**SQL Interview Preparation**

**Lecture 1**

**DBMS: Database Management System**

Insert img1

* Program to interact with the database.
* Create and manage the database.
* Store databases and database application.
* Interface between the database and database application or user.
* Manage 3 important things:
* The data
* Database Engine – used for storing and retrieving data
* Database Schema - database logical structure

Types of DBMS

* RDMS - Relational database management system
* NoSQL DBMS
* IMDBMS – In memory database management system
* CDBMS – Column database management system
* Cloud database management system

Popular DBMS: MySQL, Oracle, MSSQL, PSQL, IBM DB2

**RDBMS: Relational Database Management System**

* Is actually a DBMS used to manage relational database
* Relational database is the database that follows the model by E. F. Codd.
* Basis for SQL and all modern database systems.
* Stores data in related tables
* A single database can be spread across several tables.

**Lecture 2**

**Difference between DB Software, DB Engine, DB Server**

**DB Software**

Also known as DBMS.

**DB Engine**

Also known as Storage Engine.

Part of DBMS that actually stores and retrieves data.

Underlying software component used by DBMS to create, read, update and delete data.

Application Programming Interface(API) to directly control the database engine.

Many DBMS support multiple storage engines. Ex: MySQL supports InnoDB & MyISAM

**DB Server**

Program which provides database services to other computer program or computers using client-server model

A computer dedicated to running such a program

Performs tasks such as data analysis, storage, data manipulation, archiving.

**Lecture 3**

**What is a database**

Database is structured set of data for quick access, manage and update.

Database is a collection of one or more tables and tables contain data rows or records.

Example: A Refrigerator with different compartments.

Before using database, User must have appropriate privileges. User must select the database.

MySQL console commands:

1. -   SHOW DATABASE
2. -   CREATE DATABASE school;
3. -   USE school
4. -   DROP DATABASE school

**What is a table**

Table consists of vertical columns or fields and horizontal rows

Each column is a property of the item, each row is an item

Cell is the unit where a column and row intersects. Data elements or values are stores in cells.

Table have specified number of fields but can have any number of rows

Single database must contain a unique table name. In multiple databases, same table name can exists in other databases.

1. CREATE TABLE students(
2. id INT NOT NULL PRIMARY KEY,
3. name VARCHAR(40) NOT NULL,
4. **class** VARCHAR(20),
5. age INT
6. );
7. DESCRIBE students;
8. DELETE TABLE students;

**Lecture 4**

**Keywords, Identifiers, Constants, Clauses**

SQL statements are made up of special words Keywords, Identifiers, Constants & Clauses

Keywords: SQL standard words used to construct the SQL statements. Some Keywords are optional while some are mandatory

Identifiers: Names, we give to databases, tables columns

Constants: Literals representing fixed a value

Clauses: Portion of an SQL statement. Clause name corresponds to the SQL keywords.

Example of containing them all:

1. SELECT name
2. FROM students
3. WHERE studentId = 5;

SQL statement

* Is a standalone element that express some action to be carried out
* A syntax: keywords, identifiers, constants form a valid statement
* SQL is not case sensitive
* Convention is to use keywords in uppercase
* Ends with a semicolon

**Lecture 5**

**Data manipulation language & Data definition language**

DDL and DML both are SQL statements.

DDL changes the database structure while DML changes only the data.

DDL:

* Manages database objects like tables and columns
* Changes the database structure
* Tables and columns are created modified and removed
* CREATE, ALTER and DROP

DML:

* Manage the data that resides in our tables and columns
* Changes only the data
* Data inside a table is inserted updated or deleted
* INSERT, UPDATE and DELETE

**Lecture 5**

**Conditions and operators**

1. **WHERE**

* Depends up on a condition which evaluates as either be TRUE, FALSE, UNKOWN
* A condition is made up of keywords, identifiers, constants to compare values with data rows values. If a condition is matched, its called as a TRUE condition else a FALSE condition
* SELECT name
* FROM students
* WHERE studentId = 5;
* The FROM clause will create an intermediate result set containing all the data rows. Then filter records using the condition studentId = 5. It compares each row studentId column value to the constant value 6, and returns the data rows where condition is TRUE. It happens all at once.
* WHERE condition can be TRUE for one or more than one data row.

1. **BETWEEN**

* Is used to compare a range of values. WHERE age BETWEEN 6 AND 10;

1. **Numeric operators**

* Addition, Subtraction, Multiplication, Division.
* Can be used directly in the SQL statements in combination with column values.

1. **Concatenation operators**
2. SELECT firstName || ',' || lastName FROM students;
4. SELECT CONCAT(firstName,'',lastName);

* Joins two strings together. Used only for strings.

1. **Temporal operators**

* Intervals are used while using date time in SQL
* There are different functions based on the different like [MySQL](https://dev.mysql.com/doc/refman/8.0/en/datetime.html) or [PSQL](https://www.postgresql.org/docs/9.1/datatype-datetime.html) or [MSSQL](https://docs.microsoft.com/en-us/sql/t-sql/data-types/datetime-transact-sql?view=sql-server-ver15)

1. **NOT**

* Can be combined with IN or EXISTS condition to negate the condition
* It will make the condition FALSE if is TRUE.

1. **IN**

* Evaluates multiple values on a single data column
* Displays the row where any of the give values is true

1. **EXISTS**

* Is very similar with IN condition, it doesn’t evaluate constants but a subquery and if the condition is TRUE it will evaluate the main query.
* [Optimization differences.](https://stackoverflow.com/questions/24606985/sql-in-vs-exists)

1. SELECT \*
2. FROM employees
3. WHERE EXISTS(SELECT \*
4. FROM projects
5. WHERE projects.employeeId = employees.employeeId);
6. **NOT IN, NOT EXISTS**

* Same theory as NOT keyword

1. **ALL**

* [Good examples here no need to copy paste.](https://stackoverflow.com/questions/5980474/sql-any-all-operators)

1. **ANY**

* [Seems to be equivalent to IN operator](https://stackoverflow.com/questions/3699356/difference-between-in-and-any-operators-in-sql/23259089)

1. **SOME**

* [Equivalent to ANY operator](https://stackoverflow.com/questions/1383988/tsql-some-any-why-are-they-same-with-different-names)

**Lecture 6**

Order by is used to sort or order data rows in a result set. Default order is ASC.

Can sort data by multiple columns or cases for setting some specific data at the end.

I can also order by position number of columns in the select statement.

