OFFSET ... ဆိုတာကို သုံးပါတယ်။

```

D:\xampp\mysql\bin\mysql.exe

mysql> SELECT * FROM students LIMIT 2;
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 1 | Aung Aung | aungaung@gmail.com | 2009-12-12 |
| 2 | Maung Maung | maungmaung@hotmail.com | 2010-08-14 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM students
-> LIMIT 1 OFFSET 2;
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 3 | Tu Tu | tutu@hotmail.com | 2009-05-05 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM students
-> LIMIT 2, 1;
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 3 | Tu Tu | tutu@hotmail.com | 2009-05-05 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>

```

LIMIT 2 ဆိုတာ အပေါ်ဆုံးက Row နှစ်ခုကိုပဲ ပြပါလို့ ဆိုလိုပါတယ်။ LIMIT 1 OFFSET 2 ဆိုတာ ၂ ခုကျော်က ၁ ခုကို ပြပါဆို အဓိပ္ပာယ်ပါ။ ဒီတော့ အမှတ် ၁ နဲ့ ၂ ကို ပြတာ သတိထားမိပါလိမ့်မယ်။ LIMIT 2, 1 ဆိုတာကတော့ OFFSET 2, LIMIT 1 ဆိုတဲ့ အဓိပ္ပာယ်ပါ။ ဒီနေရာမှာ ပြောင်းပြန်ဖြစ်သွားတာ သတိထားမိပါလိမ့်မယ်။

Data များကို ပြန်လည် ပြုပြင်ခြင်း

Data တွေကို ပေါင်းထည့်တာ၊ ပြန်ထုတ်ကြည့်တာ စသည်ဖြင့် လေ့ကျင့်ခဲ့ကြပြီးပါပြီ။ ဒီတစ်ခါတော့ Data တွေကို ပြန်ပြင်တဲ့အပိုင်းကို လေ့လာ ကြည့်ရအောင် UPDATE ဆိုတဲ့ SQL Command နဲ့ သုံးပါတယ်။

```

D:\xampp\mysql\bin\mysql.exe

mysql> USE my_first_db
Database changed
mysql> SELECT * FROM students;
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 1 | Aung Aung | aungaung@gmail.com | 2009-12-12 |
| 2 | Maung Maung | maungmaung@hotmail.com | 2010-08-14 |
| 3 | Tu Tu | tutu@hotmail.com | 2009-05-05 |
| 4 | Goo Goo | googoo@gmail.com | 2010-06-10 |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> UPDATE students SET
-> email = 'aungaung@hotmail.com',
-> student_name = 'Aung Hla'
-> WHERE student_name = 'Aung Aung';
Query OK, 1 row affected (0.03 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM students WHERE student_name = 'Aung Hla';
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 1 | Aung Hla | aungaung@hotmail.com | 2009-12-12 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>

```

ပထမတုန်းကတော့ student_name မှာ Aung Aung ပါ။ ဒါကို Aung Hla ဆိုပြီး ပြောင်းပါတယ်။ အဲဒီလိုပဲ aungaung(at)gmail.com ကိုလည်း aungaung(at)hotmail.com လို့ ပြောင်းလိုက်ပါတယ်။ WHERE ကတော့ မပါမဖြစ်ပါ။ Student Name မှမဟုတ်ပါဘူး။ ဘယ် Column Name ကို မဆိုညွှန်းနိုင်ပါတယ်။ WHERE မပါဘူးဆိုရင်တော့ အားလုံး ပြောင်းသွားပါလိမ့်မယ်။ UPDATE ကို LIMIT နဲ့လည်း တွဲသုံးလို့ ရပါသေးတယ်။


```

mysql> SELECT * FROM students;
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 1 | Aung Hla | aungaung@hotmail.com | 2009-12-12 |
| 2 | Maung Maung | maungmaung@hotmail.com | 2009-12-12 |
| 3 | Tu Tu | tutu@hotmail.com | 2009-05-05 |
| 4 | Goo Goo | googoo@gmail.com | 2010-06-10 |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> UPDATE students SET date = '2009-12-18'
-> WHERE date = '2009-12-12' LIMIT 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM students;
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 1 | Aung Hla | aungaung@hotmail.com | 2009-12-18 |
| 2 | Maung Maung | maungmaung@hotmail.com | 2009-12-12 |
| 3 | Tu Tu | tutu@hotmail.com | 2009-05-05 |
| 4 | Goo Goo | googoo@gmail.com | 2010-06-10 |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)

```

LIMIT 1 ဆိုတော့ အပေါ်ဆုံး တစ်ခုတည်း ပြောင်းလဲသွားတာ သတိထားမိပါလိမ့်မယ်။ ကျွန်တော်က Date တွေ တူသွားအောင် ဒုတိယ Row က Date ကို အပေါ် Row နဲ့ တူအောင် ပြောင်းလိုက်ပါတယ်။ ပြီးရင် UPDATE လုပ်ကြည့်ပါတယ်။ WHERE Condition အရ date အတွက် ကိုက်ညီတာ ၂ ခုရှိပေမယ့် LIMIT 1 ဖြစ်တာကြောင့် အပေါ်ဆုံး တစ်ကြောင်းတည်း ပြောင်းသွားပါတယ်။

Data များကို ဖျက်ခြင်း

INSERT, SELECT နဲ့ UPDATE တွေ အားလုံး ပြီးသွားပါပြီ။ ဒီတစ်ခါ ဆက်လေ့လာမယ့် အပိုင်းက DELETE ဆိုတဲ့ အပိုင်းပါ။ Data တွေ ဖျက်တဲ့ အပိုင်းပေါ့။ Table ကို ဖျက်တဲ့အပိုင်းက အပေါ်မှာ လေ့ကျင့်ခဲ့ပြီးပါပြီ။ ခု ဖျက်မှာက အထဲက Data တွေပါ မဖျက်ခင်မှာ သေသေချာချာ စဉ်းစားဖို့ တော့ လိုပါလိမ့်မယ်။

1. DELETE FROM students WHERE student_name = 'Goo Goo';

```

-> WHERE date = '2009-12-12' LIMIT 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT * FROM students WHERE student_id IN(1,4);
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 1 | Aung Hla | aungaung@hotmail.com | 2009-12-18 |
| 4 | Goo Goo | googoo@gmail.com | 2010-06-10 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM students;
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 1 | Aung Hla | aungaung@hotmail.com | 2009-12-18 |
| 2 | Maung Maung | maungmaung@hotmail.com | 2009-12-12 |
| 3 | Tu Tu | tutu@hotmail.com | 2009-05-05 |
| 4 | Goo Goo | googoo@gmail.com | 2010-06-10 |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> DELETE FROM students WHERE student_name = 'Goo Goo';
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM students WHERE student_name = 'Goo Goo';
Empty set (0.00 sec)

mysql>

```

Data Table တွဲ Data Row တွေ ကို WHERE နဲ့ သုံးပြီး ဖျက်ပါတယ်။ ဒါလည်း UPDATE မှာ သုံးတဲ့ WHERE နဲ့ တူတူပါပဲ။

Table ထဲက Data များ အားလုံးကို ရှင်းလင်းခြင်း

TRUNCATE ဆိုတာကို မသုံးပဲ DELETE FROM students; ဆိုရင်လည်း အားလုံး ဖျက်သွားပါတယ်။ ဒါပေမယ့် TRUNCATE ကတော့ ပိုပြီး ပြည့်စုံပါတယ်။ AUTO_INCREMENT တွေအတွက် ဆိုရင် TRUNCATE နဲ့ ရှင်းထားရင် 1 က ပြန်စမှာ ဖြစ်ပေမယ့် DELETE ကတော့ အဲဒါမျိုး မရပါဘူး။ Counter ကတော့ နောက်ဆုံး Value ကနေ ဆက်သွားနေမှာပါ။


```
D:\xampp\mysql\bin\mysql.exe
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 1 | Aung Hla | aungaung@hotmail.com | 2009-12-18 |
| 4 | Goo Goo | googoo@gmail.com | 2010-06-10 |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> SELECT * FROM students;
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 1 | Aung Hla | aungaung@hotmail.com | 2009-12-18 |
| 2 | Maung Maung | maungmaung@hotmail.com | 2009-12-12 |
| 3 | Tu Tu | tutu@hotmail.com | 2009-05-05 |
| 4 | Goo Goo | googoo@gmail.com | 2010-06-10 |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> DELETE FROM students WHERE student_name = 'Goo Goo';
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM students WHERE student_name = 'Goo Goo';
Empty set (0.00 sec)

mysql> TRUNCATE TABLE students;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM students;
Empty set (0.00 sec)

mysql>
```

String Value တွေ ဖြစ်တဲ့ ‘ စတာတွေကို Data ထဲကို ထည့်ချင်တယ် ဆိုရင် ‘\’ ဆိုတာကို သုံးပြီး ထည့်နိုင်ပါတယ်။

```
D:\xampp\mysql\bin\mysql.exe
+-----+-----+-----+-----+
| 1 | Aung Hla | aungaung@hotmail.com | 2009-12-18 |
| 2 | Maung Maung | maungmaung@hotmail.com | 2009-12-12 |
| 3 | Tu Tu | tutu@hotmail.com | 2009-05-05 |
| 4 | Goo Goo | googoo@gmail.com | 2010-06-10 |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> DELETE FROM students WHERE student_name = 'Goo Goo';
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM students WHERE student_name = 'Goo Goo';
Empty set (0.00 sec)

mysql> TRUNCATE TABLE students;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM students;
Empty set (0.00 sec)

mysql> INSERT INTO students SET student_name = 'O\'Reilly';
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM students WHERE student_name = 'O\'Reilly';
+-----+-----+-----+-----+
| student_id | student_name | email | date |
+-----+-----+-----+-----+
| 1 | O'Reilly | NULL | NULL |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>
```

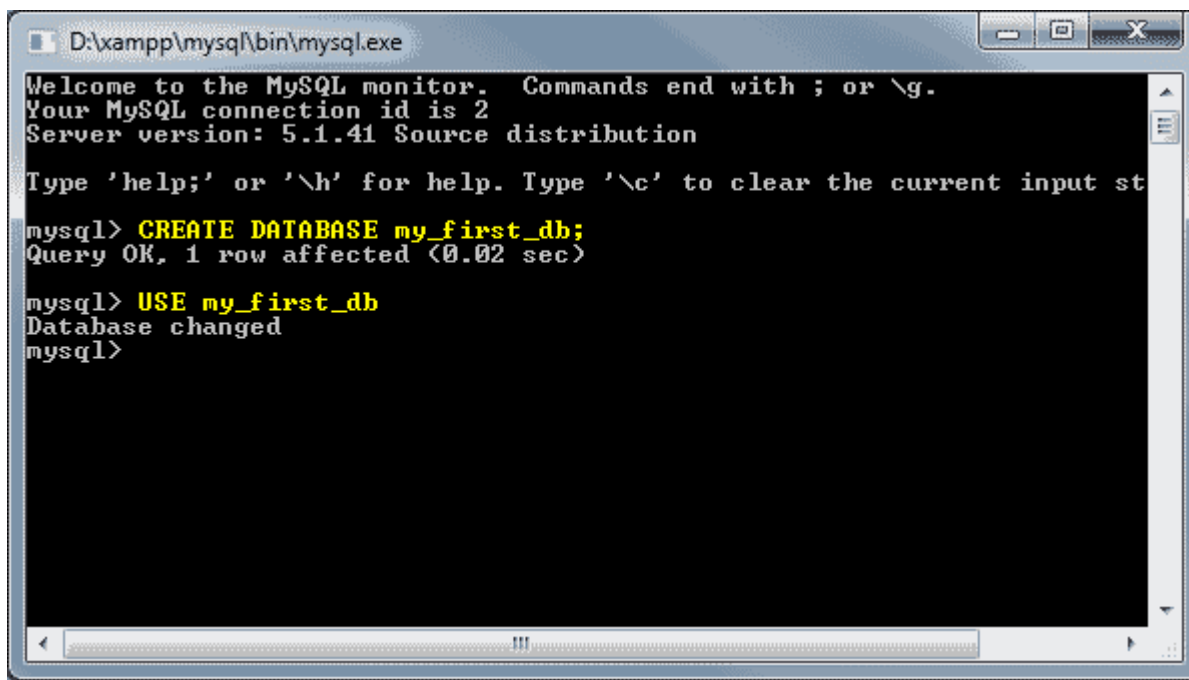
O'Reilly ဆိုတာကို ထည့်ချင်တဲ့အခါ ‘ ဆိုတဲ့ Character ပါနေပါတယ်။ အဲဒီအတွက် ရှေ့မှာ \ ထည့်ပေးဖို့ လိုပါတယ်။ ဒီနေရာမှာတော့ Escape Character အတွက် သေသေချာချာ ပြင်ဆင်ဖို့ လိုပါလိမ့်မယ်။ mysql_real_escape_string() ဆိုတဲ့ function သုံးရင်သုံး၊ ဒါမှ မဟုတ် statements တွေရေးပြီး auto escape ဖြစ်အောင် လုပ်ဖို့ လိုပါတယ်။

