The Ultimate MySQL Bootcamp: Go from Expert to Pro

Link: Online Sql Tool

slide-1: presentation

slide-1 contains: create db, insert, drop db, primary key, auto-increment, not null, default, not null default.

slide-2: presentation also known as Refining Selections

slide-2 contains: string functions, distinct, order by , limit, like,

Correlated subqueries: Correlated subqueries

Note about the SQL editor

from the next lecture will give you errors if you're using firefox or any other browser other than Chrome, Safari, or Opera. This is because the SQL Try-It Editor is using WebSQL which isn't compatible with all browsers.

We recommend using chrome for those exercises and for the remainder of the course, if possible.

First five minutes of SQL

Presentation link: Presentation


```
SELECT first_name,
       last_name,
       Count(rating)                        AS COUNT,
       Ifnull(Min(rating), 0)               AS MIN,
       Ifnull(Max(rating), 0)               AS MAX,
       Round(Ifnull(Avg(rating), 0), 2) AS AVG,
       CASE
         WHEN Count(rating) >= 10 THEN 'POWER USER'
         WHEN Count(rating) > 0 THEN 'ACTIVE'
         ELSE 'INACTIVE'
       end                                  AS STATUS
FROM   reviewers
       LEFT JOIN reviews
              ON reviewers.id = reviews.reviewer_id
GROUP  BY reviewers.id;
```

Getting started overview and installation

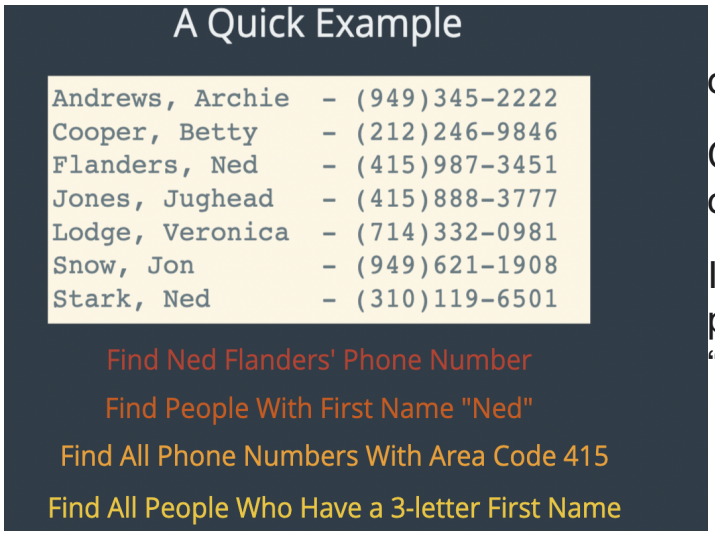
What is a db?

Figure 1: Phonebook

Another e.g of db is a phonebook.

Something archaic like this is Rolodex.

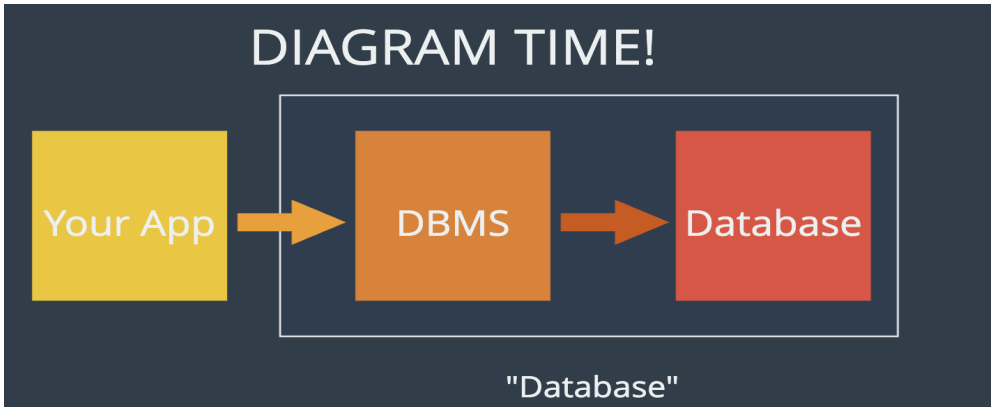
Figure 2: rolodex

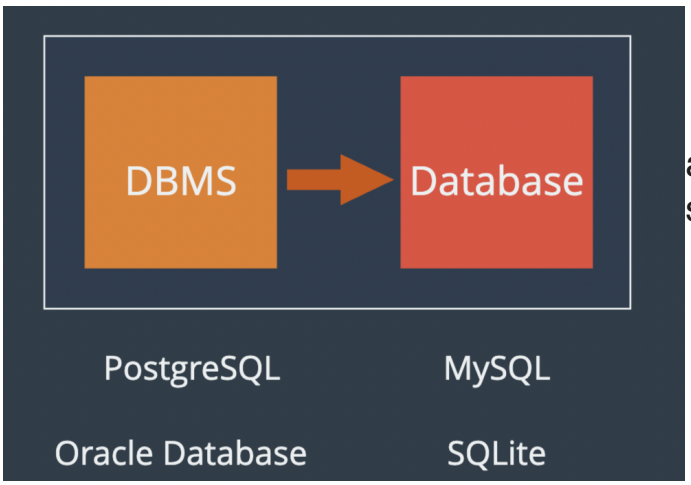


Think of phonebook as a dictionary.

Considering it is ordered based on Last name.

It's a lot difficult to find all people with the First Name "Ned".



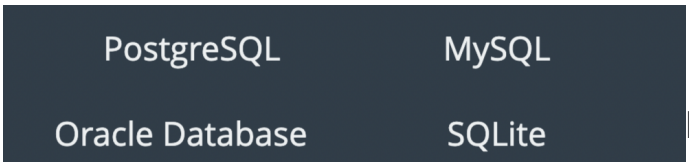


are database management system.

SQL VS MYSQL

Sql is the language we use to talk to db.

Crud basically.



Uses (SQL)structured query language.

Point is sql is not unique to mysql, that's why they all have SQL in the name.

They shared the **same** language.

MYSQL VS PostgreSQL

Are different dbms, but they both use Sql.

You'll be writing sql when you interact with this database.

There is slight difference in Syntax, they are not identical.

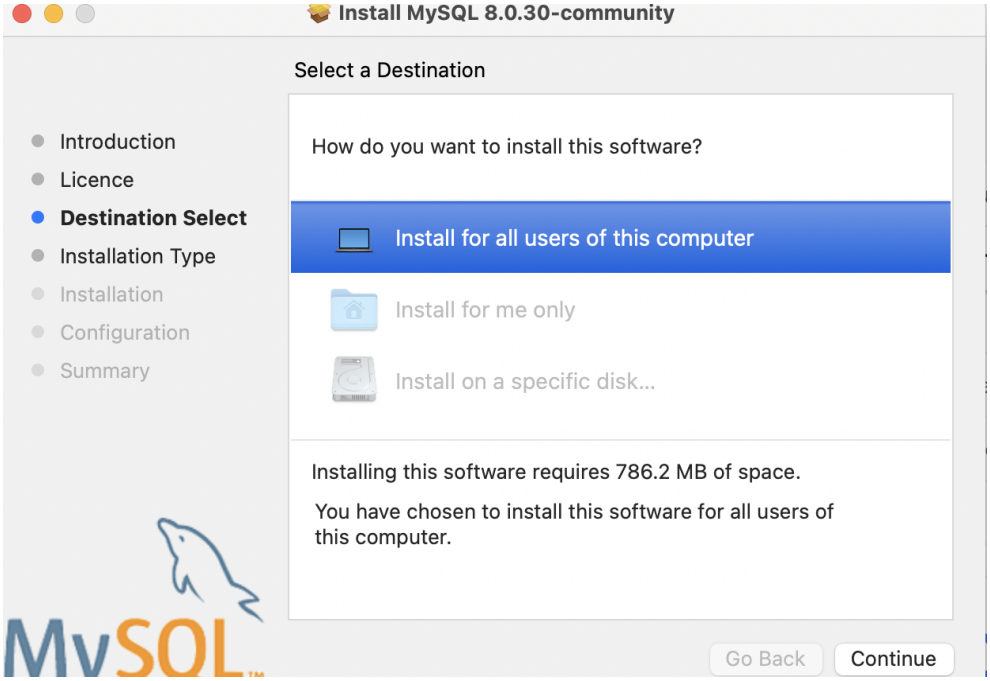
There is a sql standard and all this dbms are tasked with implementing the standard.

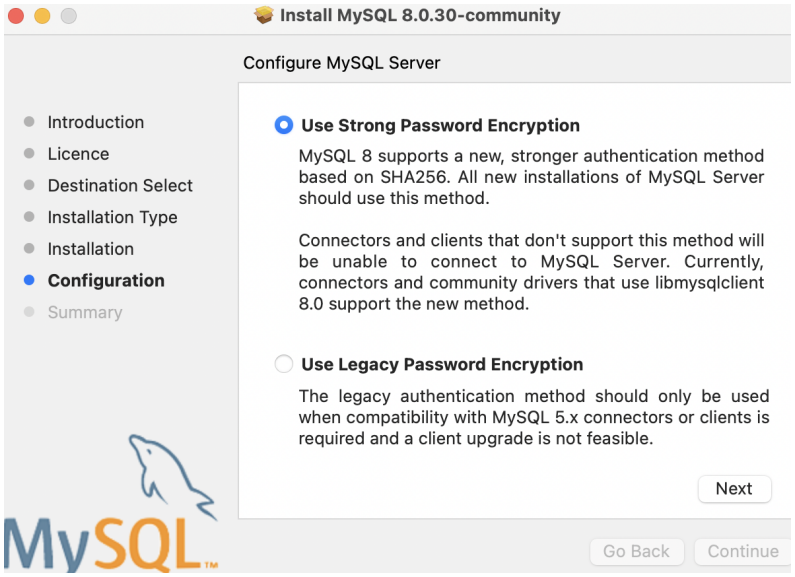
Two takeaways

1. It's preety easy to switch.

2. what makes dbms unique, Is the feature they offer. Not the language.

Factors like secure, fast, download, permission works


Install MySQL 8.0.30-community
Select a Destination
Introduction
Licence
Destination Select
Installation Type
Installation
Configuration
Summary
How do you want to install this software?
Install for all users of this computer
Install for me only
Install on a specific disk...
Installing this software requires 786.2 MB of space.
You have chosen to install this software for all users of this computer.
MySQL
Go Back
Continue


Install MySQL 8.0.30-community
Configure MySQL Server
Introduction
Licence
Destination Select
Installation Type
Installation
Configuration
Summary
Use Strong Password Encryption
MySQL 8 supports a new, stronger authentication method based on SHA256. All new installations of MySQL Server should use this method.
Connectors and clients that don't support this method will be unable to connect to MySQL Server. Currently, connectors and community drivers that use libmysqlclient 8.0 support the new method.
Use Legacy Password Encryption
The legacy authentication method should only be used when compatibility with MySQL 5.x connectors or clients is required and a client upgrade is not feasible.
Next
MySQL
Go Back
Continue

```
kazimsyed@kazims-MacBook-Air ~
% ls /usr/local/mysql/bin
ibd2sdi                                 mysql_upgrade
innochecksum                            mysqladmin
libprotobuf-lite.3.19.4.dylib           mysqlbinlog
libprotobuf.3.19.4.dylib                mysqlcheck
lz4_decompress                          mysqld
my_print_defaults                       mysqld-debug
myisam_ftdump                           mysqld_multi
myisamchk                               mysqld_safe
myisamlog                               mysqldump
myisampack                              mysqldumpslow
mysql                                   mysqlimport
mysql_config                            mysqlpump
mysql_config_editor                     mysqlshow
mysql_migrate_keyring                   mysqlslap
mysql_secure_installation               perror
mysql_ssl_rsa_setup                     zlib_decompress
mysql_tzinfo_to_sql
kazimsyed@kazims-MacBook-Air ~
% path[8]=!:1
path[8]=/usr/local/mysql/bin
```

```
kazimsyed@kazims-MacBook-Air ~
% mysql bhavans9911
ERROR 1045 (28000): Access denied for user 'kazimsyed'@'l
rd: NO)
kazimsyed@kazims-MacBook-Air ~
% mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 8.0.30 MySQL Community Server - GPL
```

If u forgot ur password, u can change it.

