NAME: VISHWAK SENAN

DATABASES

- A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS).

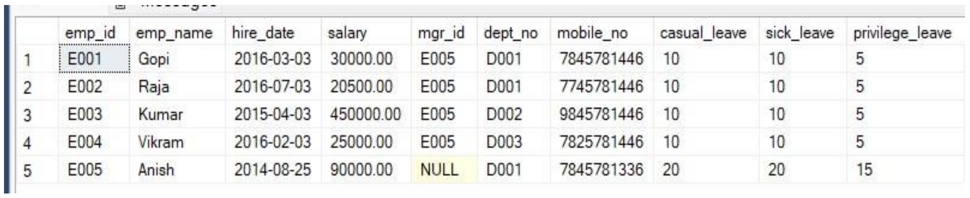
- Flat File Model:
    - A flat file is a collection of data stored in a two-dimensional database in which similar yet discrete strings of information are stored as records in a table. The columns of the table represent one dimension of the database, while each row is a separate record. • Hierarchical Model:
    - Hierarchical Data (SQL Server) A graph of links between Web pages Use hierarchic as a data type to create tables with a hierarchical structure, or to describe the hierarchical structure of data that is stored in another location. Use the hierarchic functions in Transact-SQL to query and manage hierarchical data.
- Network Model:
    - The network model allows creating more complex and more strong queries as compared to the database with a hierarchical database model. A user can execute a variety of database queries when selecting the network model. Disadvantages of a network model. The network model is a very complex database model, so the user must be very familiar with the overall structure of the database. • Relational Data Model:
    - The relational model is the theoretical basis of relational databases, which is a technique or way of structuring data using relations, which are grid-like mathematical structures consisting of columns and rows.

What is RDBMS?

- RDBMS stands for **R**elational **D**atabase **M**anagement **S**ystem. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
- A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

What is a table?

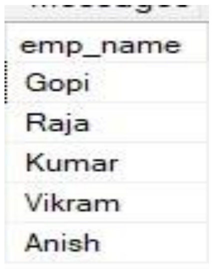
- The data in an RDBMS is stored in database objects which are called as **tables**. This table is basically a collection of related data entries and it consists of numerous columns and rows.
- The below mentioned the table of EMPLOYEE.



	emp_id	emp_name	hire_date	salary	mgr_id	dept_no	mobile_no	casual_leave	sick_leave	privilege_leave
1	E001	Gopi	2016-03-03	30000.00	E005	D001	7845781446	10	10	5
2	E002	Raja	2016-07-03	20500.00	E005	D001	7745781446	10	10	5
3	E003	Kumar	2015-04-03	450000.00	E005	D002	9845781446	10	10	5
4	E004	Vikram	2016-02-03	25000.00	E005	D003	7825781446	10	10	5
5	E005	Anish	2014-08-25	90000.00	NULL	D001	7845781336	20	20	15

What is a field?

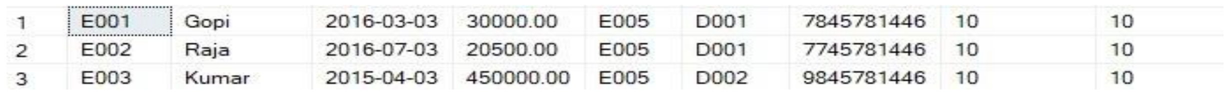
- Every table is broken up into smaller entities called fields. The fields in the EMPLOYEE table consist of emp_id, emp_name, hire_date, salary,mgr_id and etc...
- A field is a column in a table that is designed to maintain specific information about every record in the table.



emp_name
Gopi
Raja
Kumar
Vikram
Anish

What is a Record or a Row?

- A record is also called as a row of data is each individual entry that exists in a table.



1	E001	Gopi	2016-03-03	30000.00	E005	D001	7845781446	10	10
2	E002	Raja	2016-07-03	20500.00	E005	D001	7745781446	10	10
3	E003	Kumar	2015-04-03	450000.00	E005	D002	9845781446	10	10

Data Integrity

The following categories of data integrity exist with each RDBMS −

- **Entity Integrity −** There are no duplicate rows in a table.
- **Domain Integrity −** Enforces valid entries for a given column by restricting the type, the format, or the range of values.
- **Referential integrity −** Rows cannot be deleted, which are used by other records.
- **User-Defined Integrity −** Enforces some specific business rules that do not fall into entity, domain or referential integrity.

Database Normalization

Database normalization is the process of efficiently organizing data in a database. There are two reasons of this normalization process −

- Eliminating redundant data, for example, storing the same data in more than one table.
- Ensuring data dependencies make sense.

Both these reasons are worthy goals as they reduce the amount of space a database consumes and ensures that data is logically stored. Normalization consists of a series of guidelines that help guide you in creating a good database structure.

Normalization guidelines are divided into normal forms; think of a form as the format or the way a database structure is laid out. The aim of normal forms is to organize the database structure, so that it complies with the rules of first normal form, then second normal form and finally the third normal form.

It is your choice to take it further and go to the fourth normal form, fifth normal form and so on, but in general, the third normal form is more than enough.