ဒီလောက်ဆိုရင် MySQL မှ SQL အခြေခံများ ပထမအဆင့်အတွက် လုံလောက်သွားပါပြီ။

MySQL မှ SQL များ

အဲဒီ အပိုင်းတွေမှာ ခက်သလားဟု ဒေတာဘေ့(စ်) ပေါ့ပေါ့ပါးပါး SQL များကတော့ ဒုပိုင်း လိုနေပါသေးတယ်။ အဲဒါလည်း မကြာခင် လာပါလိမ့်မယ် ခင်ဗျာ။

MySQL မှ SQL အခြေခံများဆိုတဲ့ ပိုစ်မှာ ကျွန်တော်တို့ MySQL Console ကို ဘယ်လို ခေါ်သလဲဆိုတာ လေ့လာပြီး ဖြစ်ပါတယ်။ MySQL Console ကို ခေါ်လိုက်ပါ။ အဲဒီလို ခေါ်လိုက်ပြီးပြီဆိုရင် အရင်ကတည်းက ဆောက်ထားပြီးသား my_first_db ဆိုတာ ရှိပါလိမ့်မယ်။ မရှိဘူးဆိုရင်လည်း CREATE ကို သုံးပြီး ဆောက်လိုက်ပါ။



Database Indexes

Indexes (or keys) ဆိုတာတွေကို Database မှာ သုံးရခြင်း အဓိက ရှည်ရှည်ချက်ကတော့ Data တွေကို ပြန်ခေါ်တဲ့ နေရာမှာ မြန်မြန် ဆန်ဆန် ခေါ်လိုရအောင် ဆိုတဲ့ ရှည်ရှည်ချက်ပဲ ဖြစ်ပါတယ်။ Indexes (or keys) တွေဟာ Column Basic နဲ့ သွားပါတယ်။ Column တစ်ခုကို index ထားလိုက်တာနဲ့ MySQL က Lookup index ဆောက်လိုက်ပါတယ်။ အဲဒီလို ဆောက်လိုက်ခြင်း အားဖြင့် ရှာတဲ့နေရာမှာ ပိုပြီး မြန်ဆန်သွားစေပါတယ်။

အားနည်းချက် အနေနဲ့ကတော့ Column မှာ Data တွေ အပြောင်းအလဲ ရှိတိုင်း Index ကို ပြန်ပြန် ဆောက်နေ ရတာပါပဲ။ အဲဒီတော့ ကိုယ့်အနေနဲ့ အမြဲတမ်း update လုပ်နေရမယ်၊ insert လုပ်နေရမယ်၊ remove လုပ်နေရမယ် ဆိုရင် performance ပိုင်းမှာ ထိခိုက်လာနိုင်ပါတယ်။

Indexing လုပ်ဖို့ လိုအပ်တဲ့ အကောင်းဆုံး အကြောင်းပြချက်တွေကတော့

- Table တိုင်း PRIMARY KEY တစ်ခု ရှိသင့်ပါတယ်
- အကယ်၍ column တွေမှာ ရှိတဲ့ Data တွေ အနေနဲ့ Unique ဖြစ်ဖို့ တစ်ခုနဲ့ တစ်ခုလုံး မတူဖို့ လိုတယ်ဆိုရင် Unique Index ထားသင့်ပါတယ်
- Column တစ်ခုက Data Value တွေကို အမြဲတမ်း ရှာနေဖို့ လိုတယ်ဆိုရင် Index ထားသင့်ပါတယ်
- အကယ်၍ ကိုယ်သုံးနေတဲ့ Table ဟာ အခြား Table တွေနဲ့ ချိတ်ဆက်နေတယ်ဆိုရင် FOREIGN KEY ဆိုတာ ထားသင့်ပါတယ် ဒါမှမဟုတ် Regular Index လုပ်ထားသင့်ပါတယ်။

PRIMARY KEY

Table တိုင်း Table တိုင်းမှာ PRIMARY KEY တစ်ခု ရှိသင့်ပါတယ်။ အများအားဖြင့်တော့ INT Data Type နဲ့ AUTO_INCREMENT ထားတတ်ပါတယ်။

ရှေ့ပိုင်း MySQL မှာ SQL အခြေခံများမှာ ဆောက်ခဲ့တဲ့ Table အရ student_id ဆိုတာ PRIMARY KEY ပါပဲ။ အဲဒီလို သတ်မှတ် ပေးလိုက်ခြင်း အားဖြင့် Student တစ်ယောက်ခြင်းစီကို id numbers အလိုက် ရလို့ ရသွားစေပါတယ်။

PRIMARY KEY မှာ Data တွေ သိမ်းတော့မယ်ဆိုရင် Unique (တစ်ခုနဲ့ တစ်ခု လုံးမထပ်ဖို့) လိုပါတယ်။ နောက် TABLE တစ်ခုမှာ PRIMARY KEY တစ်ခုထက် ပိုပြီး ရှိလို့ မရပါဘူး။

ကဲ Table တွေ စမ်းဆောက်ကြည့်ရအောင် MySQL Console တော့ ခေါ်ထားပြီးသား ဖြစ်မယ် ထင်ပါတယ်။

1. `CREATE TABLE states (`
2. `id INT AUTO_INCREMENT PRIMARY KEY,`
3. `name VARCHAR(20)`
4. `);`

နောက်တစ်နည်းလည်း ရှိပါသေးတယ်။

1. `CREATE TABLE states (`
2. `id INT AUTO_INCREMENT,`
3. `name VARCHAR(20),`
4. `PRIMARY KEY (id)`
5. `);`

UNIQUE

Data Column တစ်ခု အတွင်းမှာ ရှိတဲ့ Data Value တွေအနေနဲ့ UNIQUE ဖြစ်ဖို့ လိုတယ်လို့ သတ်မှတ်ရင် UNIQUE Index ထည့်ပေးဖို့ လိုပါလိမ့်မယ်။

1. `CREATE TABLE states (`
2. `id INT AUTO_INCREMENT,`
3. `name VARCHAR(20),`
4. `PRIMARY KEY (id),`
5. `UNIQUE (name)`
6. `);`

ပုံမှန် Default အားဖြင့် index တွေက Column Name တွေရဲ့ နောက်မှာ သတ်မှတ်ပေးလေ့ ရှိပါတယ်။ အဲဒီလို မဟုတ်ပဲ ရှေ့မှာ သတ်မှတ်ချင်တယ် ဆိုရင်လည်း ရပါတယ်။

1. `CREATE TABLE states (`
2. `id INT AUTO_INCREMENT,`
3. `name VARCHAR(20),`
4. `PRIMARY KEY (id),`
5. `UNIQUE state_name (name)`
6. `);`

index ကို name အစား state_name အနေနဲ့ သတ်မှတ် ထားပါတယ်။

INDEX

ဒီတစ်ခါတော့ state တစ်ခု အနေနဲ့ ပထမဦးဆုံး ပါဝင်လာတဲ့ နှစ်တွေကို Column အနေနဲ့ သိမ်းချင်တယ် ဆိုရင်

1. `CREATE TABLE states (`
2. `id INT AUTO_INCREMENT,`
3. `name VARCHAR(20),`
4. `join_year INT,`
5. `PRIMARY KEY (id),`
6. `UNIQUE (name),`
7. `INDEX (join_year)`
8. `);`

join_year ဆိုတာ ပေါင်းထည့်ထားသလို index လည်း လုပ်ထားပါတယ်။ ဒါပေမယ့် Unique ဖြစ်ဖို့တော့ မသတ်မှတ်ထားပါဘူး။

INDEX အစား KEY ဆိုတာနဲ့လည်း သုံးလို့ရပါသေးတယ်။

1. `CREATE TABLE states (`
2. `id INT AUTO_INCREMENT,`
3. `name VARCHAR(20),`
4. `join_year INT,`
5. `PRIMARY KEY (id),`
6. `UNIQUE (name),`
7. `KEY (join_year)`
8. `);`

စွမ်းဆောင်ရည် ပိုင်းဆိုင်ရာ

အပေါ်မှာ ပြောခဲ့သလိုပါပဲ indexes တွေက INSERT တို့ UPDATE တို့ရဲ့ စွမ်းဆောင်ရည်ကို ကျဆင်းစေပါတယ်။ ဘာဖြစ်လို့လည်း ဆိုရင် data တွေကို table ထဲ ထည့်တိုင်း ထည့်တိုင်း index data တွေကလည်း update ကို auto လိုက်လုပ်နေပါတယ်။ အဲဒီလို လုပ်တော့ MySQL Server အနေနဲ့ အလုပ်တွေ ပိုလုပ်ဖို့ လိုလာပါတယ်။ index တွေ ထည့်တာဟာ SELECT အတွက်ကောင်းပေမယ့် INSERT တို့ UPDATE တို့အတွက်တော့ မကောင်းပါဘူး။ ဒါကြောင့် indexes တွေ ထည့်မယ်ဆိုရင် ထည့်သင့် မထည့်သင့် သေသေချာချာ စဉ်းစားသင့်ပါတယ်။