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)
```

```
mysql> use mysql
```

```
mysql> describe user;
+------------------------+-----------------------------------+------+-----+-----------------------+------
| Field                  | Type                              | Null | Key | Default               | Extra
+------------------------+-----------------------------------+------+-----+-----------------------+------
| Host                   | char(255)                         | NO   | PRI |                       |
| User                   | char(32)                          | NO   | PRI |                       |
| Select_priv            | enum('N','Y')                     | NO   |     | N                     |
| Insert_priv            | enum('N','Y')                     | NO   |     | N                     |
| Update_priv            | enum('N','Y')                     | NO   |     | N                     |
| Delete_priv            | enum('N','Y')                     | NO   |     | N                     |
| Create_priv            | enum('N','Y')                     | NO   |     | N                     |
| Drop_priv              | enum('N','Y')                     | NO   |     | N                     |
| Reload_priv            | enum('N','Y')                     | NO   |     | N                     |
```

I need to checkout the mac installatio to learn

ALTER USER ‘root@localhost’ identified by ‘password ’

Creating databases and tables

Creating databases

How to work with databases and tables:

Create database [if not exists] <db_name>;	Create database
Drop database [if exists] <db_name>;	Drop database
Drop table [if exists] <db_name>.<table_name>	Drop table while being outside the database.

Best practices:

1. Multiple db maybe running on a single Mysql server, you usually want to select the database before you begin working with it.
2. For large database project, DDL statements would be handled by a database specialist or DBA.
3. For small project, SQL programmer have to serve as DBA.
4. And for small projects, SQL programmer have to use DDL statements to create smaller tables that are needed for testing.

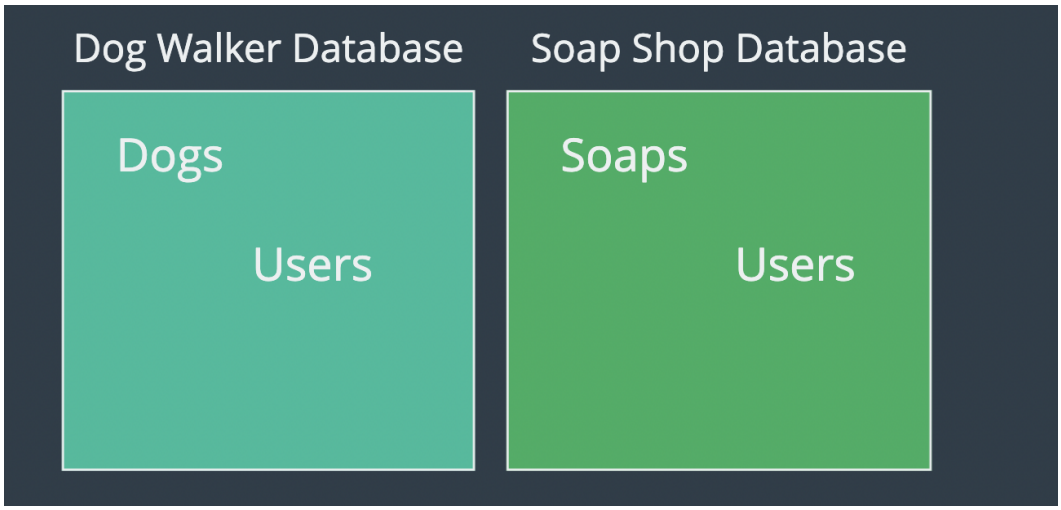


Figure 3: show databases;

Our dog walking app has users. But so does our Soap shop.

Our dog walking app has payments. But so does our Soap shop app.

They have to be **seperated**.

If we have one db and everything was using on this server, there would be a lot of crossovers and issues.

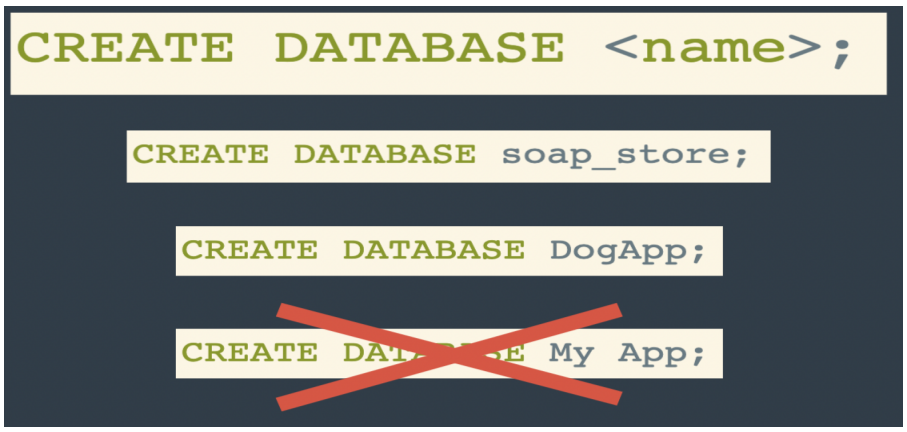


Figure 4: space isnt allowed, use underscore instead or Pascal casing.

```
DROP DATABASE <name>;
```

```
USE <database name>;
```

```
SELECT database();
```

A database is just a bunch of tables

In a relational database, at least

Giant ER diagram: [ER diagram]

Datatypes

Numeric Types	String Types	Date Types
• INT	• CHAR	• DATE
• SMALLINT	• VARCHAR	• DATETIME
• TINYINT	• BINARY	• TIMESTAMP
• MEDIUMINT	• VARBINARY	• TIME
• BIGINT	• BLOB	• YEAR
• DECIMAL	• TINYBLOB	
• NUMERIC	• MEDIUMBLOB	
• FLOAT	• LONGBLOB	
• DOUBLE	• TEXT	
• BIT	• TINYTEXT	
	• MEDIUMTEXT	
	• LONGTEXT	
	• ENUM	

INT

A Whole Number

with a max value of 4294967295

varchar

A Variable-Length String

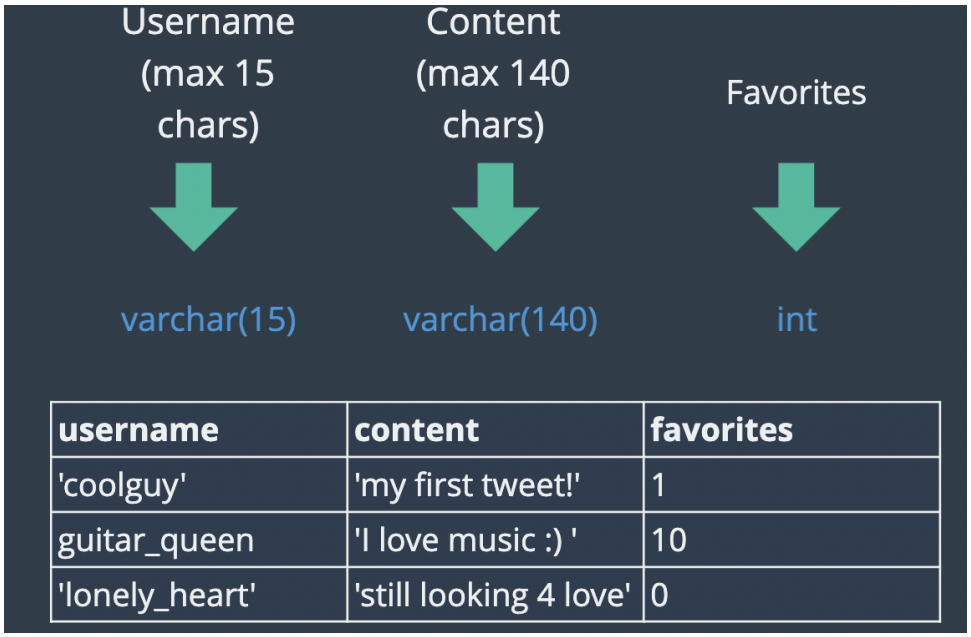
Between 1 and 255 characters

Draw a Tweets Table

At a minimum the columns must include:

- A username (max 15 chars)
- The tweet content (max 140 chars)
- Number of favorites

Make sure to specify correct MySQL datatypes!



username	content	favorites
'coolguy'	'my first tweet!'	1
guitar_queen	'I love music :) '	10
'lonely_heart'	'still looking 4 love'	0

```
CREATE TABLE tablename
  (
     column_name data_type,
     column_name data_type
  );
```

COPY

```
CREATE TABLE cats
  (
     name VARCHAR(100),
     age  INT
  );
```

Commands	Description
Show tables;	See tables inside the database.
Select columns from table_nm; or desc table_nm	Learn about table
Drop table;	Delete table
Insert into table_nm (col_1, col_2) values (< >,< >);	Insert values into a table
Insert into table_nm (col_1, col_2) values (< >,< >),(< >,< >), ...	Multiple inserts
Show warnings;	

Syntax of the Create table statement:

```
CREATE TABLE [db_name].table_name

(

        column_1 datatype [column attributes]

        [, column_2 datatype [column attributes] ]

[, table_level_constraints]

)
```

common column attributes

Attribute	Description
Not Null	Indicates that col doesn't accept null values. If ommited the column can accept null values.
Unique	Specifies that each value in the col must be unique.
Default default_value	Specifies a default value for the column.
auto_increment	Identifies a column whose value is automatically incremented by Mysql when row is added. Auto-increment col can be (float or int)

To indicate that each row in a column must contain a unique value, you can code the unique attribute.

Since two null values arent considered the same, a unique column can contains null values.

Best practice: to use not null and unique attribute.

--

Autoincrement: mysql starts numbering with 1, you can start with a value other than 1 by coding the auto_increment attribute like:

auto_increment=3

Auto-increment value can be specified for only one column in the table and that column can be defined as either the primary key or unique key.

```
SHOW TABLES;
```

```
SHOW COLUMNS FROM <tablename>;
```

Or...

```
DESC <tablename>;
```

Deleting Tables

```
DROP TABLE <tablename>;
```

INSERT

```
INSERT INTO cats(name, age)
VALUES ('Jetson', 7);
```

--

MULTIPLE INSERT

```
INSERT INTO cats(name, age)
VALUES ('Charlie', 10)
      ,('Sadie', 3)
      ,('Lazy Bear', 1);
```

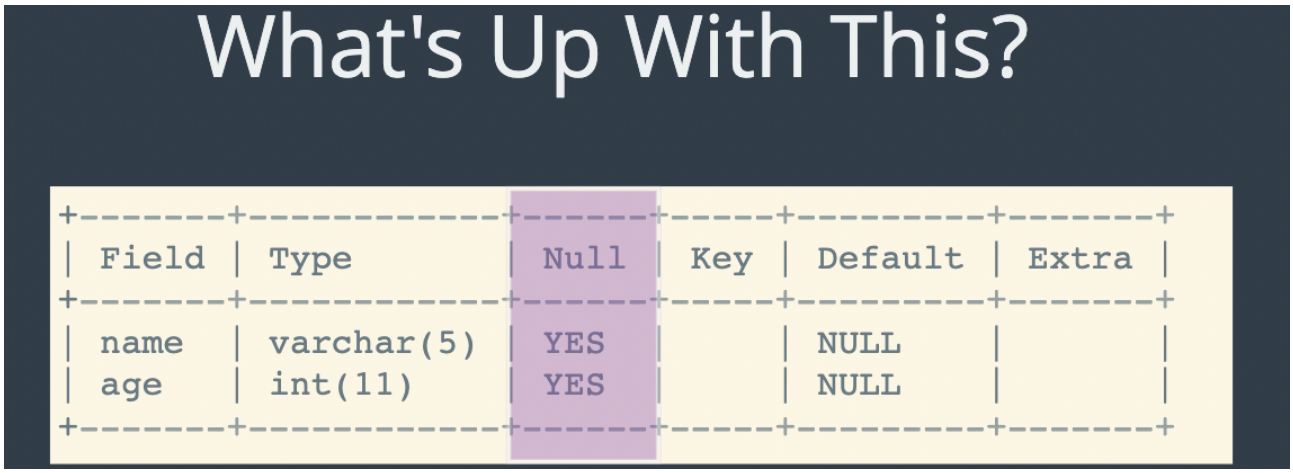
--

Warnings:


1 Warning??!
Query OK, 1 row affected, 1 warning (0.01 sec)


Let's Take A Look
SHOW WARNINGS;

Not null, default, not null



-The value isnt known.

-Null does not mean zero.

Right now, we could do this...

```
INSERT INTO cats(name)
VALUES ('Alabama');
```

Or This! *gasp*

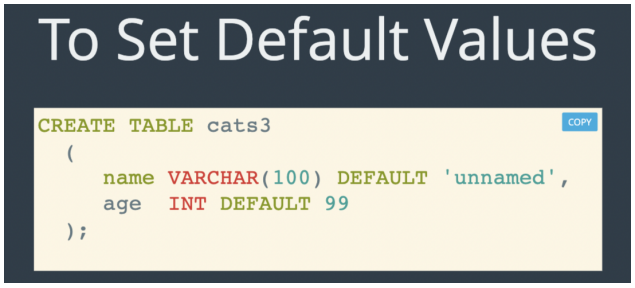
```
INSERT INTO cats()
VALUES ();
```

```
CREATE TABLE cats2
  (
     name VARCHAR(100) NOT NULL,
     age  INT NOT NULL
  );
```

Notice The Difference!

```
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| name  | varchar(100) | NO   |     | NULL    |       |
| age   | int(11)      | NO   |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
```

Figure 5: Use of not null keyword



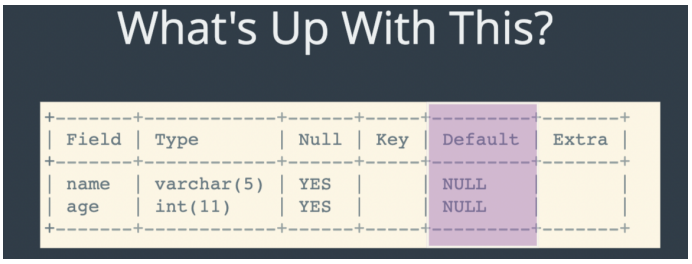
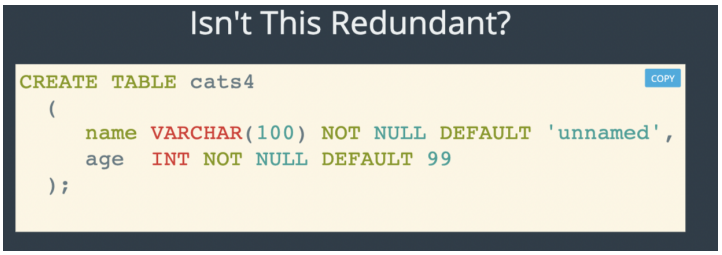
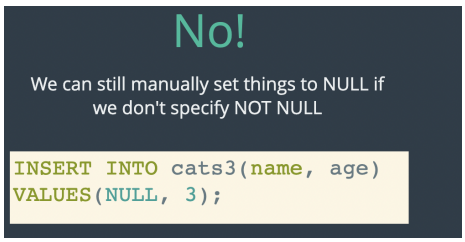


Figure 6: Use of default keyword. Here we can see that we can manually pass NULL values.

-- so u may feel:





Primary key:

What's Up With This?