SELECT Name

FROM #Fruit

ORDER BY CASE WHEN Name IS NULL THEN 1 ELSE 0 END, Name;

SELECT Name, Designation

FROM #Employee

ORDER BY CASE WHEN Designation = 'CEO' THEN 0

WHEN Designation = 'CFO' THEN 1

WHEN Designation = 'VP' THEN 2

WHEN Designation = 'Manager' THEN 3

WHEN Designation = 'Onshore Head' THEN 4

WHEN Designation = 'Trainee' THEN 5

ELSE 6

END, Name;

**Lecture 7**

**GROUP BY**

Group by is used to group identical data into groups with the help of aggregate functions. If a particular column has the same value in different rows then it will arrange these rows in a group.

1. SELECT NAME, SUM(SALARY) FROM Employee
2. GROUP BY NAME;

Group by can work on multiple columns.

**HAVING**

* Used because WHERE can’t be used with aggregate functions.
* HAVING clause filters the tuples produced by the GROUP BY clause.
* WHERE clause filters the intermediate data result rows, while HAVING clause operates on group rows.
* We can use conditions or even subqueries inside HAVING for complex queries but we can use only rows in GROUP BY.
* - 9. Write a query in SQL to find the name **and** country of the referee who managed the final match.
* SELECT sc.country\_name, rm.referee\_name
* FROM soccer\_country sc
* INNER JOIN referee\_mast rm ON sc.country\_id = rm.country\_id
* INNER JOIN match\_mast mm ON rm.referee\_id = mm.referee\_id
* GROUP BY sc.country\_name, rm.referee\_name, mm.match\_no
* HAVING mm.match\_no = (SELECT MAX(match\_no)
* FROM match\_mast)

**Lecture 8**

* 1. **Aliasses**
* Aliases are giving a temporary name to a table or column and are making the query more readable.
* Using AS keyword it will give the alias to your column or table. Names of columns or tables are not accessible anymore.
* Very useful for self joins, where we can give to the same table different aliases and join the table by a column in a WHERE clause. Also we can have a subquery in the from clause with an alias and join it in the WHERE clause.
* Aliases can be given to subqueries.
  1. **Wildcards**
* A wildcard character is used as a substitute for any other character in a string.
* Are used to search for data in a given column. They are used with LIKE operator.
* The percent (%) means zero or more character and underscore (\_) means exactly one character.
* WHERE CustomerName LIKE 'a%'    Finds any values that starts with "a"
* WHERE CustomerName LIKE '%a'    Finds any values that ends with "a"
* WHERE CustomerName LIKE '%or%'  Finds any values that have "or" in any position
* WHERE CustomerName LIKE '\_r%'   Finds any values that have "r" in the second position
* WHERE CustomerName LIKE 'a\_%\_%' Finds any values that starts with "a" **and** are at least 3 characters in length
* WHERE ContactName LIKE 'a%o'    Finds any values that starts with "a" **and** ends with "o"
  1. **TOP/LIMIT**

Specify the numbers of rows to return

1. SELECT TOP 5 FROM students - MMSQL
2. SELECT \* FROM students WHERE ROWNUM <=4; ORACLE
3. SELECT \* FROM students LIMIT 5 MySQL
   1. **INSERT INTO**

* Used to insert data into a table
* INSERT INTO Customers (CustomerName, City, Country)
* VALUES ('Cardinal', 'Stavanger', 'Norway');
* Can insert multiple rows at once
* INSERT INTO MyTable
* ( Column1, Column2, Column3 )
* VALUES
* ('John', 123, 'Lloyds Office'),
* ('Jane', 124, 'Lloyds Office'),
* ('Billy', 125, 'London Office'),
* ('Miranda', 126, 'Bristol Office');
  1. **INSERT INTO SELECT**
* Copy data rows from one table to another table. Copying from source table to the target table both actions are used in a single SQL statement.
* Is required similar table definition on both tables.
* Any existing data rows or records in target table remain unaffected.
* INSERT INTO Customers (CustomerName, City, Country)
* SELECT SupplierName, City, Country FROM Suppliers
* WHERE Country='Germany';
  1. **UPDATE**
* Used to update the data rows in a table
* Can update multiple values in a single SQL statement
* WHERE clause is used to specify which data row to be updated, if there is no clause it will update all rows from the table for the column in the SET statement.
* UPDATE Customers
* SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
* WHERE CustomerID = 1;
  1. **DELETE**
* Same theory as in UPDATE but different syntax.
* DELETE FROM Customers
* WHERE CustomerName='Alfreds Futterkiste';
  1. **SQL Injection**
* Is a method where malicious user can inject some SQL commands to display other information or destroy the database, using form fields on a web page or application.
* SQL statements are used to manage the database from a web page or application. Users can interact with the database using form fields. SQL statements are text keywords and can be changed dynamically.
* By using SQL injection, a hacker my get access to sensitive data.
* Mainly 3 types of SQL injections:
  + Using 1 = 1;
  + Using “”=””
  + Using batched SQL statements

1. Form field:
2. 15; DROP TABLE students;
3. PHP code:
4. Some POST method specific to the language
5. Final SQL statement:
6. SELECT \*
7. FROM users
8. WHERE userid = 15;
9. DROP TABLE students;

# Extra references:

# [SQL injection ,why should you write 0 or 1=1](https://stackoverflow.com/questions/44629836/sql-injection-why-should-you-write-0-or-1-1)

# [What is SQL injection?](https://stackoverflow.com/questions/601300/what-is-sql-injection)

**Lecture 9**

**JOINS**

**1. JOIN or INNER JOIN**

**2. OUTER JOIN**

**2.1 LEFT OUTER JOIN or LEFT JOIN**

**2.2 RIGHT OUTER JOIN or RIGHT JOIN**

**2.3 FULL OUTER JOIN or FULL JOIN**

**3. NATURAL JOIN**

**4. CROSS JOIN**

**5. SELF JOIN**

# All the theory I need:

# [SQL JOIN and different types of JOINs](https://stackoverflow.com/questions/17946221/sql-join-and-different-types-of-joins)

**Lecture 9**

**UNION & UNION ALL**

Concatenate two tables via two SELECT statements. UNION ALL allows duplicates, UNION will show distinct values

# [What is the difference between UNION and UNION ALL?](https://stackoverflow.com/questions/49925/what-is-the-difference-between-union-and-union-all)

**Lecture 10**

**VIEWS**

View is a database object that can be created like a table.

A view is similar to a virtual table. Unlike tables views don’t store data.

Views are complex SELECT statements used as a virtual tables for ease of reference and reuse.

The views are useful for storing complex SQL statement as a virtual table and request the view as single table instead of a complex query.

We can restrict users from accessing underlying tables and instead give access to views with limited columns

Everytime a user request a view, database engine recreate the result set, which always returns up-to-date data rows from views.

More info: [What is the purpose of views in SQL? [duplicate]](https://stackoverflow.com/questions/2381195/what-is-the-purpose-of-views-in-sql)

**Lecture 10**

**FUNCTIONS**

* 1. **AGGREGATE FUNCTIONS:**

Only work on group of rows to return single aggregate result value.

