Creating Databases

A bear walks into a bar and says to the bartender "I'll have a gin.....and tonic". The bartender
es "why such a big pause?"

A bear walks into a bar and says to the bartender "I'll have a gin.....and tonic". The bartender ɜs "why such a big pause?"

The bear replies ”Because I was born with them."

A bear walks into a bar and says to the bartender "I'll have a gin.....and tonic". The bartender
es "why such a big pause?"

The bear replies ”Because I was born with them."

Data Correctness

WIKIPEDIA
The Free Encyclopedia

Data quality refers to the condition of a set of values of qualitative or quantitative variables. There are many definitions of data quality but data is generally considered high quality if it is "fit for [its] intended uses in operations, decision making and planning". Alternatively, data is deemed of high quality if it correctly represents the real-world construct to which it refers.

Constraints

MySQL CONSTRAINT is used to define rules to allow or restrict what values can be stored in columns. The purpose of inducing constraints is to enforce the integrity of a database.

MySQL CONSTRAINTS are used to limit the type of data that can be inserted into a table.

MySQL CONSTRAINTS can be classified into two types - column level and table level.

The column level constraints can apply only to one column where as table level constraints are applied to the entire table.

CONSTRAINT	DESCRIPTION
NOT NULL	In MySQL NOT NULL constraint allows to specify that a column can not contain any NULL value. MySQL NOT NULL can be used to CREATE and ALTER a table.
UNIQUE	The UNIQUE constraint in MySQL does not allow to insert a duplicate value in a column. The UNIQUE constraint maintains the uniqueness of a column in a table. More than one UNIQUE column can be used in a table.
PRIMARY KEY	A PRIMARY KEY constraint for a table enforces the table to accept unique data for a specific column and this constraint creates a unique index for accessing the table faster.
FOREIGN KEY	A FOREIGN KEY in MySQL creates a link between two tables by one specific column of both tables. The specified column in one table must be a PRIMARY KEY and referred by the column of another table known as FOREIGN KEY.
CHECK	A CHECK constraint controls the values in the associated column. The CHECK constraint determines whether the value is valid or not from a logical expression.
DEFAULT	In a MySQL table, each column must contain a value (including a NULL). While inserting data into a table, if no value is supplied to a column, then the column gets the value set as DEFAULT.

NULL Values

In SQL, **NULL** is the term used to represent a missing value.

A **NULL** value in a table is a value in a field tha appears to be blank.

A field with a **NULL** value is a field with ***no*** value.

Not NULL Constraint

By default, a column in a table can hold **NULL** values. The **NOT NULL** constraint enforces a column to **NOT** accept **NULL** values. This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

```
use ist210mrg415;
create table emp (id  int not null,
                  name varchar(50) not null,
                  age tinyint not null,
                  address varchar(30),
                  salary float not null );
```

```
describe emp
```

+ Options

Field	Type	Null	Key	Default	Extra
id	int(11)	YES		*NULL*	
name	varchar(50)	YES		*NULL*	
age	tinyint(4)	YES		*NULL*	
address	varchar(30)	YES		*NULL*	
salary	float	YES		*NULL*	

Run SQL query/queries on database ist210mrg415:

```
1 alter table emp modify id INT(11) not null;
```

+ Options

Field	Type	Null	Key	Default	Extra
id	int(11)	NO		*NULL*	
name	varchar(50)	YES		*NULL*	
age	tinyint(4)	YES		*NULL*	
address	varchar(30)	YES		*NULL*	
salary	float	YES		*NULL*	

Let’s also ensure an age is entered

```
alter table emp modify age tinyint(4) not null
```

Your SQL query has been executed successfully.

```
describe emp
```

emp

+ Options

Field	Type	Null	Key	Default	Extra
id	int(11)	NO		*NULL*	
name	varchar(50)	YES		*NULL*	
age	tinyint(4)	NO		*NULL*	
address	varchar(30)	YES		*NULL*	
salary	float	YES		*NULL*	

Run SQL query/queries on table ist210mrg415.emp:

```
insert into emp (id,name,age,address,salary) values(6,'karen',NULL,'sheridan road',1500);
```

Error

SQL query:

```
insert into emp (id,name,age,address,salary) values(6,'karen',NULL,'sheridan road',1500)
```

MySQL said:

```
#1048 - Column 'age' cannot be null
```

Run SQL query/queries on table ist210mrg415.emp:

```
insert into emp (id,name,age,address,salary) values(6,'karen',54,NULL,1500);
```