```
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| name  | varchar(5) | YES  |     | NULL    |       |
| age   | int(11)    | YES  |     | NULL    |       |
+-------+------------+------+-----+---------+-------+
```

Primary Key
A Unique Identifier

Right now, this could happen!

Name	Breed	Age
Monty	Tabby	10
Monty	Tabby	10
Monty	Tabby	10
Monty	Tabby	10

How Do We Make Each Unique?

Name	Breed	Age	CatID
Monty	Tabby	10	1
Monty	Tabby	10	2
Monty	Tabby	10	3
Monty	Tabby	10	4

Code:

```
CREATE TABLE unique_cats2 (cat_id INT NOT NULL AUTO_INCREMENT
                          ,name VARCHAR(100)
                          ,age INT
                          ,PRIMARY KEY (cat_id));
```

Question:

Table 1: sol-1)when you define a constraint in a column definition is known as column level constraint. You can also define a constraint at the table level using the constraint keyword. when you code a table level constraint, you can provide a name for the constraint. Reasons for using a name for a table-level constraint: reasons to name a constraint

The syntax of a column-level primary key constraint:

column_name data_type primary key column_attributes

The syntax of a table-level primary key constraint

[constraint [constraint_name]]
primary key (**col_1**[,**col_2**]...)

A table with column level constraints

```
Create table vendors
(
vendor_id          INT PRIMARY KEY AUTO_INCREMENT,

Vendor_name    VARCHAR(50) NOT NULL UNIQUE
)
```

A table with table-level constraints

```
Create table vendors
(
vendor_id   int auto-increment
vendor_name varchar(50) not null

constraint vendors_pk primary key ( vendor_id),
constraint vendor_name_uq UNIQUE (vendor_name)
)
```

A table with two_column primary key constraints

```
create table invoice_line_items
(
invoice_id int not null,
invoice_sequence int not null,
line_item_description varchar(100) not null,

constraint line_items_pk primary key( invoice_id, invoice_sequence)

)
```

Here's some pretty basic reasons.

(1) If a query (insert, update, delete) violates a constraint, SQL will generate an error message that will contain the constraint name. If the constraint name is clear and descriptive, the error message will be easier to understand; if the constraint name is a random guid-based name, it's a lot less clear. Particulary for end-users, who will (ok, might) phone you up and ask what "FK__B__B_COL1__75435199" means.

(2) If a constraint needs to be modified in the future (yes, it happens), it's very hard to do if you don't know what it's named. (ALTER TABLE MyTable drop CONSTRAINT um...) And if you create more than one instance of the database "from scratch" and use system-generated default names, no two names will ever match.

(3) If the person who gets to support your code (aka a DBA) has to waste a lot of pointless time dealing with case (1) or case (2) at 3am on Sunday, they're quite probably in a position to identify where the code came from and be able to react accordingly.

Figure 7: reasons to name a constraint

How to code a foreign key constraint:

The syntax of a column-level foreign key constraint

[column attributes] references table-name(column_name)

[ON DELETE (CASCADE I SET NULL)]

The syntax of a table-level foreign key constraint

[constraint constraint_name]

FOREIGN KEY (col_name [,col_name] …)
Referecnes table_name (col_name [,col_name] …)
[ON DELETE (cascade I set null)

It also let you to set foreign key for multiple columns.

e.g

A table with column-level foreign key constraint

```
Create table invoices
(
invoice_id   int   primary key
vendor_id   int   references vendors ( vendor_id ),
invoice number varchar(50)  not null unique

)
```

A table with table-level foreign key constraint

```
create table invoices
(
invoice_id int primary key,
vendor_id int not null,
invoice number varchar not null unique,

constraint invoices_fk_vendors
   foreign_key (vendor_id) references vendors (vendor_id)
)
```

A constraint that uses the ON DELETE clause

```
constraint invoices_fk_vendors
foreign_key (vendor_id) references vendors (vendor_id)
ON DELETE CASCADE
```

Foreign key enforces referential integrity.

In some cases though, you may want to automatically delete the related rows in the invoices table when a row in the vendors table is deleted.

To do that, you can code the on delete clause on the foreign key as in the last example.

So, when you delete a row from the priamary key table , the delete is cascaded to the related rosw, in the foreign key table.

For e.g you delte a row from the vendors tbale, all related rows in the invoices table will also be deleted.

CAUTION:

Cascading delete makes it easier to delete data that you didn't intend to delete, use it with caution.

You can also use not SET NULL On the ON DELETE clause.

So when you delete a row from a primary key table, the values in the foreign key column of foreign key table are set to null.

Since this creates rows in the foreign key table that arent related to the primary key table, you rarely want to use this option.

How to alter the columns of a table

After you create tables, you may need to change the columns of table.

Not table cuz for table you can use create or replace table while being cautius of the data in the table and relationships.

You may need **amd**:

add, modify, delete.

Syntax for modifying the columns of a table.

Table 2: MySql won't allow you to change a column if that change would cause data to be lost.

ALTER TABLE [db_name.]table_name (ADD column_name data_type [column_attributes] l DROP COLUMN column_name l MODIFY column_name datatype [column_attribute])

Warning: You should never alter or other database object in a production database without first consulting the DBA.

How to alter constraints of a table.

After creating a table, you may need to add or drop a constraint.

The syntax for modifying the constraints of a table. Alter table [db_name.]table_name (add primary key constraint_definition l add [constraint constraint_name] foreign key constraint definition l drop primary key drop foreign key constraint name.)
Mysql may not allow you to drop the primary key for a table. That's true if the primary key is an auto-increment column or it's referred to by a foreign key.
A table can have multiple foreign key, you must know the name of the key you want to drop. If you don't know its name, you can use MySQL Workbench to look It up.

How to rename, truncate, and drop a table.

It is useful if you want to change the name of a table without modifying its column definitions or the data that's stored in the table.

Rename table vendors to vendor	Renaming a table
Truncate table vendor OR Delete * from vendor	Truncate to delete all of the data from a table without deleting the column definitions of the table.

Note:

1-You can't truncate or drop a table if a foreign key constraints in another table refers to that table.

2-When you drop a table, all of its data, constraints, and indexes are deleted.

Warnings:

You shouldn't use these statements on a production database without first consulting the DBA.

Strategies for backing up and restoring a database

PENDING

MySQL DATA TYPES

DATE TYPE	SPEC	DATA TYPE	SPEC
CHAR	String (0 - 255)	INT	Integer (-2147483648 to 214748-3647)
VARCHAR	String (0 - 255)	BIGINT	Integer (-9223372036854775808 to 9223372036854775807)
TINYTEXT	String (0 - 255)	FLOAT	Decimal (precise to 23 digits)
TEXT	String (0 - 65535)	DOUBLE	Decimal (24 to 53 digits)
BLOB	String (0 - 65535)	DECIMAL	"DOUBLE" stored as string
MEDIUMTEXT	String (0 - 16777215)	DATE	YYYY-MM-DD
MEDIUMBLOB	String (0 - 16777215)	DATETIME	YYYY-MM-DD HH:MM:SS
LONGTEXT	String (0 - 4294967295)	TIMESTAMP	YYYYMMDDHHMMSS
LONGBLOB	String (0 - 4294967295)	TIME	HH:MM:SS
TINYINT	Integer (-128 to 127)	ENUM	One of preset options
SMALLINT	Integer (-32768 to 32767)	SET	Selection of preset options
MEDIUMINT	Integer (-8388608 to 8388607)	BOOLEAN	TINYINT(1)

LOB (LARGE OBJECT) data types that are useful for storing images, sound, video, and large amounts of character data.

We can divide numbers into integers and real numbers. Integers are numbers that don't have a decimal point, and real numbers are numbers that have a decimal point.

Table 3: this tables shows how unsigned and zerofill attributes work with negative and positive number.

Data type	Original value	Value stored	Value displayed
INT	99	99	99
INT	-99	-99	-99
INT UNSIGNED	99	99	99
INT UNSIGNED	-99	error	error
INT ZEROFILL	99	99	0000000099
INT(4) ZEROFILL	99	99	0099

The BOOL AND BOOLEAN types are synonyms for TINYINT(1).

The fixed-point and floating point types

The fixed point type:

DECIMAL(M,D) : M specifies the maximum of total digit, D specifies the number of digits to the right of the decimal. M>D. By default D=0.

The floating point type:

Float(4 bytes), double(8 bytes)

--

Description:

real number unlike integers can store negative values in the column.

If the zero fill attribute for a real number is set, the number is displayed with zeroes padded from the left, and the unsigned attribute is automatically set.

DEC, NUMERIC, FIXED are all synonyms for the DECIMAL TYPE
REAL is a synonym for Double type

The date and time types

DATE (YYYY-MM-DD), TIME(HH:MM:SS) , DATETIME(YYYY-MM-DD HH:MM:SS) , TIMESTAMP, YEAR[(2|4)]

The interesting one that need explanation is

YEAR(2|4) I.e Year in 2-digit or 4-digit format.

How MySQL interprets literal date/time values

Literal value	Value stored in Date column
'2000-05-27'	2000-05-27
'2000-5-27'	2000-05-27
'00-5-27'	2000-05-27
20000527	2000-05-27
2000.05.27	2000-05-27
00/5/27	Error
'2000-27-5'	Wrong format YYYY-MM-DD

Literal value	Value stored in time column
'7:32'	07:32:00
'19:32:11'	19:32:11
193211	19:32:11

Literal value	Value stored in datetime in timestamp column
'2000-05-27 12:00:00'	2000-05-27 12:00:00
'2000-05-27'	'2000-05-27 00:00:00'

The ENUM and SET TYPES

The main difference between the ENUM and SET types is that an ENUM column can store exactly one value, but a SET column can store, zero, one or upto 64 different values.

You can use ENUM type to store values that are mutually exclusive, such as Yes, No or Maybe.

IOW, you can use the ENUM type to represent a choice of one value, but not two.

For e.g, delivery or pickup, cash, credit, or debit, small, medium, or large; paper or plastic; soup or salad.

Although I assume you might want both soup and salad. For that, you could use a set column.

You can use a SET column when you want to choose more than one value. For e.g, the toppings on a pizza, the software on a computer, or the features of a car could be SET values.

To store a value for an ENUM column, you code a single string.

When you add a row to a table that contains an ENUM column, MySQL assigns a default value to that column, If you don't explicitly specify a value. If the column allows null values, MySQL assigns a null value to the column.

If the column doesn't allow null values, MySQL assigns the first value as the default value, then, you'll want to code that value as the first valu in the set.

To store values in a SET column, you code a single string with one or more values seperated by commas. Then, MySQL stores each acceptable value and ignores any other values. Since commas are used to seperate values, you cant use commas within a value when you define the SET column.

When storing multiple values in a SET column, the order of the values doesn't matter. That's because MySQL stores the values in the same order as in column definition.

It doesn't matter if you repeat a value because MySQL doesn't store duplicate values.

Table 4: How values are stored in SET columns

Value	Stored in column SET(‘Pepperoni’,’Mushrooms’,’Olives’)
‘Pepperoni’	‘Pepperoni’
‘Mushrooms’	‘Pepperoni’
‘Pepperoni,’Corn’	‘Pepperoni’
‘Olives,Pepperoni’	‘Pepperonoi,Olives’

The large object types

BLOB (Binary Large object) types store strings of binary data. This data type is often used to store any type of binary data.

The TEXT types work similarly to the BLOB types, they store strings of character. As a result, in other database systems, they are sometimes referred to as CLOB.

TEXT upto 65kb in length.

The basic syntax of the SELECT statement

Clause	Description
SELECT	Describes the columns in the result set.
From	Name the base table form which the query retrieves the data.
Where	Specifies the condition that must be met for a row to be included in the result set
ORDER BY	Specifies how to sort the rows in the result set
LIMIT	Specifies the number of rows in return

Select invoice_number, invoice_date, invoice_total from invoices order by invoice_total DESC
Select invoice_id, invoice_total, credit_total + payment_total as total_credits from invoices where invoice_id=17
Select invoice_number, invoice_date, invoice_total from invoices where invoice_date Between ‘2011-6-01’ and ‘2001-06-30’
Select invoice_number, invoice_date, invoice_total from invoices where invoice_total > 5000

Source	Option	Syntax
Base table value	All columns Column name	* column_name
Calculated value	Result of a calculation Result of a function	Arithmetic expression e.g Select invoice_id, invoice_total, **credit_total + payment_total as total_credits** from invoices where invoice_id=17 Functions e.g select concat(fname,’ ‘,lname) as full_name;

How to insert default values and null values

If a column allows null values, you can use the INSERT statement to insert a null value into that column.

To insert a null value into a column, you can use the NULL keyword.

If a column is defined with a default value, you can use the insert statement to insert that value.

Finally, if a column is defined as an auto increment column, you can have MySQL generate a value for that column.

To insert a default value or to have MySQL generate a value for an auto_increment column, you can use the default keyword.