* 1. **COUNT()**
* Used to count the data rows returned in the result set.
* Counts distinct or all values in data rows returned in a result set.
* Does not count NULL values

1. SELECT COUNT(\*) AS total\_rows FROM tablename;
2. SELECT COUNT(DISTINCT columnname) AS total\_rows FROM tablename;
   1. **MIN()**

* Returns the smallest value of a given column.
* Works on numeri column as well as string columns.
  1. **MAX()**
* Returns the largest value of the given column.
* The MAX function works on numeric columns as well as string columns.
  1. **SUM()**
* Returns the total sum of a given numeric column.
* The MAX function works only on numeric columns.
  1. **AVG()**
* Returns the average value of the given numeric column.
* The AVG function works only on numeric columns.
  1. **FIRST()**
* SQL SELECT FIRST() function returns the first value of selected column.
  1. **LAST()**
* SQL SELECT LAST() function returns the last value of selected column.
  1. **SCALAR FUNCTIONS:**

Work on a single input value to return a single result value.

These functions are different based on the API.

* 1. **LEN()**
* Returns the length of a given string value
* Only works on string columns
  1. **ROUND()**
* Is used to round floating point number to the given decimal numbers
* Only works on numeric columns
  1. **SUBSTRING()**
* Extracts characters from the given string data.
* Only works on string columns.
  1. **CASE()**
* Returns a value or NULL by evaluating a series of conditions.
* CASE function is made from keywords WHEN THEN ELSE and END.
* Works similar with an IF statement in programming languages.
* Evaluates string as well numeric values.
* In simple case an expression is compared to static values while in searched case an expression is compared to multiple logical conditions.
* SELECT OrderID, Quantity,
* CASE
* WHEN Quantity > 30 THEN 'The quantity is greater than 30'
* WHEN Quantity = 30 THEN 'The quantity is 30'
* ELSE 'The quantity is under 30'
* END AS QuantityText
* FROM OrderDetails;
* SELECT CustomerName, City, Country
* FROM Customers
* ORDER BY
* (CASE
* WHEN City IS NULL THEN Country
* ELSE City
* END);
  1. **NOW()**

The NOW() function returns the current date and time.

**Note:** The date and time is returned as "YYYY-MM-DD HH-MM-SS" (string) or as YYYYMMDDHHMMSS.uuuuuu (numeric).

* 1. **UCASE()**

1. SELECT UCASE("SQL Tutorial is FUN!");
   1. **LCASE()**
2. SELECT LCASE("SQL Tutorial is FUN!");
   1. **ROUND()**

If the *operation* parameter is 0 (or not provided), the ROUND function will round the result to the number of *decimal\_places*.

If the *operation* parameter is non-zero, the ROUND function will truncate the result to the number of *decimal\_places*.

1. SELECT ROUND(125.315, 2, 1);
2. Result: 125.310    (result is truncated because 3rd parameter is non-zero)
   1. **CEILING & FLOOR**

**CEILING Example:**

1. SELECT CEILING(32.65);
2. Result: 33
4. SELECT CEILING(32.1);
5. Result: 33
7. SELECT CEILING(32);
8. Result: 32
10. SELECT CEILING(-32.65);
11. Result: -32
13. SELECT CEILING(-32);
14. Result: -32

**FLOOR Example:**

1. SELECT FLOOR(5.9);
2. Result: 5
4. SELECT FLOOR(34.29);
5. Result: 34
7. SELECT FLOOR(-5.9);
8. Result: -6
   1. **FORMAT()**

[Tons of examples here no need to copy paste.](https://docs.microsoft.com/en-us/sql/t-sql/functions/format-transact-sql?view=sql-server-ver15) Might differ based on different DBMS.

* 1. **CHAR\_LENGTH()**

The CHAR\_LENGTH() function return the length of a string (in characters).

1. SELECT CustomerName, CHAR\_LENGTH(CustomerName) AS LengthOfName
2. FROM Customers;
   1. **NULLIF()**

The NULLIF() function compares two expressions and returns NULL if they are equal. Otherwise, the first expression is returned.

Returns a null value if the two specified expressions are equal. For example, SELECT NULLIF(4,4) AS Same, NULLIF(5,7) AS Different; returns NULL for the first column (4 and 4) because the two input values are the same. The second column returns the first value (5) because the two input values are different.

[Detailed Source](https://docs.microsoft.com/en-us/sql/t-sql/language-elements/nullif-transact-sql?redirectedfrom=MSDN&view=sql-server-ver15)

* 1. **COALESCE()**

Returns the first non-NULL value from a given list. If there are no non-NULL values the COALESCE() function will return NULL.

**Different sources since I am new to this:**

* 1. [Microsoft website](https://docs.microsoft.com/en-us/sql/t-sql/language-elements/coalesce-transact-sql?view=sql-server-ver15)
  2. [COALESCE Function in TSQL](https://stackoverflow.com/questions/13366488/coalesce-function-in-tsql)
  3. [Deciding between COALESCE and ISNULL in SQL Server](https://www.mssqltips.com/sqlservertip/2689/deciding-between-coalesce-and-isnull-in-sql-server/)
  4. [Three similar rows avoid NULL return](https://www.youtube.com/watch?v=VbBach66pZw)
  5. [NULLIF, ISNULL, COALESCE](https://www.youtube.com/watch?v=jSNgQ36OvS8)
  6. [Tips where I can use it](https://www.mssqltips.com/sqlservertip/1521/the-many-uses-of-coalesce-in-sql-server/)

**Lecture 11**

**DATA TYPES**

Used to create professional, normalized, relational and easy to manage database columns and table structures.

Every column in a table is defined with a data type depending up on the data value its going to store.

A data type defines what kind of value a column can store.

There are mainly 3 data types:

* Numeric data types,
* Character data types
* Temporal data types.

Numeric data types stores only numeric values.

* Integer
* Fixed-Point
* Floating-Point

Character data types can store alphabets symbols and also numbers

Temporal data types are for date and time

[How to choose the right size for data types](https://chartio.com/learn/databases/how-to-select-the-right-data-types/)

[SQL When to use Which Data Type](https://stackoverflow.com/questions/595489/sql-when-to-use-which-data-type)

[Data types choices.](https://www.sentryone.com/white-papers/data-type-choice-affects-database-performance)

**Lecture 12**

**DDL**

Is used to manipulate the data structures .

The databases are created by using CREATE DATABASE statement.

A valid database name may contain only letters, numbers and underscore symbols. The database name must contain at least one alphabet letter.

The database names are always stored in small letters.

Its good practice to name the database with some meaningful characters, like company name, website name, business name etc.

The existing databases are displayed by using the SHOW DATABASE statement.

