* **Definitions: database, query, table, column, row, RDBMS.**
* **What is a relational database?**
* **Is the most recent entry is the most recent entry?**
* **Can you have duplicate records in the table?**
* **SQL is the language of what kind of databases?**
* **Steps for creating a table.**

|  |
| --- |
| **Database** – container that holds tables and other SQL structures related to those tables. |
| **Query** – retrieving information from database. |
| **Table** – the structure inside database that contains data, organized in **columns** and **rows**.  **Column [field]** – category, a piece of data; **Row [record]** – set of columns that describe one object. |
| **RDBSM** – Relational Database Management System. SQL is the language of the relational databases. |
| **Relational:** consider how the columns relate to each other to describe a thing. |
| The last record doesn’t have to be the newest; you can’t rely on the rows in the table being in chronological order. |
| Duplicate records should never exist in the table waste memory, long search time - must have a good table design. |
| 1. Pick **one object** that the table will describe. 2. Make a list of the information you need to know about your object. 3. Break down the information into pieces for easy query –**columns that contain atomic data for easy queries**. |

* **Benefits of normal tables.**
* **What is 1NF?**
* **What is an atomic data?**
* **Atomic data rules.**

|  |
| --- |
| **The benefits of normal tables:** no duplicate data[**saves memory**], less data [**fast search**] |
| **First Normal Form (1NF)** – each row must contain **atomic values**, each row must have a unique identifier, **primary key.** |
| An **atomic data** is the smallest pieces of data that can’t be or shouldn’t be divided. |
| **Atomic data rules:** each column of the table must hold one piece of different data. **1.** A column with atomic data can’t have severalvalues of the same type of data in that column. [Columns hold different info]  **2.** A table with atomic data can’t have several columns of the same type of data in that table. [Column has only one piece of info] |

* **Primary key rules.**
* **Synthetic vs. Natural Keys.**
* **How many AUTO\_INCREMENT field can table have?**
* **Create a new table with primary key.**
* **Insert value into the table containing primary key.**
* **Add a primary key to an existing table.**

|  |
| --- |
| **Primary key rules:** can’t be NULL, must be given a value when the record is inserted, must be compact, value can’t change. |
| **Synthetic** **key** – an extra data that is not part of the table. **Natural key** – data that is already in the table. |
| There is only one AUTO\_INCREMENT field per table, has to be an INTEGER and can’t contain NULL. |
| **CREATE TABLE** table\_name - create the table with the column that always must have a value.  **(**  column\_id **INT NOT NULL AUTO\_INCREMENT,**  column\_name2 **VARCHAR(10),**  column\_name3 **DEC(3,2) NOT NULL DEFAULT 2.00**  **PRIMARY KEY(**column\_id**)**  **);** |
| **INSERT INTO** table\_name - insert values [no column names] into the table in the same order as column names.  **VALUES**  **(** NULL, ‘value2’, value3’**),**- the AUTO\_IMCREMENT ignores the NULL.  **(**‘’, ‘value2’, value3’**),**  **(**‘value2’, value3’**),**  **(**99, ‘value2’, value3’**);** - 99 overrides the primary key in the AUTO\_INCREMENT column. |
| **ALTER TABLE** table\_name **ADD COLUMN** col\_id **NOT NULL AUTO\_INCREMENT FIRST, ADD PRIMARY KEY(**col\_id**);** |

* **Data Types: CHAR, VARCHAR, BLOB, DEC, INT, DATETIME, TIMESTAMP,** **TIME**, **DATE.**

|  |
| --- |
| **CHAR** or **CHARACTER** – set length. **Ex**: CHAR(7) |
| **VARCHAR** – text data of up to 256 characters. **Ex**: VARCHAR(20) |
| **BLOB** – text data of 256 or more characters – can’t run some string operations that you can on VARCHAR or CHAR. |
| **DEC** or **DECIMAL. Ex: DEC(5, 2)** – shows how many digits are in front of decimal and how many after. |
| **INT** or **INTEGER** – no decimals or fractions. |
| **DATETIME** [future]or **TIMESTAMP** [current]– date and time; **TIME** - time only; **DATE** – date only. |