```
Insert into color_sample values (DEFAULT, DEFAULT, NULL)
```

How to update exisiting rows

The syntax of the update statement

```
UPDATE table_nm
set column_name=expression [, column_name=expression]…
[ where search_condition ]
```

By default, MySQL Workbench runs in safe update mode. This prevents you from updating rows if the Where clause is ommited or doesn't refer to a primary key or foreign key column.

To get around the restrictions of safe update mode, you can turn this mode off.

To do that, select the Edit → Preferences command, select the SQL editor tab, change the "safe update" option, and restart the MySQL Workbench.

Warning:

If you turn off safe update mode and omit the where clause, all rows in the table will be updated.

How to delete existing rows

To delete one or more rows from a table, you can use the DELETE statement. If necessary, you can use subqueries in a delete statement to help identify the rows to be deleted.

How to delete rows

By default, MySQL Workbench runs in safe update mode. This prevents you from updating rows if the Where clause is ommited or doesn't refer to a primary key or foreign key column.

To get around the restrictions of safe update mode, you can turn this mode off.

To do that, select the Edit → Preferences command, select the SQL editor tab, change the "safe update" option, and restart the MySQL Workbench.

Warning:

If you turn off safe update mode and omit the where clause, all rows in the table will be updated.

If you want to make sure that you've selected the correct rows before you delete them, you can code a select statement that retrieves those rows.

Then, once you're sure the select statement is retrieving the correct rows, you can conver the select statement to a delete statement.

How to use a subquery in a delete statement

```
Delete from invoice_line_items
where invoice_id in
(
select invoice_id
from invoices
where vendor_id=115 )
```

An introduction to subqueries

A subquery is a select statement that's ended within another sql statement.

Where to code subqueries?

a subquery can be coded, introduced in the WHERE, HAVING, FROM, SELECT clause of a select statement.

-When a subquery returns a single value, you can use it anywhere you would normally use a single value. However, a subquery can also return a list of values(a result set that has one column) using the IN operator.

-A subquery can return a table of values(a result set that has multiple columns). In that case, you can use the subquery in the **FROM** clause.

Four ways to introduce a subquery in a select statement.

-In a WHERE clause as a search condition

-In a HAVING clause as a search condition

-In a FROM clause as a table specification

- In the select clause as a column specification.

-subqueries can be nested within other subqueries.

-A subquery can't include an ORDER BY clause.

You need to treat the subquery in the `FROM` clause, as a set of rows in some unspecified and undefined order, and put the `ORDER BY` on the top-level `SELECT`.

Workaround: I have discussed this in Onenote book and its using limit. But execution time increase.

When to use subqueries

Subqueries provides a way of processing that can't be done any other way.

However most subqueries can be restated as joins and most joins can be restated as subqueries.

A query that uses an inner join

```
select invoice_number, invoice_date, invoice_total
from invoices join vendors
on invoices.vendor_id= vendors.vendor_id
order by invoice_date
```

Same query reinstated with a subquery

```
Select invoice_number, invoice_date, invoice_total
from invoices where vendor_id in (select vendor_id from vendors)
order by invoice_date
```

So if you have a choice, which technique should you use?

In general, we recommend that you use the technique that results in the most readable code.

For e.g join tends to be more intuitive than a subquery when it uses an existing relationship between two tables.

On the other hand, subquery tends to be more intuitive when it uses an ad hoc relationship.

Another good point

you should realize that when you use a subquery in a where clause, its results cant be included in the final result set.

For instance, the second e.g above cant include the vendors name from the vendors table. That's because the vendors table isnt named in the FORM clause of the main query(saying select col_nm **from players**). So if you need to include information from both tables in the result set, you need to use a join.

Advantages of joins:

- Select clause of a join can include columns from both tables.

- A join tens to be more intuitive when it uses an exisiting relationship between the two tables, such as primary key to foreign key relationship.

Advantages of subqueries:

-You can use a subquery to pass an aggregate value to the main query.

-A subquery tends to be more intuitive when it uses an adhoc relationship between the two tables.

-long, complex queries can be sometimes be easier to code using subqueries.

How to code subqueries in the WHERE clause

How to use the IN OPERATOR

In operator is used to test whether an expression is contained in a list of values.

One way to provide the list of values is to use a subquery.

Table 5: The syntax of a WHERE clause that uses an IN phrase

```
where text_expression [NOT] in (subquery)
```

e.g

Table 6 Subquery returns a list of ID's for each vendor that's in the invoices table. Then the main query returns some data about the vendors whose ID's aren't in the list

```
Get vendors without invoices

Select vendor_id, vendor_name, vendor_state
from vendors
where vendor_id not in
(
select distinct vendor id
from invoices)
order by vendor_id
```

Description:

-when you use the **in** operator, the subquery must return a single columns of values.

- A query that uses the NOT IN operator with a subquery can typically be restated as an outer join.

How to use the comparison operators

Table 7: Syntax of WHERE clause that use a comparision operator.

WHERE expression comparison_operator [some|any|all] (subquery)

Description:

If you code a search condition without the ANY, SOME, and ALL Keywords, the subquery must return a single value. --idk

If you include the ANY, SOME, or ALL keyword, the subquery can return a list of values.

** I think both the statements above are looking to something basic from the opposite directions.

For e.g if you have a single value as result of a subquery, you won't use any, some, all.

And if you have multiple values as a result of a subquery, you may use all, some, any

How to use the ALL keyword:

To use the ALL keyword to modify the comparison operator so the condition must be true for all the values returned by a subquery.

Condition	Equivalent expression	Desc
X > all(1,2)	X > 2	Evaluates to true if x is greater than the maximum value returned by the subquery
X < ALL(1,2)	X < 1	Evaluates to true if x is less than the minimum value returned by the subquery.
X = ALL(1,2)	X=1 and X=2	
X <> ALL(1,2)	X not in (1,2)	

Q.Get invoice larger than the largest invoice for vendor 34?

Also, get other info as well such as the name of the vendors.

So we will use a join as well.

To get the largest invoice of vendor 34.

select Max(invoice_total) from invoices where vendor_id=34.

But since for example sake we need a list of values.

select invoice_total from invoices where vendor_id=34.

```
Select vendor_name, invoice_number, invoice_total

from invoices I JOIN vendors on I.vendor_id = v.vendor_id

where invoice_total > ALL(select invoice_total from invoices where
vendor_id=34)

order by vendor_name;
```

> ALL(select invoice_total from invoices where vendor_id=34)

is basically equivalent to

> Max(invoice_total)

Descriptions:

if no rows are returned by the subquery, a comparision that uses the AL keyword is always true.

If all of the rows returned by the subquery contains a null value, a comparision that uses the ALL keyword is always false.

How to use the ANY and SOME keyword

How to use the ANY or SOME keywords to test whether a comparison is true for any of values returned by a subquery.

Get invoices smaller than the largest invoice for vendor 15.

Same way to find largest invoice, just like before.

This is the better way cuz easier read and recommended whenever possible

But for e.g sake.

select invoice_total from invoices where vendor_id=34.

```
Select vendor_name, invoice_number, invoice_total

from invoices I JOIN vendors on I.vendor_id = v.vendor_id

where invoice_total < ANY(select invoice_total from invoices where
vendor_id=34)

order by vendor_name;
```

How to use ANY keyword works:

Condition	Equivalenet expressions	Description
X>ANY(1,2)	X>1	
X<ANY(1,2)	X<2	
X=ANY(1,2)	X IN (1,2)	
X <> ANY(1,2)	(x <>1) or (x<> 2)	Evaluates to true if x is not equal to at least one of the values returned by the subquery.

Description:

If the subquery doesn't return any values, or if it only return null values, a comparison that uses the any keyword evaluates to false.

Correlated subqueries

Correlated subqueries To learn extensively click here.

However, think twice before using a correlated subquery in SQL. They can be slow,

Let's begin with an example of a *correlated subquery in SQL*. Suppose we want to find all employees with a salary higher than their average **departmental salary**. We would use the following query. Once again, I've bolded the subquery:

Code

```
SELECT
  lastname,
  firstname,
  salary
FROM employee e1
WHERE e1.salary > (SELECT avg(salary)
                   FROM employee e2
                   WHERE e2.dept_id = e1.dept_id)
```

Example 2: A correlated subquery in SQL

The main difference between a SQL correlated subquery and a simple subquery is that correlated subqueries reference columns from the outer table. In the above example, `e1.dept_id` iis a reference to the outer subquery table.

To identify a correlated query, just look for these kinds of references. If you find at least one, you have a SQL correlated subquery!

Let's look at another example. Suppose we want to obtain the names of departments that have more than 10 employees. We can use the following SQL correlated subquery:

Code

```
SELECT deptname
FROM department d1
WHERE 10 < (SELECT count(*)
            FROM employee e
            WHERE e.dept_id = d1.dept_id)
```

Example 3: Another correlated subquery in SQL

Here's the code:

Code

```
SELECT
  lastname,
  firstname,
  salary,
  (SELECT avg(salary)
    FROM employee e2
    WHERE e2.dep_id = e1.dep_id) AS avg_dept_salary
FROM employee e1
```

Example 4: A SQL correlated subquery in the SELECT list

Suppose we have a table called "`assigned_to_project`" that stores the names of employees assigned to projects. We want to find all employees who are *not* assigned to any projects. The solution is the following query:

Code

```
SELECT
  lastname,
  firstname,
  salary
FROM employee e1
WHERE NOT EXISTS (SELECT project_id
                  FROM assigned_to_project
                  WHERE employee_id = e1.employee_id)
```

Uncorrelate subquery is executed once for the entire query.

However you can also code a correlated query that is executed once for each row that's processed by the main query.

Since correlated subqueries can be difficult to code, you may want to test a subquery seperately before using it within another select statement.

To do that, however, you'll need to substitute a constant value for the variable that refers to a column in the outer query.

Once, you are sure that the subquery works on its own, you can replace the constant with a reference to the outer query so you can use it within a select statement.

Description:

A correlated query is a subquery that is executed once for each row in the main query.

In contrast, an uncorrelated subquery is executed only once.

A correlated subquery refers to a value that's provided by a column in the main query. For each different value that's returned by the main query for that column, the subquery returns a different value.

To refer to a column in the main query, you can qualify the column with a table name or alias. If a correlated subquery uses the same table as the main query, you can use table alias to remove ambiguity.

How to use the EXISTS operator

The query tests whether the subquery returns a result set.

IOW, it tests whether the result set exists.

When you use this operator, the subquery doesn't actually return a result set to the outer query.

Instead it returns an indication of whether any rows satisfy the search condition of the subquery.

Because of that, queries that use this operator execute quickly.

--

You typically use the EXISTS operator with the correlated query.

For e.g

The query retrieves all the vendors in the Vendors table that don't have invoices in the invoices table.

An alternative is:

Subquery returns a list of ID's for each vendor that's in the invoices table. Then the main query returns some data about the vendors whose ID's aren't in the list

It executes more quickly than either of those queries.

In this e.g, the correlated subquery selects all invoices that have the same vendor_id as the current vendor in the outer query.

Since the subquery doesn't actually return a result set, it doesn't matter what columns are included in the select clause. As a result, it's customary to just code an asterisk.

After the subquery is executed, the search condition in the WHERE clause of the main query uses the NOT EXISTS operator to test whether any invoices were found for the current vendor. IF not vendor row is included in the result set.

Table 8: Syntax of the subquery that uses the EXISTS operator.

```
Where [NOT] exists (subquery)
```

Table 9: ALL vendors that dont have invoices

```
SELECT Vendor_id, vendor_name, vendor_state
from vendors
where NOT EXISTS
( Select * from invoices where vendor_id = vendor.vendor_id)
```

Description:

- You can use the EXISTS operator to test that one or more rows are returned by the subquery.

- You can use the NOT EXISTS operator to test that no rows are returned by the subquery.

- When you use these operators with a subquery, it doesn’t matter what columns are specify in the SELECT clause. As a result, you just code an asterisk(*).

How to code subqueries in the select clause

To do that, you code the subquery in place of column specification. As a result, the subquery must return a sinlge value for that column.

In most cases, you code correlated subqueries in the Select clause.

e.g. The subquery calc the maximum invoice date for each vendor in the vendors table.

To do that, subquery refers to the vendor_id column form the vendors table in the main query.

-Difficult to read.

-You can replace with a join.