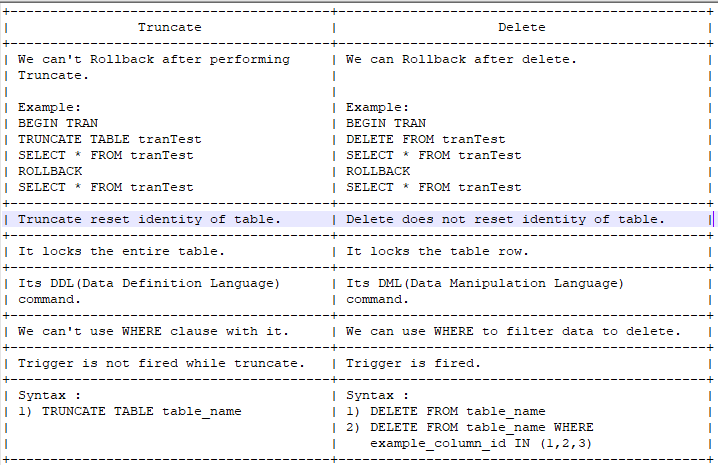
**DDL**

DDL is short name of Data Definition Language, which deals with database schemas and descriptions, of how the data should reside in the database.

* **CREATE – to create database and its objects like (table, index, views, store procedure, function and triggers)**
* -- MySQL example
* -- https://www.mysqltutorial.org/mysql-create-table/
* CREATE DATABASE database\_name;
* SHOW DATABASE;
* CREATE TABLE IF NOT EXISTS tasks (
* task\_id **INT** AUTO\_INCREMENT PRIMARY KEY,
* title VARCHAR(255) NOT NULL,
* start\_date DATE,
* due\_date DATE,
* status TINYINT NOT NULL,
* priority TINYINT NOT NULL,
* description TEXT,
* created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP
* )  ENGINE=INNODB;
* -- ORACLE/PLSQL example
* -- https://www.techonthenet.com/oracle/tables/create\_table.php
* CREATE TABLE employees
* ( employee\_number number(10) NOT NULL,
* employee\_name varchar2(50) NOT NULL,
* department\_id number(10),
* salary number(6),
* CONSTRAINT employees\_pk PRIMARY KEY (employee\_number),
* CONSTRAINT fk\_departments
* FOREIGN KEY (department\_id)
* REFERENCES departments(department\_id)
* );
* -- MSSQL example
* -- https://www.sqlservertutorial.net/sql-server-basics/sql-server-create-table/
* CREATE TABLE sales.visits (
* visit\_id **INT** PRIMARY KEY IDENTITY (1, 1),
* first\_name VARCHAR (50) NOT NULL,
* last\_name VARCHAR (50) NOT NULL,
* visited\_at DATETIME,
* phone VARCHAR(20),
* store\_id **INT** NOT NULL,
* FOREIGN KEY (store\_id) REFERENCES sales.stores (store\_id)
* );
* **ALTER – alters the structure of the existing database**
* -- MySQL example
* -- https://www.mysqltutorial.org/mysql-create-table/
* ALTER TABLE vehicles
* ADD color VARCHAR(50),
* ADD note VARCHAR(255);
* ALTER TABLE vehicles
* MODIFY year SMALLINT NOT NULL,
* MODIFY color VARCHAR(20) NULL AFTER make;
* ALTER TABLE vehicles
* DROP COLUMN vehicleCondition;
* ALTER TABLE table\_name
* RENAME TO new\_table\_name;
* ALTER TABLE vehicles
* CHANGE COLUMN note vehicleCondition VARCHAR(100) NOT NULL;
* -- ORACLE/PLSQL example
* -- https://www.techonthenet.com/oracle/tables/create\_table.php
* ALTER TABLE customers
* ADD (customer\_name varchar2(45),
* city varchar2(40) DEFAULT 'Seattle');
* ALTER TABLE customers
* MODIFY (customer\_name varchar2(100) NOT NULL,
* city varchar2(75) DEFAULT 'Seattle' NOT NULL);
* ALTER TABLE customers
* DROP COLUMN customer\_name;
* ALTER TABLE customers
* RENAME COLUMN customer\_name TO cname;
* -- MSSQL example
* -- https://www.sqlservertutorial.net/sql-server-basics/sql-server-create-table/
* ALTER TABLE supplier
* ADD (supplier\_name **char**(50),
* city **char**(45));
* ALTER TABLE table\_name
* ALTER COLUMN column\_name TYPE column\_definition,
* ALTER COLUMN column\_name TYPE column\_definition,
* ...
* ;
* ALTER TABLE supplier
* DROP COLUMN supplier\_name;
* ALTER TABLE supplier
* CHANGE COLUMN supplier\_name sname VARCHAR(100);
* ALTER TABLE supplier
* RENAME TO vendor;
* **DROP – delete objects from the database**
* -- MySQL example
* -- https://www.mysqltutorial.org/mysql-create-table/
* DROP TABLE IF EXISTS customers, suppliers;
* DROP TEMPORARY TABLE IF EXISTS customers;
* -- ORACLE/PLSQL example
* -- https://www.techonthenet.com/oracle/tables/create\_table.php
* DROP TABLE customers PURGE;
* -- MSSQL example
* -- https://www.sqlservertutorial.net/sql-server-basics/sql-server-create-table/
* DROP TABLE table\_name;
* **TRUNCATE – remove all records from a table, including all spaces allocated for the records are removed**

The TRUNCATE TABLE statement is used to remove all records from a table in MySQL. It performs the same function as a DELETE statement without a WHERE clause.

Both of these statements would result in all data from the customers table being deleted. The main difference between the two is that you can roll back the DELETE statement if you choose, but you can't roll back the TRUNCATE TABLE statement.

****

* **COMMENT – add comments to the data dictionary**
* **RENAME – rename an object**

**Select and use exsiting database**

**Drop database**

**SQL Constraints**

SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table.

Constraints can be divided into the following two types,

**Column level constraints:** Limits only column data.

**Table level constraints:** Limits whole table data.

* 1. **NOT NULL**

**NOT NULL** constraint restricts a column from having a NULL value. Once **NOT NULL** constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value.

Can be defined as string as well as numeric data type.

One important point to note about this constraint is that it cannot be defined at table level

* 1. [**UNIQUE**](https://www.sqlservertutorial.net/sql-server-basics/sql-server-unique-constraint/)

**UNIQUE** constraint ensures that a field or column will only have unique values. A **UNIQUE** constraint field will not have duplicate data. This constraint can be applied at column level or table level.

1. CREATE TABLE table\_name (
2. key\_column data\_type PRIMARY KEY,
3. column1 data\_type,
4. column2 data\_type,
5. column3 data\_type,
6. ...,
7. UNIQUE (column1,column2)
8. );
   1. [**PRIMARY KEY**](https://www.w3schools.com/sql/sql_primarykey.asp)

Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually Primary Key is used to index the data inside the table.