* **What is NULL [an undefined value –** all values are null by default unless otherwise specified**]?**
* **Create a table with all the data that require values to be inserted.**
* **Create a table with a default value of no other value is specified.**
* **Select null from a table.**

|  |
| --- |
| NULL appears in columns with no assigned value. NULL != 0 && NULL != “ ” && NULL != NULL. |
| **CREATE TABLE** table\_name - create the table with the column that always must have a value.  **(**  column\_name1 **VARCHAR(10) NOT NULL,**  column\_name1 **DEC(3,2) NOT NULL DEFAULT 2.00**  **);** |
| **Wrong!** **SELECT** column\_name **FROM** table\_name **WHERE** calories =NULL**;** - cannot select NULL directly.  **SELECT** column\_name **FROM** table\_name **WHERE** calories **IS NULL;** - select NULL indirectly only. |

* **Syntax: quotes, case sensitive, comparison operators, basic math operators, backticks.**

|  |
| --- |
| * Single quotes [no double quotes]: DATETIME, TIME or TIMESTAMP, DATE, VARCHAR, CHAR, BLOB. * No quotes for numbers, if quotes are used – no error: INT, DEC. * Backslash or an extra single quote needed when used in the text data types: ‘Vlad\’s’ or ‘Vlad’’s’. * SQL is case insensitive. Capitalization of statements is good practice in SQL. * Less than: <; Greater than: >; Equal to: =; Not Equal to: < >; Less than or equal to: <=; Greater than or equal to >=. * Comparison of Strings. **Ex: BETWEEN ‘G’ AND ‘O’** any String that begins with G, O and all letters in between. * Basic math operations can be performed on number columns. **Ex: SET** table\_name = table\_name + 1 **WHERE…** * **Backticks** whow up when we run the **SHOW CREATE TABLE** command. |

* **Is SQL case sensitive?**
* **Can you create database that already exists?**
* **Create database.**
* **Use database.**
* **Delete database.**

|  |
| --- |
| * Cannot create database that already exists. |
| **CREATE DATABASE** database\_name; - create the database which will hold all the tables. |
| **USE** database\_name; - get inside database to create the tables. |
| **DROP DATABASE** database\_name; - delete table and any data in it. |

* **Where tables are located?**
* **What is the case of table name?**
* **Can you create table that already exists?**
* **Create table.**
* **Show create table.**
* **Describe table.**
* **Delete table.**

|  |
| --- |
| * All tables must be inside of databases. * Cannot create table that already exists. * Table name should be lowercase. |
| **CREATE TABLE** table\_name - create the table containing column names and their corresponding data types.  **(**  column\_name1 **VARCHAR(10),**  column\_name2 **DATE**  **);** |
| **SHOW CREATE TABLE** table\_name; - recreate the table without any data in it. |
| **DESC** table\_name; or **DESCRIBE** table\_name; - display table format, the structure of the table. |
| **DROP TABLE** table\_name; - delete table and any data in it. |

* **Do numeric values use quotes?**
* **Can you change the order of column names?**
* **Insert values (column names).**
* **Insert values (no column names).**
* **Insert values (leave some column names out).**
* **Insert multiple records with one statement.**

|  |
| --- |
| * Numeric values do not use quotes. * You can change the order of column names as long as the matching values for each column come in the same order. |
| **INSERT INTO** table\_name - insert values [no column names] into the table in the same order as column names.  (  column\_name1,  column\_name2  )  **VALUES**  **(**  ‘value1’,  ‘value2’  **);** |
| **INSERT INTO** table\_name - insert values [no column names] into the table in the same order as column names.  **VALUES**  **(**  ‘value1’,  ‘value2’  **);** |
| **INSERT INTO** table\_name - insert values leaving some column names out.  (  column\_name1,  column\_name3  )  **VALUES**  **(**  ‘value1’,  ‘value3’  **);** |
| **INSERT INTO** table\_name - insert multiple records  **VALUES**  **(**  ‘value1’,  ‘value2’  **),**  **(**  ‘value1’,  ‘value2’  **);** |