```
Get the most recent invoice date for each vendor

Select vendor_name, ( select max(invoice_date) from invoices where
vendor_id = vendors.vendor_id) as latest_inv
from vendors
order by latest_inv desc
```

The same query restated using a join

```
Select vendor_name, max(invoice_date) as latest_inv
from vendors v
 LEFT join invoices I on v.vendor_id = I.vendor_id
Group by vendor_name
 order by latest_inv DESC
```

Description:

-When you code a subquery in the SELECT clause, the subquery must reutnr a single value.

-When you code a subquery in the select clause, you typically uses a correlated subquery.

-A query that includes a subquery in its SELECT clause can typically be restated using a join instead of the subquery. Because a join is usually faster and easier to read, subqeuries are seldom coded in the select clause.

How to code subqueries in the FROM clause

To do that you code a subquery in place of a table specification.

When you code a subquery in the from clause, it can return a result set that contains any numbers of rows and columns. This result set is sometimes referred to as an inline view.

Subqueries are typically used in the FROM clause to create inline views that provides summarized data to another summary query.

When you code a subquery in the FROM clause, you must assign a table alias to the subquery. This is required even if you don't use a table alias of t (for temporary table) to the subquery.

Get the largest invoice total for the top vendors in each state

```
select vendor_state, max(sum_of_invoices) as max_sum_of_invoices
from
(
select vendor_state, vendor_name,
sum(invoice_total) as sum of invoices
from vendors v join invoices I
on v.vendor_id = I.vendor_id
Group by vendor_state, vendor_name
) t
group by vendor_state
```

Description:

When you code a subquery in the FROM clause, you must assign an alias to it. Then, you can use that alias just as you would any other table name or alias.

When you code a subquery in the FROM clause, you should use an alias for any columns in the subquery that performs calc. Then, the inline view can use these aliases as the column names of the table.

e.g.

sum(invoices_total) as sum of invoices	max(sum of invoices)

How to code the ORDER BY clause

The order by clause specifies the sort order for the rows in a result set.

-**How to sort by a column name**:

The expanded syntax of the ORDER BY clause

```
Order by expression [ASC I DESC] [, expression [ ASC I DESC]] ...
```

```
Select vendor_name,
concat(vendor_city, ‘ , ’, ‘vendor_state’, ‘ ‘ ,vendor_zip_code) as address
from vendors
order by vendor_name DESC
```

An order by caluse that sorts by three columns.

```
Select vendor_name,
concat(vendor_city, ‘ , ’, ‘vendor_state’, ‘ ‘ ,vendor_zip_code) as address
from vendors
order by vendor_state, vendor_city, vendor_name
```

Description:

By default, in an ascending order, special characters appears first in the sort sequence, followed by number, then letters. This sort order is determined by the character set used by the server, which you can change when you start the server.

Null values appears first in the sort sequence, even if you’re using DESC.

You can sort by any columns in the base table regardless of whether its included in the SELECT Clause.

How to sort by an alias, expression, or column numbers

Table 10: An order by clause that uses an alias

```
Select vendor_name,
concat(vendor_city, ‘ , ’, ‘vendor_state’, ‘ ‘ ,vendor_zip_code) as address
from vendors
order by address, vendor_name
```

Illustration 1: Note that within the address column, the result set is also sorted by the vendor_name column

Table 11: An Order by clause that uses an expression

```
Select vendor_name,
concat(vendor_city, ‘ , ’, ‘vendor_state’, ‘ ‘ ,vendor_zip_code) as address
from vendors
order by concat( vendor_contact_last_name, vendor_contact_first_name)
```

Illustration 2: You can also use an arithmetic or string expression in the order by clause. Also note that neither of these columns is included in the SELECT CLAUSE

Table 12: An order by clause that uses column positions

```
Select vendor_name,
concat(vendor_city, ‘ , ’, ‘vendor_state’, ‘ ‘ ,vendor_zip_code) as address
from vendors
order by 2,1
```

Illustration 3: To use this technique, you code the numbers that corresponds to the column of the result set, where 1 is the first column and so on.

Description:

The order by clause can include a column alias that’s specified in the select clause if the column alias does not include spaces.

How to code the limit clause

The limit clause specifies the maximum number of rows that are returned in the result set.

For most queries, you want to see the entire result set you wont use this clause. However, there maybe times when you want to retrieve just a subset of a larger result set.

Table 13: The expanded syntax of the LIMIT clause

LIMIT [offset,] row_count

How to limit the number of rows.

mysql>(query) limit 5

How to return a range of rows

mysql> (query) limit 100,100

Description:

You can use the Limit clause to limit the number of rows returned by the select statement. This clause takes one or more two integers arguments.

If you code a single argument, it specifies the maximum row count, beginning within the first row. If you code both arguments, the offset specifies the first row to return where the offset of the first row is 0.

if you want to retirve all of the rows from a certain offset to the end of the result set, code -1 for the row count.

Typically, you'll use an ORDER by clause whenever you use the limit clause.

How to work with aggregate function

In chapter 3, you learned how to use scalar func, which operates on a single value and return a single value.

How to use aggregate functions, which operates on a series of values and return a summary of a value.

Because aggregate func typically operate on the values in columns, they are sometimes referred to as column functions.

A query that contains one or more aggregate functions is typically referred to as a summary query.

How to code aggregate functions:

Table 14: A summary query that counts unpiad invoices and calculates the total_due

```
Select count(*) as number of invoices, sum(invoice_total – payment_total –
credit_total) as total_due from invoices
where invoice_total- payment_total – credit_total > 0
```

when you use these functions, you can also code the ALL or Distinct keyword.

The ALL keyword is the Default, which means that all values are included in the calculation. The exceptions are null value, which are always excluded from these functions.

If you don't want duplicate values included, you can code the distinct keyword. In most case, you'll use Distinct only with the count function where a column is typically used as an argument.

But you can't use the ALL or Distinct keyword or an expression with Count(*).

The value of this functions are numbers of rows in the base table that satisfy the search condition of the query, including rows with null values.

You wont use distinct on Min or Max because it has no effect on those functions.

Everything above is in regard with Aggregate functions.

And it doesn't make sense to use it with the AVG and sum functions.

Description:

The expression you specify for the AVG and SUM functions must result in a numeric value. The expression for the MIN, MAX and Count functions can result in a numeric, date or string value.

By default all values are included in the calc regardless of whether they're duplicated. If you want to omit duplicate values, code the distinct keyword. The keyword typically used with the count function.

All of the aggregate func except the count(*) ignore null values.

If you code an aggregate function in the select clause, that clause can't include non-aggregate columns from the base table unless the columns are named on a **group by** clause. -- understood

e.g.

```
select aisle, sum(quantity) from groceries group by aisle;
```

In some db's, It selects the first value in the group and allows you to include non-aggregate columns.(personal experience)

Queries that uses aggregate functions

Interesting usecase:

Illustration 4: A summary query that uses the count(*), avg and sum functions

```
Select ‘After 1/1/2011’ as selection date, count(*) as number_of_invoices,
round(avg(invoice_total),2) as avg_invoice_amt ,
sum(invoice_total) as total_invoice_amt
from invoices
where invoice_date > ‘2011-01-01’
```

Here, the string ‘ after 1/1/2011’ is use tell a proper summary.

Another usecase

Illustration 5: A summary query that uses the distinct keyword

```
Select ‘After 1/1/2011’ as selection date,
count(distinct vendor_id) as number_of_vendors,
 count(*) as number_of_invoices,
round(avg(invoice_total),2) as avg_invoice_amt ,
sum(invoice_total) as total_invoice_amt
from invoices
where invoice_date > ‘2011-01-01’
```

Description:

To count all of the selected rows, you typically use the count(*) function. Alternatively(meaning to get the same result as before), you can also use the count function with the name of any column that can’t contain null values. -- understood

To count only the rows with a unique values in a specified column, you can code the count function with the distinct keyword followed by the name of the column.

How to group and summarize data

The groupby clause determines how selected rows are grouped.

Selected using the where condition.

Having clause determines which groups are included in the final result.

So,

These clauses are coded after the WHERE clause but before the ORDER BY clause.

That makes sense because the where clause is applied before the rows are grouped, and the ORDER by clause is applied after the rows are grouped.

Syntax

The syntax for the GROUP BY clause in SQL is:

```
SELECT expression1, expression2, ... expression_n,
       aggregate_function (aggregate_expression)
FROM tables
[WHERE conditions]
GROUP BY expression1, expression2, ... expression_n
[ORDER BY expression [ ASC | DESC ]];
```

Link: click here

Description:

-The groupby clause groups the rows of a result based on one, or more columns or expressions. To include two or more columns or expressions, seperate them by commas.

-If you include aggregate func in the select clause, the aggregate is calculated for each group specified by the group by clause.

-The having clause specifies a search condition for a group or an aggregate. Mysql applies this condition after it gorups the rows that satisfy the search condition in the where clause.

- when a select statement includes a group by clause, the select clause can include the columns used for grouping, aggregate functions, and expressions that result in a constant value.

Illustration 6: summary query that calc the avg invoice amount by vendor

```
Select vendor_id, round(avg(invoice_total),2) as average_invoice_amt
from invoices
group by vendor_id
having avg(invoice_total)>2000
order by average_invoice_amount desc
```

Queries that uses the group by and having clause

Illustration 7: A summary query that limits the group to those with two or more invoices.

```
Select vendor_state, vendor_city, count(*) as invoice_qty,
round(avg(invoice_total),2) as invoice_avg
from invoices join vendors
on invoices.vendor_id = vendors.vendor_id
groupby vendor_state, vendor_city
having count(*) > 2
```

By default, the groupby clause sorts the columns in ascending order. Usually, that's what you want. However, you can change this sort order by coding the DESC keyword after the column name in the groupby clause.

Or, if u don't want to sort the result set at all or you wanna get your results faster. You can code an ORDER BY NULL clause to avoids sorting overhead, it can improve the performance of your query.

How the having clause compares to the Where clause

- A where clause can refer to any column in the base table.

- A having clause can only refer to a column included in the select clause.

- A where clause can't contain aggregate functions

- A having clause can contain aggregate functions.

Illustration 8: A summary query with a search condition in the WHERE clause

```
Select vendor_name,
count(*) as invoice_qty,
round(avg(invoice_total),2) as invoice_avg
from vendors join invoices
on vendors.vendor_id = invoices.vendor_id
where invoice_total >500
group by vendor_name
order by invoice_qty DESC
```

How to code compund search conditions

Illustration 9: A summary query with a compound condition in the having clause

```
Select
invoice_date,
count(*) as invoice_qty,
sum(invoice_total) as invoice_sum
from invoices
group by invoice_date
having  invoice_date between ‘2011-05-01’ and ‘2011-05-31’
and count(*) > 1
and sum(invoice_total)>100
order by invoice_date DESC
```

The same query,

Illustration 10: a summary query with a where condition

```
Select
invoice_date,
count(*) as invoice_qty,
sum(invoice_total) as invoice_sum
from invoices
where  invoice_date between ‘2011-05-01’ and ‘2011-05-31’
group by invoice_date
having count(*) > 1
and sum(invoice_total)>100
order by invoice_date DESC
```

Description:

You can use the AND or OR operator to code compound search conditions in a having clause just as you can in a where clause.

If a search condition includes an aggregate function, it must be coded in the having clause. Otherwise, it can be coded in either the HAVING or the WHERE clause.

How to use the WITH ROLLUP operator

MySql provides an extension to standard sql that's useful for summarizing data; **WITH ROLLUP** operator.

You can use this to **add one or more summary rows** to the end of the result set that uses grouping and aggregates.

This row summarizes all the aggregate columns in the result set.

It summarizes the invoice_count and invoice_total_columns.

Since the vendor_id column cant be summarized, it assigned a null value.

Illustration 11: A summary query that includes a final summary row

```
Select vendor_id, count(*) as invoice_count, sum(invoice_total) as
invoice_total
from invoices
group by vendor_id with ROLLUP
```

Illustration 12: A summary query that includes a summary row for each grouping level

```
Select vendor_state, vendor_city, count(*) as qty_vendors
from vendors
where vendor_state in ('IA,'NJ')
group by vendor_state ASC, vendor_city ASC with ROLLUP
```

Description:

When you use the WITH ROLLUP operator, you can't use the order by clause. To start individual rows, you can code the ASC OR DESC keyword after the column in the GROUP BY clause.

When you use the WITH ROLLUP operator, you can't use the DISTINCT KEYWORD in any of the aggregate functions.

How to work with inner joins

Inner join: only those rows that satisfy the join condition are included in the result set.

A join condition names the column in each of the two tables involved in the join and indicates how the two columns should be compared.

In most cases, you use the equal operator to retrieve rows with matching columns. However, you can also use any of the other comparision operators in a join condition. --saw it once in swati-maurya lec.

Tables are typically joined on the relationship between the primary key in one table and a foreign key in the other table.

However, you can also join tables based on relationships not defined in the database. These are called ad-hoc relationships.

If the two columns in a join condition have the same name, you must quailify them with the table name so Mysql can distinguish between them.

Note: inner keyword is optional and is seldom used.

How to use table aliases

If you assign an alias to a table, you must use that alias to refer to the table throughout your query. You can’t use the original table name.

How to join to a table in another database

All tables pertaining to accounts payable such as the Vendors and Invoices tables are stored in the database, or schema, name AP.

All tables pertaining to order management are stored in a database named op.

All tables that are used by the smaller examples presented in this book are sotored in a dtabase named EX.

You may occasionally need to join to a table that's in another database.

Description:

MySql server can store tables in multiple databases. These db's are sometimes referred to as schemas.

When you run a SELECT statement against one database, you can join a table in another database if you have appropriate permissions.

To do that, you must prefix the table name in the other database with the name of that database.

How to use compound join conditions

Although a join condition typically consist of single comparision, you can include two or more comparisions in a join condition using the AND or OR operators.

For e.g

uses the AND operator to return the first and last names of all customers in the customers table whose first and last names also exist in the Employees table.

Illustration 13: An inner join with two conditions

```
Select customer_fname, customer_lname from customer c
join employee e
on c.customer_fname=e.fname
and c.customer_lname=e.lname;
```

How to use a self join

Illustration 14: A self-join that returns vendors from cities in common with other vendors

```
Select distinct v1.vendor_name, v1.vendor_city,
v1.vendor_state
from vendors v1 join vendors v2
on
v1.vendor_city=v2.vendor_city
and
v1.vendor_state=v2.vendor_state
and
v1.vendor_name <> v2.vendor_name
order by v1.vendor_state, v1.vendor_city
```

output:

vendor_name	vendor_state	vendor_city
Kazim	AZ	pheonix
sahil	AZ	pheonix
faraaz	NY	New york
...	...	...

How to use the implicit inner join syntax

Instead of coding a join condition in the from clause, you can code It in the where clause along with any search conditions. In that case, you list the tables in the FROM clause seperated by commas.

Illustration 15: join the vendors and invoices table