A table typically has a column or combination of columns that contain values that uniquely identify each row in the table. This column, or columns, is called the primary key (PK) of the table and enforces the entity integrity of the table. Because primary key constraints guarantee unique data, they are frequently defined on an identity column.

When you specify a primary key constraint for a table, the Database Engine enforces data uniqueness by automatically creating a unique index for the primary key columns. This index also permits fast access to data when the primary key is used in queries. If a primary key constraint is defined on more than one column, values may be duplicated within one column, but each combination of values from all the columns in the primary key constraint definition must be unique.

1. CREATE table Order\_Detail(
2. order\_id **int** PRIMARY KEY,
3. order\_name varchar(60) NOT NULL,
4. c\_id **int** FOREIGN KEY REFERENCES Customer\_Detail(c\_id)
5. );
   1. [**FOREIGN KEY**](https://www.w3schools.com/sql/sql_foreignkey.asp)

A foreign key (FK) is a column or combination of columns that is used to establish and enforce a link between the data in two tables to control the data that can be stored in the foreign key table. In a foreign key reference, a link is created between two tables when the column or columns that hold the primary key value for one table are referenced by the column or columns in another table. This column becomes a foreign key in the second table.

Using FOREIGN KEY constraint at Column Level

1. ALTER table Order\_Detail ADD FOREIGN KEY (c\_id) REFERENCES Customer\_Detail(c\_id);

**Behaviour of Foriegn Key Column on Delete**

There are two ways to mainting the integrity of data in Child table, when a particular record is deleted in the main table. When two tables are connected with Foriegn key, and certain data in the main table is deleted, for which a record exits in the child table, then we must have some mechanism to save the integrity of data in the child table.

**On Delete Cascade** : This will remove the record from child table, if that value of foriegn key is deleted from the main table.

**On Delete Null** : This will set all the values in that record of child table as NULL, for which the value of foriegn key is deleted from the main table.

If we don't use any of the above, then we cannot delete data from the main table for which data in child table exists. We will get an error if we try to do so.

* 1. [**CHECK**](https://www.sqlservertutorial.net/sql-server-basics/sql-server-check-constraint/)

The CHECK constraint allows you to specify the values in a column that must satisfy a Boolean expression.

The CHECK constraints reject values that cause the Boolean expression evaluates to FALSE.

Because NULL evaluates to UNKNOWN, it can be used in the expression to bypass a constraint.

For example, you can insert a product whose unit price is NULL. To fix this, you need to use a NOT NULL constraint for the unit\_price column.

1. -- Requires positive unit prices
3. CREATE TABLE test.products(
4. product\_id **INT** IDENTITY PRIMARY KEY,
5. product\_name VARCHAR(255) NOT NULL,
6. unit\_price DEC(10,2) CHECK(unit\_price > 0)
7. );
9. -- You can also assign the constraint a separate name by **using** the CONSTRAINT keyword as follows:
11. CREATE TABLE test.products(
12. product\_id **INT** IDENTITY PRIMARY KEY,
13. product\_name VARCHAR(255) NOT NULL,
14. unit\_price DEC(10,2) CONSTRAINT positive\_price CHECK(unit\_price > 0)
15. );
17. -- The **explicit** names help classify the error messages and allow you to refer to the constraints when you -- want to modify them.
19. -- If you don’t specify a constraint name **this** way, SQL Server automatically generates a name **for** you.
21. -- CHECK constraint referring to multiple columns
23. CREATE TABLE test.products(
24. product\_id **INT** IDENTITY PRIMARY KEY,
25. product\_name VARCHAR(255) NOT NULL,
26. unit\_price DEC(10,2),
27. discounted\_price DEC(10,2),
28. CHECK(unit\_price > 0),
29. CHECK(discounted\_price > 0),
30. CONSTRAINT valid\_prices CHECK(discounted\_price > unit\_price)
31. );
33. -- Add CHECK constraints to an existing table
35. ALTER TABLE test.products
36. ADD CONSTRAINT valid\_price
37. CHECK(unit\_price > discounted\_price);
39. -- Remove CHECK constraints
41. ALTER TABLE test.products
42. DROP CONSTRAINT positive\_price;
44. -- Disable CHECK constraints **for** insert or update
46. ALTER TABLE test.products
47. NO CHECK CONSTRAINT valid\_price;
    1. **DEFAULT**

The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:

1. CREATE TABLE Persons (
2. ID **int** NOT NULL,
3. LastName varchar(255) NOT NULL,
4. FirstName varchar(255),
5. Age **int**,
6. City varchar(255) DEFAULT 'Balls'
7. );
   1. **AUTO INCREMENT / SEQUENCE**

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

1. CREATE TABLE Persons (
2. Personid **int** NOT NULL AUTO\_INCREMENT,
3. LastName varchar(255) NOT NULL,
4. FirstName varchar(255),
5. Age **int**,
6. PRIMARY KEY (Personid)
7. );

[**SEQUENCE is characteristic to ORACLE/PLSQL**](file://C:\Users\test\AppData\Roaming\Microsoft\Word\In%20Oracle,%20you%20can%20create%20an%20autonumber%20field%20by%20using%20sequences.%20A%20sequence%20is%20an%20object%20in%20Oracle%20that%20is%20used%20to%20generate%20a%20number%20sequence.%20This%20can%20be%20useful%20when%20you%20need%20to%20create%20a%20unique%20number%20to%20act%20as%20a%20primary%20key.)

[**How to create id with AUTO\_INCREMENT on Oracle**](https://stackoverflow.com/questions/11296361/how-to-create-id-with-auto-increment-on-oracle)

1. CREATE SEQUENCE supplier\_seq
2. MINVALUE 1
3. MAXVALUE 999999999999999999999999999
4. START WITH 1
5. INCREMENT BY 1
6. CACHE 20;

**Other resources:** [**PK & FK**](https://docs.microsoft.com/en-us/sql/relational-databases/tables/primary-and-foreign-key-constraints?view=sql-server-ver15#FKeys)**,**

[**Database relationships**](https://www.youtube.com/watch?v=QpdhBUYk7Kk)**,** [**Difference between UNIQUE and PRIMARY KEY**](https://stackoverflow.com/questions/9565996/difference-between-primary-key-and-unique-key)**,**

[**What are DDL and DML?**](https://stackoverflow.com/questions/2578194/what-are-ddl-and-dml)

[**SQL | DDL, DQL, DML, DCL and TCL Commands**](https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/)**,**

[**MySQL API**](https://www.mysqltutorial.org/basic-mysql-tutorial.aspx)**,** [**PLSQL API**](https://www.techonthenet.com/oracle/index.php)**,** [**MSSQL API**](https://www.techonthenet.com/sql/index.php)

**Lecture 13**

**DCL (DATA CONTROL LANGUAGE)**

You can grant users various privileges to tables. These permissions can be any combination of SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALTER, or ALL.