Columns

- id
- name
- age
- address
- salary

```
select * from emp;
```

+ Options

id	name	age	address	salary
1	matt	55	brian ct	1000
2	jane	43	main st	500
3	sue	50	walsh st	1100
4	tom	52	jones st	800
5	tim	50	jazz st	1800
6	karen	54	*NULL*	1500

Let’s create a new attribute (or column) in our table to represent the type of employee

The type would be:

- Tech
- Manager
- Rep

First we need to add a column

Run SQL query/queries on table ist210mrg415.emp:

```
1 alter table emp add column emptype char(7) after id;
```

```
describe emp
```

+ Options

Field	Type	Null	Key	Default	Extra
id	int(11)	NO		*NULL*	
emptype	char(7)	YES		*NULL*	
name	varchar(50)	YES		*NULL*	
age	tinyint(4)	NO		*NULL*	
address	varchar(30)	YES		*NULL*	
salary	float	YES		*NULL*	

One of the issues we often have is what I would call “data consistency” - for example, if you adding a name to a database, and you add it as “Matthew” but when you query the database you use a person’s “nickname” – say “Matt” – you often won’t get a match, for example

“Matthew” $\neq$ “Matt”

Enumeration

An enumeration is a **complete**, ordered listing of ***all*** the items in a collection.

The term is commonly used in mathematics and computer science to refer to a listing of all of the elements of a set.

Fruits = apple, banana, pear, orange

The value of fruit may only take on one of those 4 values

This ensures that we can know the value since the enumerated list contains just the permitted set of

Run SQL query/queries on database ist210mrg415:

```
1 alter table emp modify column emptype enum("Tech","Manager","Rep")
```

+ Options

Field	Type	Null	Key	Default	Extra
id	int(11)	NO		*NULL*	
emptype	enum('Tech','Manager','Rep')	YES		*NULL*	
name	varchar(50)	YES		*NULL*	
age	tinyint(4)	NO		*NULL*	
address	varchar(30)	YES		*NULL*	
salary	float	YES		*NULL*	

select * from emp;

+ Options

id	emptype	name	age	address	salary
1	NULL	matt	55	brian ct	1000
2	NULL	jane	43	main st	500
3	NULL	sue	50	walsh st	1100
4	NULL	tom	52	jones st	800
5	NULL	tim	50	jazz st	1800
6	NULL	karen	54	NULL	1500

Run SQL query/queries on table ist210mrg415.emp:

```
1 alter table emp drop emptype
```

Do you really want to execute "alter table emp drop emptype"?

Cancel OK

SELECT * SELECT INSERT UPDATE DELETE Clear

Format Get auto-saved query

Run SQL query/queries on table ist210mrg415.emp:

```
alter table emp add column emptype char(7) after id;

alter table emp modify column emptype enum("Manager","Tech","Rep") default "Tech"
```

+ Options

Field	Type	Null	Key	Default	Extra
id	int(11)	NO		*NULL*	
emptype	enum('Manager','Tech','Rep')	YES		Tech	
name	varchar(50)	YES		*NULL*	
age	tinyint(4)	NO		*NULL*	
address	varchar(30)	YES		*NULL*	
salary	float	YES		*NULL*	

Run SQL query/queries on table ist210mrg415.emp:

```
1 insert into emp(id,name,age,address,salary) values(21,"andreea",41,"861 bedford
ave",1100)
```

Notice I skipped the emptype column on the insert – what I want to do it allow the default value to be used

+ Options

id	emptype	name	age	address	salary
1	*NULL*	matt	55	brian ct	1000
2	*NULL*	jane	43	main st	500
3	*NULL*	sue	50	walsh st	1100
4	*NULL*	tom	52	jones st	800
5	*NULL*	tim	50	jazz st	1800
6	*NULL*	karen	54	*NULL*	1500
20	*NULL*	jon	49	111 east ave	1300
21	Tech	andreea	41	861 bedford ave	1100

Run SQL query/queries on table ist210mrg415.emp:

```
insert into emp(id,emptype,name,age,address,salary) values(22,"Prof","Rick",47,"11 Deerhill",1200)
```

+ Options

id	emptype	name	age	address	salary
1	*NULL*	matt	55	brian ct	1000
2	*NULL*	jane	43	main st	500
3	*NULL*	sue	50	walsh st	1100
4	*NULL*	tom	52	jones st	800
5	*NULL*	tim	50	jazz st	1800
6	*NULL*	karen	54	*NULL*	1500
20	*NULL*	jon	49	111 east ave	1300
21	Tech	andreea	41	861 bedford ave	1100
22		Rick	47	11 Deerhill	1200

The value will either be one of the enumerated values or a blank (not a null)

Unique Columns

Auto-increment feature in MySQL allows a unique number to be generated automatically when a new record is inserted into a table. This is often useful if we want to create A Surrogate Key for our database (remember that from week 4 ?)