```
Select invoice_number, vendor_name
from vendors v, invoices I
where v.vendor_id = I.vendor_id
order by invoice_number;
```

How to work with String functions

Function	Description
concat(str1[,str2]...)	
concat_ws(sep,str1[,str2] ...)	
LTRIM	Return the string with any leading spaces removed.
RTRIM	Return the string with any trailing spaces removed.
TRIM ([BOTH | LEADING | TRAILING] [remove] from str)	Return the string without leading or trailing occurrence of the specified removed string. If remove string is omitted spaces are removed.
LENGTH(str)	
locate(find,search[,start])	Returns the position of the first occurrence of the find string in the search string, starting at the specified start position. If the start position is omitted, the search starts at the beginning of the string. If the string isnt found, the function returns zero.
LEFT(str,length)	Returns the specified number of characters from the beginning of the string.
RIGHT(str,length)	Returns the specified number of characters from the end of the string.
substring_index(str,delimiter,count)	Returns the substring before the specified number of occurences of the specified delimeter string. If count is positive, it returns from the beginning of the string. If count is negative, it returns form the end of the string.
substring(str,start[,length])	Returns the specified number of characters from the string starting at the specified start position. If length is ommited, it returns from the start positions to the end of the string.
REPLACE(search, find, replace)	Replace the search string with all the occurrence of the find string with the replace string.
INSERT(str, start, length, insert)	Returns the string with the specified insert string into it starting at the specified start position to the end of the string.
Reverse	
Lower	
Upper	
LPAD(str, length, pad)	Returns the string padded on the left with the specified paded string until its the specified length. If the string is longer than the length. It’s truncated
RPAD	
SPACE(count)	Returns the space character repeated count times
Repeat(str,count)	Returns the specified string repeated count times.

Function	Result
concat(‘last’ , ‘First’)	‘lastFirst’
concat_ws(‘_’,’last’, ‘first‘)	‘last_first’
LTRIM(‘ MySql ‘)	‘MySql ‘
RTRIM	
TRIM(BOTH ‘*’ from ‘******MySql ***’	‘MySql’
left(‘MySql,3)	‘MyS’
right(‘MySql’,3)	‘Sql’
Substring(‘(555) 555-1212’, 7, 8)	‘555-1212’
substring_index(‘https://www.youtube.com/’, ‘.‘, -2)	‘Youtube.com’
length(‘Mysql’)	5
Length(‘ Mysql ’)	8
locate(‘sql’, ‘ mysql’)	5
Locate(‘-’,’(559) 555-1212)	10
Replace(right(‘(559) 555-1212’, 13), ‘)’, ‘-’)	559-555-1212
insert(‘MySql’,1,0,’Murach’s)	‘Murach’s MySql’

How to sort by a string column that contains number

When you attempt to sort string data that's stored in a numeric column.

Table 15: Sorted by the emp_id column explicitly cast as an integer

```
Select *
from string_example
order by cast(emp_id as signed)
```

Table 16: Sorted by the emp_id column implicitly cast as an integer

```
Select *
from string_example
order by emp_id + 0
```

Table 17: Sorted by the emp_id column after it has been padded with leading zeroes

```
Select LPAD(emp_id,2,'0') as emp_id, emp_name
from string_example
order by emp_id
```

How to parse a string

Another problem you may

substring_index(col_name,'delimeter',1)

```
Select emp_name,
substring_index(emp_name,' ',1) as first_name,
substring_index(emp_name,' ',-1) as last name
from string_sample;
```

```
Select emp_name,
locate(' ',emp_name,1) as first_name, substring_index(emp_name,' ',-1) as
last name from string_sample;
```

locate(' ',col_name,1) as first_name

locate('',col_name,locate(' ',col_name,1)+1)as last name

select emp_name,

substring(col_name,index,length)

substring(col_name, locate)

```
Select emp_name,
substring(emp_name,1,locate(' ',emp_name) ) as first_name
substring(emp_name,locate(' ',emp_name) + 1) as last_name
from string_sample
```

Desc:

if a string consists of two or more components, you can parse it into its individual components. To do that, you can use the SUBSTRING_INDEX, SUBSTRING and locate functions.

How to work with numeric data

Functions	Description
Round(number [, length])	Returns the number rounded to the precision specified by len. If len is zero, the decimal digits are omitted. If len < 0 then the digits left to the decimal points are rounded.
Truncate(number [,length])	Same as above. Just len < 0 isnt defined in the book definition.
Ceiling(number)	Return the smallest integer that is **greater** than or equal to that number
floor(number)	Return the largest integer that is less than or equal to that number.
ABS(number)	Return the absolute value of the number.
SIGN(number)	Return the sign for the number as -1 for a negative number. 1 for positive number ,and zero if the number is 0
SQRT(number)	
Power(number , power)	
RAND([integer])	It returns a floating point number between 0-1. Integer provides a seed value for the random number generator.

Function	Result
Round(12.49,0)	12
Round(12.50,0)	13
Round(12.49,1)	12.5
Truncate(12.51,0)	12
Truncate(12.49,1)	12.4
Ceiling(12.5)	13
ceiling(-12.5)	12
Floor(-12.5)	-12
floor(12.5)	12
ABS(-1.25)	1.25
ABS(1.25)	1.25
SIGN(-1.25)	-1
SIGN(1.25)	1

How to search for floating-point numbers

floating point types such as double or float types stores approximate values, not exact value.

The details of why are beyond the scope of the book.

From a practical pov, you don't want to search for exact values when you're working with floating point numbers. If you do, you'll miss values that are approximately equal to the value you're looking for.

e.g

float_id	float_value
1	0.999999999999999
2	1
3	0.100000000000000001
4	12345678

Table 18: A search for an exact value that doesnt include two approximate values

```
>
select *
from float_sample
where float_value=1
```

float_id	float_value
2	1

```
1 record
query successful (0.23 s)
```

Instead search for approximate values,

```
Select * from float_sample where float_value between 0.99 and 1.01
Select * from float_sample where round(float_value,2)=1.0
```

float_id	float_value
1	0.999999999999999
2	1
3	0.100000000000000001

How to work with users and privileges

Working with users and privileges. This includes creating and dropping users, granting and revoking privileges, and changing the password for an existing user.

How to create, rename, and drop users

To start, when you use the CREATE user statement, you typically specify the name of the user, followed by the @ sign, followed by the name of the host that the user can connect from.

This is usually followed by the IDENTIFIED BY clause, which specifies a password for the user.

How to create a user:

Table 19: The syntax of the create user statement.

```
CREATE USER username IDENTIFIED BY password
```

Illustration 16: A statement that creates a user from a specific host

```
CREATE USER joel@localhost IDENTIFIED BY 'sesame'
```

Illustration 17 creates a user named Joel that can connect from the host named localhost with a password 'sesame'. IOW, Joel can only connect from the same computer where MySql server is running.

Illustration 18: A statement that creates a user from any host

```
>CREATE USER jane identified by 'sesame'
                                        -- creates jane@%
```

Illustration 19 if you don't use the @ symbol to specify the host. MySql uses the '%' as the name of the host. This indicates that the user can connect from any host.

After you create a user, the user has no privileges. However, you can use the GRANT statement to assign privileges to that person.
Just like in serviceNow

How to rename a user

Illustration 20: Syntax of rename user statement

```
RENAME USER username to new_username
```

Illustration 21: A statement that renames a user from a specific host

```
Rename user joel@localhost to kazim@localhost
```

How to drop a user

Illustration 22: The syntax of drop user statement

```
Drop user username
```

Illustration 23: A statement that drops user from a specified host

```
DROP user kazimsyed@localhost
```

Illustration 24: A statement that drops a user from any host

```
DROP user jane
                -- drops jane@%
```

Description:

you use the CREATE USER stmt to create a user that has no privileges.

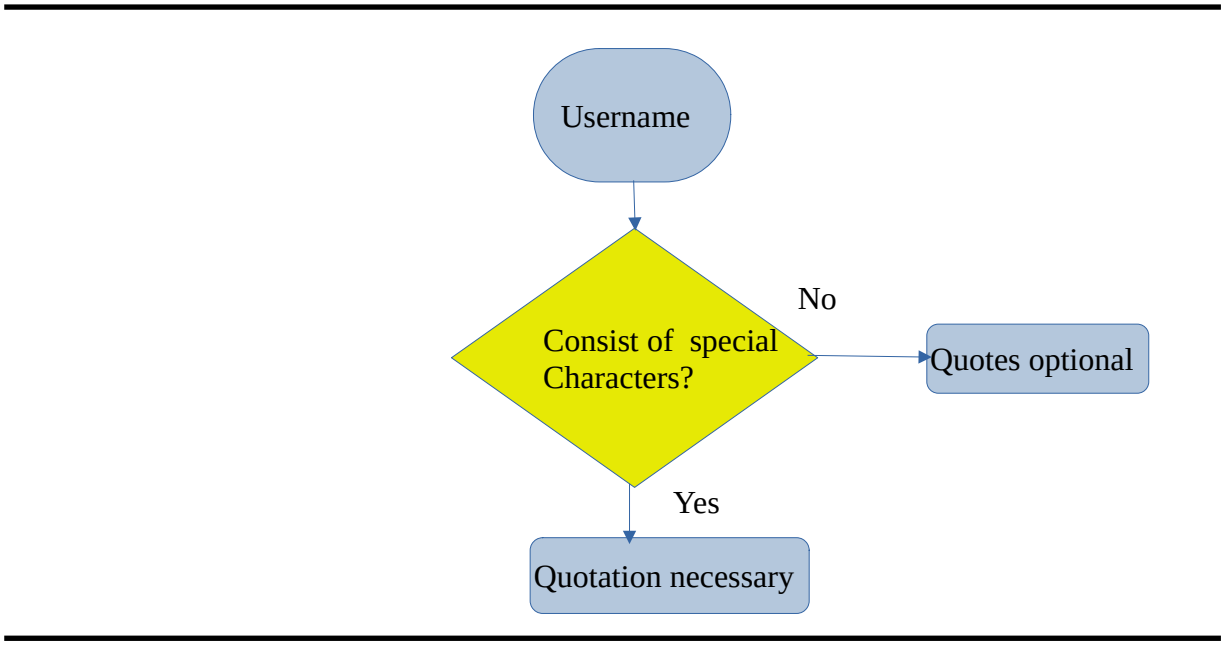
If you want to specify the host that user can connect from, you can code the username, followed by the @character, followed by the hostname.

If you create a user without specifying a hostname, MySql uses a percent sign (%) as wildcard character to indicate that the user can connect from any host.

You can use the RENAME USER stmt to change the name of a user.

You can use the DROP USER stmt to drop a user.

How to specify user account names



Another way to code a user which is synonymous with the localhost keyword. You can use the IP address to identify a remote server too if necessary.

I.e

Illustration 25: The same user with an IP address instead of the localhost keyword

```
John@127.0.0.1
```

Illustration 26: A user that can connect from any computer

```
john
```

Or

Illustration 27: the same user but with the wildcard character explicitly coded

```
John@'%'
```

Illustration 28: A user that can only connect from the murach.com domain.

```
John@'%.murach.com'
```

How to grant privileges

It's good practice to use the *CREATE USER* stmt to create users with no privileges and then use GRANT stmt to grant privileges.
However, MySql allows you to use the *GRANT* stmt to create users and grant privileges in a single stmt.

Illustration 29: The syntax of the GRANT statement.

```
GRANT privilege_list
on [db_name].table
To user1 [ identified by 'password'] [,
 user 2 [ identified by 'password]
[with GRANT OPTION]
```

Illustration 30: A statement that creates a user with no privileges

```
GRANT USAGE
ON *.*
To joel@localhost identified by 'sesame'
```

Creates a user that has no privileges. Because the on clause is required, its coded with an asterisk for both the database and table name.

The asterisk are wildcard that indicates that user have privileges on all db's and tables. Even though no privileges is given. IOW the user is given global level privilege.

Illustration 31: A Statement that creates a user with database privileges

```
GRANT USAGE
ON *.*
To joel@localhost identified by 'sesame'
with GRANT OPTION
```

Illustration 32 The grant option clause, user can grant privileges to other users.

Although using the ALL keyword makes it easy to grant all privileges to a user, it also makes it easy to grant more privileges than the user needs

Good practice is to grant a user just the privileges that he or she needs

Illustration 33: A statement that creates a user with database privileges

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON ap.*
TO  ap_user@localhost IDENTIFIED BY ‘password’
```

Illustration 34: A statement that grants global privileges to a user.

```
GRANT Select, insert, update
on *.* to ap_user@localhost
```

If you connect as a user other than the root user, even if you use the ALL keyword, only the privileges that the current user has are granted to that users.

All the above is only if he have the grant option privilege in the first place.

To grant all privileges to a user, you must connect as a user such as root user that has the appropriate user.