* 1. **GRANT**
* **GRANT command** gives user's access privileges to the database.
* This command allows specified users to perform specific tasks.
  1. **REVOKE**
* **REVOKE command** is used to cancel previously granted or denied permissions.
* This command withdraw access privileges given with the GRANT command.
* It takes back permissions from user.

1. GRANT SELECT, INSERT, UPDATE, DELETE ON employees TO smithj;
3. GRANT ALL ON employees TO smithj;
5. REVOKE privileges ON object FROM user;
7. REVOKE ALL ON employees FROM anderson;
9. REVOKE SELECT ON employees FROM **public**;

**Other resources:** [**GRANT & REVOKE practical example in oracle**](https://www.youtube.com/watch?v=QmRQ9OvBVZQ)**,** [**GRANT & REVOKE second practical example**](https://www.youtube.com/watch?v=m3QQ56xKr0k)**,** [**Article 1**](https://www.techonthenet.com/sql_server/grant_revoke.php)

**Lecture 14**

**TCL (TRANSACT-SQL)**

A transaction is a single unit of work. If a transaction is successful, all of the data modifications made during the transaction are committed and become a permanent part of the database. If a transaction encounters errors and must be canceled or rolled back, then all of the data modifications are erased.

SQL Server operates in the following transaction modes:

* 1. **Autocommit transactions**  
     Each individual statement is a transaction.
  2. **Explicit transactions**  
     Each transaction is explicitly started with the BEGIN TRANSACTION statement and explicitly ended with a COMMIT or ROLLBACK statement.
  3. **Implicit transactions**  
     A new transaction is implicitly started when the prior transaction completes, but each transaction is explicitly completed with a COMMIT or ROLLBACK statement.
  4. **Batch-scoped transactions**  
     Applicable only to multiple active result sets (MARS), a Transact-SQL explicit or implicit transaction that starts under a MARS session becomes a batch-scoped transaction. A batch-scoped transaction that is not committed or rolled back when a batch completes is automatically rolled back by SQL Server.

**What are the ACID Database Properties?**

The ACID database properties define SQL database key properties to ensure consistent, safe and robust database modification when saved.

ACID is an acronym that helps to remember the fundamental principles of a transnational system. ACID stands for Atomic, Consistent, Isolation, and Durability. Here are some informal definitions:

1. Atomic – In a transaction involving two or more pieces of information, either all or none of the information is saved. Basically, a rule of “all or nothing” is observed.
2. Consistent – The saved data cannot violate the integrity of the database. Interrupted modifications are rolled back to ensure the database is in a state before the change takes place.
3. Isolation – No other transactions take place and affect the transaction in question. This prevents “mid-air collisions.”
4. Durable – Once the transaction is committed, any failure or system restart, returns the data in a correct state. Stated another way, once a transaction is committed, it will remain so, regardless of a subsequent system failure.

What’s an Example of ACID?

It might help to look at ACID and its concepts using an example. Consider a banking transaction where you’re withdrawing money from checking to deposit in your saving account. As part of the transaction, a journal entry is made as an audit record. How would ACID help in this situation?

Since the transaction is Atomic, the money can’t be taken out of your checking account without being subsequently deposited in savings. If the transaction was interrupted for some reason, your account balance would remain unchanged.

Since a record of every transaction is kept in a journal, Consistency ensures that the transfer can’t complete without successfully writing the journal entry. If the journal is full, then the transfer is aborted. Your account balances are returned to their original balances.

Isolation ensures that other banking transactions don’t affect the outcome of your transfer. Other transaction to alter your checking balance must wait until your transaction completes.

Being Durable, once the transaction is saved or committed, it can’t be “lost.” That is, a power outage or system crash won’t cause any of the data to go missing.

**Transaction commands**

* 1. **COMMIT**
     + COMMIT is a transactional command.
     + It saves all transaction to the database since the last COMMIT or ROLLBACK.
  2. **ROLLBACK**
     + ROLLBACK is a transactional command.
     + It undo the transaction that have not been modified in the database
* mysql> begin;
* Query OK, 0 rows affected (0.00 sec)
* mysql> update emp set full\_name = 'Tanya jain' where id = 3;
* Query OK, 1 row affected (0.00 sec)
* Rows matched: 1  Changed: 1  Warnings: 0
* mysql> select \* from emp;
* -- ROLLBACK
* mysql> rollback;
* Query OK, 0 rows affected (0.00 sec)

**3. SAVEPOINT**

* + 1. SAVEPOINT is a transactional command.
    2. It rollback the transaction from a certain SAVEPOINT without rollback the entire transaction.

1. mysql> start transaction;
2. Query OK, 0 rows affected (0.00 sec)
4. mysql> savepoint sp;
5. Query OK, 0 rows affected (0.00 sec)
7. mysql> insert into emp values(9,'raj jain');
8. Query OK, 1 row affected (0.00 sec)
10. mysql> insert into emp values(10,'arpit jain');
11. Query OK, 1 row affected (0.00 sec)
13. mysql> select \* from emp;
15. -- ROLLBACK
17. mysql> rollback to savepoint sp;
18. Query OK, 0 rows affected (0.00 sec)
20. mysql> select \* from emp;

**Other resources:**

[**Transactions syntax example**](https://stackoverflow.com/questions/48759706/t-sql-transaction-syntax/48759748)

[**Transactions syntax example 2**](https://stackoverflow.com/questions/506602/best-way-to-work-with-transactions-in-ms-sql-server-management-studio)

[**Transactions (Transact-SQL)**](https://docs.microsoft.com/en-us/sql/t-sql/language-elements/transactions-transact-sql?view=sql-server-ver15)

[**Commit, Rollback and Savepoint SQL commands**](https://www.studytonight.com/dbms/tcl-command.php)

[**ACID**](https://www.ibm.com/support/knowledgecenter/SSGMCP_5.4.0/product-overview/acid.html)

[**SQL Server Programming Part 14 - Transactions**](https://www.youtube.com/watch?v=is03uRYFgqc)

[**Isolation Levels in TRANSACT-SQL**](https://www.youtube.com/watch?v=ZtPj09tJjnQ)

[**MSSQL - Understanding Isolation Level by Example (Read Uncommitted)**](https://www.youtube.com/watch?v=9ZyxJbPlw-E)

[**Transactions and locking in SQL Server and Oracle (or any sql database)**](https://www.youtube.com/watch?v=FRs6tDVk-FY)

[**DDL commands in Transactions in SQL Server versus Oracle**](https://www.mssqltips.com/sqlservertip/4591/ddl-commands-in-transactions-in-sql-server-versus-oracle/)

[**SQL - Transactions**](https://www.tutorialspoint.com/sql/sql-transactions.htm)

**Lecture 15**

**Database relationships**

Database relationship mean how the data in the one table is related to the data in another table.