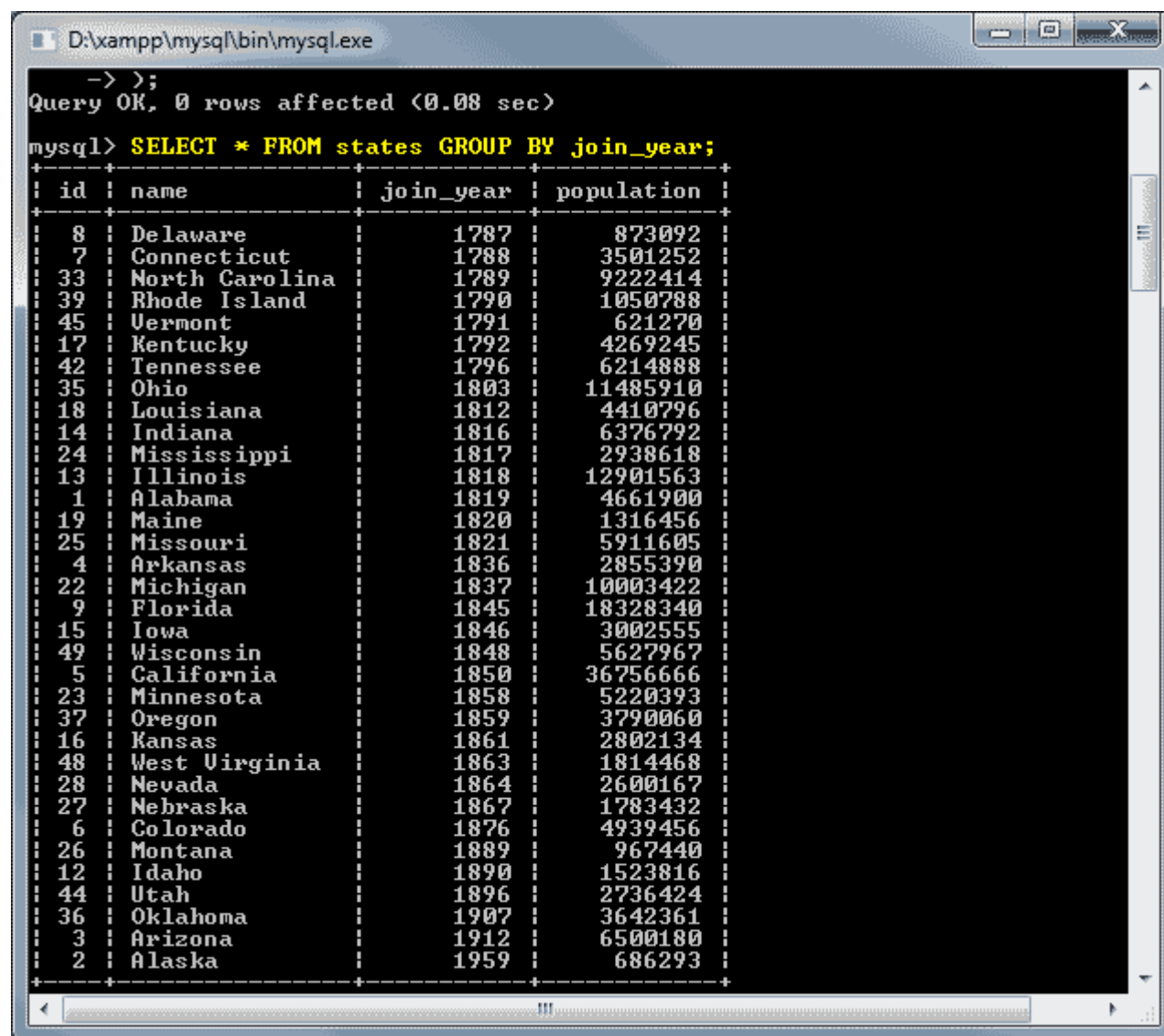
Sample Table

ဒီထက်ပိုပြီး Queries တွေကို သွားနိုင်ဖို့အတွက် sample table တစ်ခုနဲ့ sample data တွေ ထည့်ပေးဖို့ လိုပါမယ်။ ဒါကိုတော့ MySQL Console မှာ ထည့်ပေးဖို့ လိုပါမယ်။

```
1. CREATE TABLE states (  
2.     id INT AUTO_INCREMENT,  
3.     name VARCHAR(20),  
4.     join_year INT,  
5.     population INT,  
6.     PRIMARY KEY (id),  
7.     UNIQUE (name),  
8.     KEY (join_year)  
9. );  
10.  
11.INSERT INTO states VALUES  
12.(1, 'Alabama', 1819, 4661900),  
13.(2, 'Alaska', 1959, 686293),  
14.(3, 'Arizona', 1912, 6500180),  
15.(4, 'Arkansas', 1836, 2855390),  
16.(5, 'California', 1850, 36756666),  
17.(6, 'Colorado', 1876, 4939456),  
18.(7, 'Connecticut', 1788, 3501252),  
19.(8, 'Delaware', 1787, 873092),  
20.(9, 'Florida', 1845, 18328340),  
21.(10, 'Georgia', 1788, 9685744),  
22.(11, 'Hawaii', 1959, 1288198),  
23.(12, 'Idaho', 1890, 1523816),  
24.(13, 'Illinois', 1818, 12901563),  
25.(14, 'Indiana', 1816, 6376792),  
26.(15, 'Iowa', 1846, 3002555),  
27.(16, 'Kansas', 1861, 2802134),  
28.(17, 'Kentucky', 1792, 4269245),  
29.(18, 'Louisiana', 1812, 4410796),  
30.(19, 'Maine', 1820, 1316456),  
31.(20, 'Maryland', 1788, 5633597),  
32.(21, 'Massachusetts', 1788, 6497967),  
33.(22, 'Michigan', 1837, 10003422),  
34.(23, 'Minnesota', 1858, 5220393),  
35.(24, 'Mississippi', 1817, 2938618),  
36.(25, 'Missouri', 1821, 5911605),  
37.(26, 'Montana', 1889, 967440),  
38.(27, 'Nebraska', 1867, 1783432),  
39.(28, 'Nevada', 1864, 2600167),  
40.(29, 'New Hampshire', 1788, 1315809),  
41.(30, 'New Jersey', 1787, 8682661),  
42.(31, 'New Mexico', 1912, 1984356),  
43.(32, 'New York', 1788, 19490297),  
44.(33, 'North Carolina', 1789, 9222414),  
45.(34, 'North Dakota', 1889, 641481),  
46.(35, 'Ohio', 1803, 11485910),  
47.(36, 'Oklahoma', 1907, 3642361),  
48.(37, 'Oregon', 1859, 3790060),  
49.(38, 'Pennsylvania', 1787, 12448279),  
50.(39, 'Rhode Island', 1790, 1050788),  
51.(40, 'South Carolina', 1788, 4479800),  
52.(41, 'South Dakota', 1889, 804194),  
53.(42, 'Tennessee', 1796, 6214888),  
54.(43, 'Texas', 1845, 24326974),  
55.(44, 'Utah', 1896, 2736424),  
56.(45, 'Vermont', 1791, 621270),  
57.(46, 'Virginia', 1788, 7769089),  
58.(47, 'Washington', 1889, 6549224),  
59.(48, 'West Virginia', 1863, 1814468),  
60.(49, 'Wisconsin', 1848, 5627967),  
61.(50, 'Wyoming', 1890, 532668);
```

GROUP BY (Data များကို Group ခွဲခြင်း)

GROUP BY ကို အသုံးပြုတဲ့ ပုံစံကို လေ့လာကြည့်ပါမယ်။

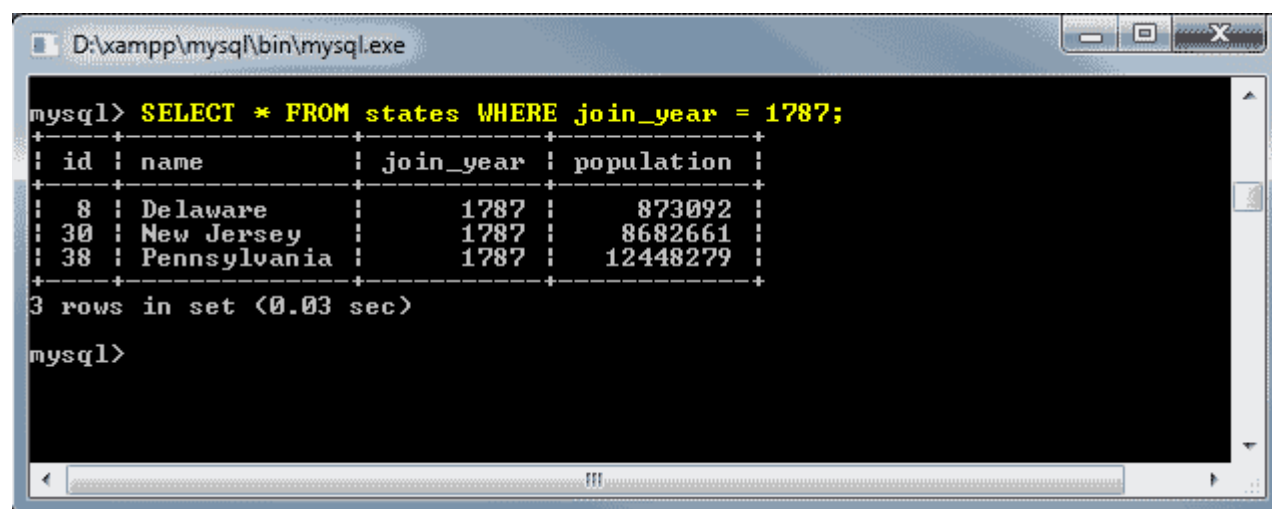


```
D:\xampp\mysql\bin\mysql.exe
-> >;
Query OK, 0 rows affected (0.08 sec)

mysql> SELECT * FROM states GROUP BY join_year;
+----+-----+-----+-----+
| id | name       | join_year | population |
+----+-----+-----+-----+
| 8  | Delaware  | 1787      | 873092     |
| 7  | Connecticut | 1788      | 3501252    |
| 33 | North Carolina | 1789      | 9222414    |
| 39 | Rhode Island | 1790      | 1050788    |
| 45 | Vermont    | 1791      | 621270     |
| 17 | Kentucky   | 1792      | 4269245    |
| 42 | Tennessee  | 1796      | 6214888    |
| 35 | Ohio        | 1803      | 11485910   |
| 18 | Louisiana   | 1812      | 4410796    |
| 14 | Indiana     | 1816      | 6376792    |
| 24 | Mississippi | 1817      | 2938618    |
| 13 | Illinois    | 1818      | 12901563   |
| 1  | Alabama     | 1819      | 4661900    |
| 19 | Maine       | 1820      | 1316456    |
| 25 | Missouri    | 1821      | 5911605    |
| 4  | Arkansas    | 1836      | 2855390    |
| 22 | Michigan    | 1837      | 10003422   |
| 9  | Florida     | 1845      | 18328340   |
| 15 | Iowa        | 1846      | 3002555    |
| 49 | Wisconsin   | 1848      | 5627967    |
| 5  | California   | 1850      | 36756666   |
| 23 | Minnesota   | 1858      | 5220393    |
| 37 | Oregon      | 1859      | 3790060    |
| 16 | Kansas      | 1861      | 2802134    |
| 48 | West Virginia | 1863      | 1814468    |
| 28 | Nevada      | 1864      | 2600167    |
| 27 | Nebraska     | 1867      | 1783432    |
| 6  | Colorado     | 1876      | 4939456    |
| 26 | Montana     | 1889      | 967440     |
| 12 | Idaho        | 1890      | 1523816    |
| 44 | Utah         | 1896      | 2736424    |
| 36 | Oklahoma     | 1907      | 3642361    |
| 3  | Arizona      | 1912      | 6500180    |
| 2  | Alaska       | 1959      | 686293     |
+----+-----+-----+-----+
```

ရှိတာက row 50 ရှိပေမယ့် တကယ်ပြောရင် 34 rows ပဲ ပြပါတယ်။ Group By လုပ်လိုက်တဲ့ အတွက် ထပ်နေတဲ့ year တွေကို ဖယ်ထုတ် လိုက်ပါတယ်။ ဒါကြောင့် 34 rows ပဲ ရှိတော့တာပါ။

ဥပမာ 1878 ခုနှစ်မှာ state ၃ ခုရှိနေပါတယ်။ SELECT လုပ်ကြည့်ရအောင်



```
D:\xampp\mysql\bin\mysql.exe

mysql> SELECT * FROM states WHERE join_year = 1787;
+----+-----+-----+-----+
| id | name       | join_year | population |
+----+-----+-----+-----+
| 8  | Delaware  | 1787      | 873092     |
| 30 | New Jersey | 1787      | 8682661    |
| 38 | Pennsylvania | 1787      | 12448279   |
+----+-----+-----+-----+

3 rows in set (0.03 sec)

mysql>
```

1787 ခုနှစ်မှာ state ၃ ခုရှိနေပါတယ်။ Delaware က ထိပ်ဆုံးမှာ ဖြစ်နေတော့ GROUP BY လုပ်ထားတော့ ထိပ်ဆုံးက တစ်ခုပဲ ပြပါ တော့တယ်။ GROUP BY ကို ဒီအတိုင်းပဲ သုံးတယ်ဆိုရင် သိပ်အဓိပ္ပာယ် မရှိသလိုပါပဲ။ ဒါကြောင့် တစ်ခြား Function တွေနဲ့ တွဲသုံးဖို့ လိုပါတယ်။ နောက်တစ်ပိုင်းမှာ အဲဒါကို ဆက်လေ့လာပါမယ်။

MySQL မှ SQL များ (၂)

COUNT(*): Counting Rows

ဒီ အသုံးကတော့ GROUP BY နဲ့ အမြဲတမ်းတွဲသုံးပါတယ်။ Group တစ်ခုခြင်စီမှာ ရှိတဲ့ Rows အရေအတွက် ကို ဖော်ပြပေးပါတယ်။

အောက်က ဥပမာအရဆိုရင် join_year ထပ်နေတဲ့ states အရေအတွက်ကို ရှာလို့ရပါတယ်။


```
D:\xampp\mysql\bin\mysql.exe
mysql> USE my_first_db;
Database changed
mysql> SELECT COUNT(*), join_year FROM states GROUP BY join_year;
+-----+-----+
| COUNT(*) | join_year |
+-----+-----+
| 3        | 1787      |
| 8        | 1788      |
| 1        | 1789      |
| 1        | 1790      |
| 1        | 1791      |
| 1        | 1792      |
| 1        | 1796      |
| 1        | 1803      |
| 1        | 1812      |
| 1        | 1816      |
| 1        | 1817      |
| 1        | 1818      |
| 1        | 1819      |
| 1        | 1820      |
| 1        | 1821      |
| 1        | 1836      |
| 1        | 1837      |
| 2        | 1845      |
| 1        | 1846      |
| 1        | 1848      |
| 1        | 1850      |
| 1        | 1858      |
| 1        | 1859      |
| 1        | 1861      |
| 1        | 1863      |
| 1        | 1864      |
| 1        | 1867      |
| 1        | 1876      |
| 4        | 1889      |
| 2        | 1890      |
| 1        | 1896      |
| 1        | 1907      |
| 2        | 1912      |
| 2        | 1959      |
+-----+-----+
34 rows in set (0.13 sec)
```

Grouping Everything

GROUP BY ကို မပါဘဲ COUNT(*) ဆိုတာမျိုးကို တန်သုံးမယ်ဆိုရင် ရှိသမျှ အထဲမှာ ရှိတဲ့ Row တွေရဲ့ Count ကို ပြပါလိမ့်မယ်။

```
D:\xampp\mysql\bin\mysql.exe
+-----+-----+
| 1        | 1896      |
| 1        | 1907      |
| 2        | 1912      |
| 2        | 1959      |
+-----+-----+
34 rows in set (0.13 sec)

mysql> SELECT COUNT(*) FROM states;
+-----+
| COUNT(*) |
+-----+
| 50        |
+-----+
1 row in set (0.03 sec)

mysql>
```

WHERE နဲ့ တွဲသုံးကြည့်မယ်ဆိုရင်

```
D:\xampp\mysql\bin\mysql.exe
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.1.41 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement

mysql> USE my_first_db;
Database changed
mysql> SELECT COUNT(*) FROM states WHERE join_year = 1787;
+-----+
| COUNT(*) |
+-----+
| 3        |
+-----+
1 row in set (0.09 sec)

mysql>
```

MIN(), MAX() and AVG()

MIN() ဆိုတော့ အနည်းဆုံး၊ MAX() ဆိုတော့ အများဆုံး၊ AVG() ဆိုတော့ ပျဉ်းမျှပေါ့


```
D:\xampp\mysql\bin\mysql.exe

Type 'help;' or '\h' for help. Type '\c' to clear the current input state

mysql> USE my_first_db;
Database changed
mysql> SELECT COUNT(*) FROM states WHERE join_year = 1787;
+-----+
| COUNT(*) |
+-----+
|        3 |
+-----+
1 row in set (0.09 sec)

mysql> SELECT MIN(population), MAX(population), AVG(population)
-> FROM states;
+-----+-----+-----+
| MIN(population) | MAX(population) | AVG(population) |
+-----+-----+-----+
|          532668 |          3675666 | 6069357.8200    |
+-----+-----+-----+
1 row in set (0.07 sec)

mysql>
```

GROUP_CONCAT()