A **primary key** is a special relational **database** table column (or c
of columns) designated to uniquely identify all table records.

A **primary key's** main features are: It must contain a unique value
each row of data. It cannot contain null values.

Let’s create a new table (called emp2) that contains the following items (columns)

ID (created unique)
Name
emptype

```
use ist210mrg415;

create table emp2( id int AUTO_INCREMENT ,
                                name varchar(50),
                                emptype
ENUM('Manager','Tech','Rep') default "Tech",
                                     primary key(id) );

insert into emp2 (name,emptype) values("matt","Tech");
insert into emp2 (name,emptype) values("karen","Manager");
insert into emp2 (name) values("fred");
```

```
SELECT * FROM `emp2`
```

Show all | Number of rows: 25 Filter rows: Search th

+ Options

←T→			id	name	emptype
Edit	Copy	Delete	1	matt	Tech
Edit	Copy	Delete	2	karen	Manager
Edit	Copy	Delete	3	fred	Tech

Notice the default value

Notice the ID autoincremented

```
create table emp2( id int AUTO_INCREMENT,
                                    name varchar(50),
                                    emptype enum('Manager','Tech','Rep')
default 'Tech',
                                    primary key (id),
                                    CONSTRAINT name_check unique(name));

insert into emp2(name,emptype) values("matt","Tech");
insert into emp2(name,emptype) values('karen","Manager");
insert into emp2(name,emptype) values('matt","Tech");
```

Error

SQL query:

```
insert into emp2(name,emptype) values('matt","Tech")
```

MySQL said:

```
#1062 - Duplicate entry 'matt' for key 'name_check'
```

```
create table emp2( id int AUTO_INCREMENT,
                                   first varchar(20) not
NULL,
                                   last  varchar(20) not
NULL,
                                   primary key(id),
                                  CONSTRAINT name_check
unique(first,last));
```

```
values("matt","ganis");
values("karen","ganis");
```

+ Options

	id	first	last
Edit Copy Delete	2	karen	ganis
Edit Copy Delete	1	matt	ganis

```
insert into emp2 (first,last) values("mat[illegible]is");
```

Error

SQL query:

```
insert into emp2(first,last) values("matt","ganis")
```

MySQL said:

```
#1062 - Duplicate entry 'matt-ganis' for key 'name_check'
```

+ Options

id	emptype	name	age	address	salary
1		matt	55	brian ct	1000
2		jane	43	main st	500
3		sue	50	walsh st	1100
4		tom	52	jones st	800
5		tim	50	jazz st	1800
6		karen	54	*NULL*	1500
20		jon	49	111 east ave	1300
21		andreea	41	861 bedford ave	1100
22		Rick	47	11 Deerhill	1200
23		Fred	53	LoneStar Highway	1900

Run SQL query/queries on table ist210mrg415.emp:

```
update emp set emptype="Manager" where id=1;
update emp set emptype="Manager" where id=2;
update emp set emptype="Rep" where id=3;
update emp set emptype="Tech" where id=4;
update emp set emptype="Tech" where id=5;
update emp set emptype="Manager" where id=6;
update emp set emptype="Tech" where id=20;
update emp set emptype="Rep" where id=21;
update emp set emptype="Tech" where id=22;
update emp set emptype="Manager" where id=23;
```

+ Options

id	emptype	name	age	address	salary
1	Manager	matt	55	brian ct	1000
2	Manager	jane	43	main st	500
3	Rep	sue	50	walsh st	1100
4	Tech	tom	52	jones st	800
5	Tech	tim	50	jazz st	1800
6	Manager	karen	54	*NULL*	1500
20	Tech	jon	49	111 east ave	1300
21	Rep	andreea	41	861 bedford ave	1100
22	Tech	Rick	47	11 Deerhill	1200
23	Manager	Fred	53	LoneStar Highway	1900