In RDBMS. The term relational refers to the tables with relationships

Relationships between tables are created using keys. A key in one table will normally relate to a key in another table.

* 1. **One-To-One:**

**If only one data in one table relates to the only one data in another table.**

**Example**

In the database, we want to store employees, but also their valid identity cards. We’re not interested in storing any other types of documents or identity cards that were previously valid, so we need exactly 1 (or none) identity card for 1 employee.

Let’s check this truly is a one-to-one relation. We’ve been given these rules: **One** employee could have only **one** valid identity card in our system. **One** identity card could belong to only **one** employee. We haven’t used the word “many”, so this can’t be any type of relation including the word “many”.

We could do two things here:

* Store identity card details in the **employee** table. This is how it’s usually done and the reason for doing it differently (as mentioned below) is some kind of exception
* Store identity card details in a separate table and relate these two tables with a foreign key. But that foreign key (**identity\_card.employee\_id**), referencing **employee.id**, should, at the same time, be the primary key of the **identity\_card** table. This way we could have only 1 record per employee

We could decide to go with the second option if we want:

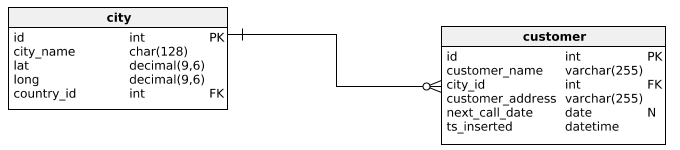
* To keep identity card data separately because we want to keep the model clear and follow the same logic in the whole model (each entity from the real-world has its’ own table in the data model)
* Maybe not all employees will have identity cards, so we’ll spare some storage space this way

Please notice that one-to-one was also implemented in the same manner as one-to-many (1 relation) but with the additional condition (the foreign key is also the primary key).

* 1. **One-To-Many:**

**If only one data in one table relates to the multiple data in another table.**

**Example**

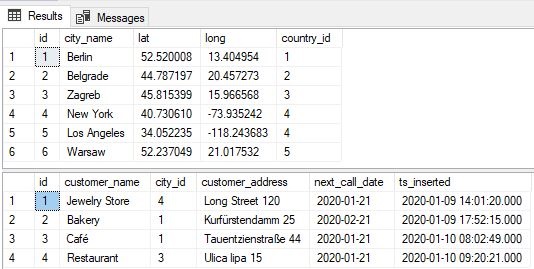


Imagine that we want to store a list of all our customers in the database. For each customer, we also want to store the city where this customer is located, and we know that the customer will be in exactly one city.

We simply established a relation from the **city.id** to **customer.city\_id**. And this works, because the customer can be only in one city and the city could have many different customers located in it.

When you want to determine the nature of the relation you need to establish between two tables just do this. In our example – For **one** city, we could have **many** different customers located in it. And the other way around – For **one** customer, we can have only **one** city it’s located in.

So, how to choose between these 3 different types of relations? If you said the word “many” only once, then this is one-to-many relation. If you would use the word “many” two times, the relation would be many-to-many. And if you wouldn’t use it at all, then it would be one-to-one.

****

We can easily notice few things:

* Not all cites were used (only these with ids 1, 3 and 4 were)
* Each customer had exactly one city it belongs to (**customer.city\_id**)
  1. **Many-To-Many:**

**If multiple data in one table relates to the multiple data in another table.**

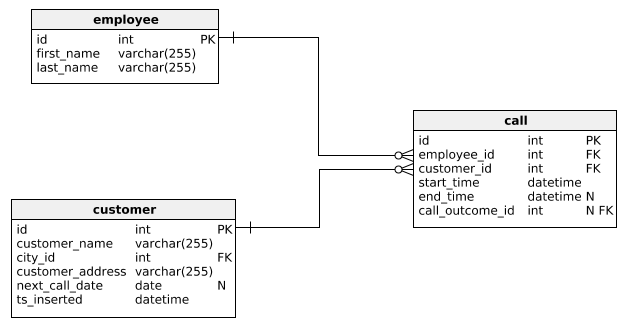
### **Example**

We need to store calls between employees and customers.

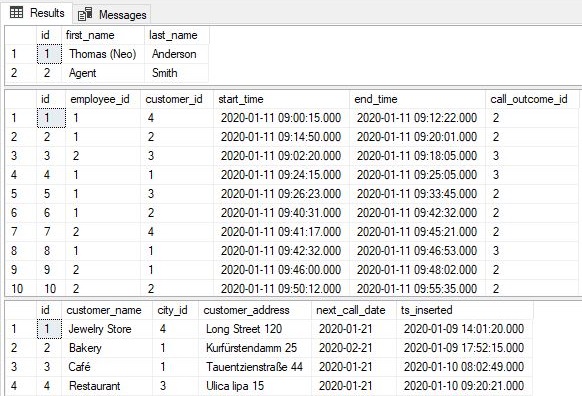
**One** employee, during the time, could call **many** customers. Also, **one** customer, during the time, could receive calls from **many** employees.

Notice that we’ve mentioned the word “many” two times. This is the signal we need to resolve this using many-to-many relation (out of 3 types of relations we have on disposal). To solve it we’ll:

* Add a table between tables **employee** and **customer**
* Add foreign keys (**employee\_id** & **customer\_id**) to that new table (**call**)

****

Now, when we look from the employee perspective, **one** employee could make **many** (multiple) calls. On the other hand, **one** customer could be related to **many** (multiple) calls. Therefore, many-to-many relation is implemented with adding a new table and one-to-many relations from both sides.

****

You can easily notice that the table **call** has attributes **employee\_id** related to the **employee.id** and **customer\_id** related to the **customer.id**. Since they are foreign keys, they hold only values from the set defined in the referenced tables (**employee** & **customer**).

Actually, we have 1 more foreign key here and that is the **call.call\_outcome\_id**. The relation between tables **call** and **call\_outcome** is one-to-many. This means that the table **call** actually relates three tables – **customer**, **employee**, and **call\_outcome**.

The attribute **call.call\_outcome\_id** could contain NULL value (e.g. when the call starts, we still don’t know the outcome and it shall be defined later). That is the reason why, on the relation line, close to the table **call\_outcome**, you can see a little circle (representing “zero”). Other one-to-many relations have a vertical line (representing “one”).