Illustration 35: A stmt that grants db privileges to a user

```
GRANT SELECT, INSERT, UPDATE
on ap.* to joel@localhost
```

Illustration 36: A stmt that grants table privileges to a user

```
GRANT SELECT, INSERT, UPDATE
on ap.vendors to joel@localhost
```

Illustration 37: A stmt that grants db privileges to a user

```
GRANT SELECT(vendor_name, vendor_state, vendor_rip_code), INSERT,
UPDATE(vendor_address1)
on ap.* to joel@localhost
```

Illustration 38: A stmt that uses the current database.

```
GRANT SELECT, INSERT, UPDATE,DELETE
on ap.vendors to joel@localhost
```

Description:

- You use the GRANT stmt to grant privileges to a user. If the user account doesn't exist, the user is created.

- The ON clause determines the level at which the privileges are granted. You can use the asterisk(*) to specify all databases or tables. If you don't specify a database, MySql uses the current database.

-The with grant option clause allows the user to grant their privileges to other users.

How to view privileges

When you're done granting or revoking privileges, you may want to view the privileges that have been granted to make sure that you have granted the correct privileges to each user.

To start, if you want to get a list of users for the current server.

```
Select User, HOST from mysql.user
```

Illustration 39: The syntax of the show GRANTS statement.

```
SHOW GRANTS [FOR user]
```

Illustration 40: a stmt that shows the privileges for the current user

```
Show GRANTS
```

Illustration 41: A statement that shows the privileges for a user from any host

```
Show grants for jim
```

Illustration 42: A stmt that shows the privileges for a user from a specific host

```
Show grants for jim@localhost
```

Descripton:

You can query the User table in the mysql database to get a list of users for the current MySql server.

You can use the show grants stmt to display the privileges for a user.

How to revoke privileges

After you've created users and granted privileges to them, you may need to revoke privileges. For e.g, you may need to revoke some or all of a user privileges if the user abuses those privileges.

Illustration 43 The syntax of the REVOKE stmt for all privileges

```
REVOKE ALL[ Privileges], GRANT OPTION
From user1 [,user2]
```

Illustration 44: The syntax of the REVOKE stmt for specific privileges

```
REVOKE ALL[ Privileges], GRANT OPTION
[ON <db.table]
From user1 [,user2]
```

This revoke all privileges from the user on all db's. To be able to use this syntax, you must've logged in as a user that has the CREATE USER privilege. IOW, You must be logged in as a user that has the grant option privilege and the privilege you're invoking.

Meaning users with (grant option privilege + select and update only) privilege can't revoke the DELETE privilege of some user.

Although the revoke stmt removes privileges. It does't remove the user from the db that MySql uses to keep track of users.
To remove a user account entirely, use the DROP USER stmt.

Description:

-You can use the Revoke statements to revoke privileges from a user.

- To revoke all privileges, you must have the global create user privilege.

-To revoke specific privilege, you must have the GRANT OPTION privilege and you must have the privilege that you're revoking.

How to change passwords

Table 20: The syntax of the SET PASSWORD Statement.

```
SET PASSWORD [FOR USER] = PASSWORD('password')
```

Table 21: A statement that changes a user password

```
SET PASSWORD FOR JOHN = PASSWORD('password')
```

Table 22: A statement that changes the current user password

```
SET PASSWORD = PASSWORD('secret')
```

Illustration 45 to change the password for a user who's currently logged in.

Table 23: A Grant stmt that changes the user's password

```
GRANT USAGE ON *.* TO JOHN IDENTIFIED BY 'PASSWORD'
```

Illustration 46 You can also change a password using the grant statement. Here the usage privilege is use so the current privileges aren't changed. The IDENTIFIED clause is used to specify the new password.

For security reasons, you should always assign passwords to each user.
To make sure that every user has a password.
You can execute a

Illustration 47: A select stmt that selects all user that don't have passwords

```
SELECT HOST, User
from mysql.user
where password = ''
```

Also, if the users aren't needed you can drop them.

Description:

-To change a user's password, use the SET PASSWORD stmt.

- To change the password for another user's account, you must have the update privilege.

- you can change the current user's privilege by using the SET PASSWORD stmt without a FOR CLAUSE.

- you can also change a passwd using the GRANT statement with the USAGE privilege and an IDENTIFIED BY clause.

- To be sure you've assinged password to all users, you can select data from the USER table of the mysql database for all users without passwords.

An introduction to views

A view is a select statement that's stored in the database as a database object.

-**virtual table** that **only** consist of rows and columns specified in the CREATE View statement.

-the table or tables that are listed in the FROM clause are called the **base table for the view**.

-refers back to the base table, so it doesn't store any data, and it always reflects the most current data in the base tables.

Table 24: Create view statement for a view named vendors_min

```
CREATE VIEW vendors_min AS
select vendor_name, vendor_state, vendor_phone
from vendors
```

Illustration 48 when you create a view like this one, the view is updatable. As a result, it's possible to use the view in an INSERT, UPDATE, or DELETE statement.

Illustration 49: An update stmt that uses a view to update the base table.

```
UPDATE vendors_min
set vendor_phone ='(800) 555-3941'
where vendor_name='Register of Copyrights'
```

Because a view is stored as object in a database, it can be used by anyone who has appropriate privileges.
That may include users who have access to the database through applications that provide for ad-hoc queries and report generation.
In addition, that may include custom applications that are written specifically to work with the data in the database. In fact, views are often designed to be used with these types of applications.

Illustration 50: A statement that drops a view

```
DROP VIEW vendors_min
```

Description:

A view consist of a SELECT statement that's stored as an object in the database. The tables referenced in the SELECT stmt are called the base tables for the view.

When you create a view, you can refer to the view anywhere you would normally. Use a table in a SELECT, INSERT, UPDATE, or DELETE stmt.

Although a view behaves like virtual table, it doesn't store any data. Instead, a view always refers back to its base table.

A view can also be referenced to as viewed table because it provides a view to the underlying base table.

Benefits of using views

-limit exposure of the tables in your database to external users and applications.

To illustrate, suppose a view refers to a table that you've decided into two tables. To accommodate this change, you simply modify the view. IOW, you don't have to modify any statements that refer to that view. That means you don't have to be aware of the change in the database structure, and application programs that use the view don't have to be modified.

-Restrict access to the database.

To do that, you include **just** the columns and rows you want a user or an application to have access to in the views. Then you let the user or application to access the data only throught the views.

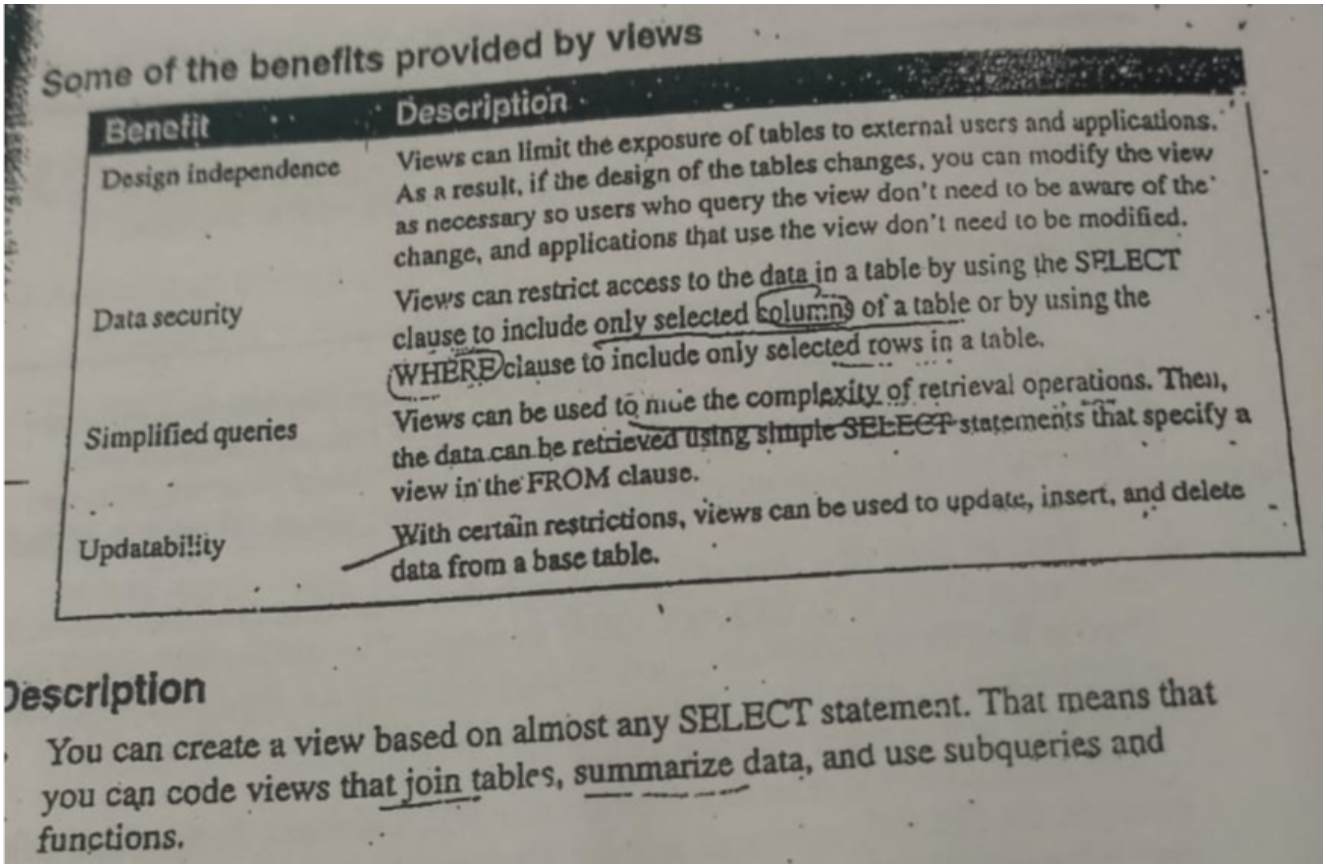
For e.g lets say you have an Employee table that has a salary column. So u create a view that doesn't incude a salary column for users who need to view and maintain the table, but who shouldn't be able to view salaries.

And you create another view for the users who need to view and maintain the salary information.

- Hide the complexity of a select statement.

For e.g. if u have a long and unwieldy SELECT stmt that joins multiple tables. You can create a view for that statement. That makes it easier for u and other database user to work with this data.

- when you create a view, you can allow data in the base table to be updated though the view. To do that, you use INSERT, UPDATE OR DELETE statements that work with the view.



Some of the benefits provided by views

Benefit	Description
Design independence	Views can limit the exposure of tables to external users and applications. As a result, if the design of the tables changes, you can modify the view as necessary so users who query the view don't need to be aware of the change, and applications that use the view don't need to be modified.
Data security	Views can restrict access to the data in a table by using the SELECT clause to include only selected columns of a table or by using the WHERE clause to include only selected rows in a table.
Simplified queries	Views can be used to hide the complexity of retrieval operations. Then, the data can be retrieved using simple SELECT statements that specify a view in the FROM clause.
Updatability	With certain restrictions, views can be used to update, insert, and delete data from a base table.

Description

You can create a view based on almost any SELECT statement. That means that you can code views that join tables, summarize data, and use subqueries and functions.

How to work with views

Table 25: The syntax of the CREATE VIEW Stmt

```
CREATE [OR REPLACE] view view_name
[ column_alias_1 [ ,column_alias_2] … ]
AS
select_statment
[ WITH CHECK OPTION] [CONSTRAINT constraint_name]
```

Table 26: A view of vendors that have invoices

```
CREATE VIEW vendors_phone_list AS
SELECT vendor_name, vendor_contact_last_name,
vendor_contact_first_name, vendor_phone
from vendors
where vendor_id in (select distinct vendor_id from invoices)
```

Table 27: A view that uses join

```
CREATE OR REPLACE view vendor_invoices
SELECT Vendor_name, invoice_number, invoice_date, invoice_total
from vendors
JOIN invoices on vendors.vendor_id = invoices.vendor_id
```

Table 28: A view that uses a limit clause

```
CREATE OR REPLACE VIEW top5_invoice_total AS
select vendor_id, invoice_total
from invoices
order by invoice_total DESC
limit 5
```

Table 29: A view that names all of its columns in the CREATE VIEW CLAUSE

```
CREATE OR REPLACE VIEW invoices_outstanding
(invoice_number, invoice_date, invoice_total, balance_due)
as
select invoice_number, invoice_date, invoice_total, invoice_total-
payment_total – credit_total from invoices
where invoice_total – payment_total – credit_total > 0
```

OR

Table 30: A view that names just the calc column in its select clause.

```
CREATE OR REPLACE VIEW invoices_outstanding
as
select invoice_number, invoice_date, invoice_total, invoice_total-
payment_total – credit_total as baalance_due from invoices
where invoice_total – payment_total – credit_total > 0
```

--

Table 31: A view that summarizes invoices by vendor

```
CREATE OR REPLACE VIEW invoice_summary as
select vendor_name
count(*) as invoice_count
sum(invoice_total) as invoice_total_sum
from vendors
join invoices on vendors.vendor_id = invoices.vendor_id
group by vendor_name
```

Illustration 51 shows that view can use aggregate functions and the GROUP BY clause to summarize the data. In this case, the rows are grouped by vendor_name, and a count of the invoices and the invoice_total are calculated for each vendor.

When you create a view, the SELECT stmt you code within the definition of the view can refer to another view instead of base table.
IOW, views can be nested. In theory, nested views can make it easier to present data to your users.
In pracitce, using nested views can make the dependencies between tables and views confusing, which can make your code difficult to maintain.
As a result, if you use nested views, you should use them carefully.

description:

If you name the columns of a view in the CREATE VIEW clause, you have to name all of the columns. In contrast, if you name the columns in the SELECT clause, you can name just the columns you need to name.