ဒီ GROUP_CONCAT ကိုတော့ Group တစ်ခုခြင်းစီမှာ ရှိတဲ့ String တွေ အားလုံးကို ပေါင်းပြီး Single String တစ်ခုတည်း အနေနဲ့ ဖော်ပြပေးမှာ ဖြစ်ပါတယ်။ Separator တွေ ဘာတွေလည်း သတ်မှတ်လို့ရပါတယ်။ GROUP_CONCAT() ကို လေ့လာကြည့်ရအောင်

```
D:\xampp\mysql\bin\mysql.exe

mysql> SELECT GROUP_CONCAT(name SEPARATOR ', '), join_year
-> FROM states GROUP BY join_year;
+-----+-----+
| GROUP_CONCAT(name SEPARATOR ', ') | join_year |
+-----+-----+
| New Jersey, Pennsylvania, Delaware | 1787 |
| Connecticut, Maryland, New Hampshire, Massachusetts, Georgia, New York, South Carolina, Virginia | 1788 |
| North Carolina | 1789 |
| Rhode Island | 1790 |
| Vermont | 1791 |
| Kentucky | 1792 |
| Tennessee | 1796 |
| Ohio | 1803 |
| Louisiana | 1812 |
| Indiana | 1816 |
| Mississippi | 1817 |
| Illinois | 1818 |
| Alabama | 1819 |
| Maine | 1820 |
| Missouri | 1821 |
| Arkansas | 1836 |
| Michigan | 1837 |
| Florida, Texas | 1845 |
| Iowa | 1846 |
| Wisconsin | 1848 |
| California | 1850 |
| Minnesota | 1858 |
| Oregon | 1859 |
| Kansas | 1861 |
| West Virginia | 1863 |
| Nevada | 1864 |
| Nebraska |
```

ကြည့်ရတာ နည်းနည်းတော့ ရှုပ်သွားပါတယ်။ ဒါပေမယ့် တော်တော် အသုံးဝင်တဲ့ Command တစ်ခုပါပဲ။ ရေထားတဲ့ Code ကို မမြင်ရရင်

1. `SELECT GROUP_CONCAT(name SEPARATOR ', '), join_year`
2. `FROM states GROUP BY join_year;`

SUM()

ဒါကတော့ ပေါင်းလဒ်ရှာချင်တဲ့အခါမှာ သုံးပါတယ်။

```
D:\xampp\mysql\bin\mysql.exe

mysql> SELECT SUM(population) AS usa_population FROM states;
+-----+
| usa_population |
+-----+
| 303467891      |
+-----+
1 row in set (0.03 sec)

mysql> _
```

IF() & CASE: Control Flow

IF()

```
D:\xampp\mysql\bin\mysql.exe

+-----+
| usa_population |
+-----+
| 303467891      |
+-----+
1 row in set (0.03 sec)

mysql> SELECT IF(true, 'foo', 'bar');
+-----+
| IF(true, 'foo', 'bar') |
+-----+
| foo                     |
+-----+
1 row in set (2.08 sec)

mysql> SELECT IF(false, 'foo', 'bar');
+-----+
| IF(false, 'foo', 'bar') |
+-----+
| bar                     |
+-----+
1 row in set (0.00 sec)

mysql> _
```

အပေါ်မှာ စမ်းပြတာက နမူနာပါ။ ခုမှ တကယ်စမ်းကြည့်ပါမယ်။

1. `SELECT`
- 2.
3. `SUM(`
4. `IF(population > 5000000, 1, 0)`
5. `) AS big_states,`
- 6.
7. `SUM(`
8. `IF(population <= 5000000, 1, 0)`
9. `) AS small_states`
- 10.
11. `FROM states;`

၀၀၀၀ SUM() population 5 million ကျော်တဲ့ big states တွေ၊ ဒုတိယ SUM() ကတော့ small states တွေရဲ့ Counts

```
D:\xampp\mysql\bin\mysql.exe

+-----+
| bar |
+-----+
1 row in set (0.00 sec)

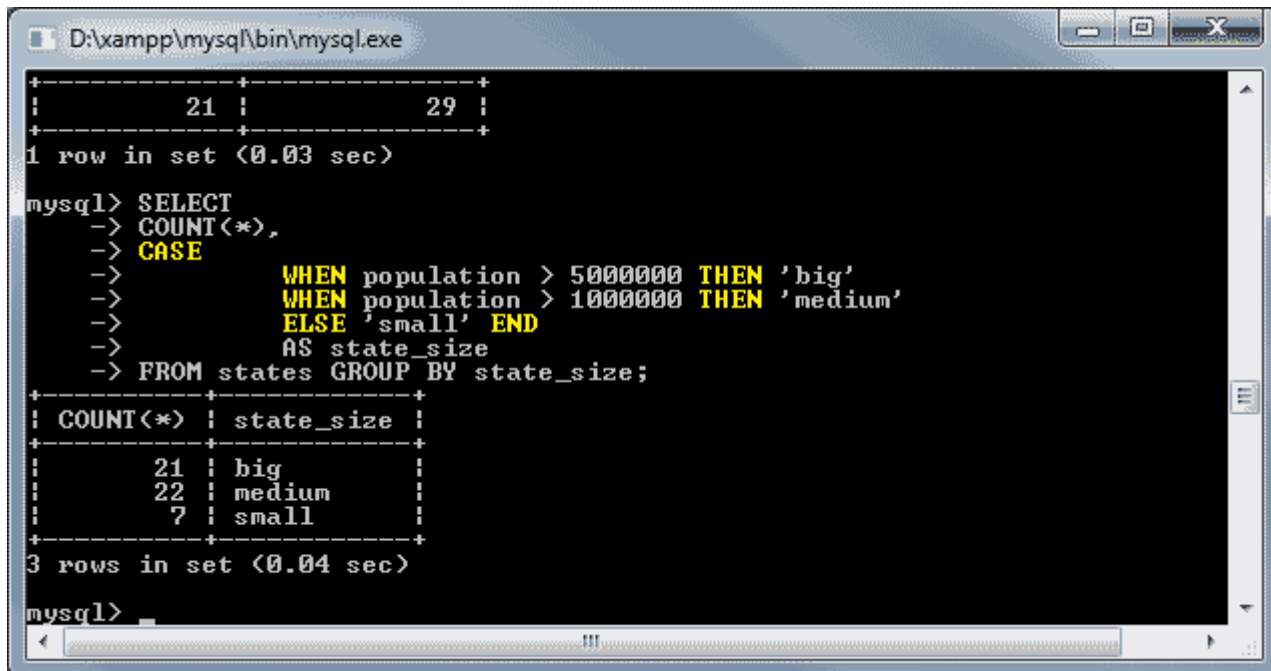
mysql> SELECT
-> SUM(
-> IF(population > 5000000, 1, 0)
-> ) AS big_states,
-> SUM(
-> IF(population <= 5000000, 1, 0)
-> ) AS small_states
-> FROM states;
+-----+-----+
| big_states | small_states |
+-----+-----+
| 21         | 29          |
+-----+-----+
1 row in set (0.03 sec)

mysql> _
```

CASE

CASE ဆိုတဲ့ အသုံးကတော့ switch-case statements နဲ့ တူမယ်ထင်ပါတယ်။

1. `SELECT`
2. `COUNT(*)`,
3. `CASE`
4. `WHEN population > 5000000 THEN 'big'`
5. `WHEN population > 1000000 THEN 'medium'`
6. `ELSE 'small' END`
7. `AS state_size`
8. `FROM states GROUP BY state_size;`



```
D:\xampp\mysql\bin\mysql.exe
+-----+-----+
| 21 | 29 |
+-----+-----+
1 row in set (0.03 sec)

mysql> SELECT
-> COUNT(*),
-> CASE
->     WHEN population > 5000000 THEN 'big'
->     WHEN population > 1000000 THEN 'medium'
->     ELSE 'small' END
-> AS state_size
-> FROM states GROUP BY state_size;
+-----+-----+
| COUNT(*) | state_size |
+-----+-----+
| 21 | big |
| 22 | medium |
| 7 | small |
+-----+-----+
3 rows in set (0.04 sec)

mysql>
```

MySQL မှ SQL များ (၃)

MySQL မှ SQL များ ဆိုတဲ့ အပိုင်းတွေကို အရင် MySQL မှာ SQL အခြေခံများရဲ့ အဆက်အနက်နဲ့ ရေးဖြစ်ခဲ့တာပါ။ အဓိက ရေးဖြစ်ရတဲ့ အကြောင်းက နောင် ကျွန်တော် ဆက်ရေးမယ့် အကြောင်း ဖြစ်တဲ့ PHP and MySQL ဆိုတဲ့ အပိုင်းကို သွားဖို့အတွက် ပြင်ဆင်တဲ့ အနေနဲ့ရေးနေတာ ဖြစ်ပါတယ်။ အဲဒီအပိုင်းကို လေ့လာဖို့ စိတ်ဝင်စားတဲ့ သူတွေ အနေနဲ့ PHP အကြောင်းတွေ ရေးနေတဲ့ ပိုစ်တွေကိုလည်း ဖတ်ထားဖို့ လိုမှာ ဖြစ်ပါတယ်။ ဒီတော့ မပြီးသေးတဲ့ MySQL မှ SQL များဆိုတဲ့ အပိုင်းတွေကို ရှေ့ဆက်ရမှာ ဖြစ်ပါတယ်။ ဒါတွေအားလုံးကို ကျွန်တော် tutsplus ကနေ ကိုယ်ကိုယ်တိုင် စမ်းသပ် Screenshot ဖမ်းပြီး ပြန်ရေးပေးနေတာပါ။ နောက်တစ်ချိန် PHP နဲ့ MySQL ကို တွဲပြီး လေ့လာဖို့အတွက် Foundation ချနေတယ် ဆိုပါတော့ဗျာ။

HAVING: Conditions on Hidden Fields

HAVING ဆိုတာကို MySQL က SQL မှာ Hidden Field တွေကို စစ်တဲ့နေရာမှာ သုံးပါတယ်။ အများအားဖြင့် GROUP BY ဆိုတဲ့ အသုံးနဲ့ တွဲသုံးလေ့ရှိပါတယ်။ ကဲ တစ်ဆင့်ခြင်း စမ်းကြည့်ရအောင်ဗျာ။

1. `SELECT COUNT(*), join_year FROM states GROUP BY join_year;`

ဒါဆိုရင်တော့ ထုံးစံအတိုင်း 34 rows ရပါလိမ့်မယ်။

```

D:\xampp\mysql\bin\mysql.exe
Database changed
mysql> SELECT COUNT(*), join_year FROM states GROUP BY join_year;
+-----+-----+
| COUNT(*) | join_year |
+-----+-----+
| 3        | 1787      |
| 8        | 1788      |
| 1        | 1789      |
| 1        | 1790      |
| 1        | 1791      |
| 1        | 1792      |
| 1        | 1796      |
| 1        | 1803      |
| 1        | 1812      |
| 1        | 1816      |
| 1        | 1817      |
| 1        | 1818      |
| 1        | 1819      |
| 1        | 1820      |
| 1        | 1821      |
| 1        | 1836      |
| 1        | 1837      |
| 2        | 1845      |
| 1        | 1846      |
| 1        | 1848      |
| 1        | 1850      |
| 1        | 1858      |
| 1        | 1859      |
| 1        | 1861      |
| 1        | 1863      |
| 1        | 1864      |
| 1        | 1867      |
| 1        | 1876      |
| 4        | 1889      |
| 2        | 1890      |
| 1        | 1896      |
| 1        | 1907      |
| 2        | 1912      |
| 2        | 1959      |
+-----+-----+
34 rows in set (0.10 sec)

```

အကယ်၍ အဲဒီအပေါ်က 34 Rows ထဲက COUNT() 1 ထက်ကြီးတာတွေကို ကြည့်ချင်တယ်ဆိုရင် WHERE ကိုသုံးလို့ အဆင်မပြေတော့ပါဘူး။

```

D:\xampp\mysql\bin\mysql.exe
+-----+-----+
| 1        | 1864      |
| 1        | 1867      |
| 1        | 1876      |
| 4        | 1889      |
| 2        | 1890      |
| 1        | 1896      |
| 1        | 1907      |
| 2        | 1912      |
| 2        | 1959      |
+-----+-----+
34 rows in set (0.10 sec)

mysql> SELECT COUNT(*), join_year FROM states WHERE COUNT(*) > 1
-> GROUP BY join_year;
ERROR 1111 (HY000): Invalid use of group function
mysql> _

```

အဲဒီလို အခြေအနေမှာ ဆိုရင် HAVING ကို သုံးဖို့ လိုပါတယ်။

```

D:\xampp\mysql\bin\mysql.exe
mysql> SELECT COUNT(*), join_year FROM states WHERE COUNT(*) > 1
-> GROUP BY join_year;
ERROR 1111 (HY000): Invalid use of group function
mysql> SELECT COUNT(*), join_year FROM states
-> GROUP BY join_year
-> HAVING COUNT(*) > 1;
+-----+-----+
| COUNT(*) | join_year |
+-----+-----+
| 3        | 1787      |
| 8        | 1788      |
| 2        | 1845      |
| 4        | 1889      |
| 2        | 1890      |
| 2        | 1912      |
| 2        | 1959      |
+-----+-----+
7 rows in set (0.00 sec)

mysql>