* **What happens WHERE clause is left out?**
* **Can statement have more than one SET clause?**
* **How does UPDATE clause work?**
* **Delete a single or multiple record. [Do select first! Because other data may be targeted.]**
* **Change data of one column with UPDATE…SET…WHERE clause.**
* **Change data of two columns with UPDATE…SET…WHERE clause.**
* **Use a current column to fill a new column.**
* **Populate the new column using UPDATE.**
* **The CASE expression combines all the UPDATE statements by checking an existing column value with a condition.**

|  |
| --- |
| * If WHERE clause is left out, every column in the SET clause will be updated with a new value. * The statement cannot have more than one SET clause. * UPDATE is not deleting anything, it recycles the old record into the new record, existing data are overriden. |
| **DELETE FROM** table\_name **WHERE** column\_name = ‘value’; - delete a single record, include a precise WHERE clause. |
| **UPDATE** table\_name **SET** column\_name = ‘value’ **WHERE** column\_name = ‘value1’; |
| **UPDATE** table\_name **SET** column\_name = ‘value’ **WHERE** column\_name1 = ‘value1’, column\_name2 = ‘value2’; |
| **UPDATE** table\_name **SET** column\_name = **RIGHT(**column\_name, 2**);** |
| **INSERT INTO** table\_name - insert multiple records  **VALUES**  **(**  ‘value1’,  ‘value2’  **),**  **(**  ‘value1’,  ‘value2’  **);** |
| **UPDATE** table\_name  **SET** column\_name =  **WHEN** column\_name1 = ‘value’  **THEN** ‘new\_value1’  **WHEN** column\_name2 = ‘value’ **AND** column\_name2 = ‘value’  **THEN** ‘new\_value2’  **ELSE** ‘new\_value3’  **END**; |

* **Select all columns.**
* **Select all columns using WHERE clause and LIMIT clause [SQL start counting at 0].**
* **Select columns from 20 to 30 out of 100.**
* **Select specific columns.**
* **Select specific columns with AND/OR clause and WHERE clause.**
* **Select columns with wildcard [searches for part of the text].**
* **Select columns with BETWEEN…AND… clause. [Equivalent to <=, >= symbols] .**
* **Select columns with IN clause.**
* **Select columns with NOT clause.**
* **Select one column with ORDER BY clause.** [**Order:** null,numbers, upper letters, lower letters, symbols]
* **Select two columns with ORDER BY clause and reversed order.** [By default, SQL returns results in **ASC** order]

|  |
| --- |
| **SELECT \* FROM** table\_name; |
| **SELECT \* FROM** table\_name **WHERE** column\_name = ‘value’ **LIMIT 2**; |
| **SELECT \* FROM** table\_name **WHERE** column\_name = ‘value’ **LIMIT 20, 30**; |
| **SELECT** column\_name1, column\_name4 **FROM** table\_name **WHERE** column\_name = 10; |
| **SELECT** column\_name **FROM** table\_name **WHERE** column\_name1 = 10 **AND** column\_name2 = ‘value’; |
| **SELECT** column\_name **FROM** table\_name **WHERE** column\_name1 < 10 **OR** column\_name2 <> ‘value’; |
| **SELECT** column\_name **FROM** table\_name **WHERE** column\_name **BETWEEN** 30 **AND** 60; |
| **Instead of: SELECT \* FROM** table\_name **WHERE** column\_name1 = ’value1’ **OR** column\_name1 = ’value2’;  **Use: SELECT \* FROM** table\_name **WHERE** column\_name1 **IN**(’value1’, ’value2’); |
| **SELECT \* FROM** table\_name **WHERE NOT** column\_name1 = ’value1’ **AND NOT** column\_name2 **LIKE** ’%val’; |
| **Wildcard LIKE or \_: SELECT \* FROM** table\_name **WHERE** column\_name **LIKE** ‘**%**ue’ **OR** ‘**\_**ue**%**’ **OR** ‘**%**al**%**’; |
| **SELECT** column\_name1, column\_name4 **FROM** table\_name **WHERE** column\_name = 10 **ORDER BY** column\_name1; |
| **SELECT** column\_name1, column\_name4 **FROM** table\_name **ORDER BY** column\_name1 **ASC**, column\_name2 **DESC**; |