How to create an updatable view

Once you create a view, you can refer to it in a select statement.

In addition, you can refer to it in insert, update and delete statement. Meaning CRUD is possible to modify the data that's stored in an underlying table.

To do that view must be updatable view.

Requirements:

1. You can't use DISTINCT, AGGREGATE FUNC's, GROUP BY, HAVING Clause and two select statements can't be joined by union operations.

Meaning you can only refine the data using where, customize the data and limit the data and fields. My words.

You can't use DISTINCT if u want an updatable view. Cuz you are not aware of all the data records. You'll be missing out.

Same reason for the rest.

2. You can't update calculated columns.

Duh-uh. They are derived columns, they don't exist.

3. when you update data through a view, you can only update the data in a single base table at a time, even if the view refers to two or more tables.

For instance, the view includes data from two base tables: vendors and invoices. Because of that, you can code an UPDATE statement that updates the data in the vendors table or the data in the invoices_table. BUT NOT IN both tables.

Requirements for creating updatable views:

-The select list can’t include DISTINCT clause

- The select list can’t include AGGREGATE functions.

- the selct stamtnet can’t include a GROUP BY or HAVING clause.

-The view can’t include the UNION operator.

A create view statement that creates an updatable view

```
CREATE OR REPLACE view balance_due_view as
select vendor_name, invoice_number,
invoice_total,payment_total, credit_total,
invoice_total – payment_total -credit_total as balance_due
from vendors join invoices on vendors.vendor_id= invoices.vendor_id
where invoice_total – payment_total – credit_total > 0
```

```
UPDATE balance_due_view
set credit_total=100
where invoice_number=’82392’
```

Description:

An updatable view is a view that can be used in an INSERT, UPDATE, or DELETE stmt to update the data in the base table. If a view isnt updatable, its called a read-only-view.

How to use the WITH CHECK OPTION clause

The WITH CHECK OPTION clause to prevent an update if it causes the row to be excluded from the view.

```
CREATE OR REPLACE view vendor_payment as
select vendor_name, invoice_number,
invoice_total,payment_total, credit_total
from vendors join invoices on vendors.vendor_id= invoices.vendor_id
where invoice_total – payment_total – credit_total > 0
WITH CHECK OPTION
```

```
Select * from vendor_payment
where invoice_number=’p-608’
```

Name	Number	Date	payment_date	invoice_total	credit_total	Payment_total
Kazim	p-608	27-5-2000	**NA**	20551.8	1200.00	**0.00**

```
UPDATE vendor_payment
set payment_total=’400.00’
payment_date=’2011-08-01’
where invoice_number=’p-608’

(1 row affected)
```

Name	Number	Date	payment_date	invoice_total	credit_total	Payment_total
Kazim	p-608	27-5-2000	**2011-08-01**	20551.8	1200.00	**400.00**

```
UPDATE vendor_payment
set payment_total=’3000.00’
payment_date=’2011-08-01’
where invoice_number=’p-608’

ERROR CODE CHECK OPTION failed
```

Illustration 52 this can prevent users from storing invalid data in the database, this clause can be useful in some situations. This error occur cuz the update statement exclude the rows from the view.

How to insert or delete rows from a view

It is generally more common to work directly with base tables when inserting or deleting rows.

Cuz when u inserting data through a view to the base table, you must have all the required columns and the constructs of the views are not suitable for this. As a result, to use a view to insert rows, you must design a view that includes all required columns for the underlying table.
Also, an Insert statement that uses a view can insert rows into only one table. That's true even if the view is based on two or more tables and all of the required columns for those tables are included in the view.
In that case, you could use a seperate INSERT stmt to insert rows into each table through a view.

DELETE STMT fails cuz of foreign key constraint. So do deal with that first.

How to alter or drop a view

Although MySql supports an ALTER view statement, its usually eaiser to alter a view by using create or repalce view stmt.

And its fine,cuz view doesn't store the data. It's a virtual table.

To drop a view

```
DROP view view_name
```

Section 5: CRUD COMMANDS

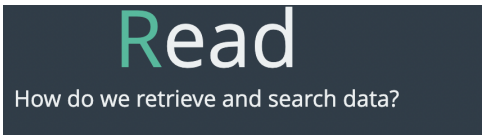
Crud is an acroynm for create, read, update, delete.

Update and delete: how do we change things in a db if we make mistakes.

Lets say User forgot their password. We gonna send a reset link.

We need to change it in the db.

Or lets say user leave our website for some really bad reason, so we can delete that user.




```
UPDATE cats SET breed='Shorthair'
WHERE breed='Tabby';
```

Best practices:

- Try selecting before you update.

--

How to change datatype of a column.

alter table shirts modify column shirt_size varchar(2);

How to update multiple columns together

update shirts set shirt_size = "XS" , color = "off white" where color='white';

The world of string functions.

things I could do with the data.

For numbers, I could add them together.

Or I might to reverse a string.

Or replace all spaces with certain character, maybe _.

Or conjoin first name or last name.

We can concatanate.

We can substring.

We can uppercase, lowercase.

Running SQL files

```
source file_name.sql
```

Text functions

```
% cat data1
+--------------+----------------+
| author_fname | author_lname   |
+--------------+----------------+
| Jhumpa       | Lahiri         |
| Neil         | Gaiman         |
| Neil         | Gaiman         |
| Jhumpa       | Lahiri         |
| Dave         | Eggers         |
| Dave         | Eggers         |
| Michael      | Chabon         |
| Patti        | Smith          |
| Dave         | Eggers         |
| Neil         | Gaiman         |
| Raymond      | Carver         |
| Raymond      | Carver         |
| Don          | DeLillo        |
| John         | Steinbeck      |
| David        | Foster Wallace |
| David        | Foster Wallace |
| Adam         | Smith          |
+--------------+----------------+
```

```
kazimsyed@kazims-MacBook-Air ~/Desktop/mysql_learn
% sed -n 's/|/("/1
s/|/","/1
s/|/")/1
s/ //g
quote> 4,$p' data1
("Jhumpa","Lahiri")
("Neil","Gaiman")
("Neil","Gaiman")
("Jhumpa","Lahiri")
("Dave","Eggers")
("Dave","Eggers")
("Michael","Chabon")
("Patti","Smith")
("Dave","Eggers")
("Neil","Gaiman")
("Raymond","Carver")
("Raymond","Carver")
("Don","DeLillo")
("John","Steinbeck")
("David","FosterWallace")
("David","FosterWallace")
("Adam","Smith")
+--------------+----------------+
```

code:

```
sed -n 's/|/("/1
s/|/","/1
s/|/")/1
s/ //g
4,$p' data1
```

```
kazimsyed@kazims-MacBook-Air ~/Desktop/mysql_learn/filter
% gawk 'BEGIN{ORS=","}{print $0}' data1
("Jhumpa","Lahiri"),("Neil","Gaiman"),("Neil","Gaiman"),("Jhumpa","Lahiri"),("Dave","
"),("Patti","Smith"),("Dave","Eggers"),("Neil","Gaiman"),("Raymond","Carver"),("Raymo
nbeck"),("David","FosterWallace"),("David","FosterWallace"),("Adam","Smith"),+-------
```

```
gawk 'BEGIN{ORS=","}{print $0}' data1
```

string_fun.sql (~/Desktop/mysql_learn) - gedit

```
1
2 create database learn_str;
3 use learn_str;
4 create table names(id int auto_increment primary key, fname varchar(30), lname varchar(30));
5
6 insert into names (fname,lname) values ("Jhumpa","Lahiri"),("Neil","Gaiman"),("Neil","Gaiman"),("Jhumpa","Lahiri"),
  ("Dave","Eggers"),("Dave","Eggers"),("Michael","Chabon"),("Patti","Smith"),("Dave","Eggers"),("Neil","Gaiman"),
  ("Raymond","Carver"),("Raymond","Carver"),("Don","DeLillo"),("John","Steinbeck"),("David","FosterWallace"),
  ("David","FosterWallace"),("Adam","Smith");
```

data:

```
mysql> select * from names;
+----+---------+---------------+
| id | fname   | lname         |
+----+---------+---------------+
|  1 | Jhumpa  | Lahiri        |
|  2 | Neil    | Gaiman        |
|  3 | Neil    | Gaiman        |
|  4 | Jhumpa  | Lahiri        |
|  5 | Dave    | Eggers        |
|  6 | Dave    | Eggers        |
|  7 | Michael | Chabon        |
|  8 | Patti   | Smith         |
|  9 | Dave    | Eggers        |
| 10 | Neil    | Gaiman        |
| 11 | Raymond | Carver        |
| 12 | Raymond | Carver        |
| 13 | Don     | DeLillo       |
| 14 | John    | Steinbeck     |
| 15 | David   | FosterWallace |
| 16 | David   | FosterWallace |
| 17 | Adam    | Smith         |
+----+---------+---------------+
17 rows in set (0.00 sec)
```

text functions:

function	about	reference
Concat(columns, ..)		
Concat_ws(fs,column1, columns2, ...)		concat with seperator
substr		substr
replace		Replace
char_length		
upper		
lower		
reverse		

```
mysql> select concat_ws('_',fname,lname,'alaska') from names;
+-----------------------------------+
| concat_ws('_',fname,lname,'alaska') |
+-----------------------------------+
| Jhumpa_Lahiri_alaska              |
| Neil_Gaiman_alaska                |
| Neil_Gaiman_alaska                |
| Jhumpa_Lahiri_alaska              |
| Dave_Eggers_alaska                |
| Dave_Eggers_alaska                |
| Michael_Chabon_alaska             |
| Patti_Smith_alaska                |
| Dave_Eggers_alaska                |
| Neil_Gaiman_alaska                |
| Raymond_Carver_alaska             |
| Raymond_Carver_alaska             |
| Don_DeLillo_alaska                |
| John_Steinbeck_alaska             |
| David_FosterWallace_alaska        |
| David_FosterWallace_alaska        |
| Adam_Smith_alaska                 |
+-----------------------------------+
17 rows in set (0.00 sec)
```

Figure 8: concat with seperator

```
mysql> select concat_ws(' ',fname,substr(lname,1,3),'...') from names;
+-----------------------------------------------+
| concat_ws(' ',fname,substr(lname,1,3),'...') |
+-----------------------------------------------+
| Jhumpa Lah ...                                |
| Neil Gai ...                                  |
| Neil Gai ...                                  |
| Jhumpa Lah ...                                |
| Dave Egg ...                                  |
| Dave Egg ...                                  |
```

Figure 9: substr

-count starts from 1.

- if the last argument isnt given, then it is consider till end.

e.g1-

```
SELECT SUBSTRING('Hello World', 7);
```

World

e.g2-

```
SELECT SUBSTRING('Hello World', -3);
```

rld

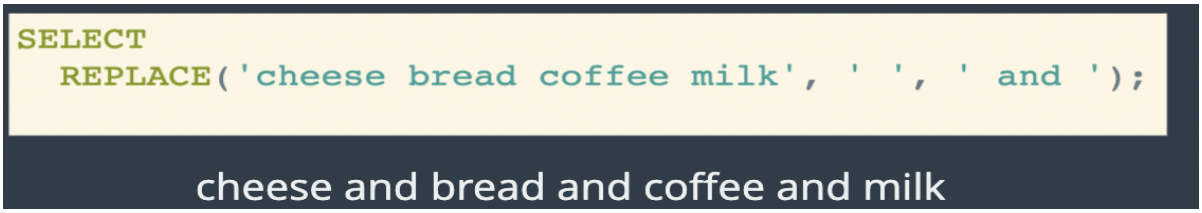
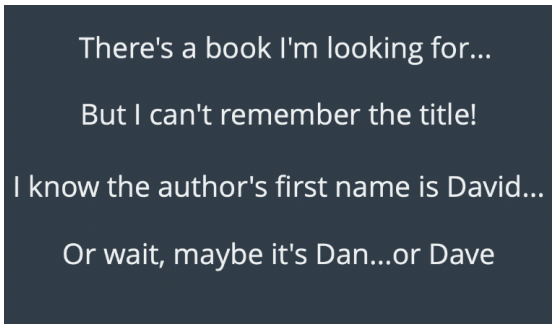


Figure 10: Replace

Like keyword



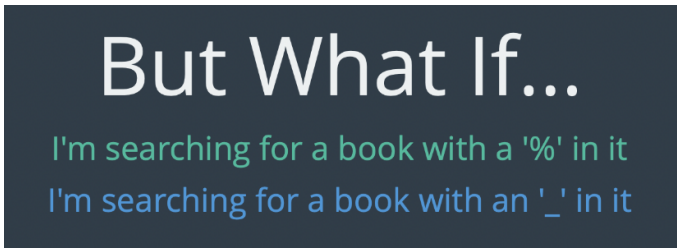


Figure 11: we need to escape.

Aggregate functions:

```
mysql> select distinct count(writer_name) from books_view;
+---------------------+
| count(writer_name) |
+---------------------+
|                  16 |
+---------------------+
1 row in set (0.00 sec)

mysql> select  count(distinct writer_name) from books_view;
+-----------------------------+
| count(distinct writer_name) |
+-----------------------------+
|                           9 |
+-----------------------------+
1 row in set (0.00 sec)

mysql> select  count(all writer_name) from books_view;
+-------------------------+
| count(all writer_name) |
+-------------------------+
|                      16 |
+-------------------------+
1 row in set (0.00 sec)
```

All or distinct keywords specify whether duplicate rows are returned.