```

ဒီစနစ်က တစ်ခြား Database System တွေ ရှိမရှိတော့ မသေချာပါဘူး။

Subqueries

Query တစ်ခု အတွင်းမှာ နောက်ထပ် Query တစ်ခုအနေနဲ့ ရေးချင်တယ်ဆိုရင်လည်း ရေးလို့ရပါတယ်။ ဒါလည်း အင်မတန် အသုံးဝင်ပါတယ်။

လူဦးရေအများဆုံး State ကို ရှာကြည့်ချင်တယ်ဆိုရင်

1. `SELECT * FROM states WHERE population = (`
2. `SELECT MAX(population) FROM states`
3. `);`

```

D:\xampp\mysql\bin\mysql.exe
+-----+
| id | name | join_year | population |
+-----+
| 3 | | 1787 | |
| 8 | | 1788 | |
| 2 | | 1845 | |
| 4 | | 1889 | |
| 2 | | 1890 | |
| 2 | | 1912 | |
| 2 | | 1959 | |
+-----+
7 rows in set (0.00 sec)

mysql> SELECT * FROM states WHERE population = (
-> SELECT MAX(population) FROM states
-> );
+-----+
| id | name | join_year | population |
+-----+
| 5 | California | 1850 | 36756666 |
+-----+
1 row in set (0.00 sec)

mysql>

```

အတွင်းမှာ ရှိတဲ့ Query က အမြင့်ဆုံး Population အရေအတွက်ကို ထုတ်ပေးပြီး အပြင်ဖက်က Query ကတော့ အဲဒီ အမြင့်ဆုံး အရေအတွက်နဲ့ ကိုက်တဲ့ အချက်အလက်ကို ထပ်ရှာပါတယ်။ ဒါပေမယ့် ဒီလိုရေးလို့ မရဘူးလားလို့ စေ့က တက်စရာ ရှိပါတယ်။

```

D:\xampp\mysql\bin\mysql.exe
7 rows in set (0.00 sec)

mysql> SELECT * FROM states WHERE population = (
-> SELECT MAX(population) FROM states
-> );
+-----+
| id | name | join_year | population |
+-----+
| 5 | California | 1850 | 36756666 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM states ORDER BY population DESC LIMIT 1;
+-----+
| id | name | join_year | population |
+-----+
| 5 | California | 1850 | 36756666 |
+-----+
1 row in set (0.03 sec)

mysql>

```

ရုတ်တရက်ကြည့်လိုက်ရင်တော့ အဖြေနှစ်ခုက တူနေပါတယ်။ ဒါပေမယ့်

1. `SELECT * FROM states WHERE join_year = (`
2. `SELECT MAX(join_year) FROM states`
3. `);`

```

D:\xampp\mysql\bin\mysql.exe
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.1.41 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement

mysql> USE my_first_db;
Database changed
mysql> SELECT * FROM states WHERE join_year = (
-> SELECT MAX(join_year) FROM states
-> );
+-----+
| id | name | join_year | population |
+-----+
| 2 | Alaska | 1959 | 686293 |
| 11 | Hawaii | 1959 | 1288198 |
+-----+
2 rows in set (0.12 sec)

mysql>

```

ORDER BY ... LIMIT 1 နဲ့ ဆက်ပြီး ရေးကြည့်ရအောင်


```
D:\xampp\mysql\bin\mysql.exe

mysql> USE my_first_db;
Database changed
mysql> SELECT * FROM states WHERE join_year = <
-> SELECT MAX(join_year) FROM states
-> >;
+-----+-----+-----+-----+
| id | name | join_year | population |
+-----+-----+-----+-----+
| 2 | Alaska | 1959 | 686293 |
| 11 | Hawaii | 1959 | 1288198 |
+-----+-----+-----+-----+
2 rows in set (0.12 sec)

mysql> SELECT * FROM states ORDER BY join_year DESC LIMIT 1;
+-----+-----+-----+-----+
| id | name | join_year | population |
+-----+-----+-----+-----+
| 11 | Hawaii | 1959 | 1288198 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>
```

အပေါ်မှာတော့ တစ်ခုတည်း ဖြတ်သွားတာ တွေ့ရပါလိမ့်မယ်။

IN()

တစ်ခါတစ်လေမှာ အတွင်းဖက် Query ကနေ တစ်ခုထက်ပိုတဲ့ Result တွေအတွက် စစ်ချင်တယ်ဆိုရင် = နဲ့ သုံးလို့ မရတော့ပါဘူး။ အဲဒီလို အခြေအနေအတွက် IN ဆိုတာကို သုံးရပါတယ်။

1. `SELECT * FROM states WHERE join_year IN (`
2. `SELECT join_year FROM states`
3. `GROUP BY join_year`
4. `HAVING COUNT(*) > 1`
5. `) ORDER BY join_year;`

```
D:\xampp\mysql\bin\mysql.exe

1 row in set (0.00 sec)

mysql> SELECT * FROM states WHERE join_year IN <
-> SELECT join_year FROM states
-> GROUP BY join_year
-> HAVING COUNT(*) > 1
-> > ORDER BY join_year;
+-----+-----+-----+-----+
| id | name | join_year | population |
+-----+-----+-----+-----+
| 38 | Pennsylvania | 1787 | 12448279 |
| 8 | Delaware | 1787 | 873092 |
| 30 | New Jersey | 1787 | 8682661 |
| 32 | New York | 1788 | 19490297 |
| 40 | South Carolina | 1788 | 4479800 |
| 21 | Massachusetts | 1788 | 6497967 |
| 20 | Maryland | 1788 | 5633597 |
| 29 | New Hampshire | 1788 | 1315809 |
| 46 | Virginia | 1788 | 7769089 |
| 10 | Georgia | 1788 | 9685744 |
| 7 | Connecticut | 1788 | 3501252 |
| 9 | Florida | 1845 | 18328340 |
| 43 | Texas | 1845 | 24326974 |
| 41 | South Dakota | 1889 | 804194 |
| 26 | Montana | 1889 | 967440 |
| 47 | Washington | 1889 | 6549224 |
| 34 | North Dakota | 1889 | 641481 |
| 50 | Wyoming | 1890 | 532668 |
| 12 | Idaho | 1890 | 1523816 |
| 31 | New Mexico | 1912 | 1984356 |
| 3 | Arizona | 1912 | 6500180 |
| 11 | Hawaii | 1959 | 1288198 |
| 2 | Alaska | 1959 | 686293 |
+-----+-----+-----+-----+
23 rows in set (0.03 sec)

mysql> _
```

Sub Queries တွေ ဒီထက်ပိုပြီး ရှုပ်ထွေးပါသေးတယ်။ ဒါကြောင့် ရှေ့ဆက်ပြီး လေ့လာချင်တယ် ဆိုရင် MySQL Manual မှာ ကြည့်နိုင်ပါတယ်။ Sub Queries တွေကို SQL ထဲမှာ သုံးမယ်ဆိုရင် တစ်ခါတစ်လေ ပြဿနာများတတ်ပါတယ်။ သတိထား သုံးဖို့ လိုပါတယ်။ အဲဒီလို Sub Queries တွေကို Program ဖက်ကနေလည်း သုံးလို့ရပါတယ်။ ဒါကိုတော့ နောက်ပိုင်း PHP, MySQL ကို တွဲပြီး လေ့လာတဲ့အခါ ဆက်ကြည့်ကြမှာ ဖြစ်ပါတယ်။

UNION: Combining Data

UNION ကတော့ Query တွေကို ပေါင်းတာ ဖြစ်ပါတယ်။

ဥပမာ အနေနဲ့ 'N' နဲ့ စတဲ့ states တွေနဲ့ လူဦးရေ အများဆုံး ရှိတဲ့ states တွေကို ပေါင်းပြီး ပြချင်တဲ့ အခါမျိုးမှာ

1. `(SELECT * FROM states WHERE name LIKE 'n%')`
2. `UNION`
3. `(SELECT * FROM states WHERE population > 10000000);`

```
D:\xampp\mysql\bin\mysql.exe

mysql> USE my_first_db;
Database changed
mysql> (SELECT * FROM states WHERE name LIKE 'n%')
-> UNION
-> (SELECT * FROM states WHERE population > 10000000);
+-----+-----+-----+-----+
| id | name          | join_year | population |
+-----+-----+-----+-----+
| 27 | Nebraska      | 1867      | 1783432    |
| 28 | Nevada       | 1864      | 2600167    |
| 29 | New Hampshire | 1788      | 1315809    |
| 30 | New Jersey   | 1787      | 8682661    |
| 31 | New Mexico   | 1912      | 1984356    |
| 32 | New York     | 1788      | 19490297   |
| 33 | North Carolina | 1789      | 9222414    |
| 34 | North Dakota  | 1889      | 641481     |
| 5  | California   | 1850      | 36756666   |
| 9  | Florida      | 1845      | 18328340   |
| 13 | Illinois     | 1818      | 12901563   |
| 22 | Michigan     | 1837      | 10003422   |
| 35 | Ohio         | 1803      | 11485910   |
| 38 | Pennsylvania  | 1787      | 12448279   |
| 43 | Texas        | 1845      | 24326974   |
+-----+-----+-----+-----+
15 rows in set (0.06 sec)

mysql> _
```

ဒီနေရာမှာ New York အနေနဲ့ N နဲ့ စတဲ့နေရာမှာလည်း ပါသလို လူဦး 10000000 ကျော်တဲ့အထဲမှာလည်း ပါပါတယ်။ ပုံမှန်ဆိုရင် နှစ်ခု ပြရမှာ ဖြစ်ပေမယ့် တကယ်ပြတဲ့အခါမှာ ထပ်နေတဲ့ Row ကို အလိုအလျှောက် ဖယ်ထုတ်လိုက်ပါတယ်။ အဲဒီ UNION ရဲ့ အားသာချက် တစ်ခုကတော့ မတူညီတဲ့ Table တွေကို ပေါင်းလို့ ရတာပါပဲ။

ဥပမာ အနေနဲ့ employees, managers နဲ့ customers ဆိုပြီး table သုံးမျိုး ရှိတယ် ဆိုကြပါစို့။ table အားလုံးရဲ့ အထဲမှာလည်း email တွေ ပါတယ်။ ကိုယ်ကလည်း အားလုံးရဲ့ email တွေကို သိချင်တဲ့အခါ

1. (SELECT email FROM employees)
2. UNION
3. (SELECT email FROM managers)
4. UNION
5. (SELECT email FROM customers WHERE subscribed = 1);

အဲဒီလို ဆိုရင် Table အားလုံးရဲ့ အထဲက email တွေကိုပဲ ဆွဲထုတ်ပေးမှာ ဖြစ်ပါတယ်။

INSERT နှင့် ပတ်သက်ပြီး ရှေ့ဆက်သိသင့်သည်များ

INSERT အကြောင်းကို ရှေ့က ပို့စ်တွေမှာ လေ့လာခဲ့ပြီး ဖြစ်ပါတယ်။ အခု INSERT က ပိုပြီး အဆင့်မြင့်တဲ့ INSERT အပိုင်းပဲ ဖြစ်ပါတယ်။

DUPLICATE KEY UPDATE

စမ်းသပ်ကြည့်ဖို့အတွက် TABLE တစ်ခု ထပ်ဆောက်ပါမယ်။

1. CREATE TABLE products (
2. id INT AUTO_INCREMENT PRIMARY KEY,
3. name VARCHAR(20),
4. stock INT,
5. UNIQUE (name)
6.);

```
D:\xampp\mysql\bin\mysql.exe

15 rows in set (0.06 sec)

mysql> CREATE TABLE products (
-> id INT AUTO_INCREMENT PRIMARY KEY,
-> name VARCHAR(20),
-> stock INT,
-> UNIQUE (name)
-> );
Query OK, 0 rows affected (0.19 sec)

mysql> INSERT INTO products SET name = 'breadmaker', stock = 10;
Query OK, 1 row affected (0.04 sec)

mysql> _
```

Table အရ name က UNIQUE ဖြစ်ဖို့ လိုမှာ ဖြစ်ပါတယ်။ အဲဒီလို UNIQUE ဖြစ်ရမယ့် နေရာမှာ နောက်ထပ် DUPLICATE ဖြစ်တဲ့ တန်ဖိုးတစ်ခုကို ထည့်ချင်တဲ့အခါ

```
D:\xampp\mysql\bin\mysql.exe
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.1.41 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement

mysql> USE my_first_db;
Database changed
mysql> INSERT INTO products SET name = 'breadmaker';
ERROR 1062 (23000): Duplicate entry 'breadmaker' for key 'name'
mysql>
```

အဲဒီလို ထည့်လိုက်တာနဲ့ ERROR တက်လာပါတော့တယ်။ ဒါပေမယ့် ကိုယ့်ဆီမှာကလည်း breadmaker နောက်ထပ် အသစ်တစ်ခု ရှိနေတယ်၊ ထပ်ဖြည့်ချင်တယ် ဆိုရင် DUPLICATE KEY ဆိုတဲ့ Keyword ကို သုံးရမှာ ဖြစ်ပါတယ်။

```
D:\xampp\mysql\bin\mysql.exe
mysql> USE my_first_db;
Database changed
mysql> INSERT INTO products SET name = 'breadmaker';
ERROR 1062 (23000): Duplicate entry 'breadmaker' for key 'name'
mysql> INSERT INTO products SET name = 'breadmaker', stock = 1
    -> ON DUPLICATE KEY UPDATE stock = stock + 1;
Query OK, 2 rows affected (0.00 sec)

mysql> SELECT * FROM products;
+----+-----+-----+
| id | name   | stock |
+----+-----+-----+
|  1 | breadmaker |    11 |
+----+-----+-----+
1 row in set (0.00 sec)

mysql> _
```