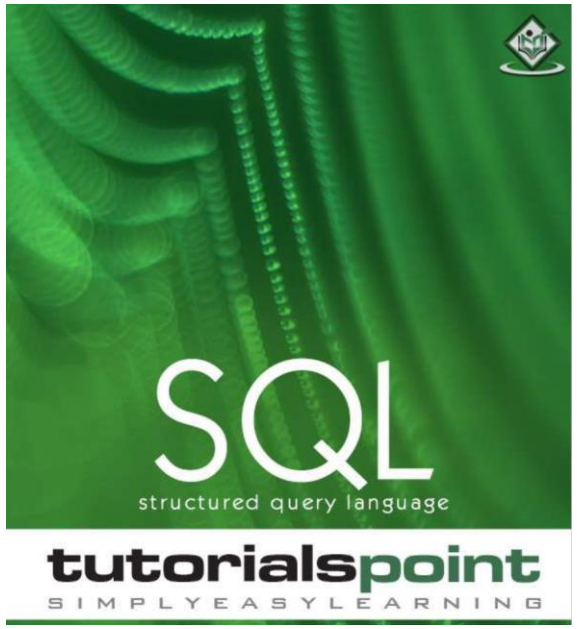
Select Statements

How many records in our database ?

```
select count(*) from emp;
```

+ Options

count(*)
10



Look here starting at page 47

How many managers do we have ?

```
Select count(emptype) from emp where emptype="Manag
```

+ Options

count(*)
4

Create a different header in our report table – for example if we want to know how many managers we have, rather than the header being count(*) like above we want

+ Options

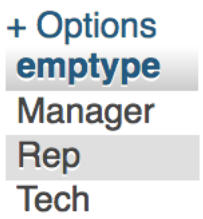
manager
4

Use: `select count(emptype) as "manager" from emp where emptype="Manager"`

Select Statements

How many different types of employees do we have ?

```
select distinct emptype from
```




```
SELECT count(distinct emptype) from emp  (will show just a count or 3)
```

Which employee types are making more than $1500 ?

```
select distinct emptype from emp where salary>1500
```

Select Statements with MySQL functions

What is our total payroll ?

```
select sum(salary) from emp;
```

+ Options

sum(salary)
12200

What is our total payroll for just the Tech's in the company ?

```
select sum(salary) from emp where emptype='
```

+ Options

sum(salary)
5100

What is the average salary of our Managers ?

```
SELECT avg(salary) from emp where emptype='Manag
```

+ Options

avg(salary)
1225

You are currently employed in a vintage record store (you know, those old, round, black things that would produce music ;-). Your boss would like you to create a database for the store’s inventory. Currently the media you carry are: Albbums (Long Playing albums), Singles (smaller records) and CD’s (compact disc’s). In anticipation of needing a unique key, you should assign an ID to each entry (start the ID values at 1000) as a database key

The current inventory is listed on the next page - the number in front of each name is the quantity in stock and price is in parenthesis

Be sure to define the database in a way that ensures we don’t have any data complications (ie, data correctness) - be sure there are not duplicate entries made (artist and media type)

Create a single table database to represent this data

Artist: Journey

Albums	CD's	Singles
3 - Escape ($19.99)	5 - Revelation ($15.99)	
10 - Frontiers ($20.15)	0 - Infinity ($16.50)	
2- Trial by Fire ($15.99)		

Artist: Kiss

Albums	CD's	Singles
2 - Love Gun ($19.99)	3 - Dressed to Kill ($13.99)	0 - Beth ($12.99)
8 - Destroyer ($19.99)		4 - Detroit Rock City ($14.99)
3 - Alive 2 ($25.99)		6- Rock and Roll all Night ($12.99)

Artist: Elton John

Albums	CD's	Singles
3 - Goodbye Yellowbrick Road ($30.00)	3 - Goodbye Yellowbrick Road ($23.99)	3 - Harmony ($12.99)
2 - Captain Fantastic ($18.00)	6 –Rock of the Westies ($21.99)	2 - Someone Saved my Life ($11.99)
3- Elton John ($15.99)		5 - Your Song ($16.99)
0 - Caribou ($21.99)		

Create Database queries to answer these questions:

What stock items do we need to reorder (ie, we have none) ?

In order to sell more, perhaps we need to lower the price on items where we have 5 or more. Which ones ?

How many total Singles, Records and CD’s do we have in stock ?