* **Explain change, modify, add, drop.**
* **Add a column to the end of the table.**
* **Add a column to a table using AFTER clause [FIRST, SECOND, BEFORE** column\_name, **LAST], a specified location.**
* **Rename the table.**
* **Change column name, set it to increment, change data type, and make it a primary key.**
* **Change two columns with one statement.**
* **Change data type of an existing table.**
* **Drop a column.**
* **Drop a primary key.**

|  |
| --- |
| * **CHANGE** both the name and data type of an existing column – if wrong data type, may lose data. * **MODIFY** the data type or position of an existing column. Can’t modify the order of columns of an existing table. * **ADD** both the name and data type. Change the column order when adding a new column. * **DROP** a column with all the data in it. |
| **ALTER TABLE** table\_name **ADD COLUMN** column\_name1 **VARCHAR(10);** |
| **ALTER TABLE** table\_name **ADD COLUMN** column\_name2 **VARCHAR(20) AFTER** column\_name**;** |
| **ALTER TABLE** table\_name **RENAME TO** table\_name2; |
| **ALTER TABLE** table\_name **CHANGE COLUMN** column\_name column\_id **INT NOT NULL AUTO\_INCREMENT, ADD PRIMARY KEY(**column\_id**);** |
| **ALTER TABLE** table\_name **CHANGE COLUMN** column\_name column\_name2 **VARCHAR(20), CHANGE COLUMN** column\_name3 column\_name4 **INT;** |
| **ALTER TABLE** table\_name **CHANGE COLUMN** column\_name column\_name **NEWTYPE;**  **or: ALTER TABLE** table\_name **MODIFY COLUMN** column\_name **NEWTYPE;** |
| **ALTER TABLE** table\_name **DROP COLUMN** column\_name; |
| **ALTER TABLE** table\_name **DROP PRIMARY KEY;** |

**\*\*\* String functions do not change the data stored in the table, the result gets returned. \*\*\***

* **Select specified number of characters from right/left side of String.**
* **Select everything in front of the first comma.**
* **Select String, starting position and length that you get back.**
* **Select String to upper/lower case.**
* **Reverse String.**
* **Remove white spaces from left or right side of String.**
* **Get the length of String.**

|  |
| --- |
| **SELECT RIGHT(**column\_name, 2**);** |
| **SELECT SUBSTRING\_INDEX(**column\_name, ‘,’,1**) FROM** table\_name**;** |
| **SELECT SUBSTRING(‘**Brooklyn, NY**’,** 5, 3**);** |
| **SELECT UPPER(‘**aaa**’); or SELECT LOWER(‘**AaA**’);** |
| **SELECT REVERSE(‘**Hello!**’);** |
| **SELECT LTRIM (‘** hello **’); or SELECT RTRIM (‘** hello **’);** |
| **SELECT LENGTH(**‘string’**);** |

* **Sum the values using SUM clause**
* **Sum the columns and group them with GROUP BY clause.**
* **Find the average of the column with AVG clause and group them with GROUP BY clause.**
* **Find the largest or the smallest value in the table.**
* **Select distinct values, no duplicates.**
* **Count distinct values in the column.**

|  |
| --- |
| **SELECT SUM(**column\_name2**) FROM** table\_name **WHERE** column\_name1 = ‘value’**;** |
| **SELECT** column\_name, **SUM(**column\_name2**) FROM** table\_name **GROUP BY** column\_name1 **ORDER BY SUM(**column\_name**) DESC;** |
| **SELECT** column\_name, **AVG(**column\_name2**) FROM** table\_name **GROUP BY** column\_name**;** |
| **SELECT** column\_name, **MAX(**column\_name2**) FROM** table\_name **GROUP BY** column\_name**;** |
| **SELECT DISTINCT** column\_name **FROM** table\_name **ORDER BY** column\_name; |
| **SELECT COUNT(DISTINCT** column\_name) **FROM** column\_name1**;** |