REPLACE INTO

REPLACE INTO ကို duplicate row တွေ ရှိတယ်ဆိုရင် ဖျက်ပစ်ပြီး INSERT ရဲ့ function ကို သုံးပြီး REPLACE လုပ်လိုက် ပါတယ်။

```
D:\xampp\mysql\bin\mysql.exe
mysql> SELECT * FROM products;
+----+-----+-----+
| id | name   | stock |
+----+-----+-----+
|  1 | breadmaker |    11 |
+----+-----+-----+
1 row in set (0.00 sec)

mysql> REPLACE INTO products SET name = 'breadmaker', stock = 5;
Query OK, 2 rows affected (0.00 sec)

mysql> SELECT * FROM products;
+----+-----+-----+
| id | name   | stock |
+----+-----+-----+
|  2 | breadmaker |     5 |
+----+-----+-----+
1 row in set (0.00 sec)

mysql>
```

REPLACE က ရှိသမျှ ထပ်နေတဲ့ Row တွေ အားလုံးကို ဖျက်ပစ်ပြီး အသစ်တစ်ခုနဲ့ အစားထိုး လိုက်တာပါ။ ဒါကြောင့် ID လည်း INCREMENT ဖြစ်သွားပါတယ်။

INSERT IGNORE

ဒါကတော့ duplicate error ဖြစ်မှာကို ရှောင်သွားတဲ့ သဘောပါ။

```
D:\xampp\mysql\bin\mysql.exe
mysql> SELECT * FROM products;
+----+-----+-----+
| id | name   | stock |
+----+-----+-----+
|  2 | breadmaker |     5 |
+----+-----+-----+
1 row in set (0.00 sec)

mysql> INSERT IGNORE INTO products SET name = 'breadmaker', stock = 1;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM products;
+----+-----+-----+
| id | name   | stock |
+----+-----+-----+
|  2 | breadmaker |     5 |
+----+-----+-----+
1 row in set (0.00 sec)

mysql>
```

ERROR လည်း တစ်ခုမှ မပြသလို row တစ်ခုမှလည်း Update လုပ်မှာ မဟုတ်ပါဘူး။

MySQL မှ SQL များ (၄)

ဒီတစ်ပိုင်းပြီးပြီ ဆိုရင်တော့ PHP and MySQL ကို ဆက်လို့ရပြီ ထင်ပါတယ်။ Myanmar Tutorials မှာ Database နဲ့ ပတ်သက်ရင် တော်တော် စုံလာပြီလို့ ပြောလိုရပါတယ်။ Ms SQL Server နဲ့ ပတ်သက်တဲ့ အပိုင်းတွေတော့ ကျန်သေးတာပေါ့။ အဲဒီအပိုင်းတွေကို ရေးပေးမယ့် သူတွေလည်း ခုချိန်မှာ ရှိလာပါပြီ။ နောင်ဆိုရင် Database နဲ့ ပတ်သက်ရင် တော်တော်စုံသွားတဲ့ အနေအထားတစ်ခုကို ရောက်သွားပါလိမ့်မယ်။ ကဲ အရင်တစ်ခါက မပြီးပြတ်ခဲ့တာလေး ဆက်လိုက်ရအောင် ...

ဒီတစ်ခါတော့ MySQL မှာ ရှိတဲ့ Data Types တွေကို လေ့လာကြည့်မှာ ဖြစ်ပါတယ်။ ကျွန်တော်တို့ အနေနဲ့ 1234 အစရှိတဲ့ ဂဏ္ဍန်းတွေကို သုံးဖို့လည်း လိုသလို ABCD အစရှိတဲ့ စာလုံးတွေကိုလည်း သုံးဖို့ လိုပါတယ်။ အဲဒီအပြင် နေ့စွဲဆိုတဲ့ Data Type ကိုလည်း လိုမှာ ဖြစ်ပါတယ်။

Data Types

Table တစ်ခုခြင်းစီမှာ ရှိတဲ့ Column တွေ အနေနဲ့ Data Type တစ်ခုစီကို ကိုယ်စားပြုဖို့ လိုပါလိမ့်မယ်။ အပေါ်ဖက်က MySQL Command တွေမှာ ကျွန်တော်တို့ သုံးခဲ့တာတွေက INT, VARCHAR နဲ့ DATE ဆိုတာကို သုံးခဲ့တာ မှတ်မိပါလိမ့်မယ်။ အဲဒီအပြင် အခြား Data Types တွေလည်း အများကြီး ရှိပါသေးတယ်။

ပထမဦးဆုံး စလေ့လာမှာက Numeric Data Types တွေပါ။ Group နှစ်ခု ခွဲလို့ ရပါလိမ့်မယ်။ Integers Vs Non-integers ဆိုပြီး နှစ်ခုခွဲလို့ ရပါလိမ့်မယ်။

Integer Data Types

Integer တွေအနေနဲ့ ရိုးရိုး Natural Numbers တွေကိုပဲ သိမ်းလို့ရမှာ ဖြစ်ပါတယ်။ ပြောရရင် ဒဿမကိန်း မပါဘူးဆိုပါတော့။ ပုံမှန်အားဖြင့်တော့ (-) ကော (+) ပါ ဖြစ်နိုင်ပါတယ်။ ဒါပေမယ့် UNSIGNED ဆိုတဲ့ option ကို ရွေးထားမယ် ဆိုရင်တော့ အပေါင်း ဂဏ္ဍန်းတွေပဲ သိမ်းလို့ ရပါလိမ့်မယ်။

MySQL အနေနဲ့ Integers အမျိုးအစား ငါးမျိုးကို အထောက်အပံ့ ပေးပါတယ်။ တစ်ခုနဲ့ တစ်ခု Sizes တွေ Ranges တွေ မတူပါဘူး။

Type	Bytes	Minimum Value		Maximum Value	
		Signed	Unsigned	Signed	Unsigned
TINYINT	1	-128		127	
			0		255
SMALLINT	2	-32768		32767	
			0		65535
MEDIUMINT	3	-8388608		8388607	
			0		16777215
INT	4	-2147483648		2147483647	
			0		4294967295
BIGINT	8	-9223372036854775808		9223392036854775807	
			0		18446744073709551615

Non-Integer Numeric Data Types

Non-integer Data Type အနေနဲ့ Decimal Numbers တွေ ပါမှာ ဖြစ်ပါတယ်။ FLOAT, DOUBLE နဲ့ DECIMAL ဆိုပြီး သုံးခုရှိပါတယ်။ FLOAT ကတော့ Size အနေနဲ့ 4 bytes သုံးပြီး DOUBLE ကတော့ 8 bytes သုံးပါတယ်။ အလုပ်လုပ်ပုံက အတူတူပဲ ဆိုပေမယ့် DOUBLE ကတော့ precision ပိုကောင်းပါတယ်။ DECIMAL (M,N) ကတော့ Precision Level အပေါ်မူတည်ပြီး Size တွေ ကွာနိုင်ပါတယ်။ တစ်ခြား Non-integer Data Types တွေနဲ့ မတူတဲ့ အချက်ကတော့ maximum နဲ့ Decimal point ကို သတ်မှတ်လို့ ရတာပါပဲ။ M ဆိုတာ maximum number of digits ကို ကိုယ်စားပြုပြီး N ကတော့ decimal point အရေအတွက်ကို ကိုယ်စားပြုပါတယ်။ ဥပမာအနေနဲ့ DECIMAL(13, 4) ဆိုရင် maximum 9 integer digits နဲ့ 4 decimal point ကို ကိုယ်စားပြုပါတယ်။

String Data Types

String Data Type ကတော့ Text String တွေကို မှတ်သားတဲ့ နေရာမှာ သုံးပါတယ်။ ကျွန်တော်တို့ အပေါ်က Database တွေမှာ သုံးသွားတဲ့ Data Type ကတော့ VARCHAR ဆိုတဲ့ Data Type ကို သုံးသွားပါတယ်။

CHAR(N)

CHAR(N) အနေနဲ့ Character ကို N အရေအတွက်အထိ သိမ်းနိုင်ပါတယ်။ Size အနေနဲ့လည်း Fixed Size ပဲ သုံးပါတယ်။ ကိုယ်က CHAR(50) လို့ သတ်မှတ်ထားရင် ငါးဆယ်လုံး ထည့်တာ မထည့်တာ အပထား Size အနေနဲ့တော့ အလုံး (၅၀)စာ သိမ်းထားပါလိမ့်မယ်။ Maximum အနေနဲ့ 255 အထိ သုံးလို့ရပါတယ်။

VARCHAR(N)

VARCHAR(N) ကတော့ CHAR(N) နဲ့ အလုပ်လုပ်ပုံခြင်း တူပါတယ်။ ဒါပေမယ့် မတူတာကတော့ N အနေနဲ့ Maximum အတွက် သတ်မှတ်ပေးတဲ့ နေရာမှာပဲ သုံးပါတယ်။ Size အနေနဲ့ String ဘယ်လောက်ပါလဲ အပေါ်မူတည်ပြီး ကွဲပြား သွားပါလိမ့်မယ်။ VARCHAR(50) ဆိုရင် Maximum အနေနဲ့ 50 ထည့်နိုင်ပြီး ကိုယ်က tutu လို့ထည့်ရင် အဲဒီ tutu ဆိုတဲ့ လေးလုံးစာအတွက်ပဲ Size က ရှိမှာ ဖြစ်ပါတယ်။ တကယ့် သုံးနိုင်တဲ့ Maximum ကတော့ 65535 Characters သုံးနိုင်ပါတယ်။

TEXT

TEXT ဆိုတဲ့ Data Type ကတော့ String အရှည်ကြီးတွေအတွက် အသုံးဝင်ပါတယ်။ Blog Post တွေကို သိမ်းတဲ့ နေရာမျိုးတွေမှာ သုံးနိုင်ပါတယ်။ TEXT မှာ 65535 characters၊ MEDIUMTEXT မှာ 16.7 million characters နဲ့ LONGTEXT မှာ 4.3 billion characters သိမ်းနိုင်ပါတယ်။ MySQL အနေနဲ့ အဲဒီ TEXT Data Type တွေကို သီးသန့် နေရာမှာ သိမ်းလေ့ရှိပါတယ်။ အဲဒီလို သိမ်းလိုက်တဲ့ အတွက် Table အနေနဲ့ သေးသေးနဲ့ မြန်မြန် သုံးနိုင်စေပါတယ်။

Date Types

Date Types ကလည်း မရှိမဖြစ် Data Type ထဲမှာ ပါပါတယ်။

DATE

DATE ကတော့ ရက်စွဲတန်ဖိုးကို သိမ်းပေးနိုင်ပါတယ်။ Format အနေနဲ့ “YYYY-MM-DD” သိမ်းပေးနိုင်ပါတယ်။ YYYY ဆိုတာက Year ကိုယ် ကိုယ်စားပြုပါတယ်။ 2010 ဆိုပါတော့။ MM ကတော့ Month ပါ။ DD ကတော့ Day ပါ။

DATETIME

DATETIME အနေနဲ့ကတော့ date ကော time ပါ ကိုယ်စားပြုပြီး မှတ်သားနိုင်ပါတယ်။ Format အနေနဲ့ ‘YYYY-MM-DD HH:MM:SS’ ဆိုပြီး သိမ်းနိုင် မှတ်နိုင်ပါတယ်။ Space အနေနဲ့ 8 bytes ယူပါတယ်။

TIMESTAMP

သူကတော့ DATETIME နဲ့ ဆင်ပါတယ်။ မတူတာက သူက 4 Bytes ပဲ နေရာယူပြီး Range အနေနဲ့ 1970-01-01 00:00:01 ကနေ 2038-01-19 03:14:07 အထိ ရှိပါတယ်။

TIME

သူကတော့ အချိန်သီးသန့် မှတ်ပေးပါလိမ့်မယ်။

YEAR

သူကတော့ ခုနှစ် သီးသန့် မှတ်ပါလိမ့်မယ်။

အပေါ်မှာ ပြသွားတဲ့ Data Type တွေကတော့ အသုံးအများဆုံး Data Type တွေပါပဲ။ တစ်ခြား MySQL က Support လုပ်တဲ့ Data Type တွေကို လေ့လာချင်တယ် ဆိုရင် [ဒီနေရာ](#)မှာ လေ့လာနိုင်ပါတယ်။ ရွှင်လန်းချမ်းမြေ့ပါစေ။

Normalization and Common Database Design Patterns (0)