**Other resources:**

[**Difference between one-to-many and many-to-one relationship**](https://stackoverflow.com/questions/4601703/difference-between-one-to-many-and-many-to-one-relationship)**,**

[**DATABASE TABLE RELATIONSHIPS: ONE-TO-ONE VS. ONE-TO-MANY VS. MANY-TO-MANY**](http://www.elcoderino.com/database-table-relationships-one-to-one-vs-one-to-many-vs-many-to-many/)**,**

[**BEST LINK**](https://launchschool.com/books/sql/read/table_relationships)

**Lecture 16**

**Database normalization**

**Most important factor in Database design or Data modeling.**

**Eliminate data redundancies and store the data logically to make the data management easier.**

**Database relationships and keys are useful**

**The series of rules in normalization are called Normal Forms. There mainly 3 types of normal forms:**

* [**First normal form (1NF):**](https://www.studytonight.com/dbms/first-normal-form.php)

#### **Rule 1: Single Valued Attributes**

Each column of your table should be single valued(atomic) which means they should not contain multiple values. We will explain this with help of an example later, let's see the other rules for now.

#### **Rule 2: Attribute Domain should not change**

This is more of a "Common Sense" rule. In each column the values stored must be of the same kind or type.

**For example:** If you have a column dob to save date of births of a set of people, then you cannot or you must not save 'names' of some of them in that column along with 'date of birth' of others in that column. It should hold only 'date of birth' for all the records/rows.

#### **Rule 3: Unique name for Attributes/Columns**

This rule expects that each column in a table should have a unique name. This is to avoid confusion at the time of retrieving data or performing any other operation on the stored data.

If one or more columns have same name, then the DBMS system will be left confused.

#### **Rule 4: Order doesn't matters**

This rule says that the order in which you store the data in your table doesn't matter.

**Example**: Do not store values with comma, just a new row even if there are duplicate values.

* [**Second normal form (2NF):**](https://www.studytonight.com/dbms/second-normal-form.php)

For a table to be in the Second Normal Form, it must satisfy two conditions:

1. The table should be in the First Normal Form.
2. There should be no Partial Dependency.

Let's take an example of a **Student** table with columns student\_id, name, reg\_no(registration number), branch and address(student's home address).

In this table, student\_id is the primary key and will be unique for every row, hence we can use student\_id to fetch any row of data from this table

Even for a case, where student names are same, if we know the student\_id we can easily fetch the correct record.

Hence we can say a **Primary Key** for a table is the column or a group of columns(composite key) which can uniquely identify each record in the table.

I can ask from branch name of student with student\_id **10**, and I can get it. Similarly, if I ask for name of student with student\_id **10** or **11**, I will get it. So all I need is student\_id and every other column **depends** on it, or can be fetched using it.

This is **Dependency** and we also call it **Functional Dependency**.

Now that we know what dependency is, we are in a better state to understand what partial dependency is.

For a simple table like Student, a single column like student\_id can uniquely identfy all the records in a table.

But this is not true all the time. So now let's extend our example to see if more than 1 column together can act as a primary key.

Let's create another table for **Subject**, which will have subject\_id and subject\_name fields and subject\_id will be the primary key.

Now we have a **Student** table with student information and another table **Subject** for storing subject information.

Let's create another table **Score**, to store the **marks** obtained by students in the respective subjects. We will also be saving **name of the teacher** who teaches that subject along with marks.

In the score table we are saving the **student\_id** to know which student's marks are these and **subject\_id** to know for which subject the marks are for.

Together, student\_id + subject\_id forms a **Candidate Key**(learn about [Database Keys](https://www.studytonight.com/dbms/database-key.php)) for this table, which can be the **Primary key**.

Confused, How this combination can be a primary key?

See, if I ask you to get me marks of student with student\_id 10, can you get it from this table? No, because you don't know for which subject. And if I give you subject\_id, you would not know for which student. Hence we need student\_id + subject\_id to uniquely identify any row.

### **But where is Partial Dependency?**

Now if you look at the **Score** table, we have a column names teacher which is only dependent on the subject, for Java it's Java Teacher and for C++ it's C++ Teacher & so on.

Now as we just discussed that the primary key for this table is a composition of two columns which is student\_id & subject\_id but the teacher's name only depends on subject, hence the subject\_id, and has nothing to do with student\_id.

This is **Partial Dependency**, where an attribute in a table depends on only a part of the primary key and not on the whole key.

## **How to remove Partial Dependency?**

There can be many different solutions for this, but out objective is to remove teacher's name from Score table.

The simplest solution is to remove columns teacher from Score table and add it to the Subject table. Hence, the Subject table will become:

* [**Third normal Form (3NF):**](https://www.studytonight.com/dbms/third-normal-form.php)

### **What is Transitive Dependency?**

With exam\_name and total\_marks added to our Score table, it saves more data now. Primary key for our Score table is a composite key, which means it's made up of two attributes or columns → **student\_id + subject\_id**.

Our new column exam\_name depends on both student and subject. For example, a mechanical engineering student will have Workshop exam but a computer science student won't. And for some subjects you have Prctical exams and for some you don't. So we can say that exam\_name is dependent on both student\_id and subject\_id.

And what about our second new column total\_marks? Does it depend on our Score table's primary key?

Well, the column total\_marks depends on exam\_name as with exam type the total score changes. For example, practicals are of less marks while theory exams are of more marks.

But, exam\_name is just another column in the score table. It is not a primary key or even a part of the primary key, and total\_marks depends on it.

This is **Transitive Dependency**. When a non-prime attribute depends on other non-prime attributes rather than depending upon the prime attributes or primary key.

### **How to remove Transitive Dependency?**

Again the solution is very simple. Take out the columns exam\_name and total\_marks from Score table and put them in an **Exam** table and use the exam\_id wherever required.

**Other resources:**

[**Normalization - 1NF, 2NF, 3NF and 4NF**](https://www.youtube.com/watch?v=UrYLYV7WSHM)

[**Basic Concept of Database Normalization - Simple Explanation for Beginners**](https://www.youtube.com/watch?v=xoTyrdT9SZI)

[**What is Normalization in SQL? | Database Normalization Forms - 1NF, 2NF, 3NF, BCNF**](https://www.youtube.com/watch?v=ABwD8IYByfk)

[**What exactly does database normalization do?**](https://stackoverflow.com/questions/1102590/what-exactly-does-database-normalization-do)

[**Normalization in plain English**](https://stackoverflow.com/questions/2331838/normalization-in-plain-english)

[**What is Normalisation (or Normalization)?**](https://stackoverflow.com/questions/246701/what-is-normalisation-or-normalization)

[**Differences between normalizations.**](https://www.quora.com/What-is-the-difference-between-NF-2NF-and-3NF)

**Lecture 17**

**Database import/export**

**Lecture 18**

**MySQL workbench**