**What is SQL?**

**Relational databases** store data in tables, which are connected to each other in a variety of different ways. Tables contain columns and rows of information, with each column specifying the data type of the information within, and each row having a unique key.

In order to access our information, we need to use a tool that can talk to a relational database. **Structured Query Language** or **SQL** is the main tool used by programmers to work with these data structures.

SQL is a **Relational Database Management System** or **RDMS**. We can use SQL to perform many essential operations on a database, such as adding and removing data.

When we want to perform an operation on a database, we write a SQL **query**.

Ultimately, we can boil down each query’s purpose into one of the following four categories:

1. Create
2. Read
3. Update
4. Delete

**CRUD** or *Create*, *Read*, *Update*, *Delete* represents the four major operations we perform when we work with data.

## Create

When we talk about creating something in SQL, we could want to create a table, add a row, or add a column.

### Creating a Table

#### 1. Creating a Table from Scratch

**CREATE** **TABLE** li\_wedding (

guest\_id **INT**,

last\_name **VARCHAR**(**255**),

first\_name **VARCHAR**(**255**),

attending **BOOL**,

diet **VARCHAR**(**255**)

);

#### 2. Creating a Table from Another Table

**CREATE** **TABLE** johnson\_vow\_renewal

**AS** **SELECT** guest\_id, last\_name, first\_name, attending, diet

**FROM** johnson\_wedding

**WHERE** attending = **1**;

### Adding a Row

**INSERT** **INTO** johnson\_vow\_renewal

**VALUES** (**185**, "Johnson", "Eliza", **1**, "Vegan");

### Adding a column

**ALTER** **TABLE** li\_wedding

**ADD** can\_drink **boolean**;

## Read

When reading data, we don’t want to modify anything, we just want to know what is there. In order to get information from a table, we need to use a **SELECT** statement.

**SELECT** column\_name\_1, column\_name\_2, ...

**FROM** **table\_name**

**WHERE** **some** conditional **is** true.

we will use a **\*** to denote that we want all columns.

**SELECT** \*

**FROM** li\_wedding

## Update

**UPDATE** johnson\_vow\_renewal

**SET** diet="vegetarian"

**WHERE** guest\_id=**185**;

## Delete

**DELETE** **FROM** johnson\_vow\_renewal **WHERE** guest\_id=**107**;

**Joins**

A **join** combines two tables into one result set. We can use joins when we want to query two tables at the same time. Whenever we join two tables, we have to specify the condition upon which the tables need to be joined.

In SQL, there are four different types of joins:

1. Inner Join
2. Left Outer Join
3. Right Outer Join
4. Full Outer Join

No matter which join you are working with, the general syntax for the query looks like so:

|  |  |
| --- | --- |
| 1  2  3 | **SELECT** column\_name\_1, column\_name\_2, ....  **FROM** table\_a  TYPEOFJOIN **JOIN** table\_b **ON** table\_a.column\_name\_1 = table\_b.column\_name\_1; |

## Inner Join

Joining two tables with an **inner join** produces a result set that only includes the values that are present in both tables.

## Left Outer Join

Joining two tables with a **left outer join** gives us a result set which includes all values in the left table and any matching records from the right table.

## Right Outer Join

Joining two tables with a **right outer join** gives us a result set that includes all values in the right table and any matching records from the left table.

## Full Outer Join

Joining two tables with a **full outer join** gives us a result set that includes all records from both tables. Full outer joins are important to SQL, but the syntax is not supported in MySQL. Instead, to achieve a full outer join, you have to work with a left outer join and a right outer join.

## One-To-Many Relationships

A common structure for database tables is the **one-to-many relationship**. In this form, each entry in one table relates to many rows in a different table. For example, a single row in **courses** represents one class offered by a school. That single course can be taught by multiple teachers, and lots of different students will be enrolled in it. Thus, one entry in the **courses** table relates to many entries in **teachers** and **students**.

To connect the **courses** table to **teachers** in a one-to-many relationship, the following conditions must be met:

1. Each table must include a **primary key** column. A primary key is a unique, numerical identifier given to an entry.
2. The **teachers** table must include a **foreign key** column. Foreign keys are integers that tie directly to the entries in a different table. In our school example, the foreign key for a row in **teachers** matches one primary key in **courses**.

Note that different teacher entries could have the same value for the foreign key. This sets up the one-to-many link between a single row in **courses** and multiple rows in **teachers**

In a one-to-one relationship, each row in one database table is linked to 1 and only 1 other row in another table. In a one-to-one relationship between Table A and Table B, each row in Table A is linked to another row in Table B. The number of rows in Table A must equal the number of rows in Table B

In a many-to-many relationship, one or more rows in a table can be related to 0, 1 or many rows in another table. In a many-to-many relationship between Table A and Table B, each row in Table A is linked to 0, 1 or many rows in Table B and vice versa. A 3rd table called a mapping table is required in order to implement such a relationship.  
  
To illustrate the many-to-many relationship consider the sample table design for a bank below:  
  
customers table

|  |  |
| --- | --- |
| **column** |  |
| cust\_id | primary key |
| lastname |  |
| firstname |  |

products table

|  |  |
| --- | --- |
| **column** |  |
| product\_id | primary key |
| name |  |

mapping table

|  |
| --- |
| **column** |
| cust\_id |
| product\_id |