ဆော့ဝဲရေးသားတဲ့ ပုံသဏ္ဌာန်တွေ အခုအချိန်မှာ တစ်ခုပြီးတစ်ခု ပြောင်းလဲလာခဲ့တာ ဆယ်စုနှစ်တစ်ခု နှစ်ခုအတွင်းမှာ အသုံးပြုခဲ့တဲ့ Tools တွေ အသုံးပြုတဲ့ Platforms တွေ အများကြီး ပြောင်းလဲ တိုးတက်ခဲ့ပါတယ်။ ဒါပေမယ့် မပြောင်းလဲပဲ ဆက်ပြီး သုံးစွဲ နေကြရတာက Relational Database Model ဖြစ်နေပါတယ်။ အခုအချိန်အထိ Relational Model အပေါ် အခြေခံတဲ့ Database Server များကို အများဆုံး အသုံးပြု နေကြရပါတယ်။ ဒါ့ကြောင့် Relational Model ရဲ့ Database Design အတွက် အရေးပါတဲ့ Normalization Process ကို ဥပမာအချို့နဲ့ အသုံးပြုပြီးတော့ ဖတ်ရလွယ်ကူစေရန် ရေးသားဖို့ စိတ်ကူးမိပါတယ်။ ပထမဆုံးအနေနဲ့ Business Documents တွေကနေ တစ်ဆင့် Normalize လုပ်ပုံကို အဆင့်ဆင့်ဖော်ပြမှာဖြစ်ပါတယ်။ နောက်ပိုင်းများမှာတော့ Business Requirements များကိုအကျဉ်းချုပ်ဖော်ပြပြီး ကြိုတင်ပြီး Design လုပ်ထားတဲ့ Database Design ကိုသာပြသပြီးတော့ အသုံးများတတ်တဲ့ Design Patterns များကို တင်ပြမှာဖြစ်ပါတယ်။



Normalization ကို Software Development မှာ အသုံးပြုပုံ အဆင့်ဆင့်ကို ပြသခြင်း ဖြစ်တဲ့အတွက် ရေးသားထားတဲ့ ဆောင်းပါးများကို ဖတ်ရှုဖို့အတွက် ကြိုတင် လိုအပ်ချက်များ ရှိပါတယ်။ ပထမဆုံးအနေနဲ့ Database အမျိုးအစား တစ်ခုကို အသုံးပြုတတ်ဖို့ လိုအပ်ပါတယ်။ အဲဒီ လိုအပ်ချက်အတွက် ကိုနေထက် ရေးသားထားတဲ့ ခက်သလားဟု ဒေတာဘေ့စ် ဆောင်းပါးများနဲ့ ကိုသီဟရေးသားတဲ့ MySQL မှ SQL များကို ဖတ်ရှုပါက အများကြီး အထောက်အကူ ဖြစ်ပါလိမ့်မယ်။ ဒုတိယလိုအပ်ချက် တစ်ခုအနေနဲ့ Programming Language တစ်ခုခုကို တတ်ကျွမ်းရပါမယ် ထပ်မံပြီးတော့ အတိအကျ ပြောရမယ်ဆိုလျှင် တတ်ကျွမ်းတဲ့ Programming Language ကို အသုံးပြုပြီးတော့ Database တစ်ခုအတွင်းက Data များကို အသွင်း အထုတ် အသစ်ထည့် လုပ်တတ်ရပါမယ်။ ဒီလိုအပ်ချက်နဲ့ ပြည့်စုံမှသာလျှင် ဆက်လက် ဆွေးနွေးမယ့် Design Patterns များကို ကြည့်ပြီးတော့ မိမိတို့ရဲ့ Development မှာဘယ်လို အကျိုးရှိလာမယ် ဆိုတာကို သဘောပေါက်နိုင်မှာ ဖြစ်ပါတယ်။ တတိယအချက် အနေနဲ့ ဆွဲပြီးသား Database Design များကို ဖတ်ရှုဖို့အတွက် UML ကို နားလည်မယ်ဆိုရင် ပိုပြီး အဆင်ပြေ ပါလိမ့်မယ်။ တတိယ အချက်ကတော့ မဖြစ်မနေ မဟုတ်ပါဘူး။ Normalization အဆင့်ဆင့်ကို ပထမဆုံး တင်ပြမှာဖြစ်ပါတယ်။ ဒါ့အပြင် ရှုပ်ထွေးတဲ့ Diagram များပါလာမှာ မဟုတ်တဲ့အတွက် အလွယ် တစ်ကူပဲ နားလည် နိုင်ပါလိမ့်မယ်။

ပထမဆုံးအနေနဲ့ Normalization ဆိုတာ ရှုပ်ထွေးနေတဲ့ Data များကို စနစ်တစ်ကျ ပုံစံတစ်ခုအဖြစ် ပြောင်းလဲပုံ အဆင့်ဆင့်နဲ့ နည်းလမ်းများလို့ အဓိပ္ပာယ် ဖွင့်နိုင်ပါတယ်။ စီးပွားရေး လုပ်ငန်းတစ်ခုမှာ အသုံးပြုတဲ့ Software တစ်ခုကို ကြည့်မယ်ဆိုရင် အဓိကအားဖြင့် Data များကို သိမ်းဆည်း ထိန်းသိမ်းတာနဲ့ သိမ်းဆည်းထားတဲ့ Data များကို လိုအပ်တဲ့ ပုံစံအဖြစ် ပြန်လည် ထုတ်ပေးရတဲ့ အပိုင်းများသာ အဓိက ပါဝင်ပါတယ် တစ်ခါတစ်ရံမှာ သက်ဆိုင်ရာ စီးပွားရေးလုပ်ငန်းများနဲ့ သက်ဆိုင်တဲ့ တွက်ချက်မှု များပါဝင်တဲ့ အစိတ်အပိုင်းလည်း ပါဝင်တတ်ပါတယ်။ ဒီနေရာများမှာ Normalization ကို Software Development မလုပ်ခင်မှာ သိမ်းဆည်းမယ့် Data များကို မည်သို့မည်ပုံ စနစ်တစ်ကျ သိမ်းဆည်းမယ်လို့ Design လုပ်စဉ်မှာ အသုံးပြုပါတယ်။ Normalization ပြုလုပ်ဖို့အတွက် အသုံးပြုမယ့် စီးပွားရေး လုပ်ငန်းမှာ အသုံးပြုတဲ့ Data များကို ကြည့်ရှုဖို့လိုအပ်ပါတယ် လက်ရှိသုံးစွဲလျက်ရှိတဲ့ Data များဟာ Manual မှတ်သားထားတဲ့ စာရွက်စာတမ်းများ ဖြစ်နိုင်သလို Spread Sheet လို Electronic Documents များလည်း ဖြစ်နိုင်ပါတယ်။ Software Design ရဲ့ အခြေခံအောက်ဆုံး အပိုင်းမှာပါဝင်နေတာ ဖြစ်တဲ့အတွက် Development လုပ်စဉ်မှာ ပြင်ဆင်လို့ မရနိုင်တော့တာများပါတယ် ပြင်ဆင်မယ်ဆိုရင် Development လုပ်ပြီးသား အစိတ်အပိုင်းများကို ထိခိုက်မှု ပြင်ဆင်ရမှု များတဲ့အတွက် အထူး ဂရုပြုဖို့ လိုအပ်ပါတယ်။

Normalization မှာ အဆင့်အလိုက် Data များကိုရှင်းလင်းပုံကို Normal Form လို့ခေါ်ပါတယ်။ Normal Form များဟာ First ကနေ Infinity အထိရှိတယ်လို့ သိအိုရီအရ ဆိုပါတယ်။ ဒါပေမယ့် လက်တွေ့ အသုံးပြုတဲ့ Normal Form ကတော့ Third အထိသာ အများဆုံး သုံးစွဲပါတယ်။ သင်္ကေတအားဖြင့် 1NF, 2NF, 3NF စသည်ဖြင့် နောက်ပိုင်း ဆောင်းပါးများမှာ သုံးစွဲ ရေးသားပါမယ်။ အခုရေးသားတဲ့ ဆောင်းပါးများဟာ Normalization အသုံးပြုပုံကို အဓိက မြင်သာအောင် ပြသမှာ ဖြစ်တဲ့အတွက် Normal Form အဆင့်ဆင့်ရဲ့ အသေးစိတ် သိအိုရီများကို ဆွေးနွေးထားချက်တွေ ပါဝင်မှာ မဟုတ်ပါဘူး။ သိဖို့လိုအပ်တဲ့ အချက်အလက် များကိုသာ တင်ပြပြီး လက်တွေ့အားဖြင့် Normalize လုပ်ပုံကိုသာ တင်ပြမှာ ဖြစ်ပါတယ်။ ဒါ့ကြောင့် အခြေခံ အချက်အလက် များကိုလည်း သိရှိထားဖို့ လိုအပ်ပါတယ်။ သိအိုရီများကို အသေးစိတ် လေ့လာဖို့ အတွက်လည်း တိုက်တွန်းပါတယ်။ လေ့လာဖို့အတွက် အကြံပေးရမယ်ဆိုရင် C. J. Date ရေးသားတဲ့ Introduction to Database System ရယ် Elmasri နဲ့ Navathe ရေးသားတဲ့ Fundamentals of Database Systems ကိုညွှန်းချင်ပါတယ်။

ဒီကနေ့တော့ သိသင့် သိထိုက်တာများနဲ့ Normalization အကြောင်း သိကောင်းစရာ များကိုသာ နိဒါန်းအနေနဲ့ ရေးသား တင်ပြထားပါတယ်။ ဆက်လက် ရေးသားမယ့် ဆောင်းပါးမှာတော့ Normal Form များအတွက် အကျဉ်းချုပ် စည်းကမ်းများကို တင်ပြပေးပြီးတော့ Raw Data များကို ပြသပြီး အဆင့်ဆင့် Normalization လုပ်ပုံ အဆင့်ဆင့်ကို တင်ပြပေးမှာ ဖြစ်ပါတယ်။ Diagram များ ရေးဆွဲရခြင်းများ ပါနေတဲ့အတွက် ရေးသားရတဲ့ အချိန်များ ပိုမိုလိုအပ်ပါတယ်။

Normalization and Common Database Design Patterns (1)

ပထမဆုံးအနေနဲ့ နေ့စဉ်မြင်တွေ့နေကျ Data များကို စတင်လေ့လာပြီး Normalize လုပ်တဲ့ အဆင့်ဆင့်ကို စတင် တင်ပြမှာ ဖြစ်ပါတယ်။ Normalize မလုပ်ဆောင်ခင်မှာ Normal Form များကို အလွယ်ဆုံးနဲ့ အကျဉ်းချုပ်သိဖို့ လိုအပ်ပါတယ်။ Normal Form တစ်ခုစီရဲ့ Definition များမှာ စာအုပ်တစ်ခုနဲ့ တစ်ခုမှာ အနည်းအကျဉ်း ကွဲပြားတတ်ပါတယ် ဒါ့ကြောင့် အခုဖော်ပြထားတဲ့ အကျဉ်းချုပ်ဟာ ကျွန်တော် မှတ်သားထားတဲ့ အကျဉ်းချုပ် ဖြစ်တဲ့အတွက် အများနဲ့ အနည်းငယ် ကွဲပြားနိုင်ပါတယ်။

First Normal Form (1NF)

- Separate repetitive values group into new table
- Define Unique Identifier

Second Normal Form (2NF)

- Separate partial dependence values group into new table

Third Normal Form (3NF)

- Separate transitive dependence values group into new table

ပထမဆုံး အနေနဲ့ အမြဲ မြင်တွေ့နေကျ Invoice တစ်စောင်ကို နမူနာအဖြစ် အသုံးပြုပါမယ်။ Invoice Data ကို Normalize လုပ်တာဟာ လွယ်ပေမယ့် အလွန် အသုံးကျပါတယ် လုပ်ငန်း အတော်များများမှာ Invoice တွေ သုံးတဲ့အတွက် အမြဲ ပြန်လည် အသုံးပြုနေရတဲ့ Design Pattern တစ်ခုလို့ ပြောလို့လည်း ရနိုင်ပါတယ်။

Invoice No	00001
Date	1/1/2010
Customer No	C0001
Customer	Mr. A
Address	Mr. A's Address
Employee No	E0001
Employee	Mr. E

Item	Description	Price	Qty	Amount
A0001	Apple	1	10	10
B0001	Banana	1	5	5
C0001	Coconut	3	1	3
				18

အထက်မှာ ဖော်ပြထားတဲ့ Invoice မှာဆိုရင် Invoice No, Date, Customer No, Customer, Address, Employee No, Employee, Item, Description, Price, Qty, Amount, Total ဆိုပြီး မှတ်သားလို့ရတဲ့ Data အမျိုးအစားနာမည် (Column Name) များ ပါဝင်ပါတယ်။ ပထမဆုံး အနေနဲ့ အဆိုပါ Data တွေကို Tabular Format နဲ့ ချရေးရပါမယ် အထက်မှာ ပြောခဲ့တဲ့ Column Name များကို တပ်ရပါမယ်။ Tabular Format နဲ့ ချရေးထားတဲ့ Normalize မလုပ်ရသေးတဲ့ Raw Data Table ကို Normal Form အရဆိုရင် 0NF လို့ ပြောလို့ရပါတယ်။ နမူနာအဖြစ် အောက်မှာ ဖော်ပြထားတဲ့ 0NF ပြုလုပ်ထားတဲ့ Table ကိုကြည့်နိုင်ပါတယ်။

0NF

Invoice No	Date	CustomerNo	Customer	Address	Employee	Item	Description	Price	Qty	Amount	Total
00001	1/1/2010	C0001	Mr. A	Mr. A's Address	E0001	A0001	Apple	1	10	10	18
00001	1/1/2010	C0001	Mr. A	Mr. A's Address	E0001	B0001	Banana	1	5	5	18
00001	1/1/2010	C0001	Mr. A	Mr. A's Address	E0001	C0001	Coconut	1	2	2	18

ပုံကို ကလစ်ခေါက်ပြီး အကြီးချဲ့ကြည့်ပါ

0NF Table ကို ကြည့်မယ်ဆိုရင် Data အတော်များများ ထပ်ကာထပ်ကာ ပါဝင်နေတာကို တွေ့ရပါမယ်။ အဲဒါဟာ 0NF ရဲ့အားနည်းချက်ပါ အကြိမ် များစွာ ပါဝင်နေတဲ့ Data တွေဟာ ရေးသွင်းတဲ့ လူမှားယွင်းခဲ့ရင် တစ်နေရာမှာ လွဲမှားနိုင်ပါတယ်။ အဲဒီလိုသာ ဖြစ်ခဲ့မယ်ဆိုရင် Data တွေရဲ့ Consistency ကို ထိခိုက်စေပါတယ်။ ဒါကြောင့် 1NF မှာ အကြိမ်များစွာ ပါဝင်နေတဲ့ InvoiceNo, Date, CustomerNo, Customer, Address, EmployeeNo, Employee, Total တွေကို သီးသန့် Table တွေ အနေနဲ့ ခွဲထုတ် လိုက်ပါမယ်။

အောက်ဖော်ပြပါပုံမှ Invoice Table ကိုကြည့်ပါ။ အထက်မှာ ပြောခဲ့တဲ့ 1NF ရဲ့ပြုလုပ်ပုံမှာ Define Unique Identifier လို့ပါသေးတဲ့ အတွက် Table တိုင်းမှာ Key Column သတ်မှတ် ပေးရပါမယ်။ ဒါ့ကြောင့် အသစ် ခွဲထုတ်လိုက်တဲ့ Invoice Table မှာ ပြန်ပြီး မထပ်နိုင်တဲ့ Columns အဖြစ် InvoiceNo ကို အသုံးပြုလို့ ရတဲ့အတွက် InvoiceNo ကို Unique Identifier အဖြစ် အသုံးပြု ထားပါတယ်။

ဒီနေရာမှာ Table အသစ် ခွဲထုတ် လိုက်တဲ့အတွက် မူရင်း Table မှာ ခွဲထုတ်လိုက်တဲ့ Table နဲ့ ဆက်စပ်မှု ရှိဖို့ လိုအပ်တဲ့အတွက် အသစ် ခွဲထုတ်လိုက်တဲ့ Invoice Table ရဲ့ Key Columns ကို မူရင်း Table မှာ Foreign Key အဖြစ် Referential Integrity အတွက် သတ်မှတ်ပေးဖို့ လိုပါလိမ့်မယ်။ မူရင်း Table မှာလည်း 1NF နဲ့ ညီဖို့အတွက် Unique Identifier သတ်မှတ် ရပါမယ်။

InvoiceDetail မှာ Column တစ်ခုတည်း အနေနဲ့ Unique ဖြစ်တာ မရှိတဲ့အတွက် InvoiceNo နဲ့ Item နှစ်ခုပေါင်းကို Composite Key သတ်မှတ်ထားပါတယ်။ သတ်မှတ်ရခြင်း အကြောင်းကတော့ Invoice တစ်စောင်မှာ အမျိုးတူပစ္စည်းကို နှစ်ကြိမ်မှတ်သားခြင်းမရှိတဲ့အတွက် အဲဒီ Column နှစ်ခုပေါင်း Values ဟာနှစ်ကြိမ်ပြန်လည်မထပ်နိုင်တဲ့အတွက် Unique Identifier အဖြစ်အသုံးချလို့ရပါတယ်။ ဒါဆိုရင် 0NF ကနေ 1NF ကိုပြောင်းလဲတဲ့ အဆင့် တစ်ခု ပြီးစီးသွားပါပြီ။

1NF

Invoice

InvoiceNo	Date	CustomerNo	Customer	Address	EmployeeNo	Employee	Total
00001	1/1/2010	C0001	Mr. A	Mr. A's Address	E0001	Mr. E	18

InvoiceDetail

InvoiceNo	Item	Description	Price	Qty	Amount
00001	A0001	Apple	1	10	10
00001	B0001	Banana	1	5	5
00001	C0001	Coconut	3	1	3

2NF ကတော့ Dependency အပေါ်မှာ ခွဲခြားတာပါ အရှင်းလင်းဆုံး ကတော့ Table တစ်ခုထဲမှာ ပါဝင်တဲ့ Columns တွေဟာ Key Column ကိုမှီခိုရပါတယ်။ မှီခိုရာမှာ Direct or Transitive ဘယ်လိုပဲဖြစ်ဖြစ် လက်ခံပါတယ်။ Dependence မဖြစ်တဲ့ Columns များကို သီးသန့် Table အဖြစ် ခွဲထုတ်ရပါမယ်။

ပထမဆုံးအနေနဲ့ 1NF ကို ပြန်ကြည့် ရပါမယ်။ Invoice Table မှာ InvoiceNo ဟာ Key ဖြစ်ပါတယ်။ Date ဟာ Invoice ကိုဝယ်တဲ့ Date ဖြစ်တဲ့အတွက် Key ကို မှီခိုတယ် ပြောရပါမယ်။

CustomerNo ဆိုတာလည်း ဒီ Invoice ကိုဝယ်တဲ့ Customer ရဲ့ နံပါတ်ပါ။ အဲဒီအတွက် မှီခိုပါတယ်။ ဒီနေရာမှာ Customer နဲ့ Address ကတော့ Invoice ကို မမှီခိုပါဘူး။ ဒါပေမယ့် သက်ဆိုင်ရာ CustomerNo ရဲ့ အမည်နဲ့ လိပ်စာ ဖြစ်တဲ့အတွက် တစ်ဆင့်ခံ မှီခိုတယ်လို့ ပြောလို့ရပါတယ်။ အဲဒါမျိုးကို Transitive လို့ခေါ်ပါတယ်။

EmployeeNo ဟာလည်း Invoice ကို ပြုလုပ်ပေးလိုက်တဲ့ ဝန်ထမ်းရဲ့ ကိုယ်ပိုင် နံပါတ်ဖြစ်လို့ မှီခိုပါတယ်။ Employee ကတော့ ဝန်ထမ်းရဲ့ အမည်ဖြစ်လို့ တိုက်ရိုက် မမှီခိုပေမယ့် EmployeeNo အပေါ်ကနေ Transitive အနေနဲ့ မှီခိုပါတယ်။

Total ဟာလည်း Invoice တစ်ခုလုံးရဲ့ စုစုပေါင်းတန်ဖိုး ဖြစ်တဲ့အတွက် InvoiceID အပေါ်မှာ တိုက်ရိုက်မှီခိုပါတယ်။ အဲဒါကြောင့် Invoice မှာ Key အပေါ်မှာ မမှီခိုတဲ့ Column မပါဝင်တဲ့အတွက် ဘာမှ လုပ်စရာမလိုပါဘူး။

Invoice Detail ကိုကြည့်မယ်ဆိုရင် Key ဟာ Composite ဖြစ်ပါတယ် အဲဒါကြောင့် အခြား Column များဟာ နှစ်ခုပေါင်း အပေါ်မှာ မှီခိုရပါမယ်။ ဒီနေရာမှာ Description နဲ့ Price ဟာ InvoiceNo နဲ့ ပါတ်သက်မှု မရှိပါဘူး။ Item ရဲ့ ဈေးနှုန်းနဲ့ အမည် ဖော်ပြချက်သာ ဖြစ်လို့ 2NF နဲ့ညီဖို့အတွက် ခွဲထုတ်ရပါမယ်။ 1NF မှာလိုပဲ ခွဲထုတ်လိုက်ရင် မူရင်း Table မှာ Foreign Key ကို သတ်မှတ် ပေးခဲ့ရပါတယ်။ ခွဲထုတ်လိုက်တဲ့ ပုံများကို အောက်မှာ ပြသထားတဲ့ 2NF ပုံများမှာ ကြည့်ရှု နိုင်ပါတယ်။ Qty နဲ့ Amount ကတော့ သက်ဆိုင်ရာ Invoice နဲ့ Item ရဲ့ အရေအတွက်နဲ့ ကျသင့်ငွေဖြစ်လို့ Composite Key တစ်ခုလုံးကို မှီခိုတဲ့အတွက် ခွဲထုတ်စရာ မလိုအပ်ပါဘူး။

2NF

Invoice

InvoiceNo	Date	CustomerNo	Customer	Address	EmployeeNo	Employee	Total
00001	1/1/2010	C0001	Mr. A	Mr. A's Address	E0001	Mr. E	18

InvoiceDetail

InvoiceNo	Item	Qty	Amount
00001	A0001	10	10
00001	B0001	5	5
00001	C0001	1	3

Item

Item	Description	Price
A0001	Apple	1
B0001	Banana	1
C0001	Coconut	4

3NF မှာတော့ 2NF မှာခွင့်ပြုခဲ့တဲ့ အကြောင်းအရာ တစ်ခုကို ထပ်မံ တင်းကျပ်ပါတယ်။ 2NF မှာ Non-Key Columns များကို Key Column အပေါ်မှာ Direct or Transitive မှီခိုခွင့် ပေးခဲ့ပါတယ်။ 3NF မှာတော့ Non-Key Columns များဟာ Key Column အပေါ်မှာ Direct သာမှီခိုရပါမယ်။ Transitive ဖြစ်ခဲ့ရင် Table တစ်ခုအဖြစ် ခွဲထုတ်ရမယ်လို့ သတ်မှတ် ထားပါတယ်။ အဲဒီအတွက် အထက်မှာ ခွင့်ပြုခဲ့တဲ့ Employee နဲ့ Customer များကို သီးသန့် Table များအဖြစ် ခွဲထုတ် လိုက်ရပါမယ်။ အရင် လုပ်ဆောင်ချက်များလိုပဲ Referential Integrity အတွက် ခွဲထုတ်လိုက်တဲ့ Table အသစ်မှ Key ကို မူရင်း Table မှာ Foreign Key အဖြစ် သတ်မှတ် ပေးရမှာပဲ ဖြစ်ပါတယ်။ အဲဒီလို လုပ်ဆောင်ထားတဲ့ 3NF နဲ့ညီတဲ့ Table များကို အောက်မှာ ဖော်ပြထားပါတယ်။

3NF

Invoice

InvoiceNo	Date	CustomerNo	EmployeeNo	Total
00001	1/1/2010	C0001	E0001	18

InvoiceDetail

InvoiceNo	Item	Qty	Amount
00001	A0001	10	10
00001	B0001	5	5
00001	C0001	1	3

Item

Item	Description	Price
A0001	Apple	1
B0001	Banana	1
C0001	Coconut	4

Customer

CustomerNo	Customer	Address
C0001	Mr. A	Mr.A's Address

Employee

EmployeeNo	Employee
E0001	Mr. E

အထက်မှာ ဖော်ပြသွားတာတွေကတော့ 0NF to 3NF အထိ အဆင့်ဆင့် လုပ်ဆောင်ပုံကို အလွယ်တကူ တွေ့ရှိနိုင်တဲ့ Data နဲ့ ပြသထားတာ ဖြစ်ပါတယ်။ အထက်မှာ ရေးသားထားတာတွေမှာ Database နဲ့ ပါတ်သက်တဲ့ အခြေခံ အချက်အလက်များကို ရှင်းလင်း ထားခြင်း မရှိသလို ဖော်ပြ ထားခြင်းလည်း မရှိပါဘူး။ အကယ်၍ အဆုံးအထိ ဖတ်ကြည့်လို့ နေရာ အတော်များများမှာ နားမလည်ခဲ့ဘူး ဆိုရင် အခြားသော အခြေခံများကို အရင်ဆုံး ဖတ်ရှုဖို့ လိုပါတယ်။ Key, Composite Key, Foreign Key, Unique Identifier, Dependency, Transitive စတာတွေကို အသေအချာ သိရှိဖို့ လိုအပ်ပါတယ်။

ပထမဆုံး ဆောင်းပါးမှာ ညွှန်းခဲ့တဲ့ စာအုပ်နှစ်အုပ်မှာ ရှာဖွေ ဖတ်ရှုနိုင်ပါတယ် ဒါမှ မဟုတ်ရင်လည်း အခြားသော စာအုပ် သို့မဟုတ် web resource များမှာ ရှာဖွေ ဖတ်ရှုလို့ ရပါတယ်။ ယခုရေးသားထားတဲ့ ဆောင်းပါးမှ ထပ်မံ သိရှိလိုတာများ ရှိခဲ့မယ်ဆိုရင် Comment များကနေပြီး မေးမြန်း ထားခဲ့နိုင်ပါတယ်။ Comment မှာပဲ Reply အနေနဲ့ ဖြေပေးထားပါမယ်။ နောက် ဆောင်းပါးမှာတော့ ယခု ပြုလုပ်ထားတဲ့ Normalize Schema ကိုပဲ အခြားသော Notation နဲ့ ရေးဆွဲပြီးတော့ Relationship များအကြောင်း ရေးသား ပါဦးမယ်လို့ ပြောရင်းနဲ့ပဲ ဒီဆောင်းပါးကို အဆုံးသတ် လိုက်ပါတယ်။