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)

Functional Dependency

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

- $X \rightarrow Y$

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

For example:

Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

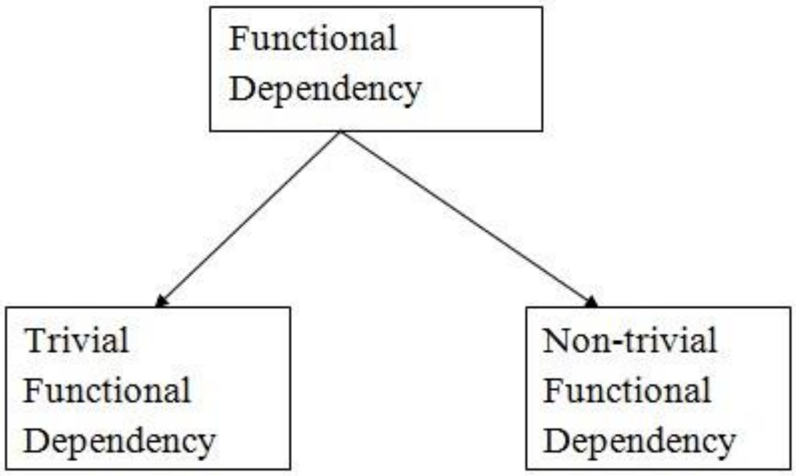
Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

- Emp_Id → Emp_Name

We can say that Emp_Name is functionally dependent on Emp_Id.

Types of Functional dependency


Functional Dependency
Trivial Functional Dependency
Non-trivial Functional Dependency

1. Trivial functional dependency

- A → B has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: A → A, B → B Example:

- Consider a table with two columns Employee_Id and Employee_Name.
- {Employee_id, Employee_Name} → Employee_Id is a trivial functional dependency as
- Employee_Id is a subset of {Employee_Id, Employee_Name}.
- Also, Employee_Id → Employee_Id and Employee_Name → Employee_Name are trivial dependencies too.

2. Non-trivial functional dependency

- A → B has a non-trivial functional dependency if B is not a subset of A.
- When A intersection B is NULL, then A → B is called as complete non-trivial.

Example:

- ID → Name,
- Name → DOB

Normalization

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.
- The normal form is used to reduce redundancy from the database table.

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
4NF	A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
5NF	A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Second Normal Form (2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Third Normal Form (3NF)

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.

1. X is a super key.
2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Example:

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in the table above:

{EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP} ....so on

3. **Candidate key:** {EMP_ID}
4. **Non-prime attributes:** In the given table, all attributes except EMP_ID are non-prime.
5. Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key (EMP_ID). It violates the rule of third normal form.
6. That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.
7. **EMPLOYEE table:**

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

8. **EMPLOYEE_ZIP table:**

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Boyce Codd normal form (BCNF)

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

9. EMP_ID → EMP_COUNTRY

10. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	**EMP_COUNTRY**
264	India
264	India

EMP_DEPT table:

EMP_DEPT	**DEPT_TYPE**	**EMP_DEPT_NO**
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	**EMP_DEPT**
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

11. EMP_ID → EMP_COUNTRY
12. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:

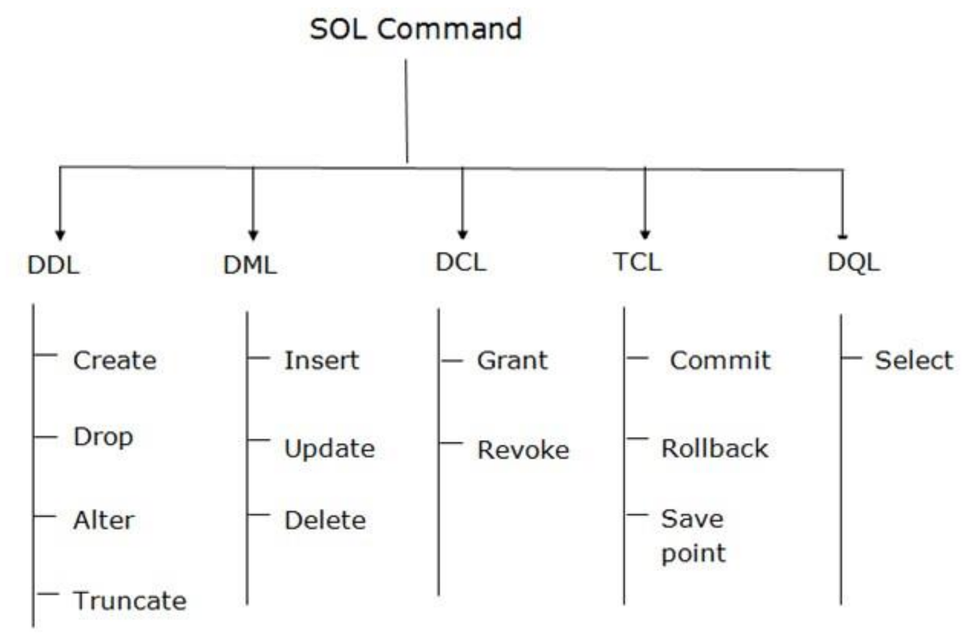
For the first table: EMP_ID
For the second table: EMP_DEPT
For the third table: {EMP_ID, EMP_DEPT}

SQL Commands

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

Types of SQL Commands

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.

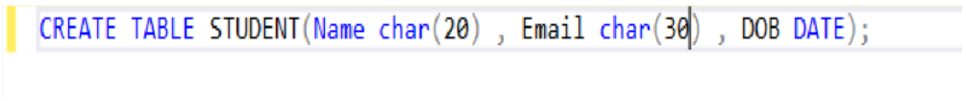


1. Data Definition Language (DDL)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

- CREATE
- ALTER
- DROP
- TRUNCATE



```
CREATE TABLE STUDENT(Name char(20) , Email char(30) , DOB DATE);
```

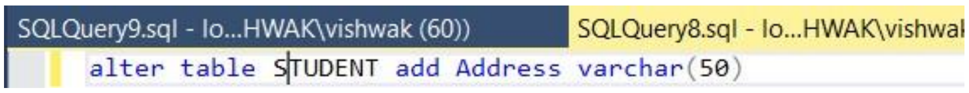
in
the database.

b. DROP: It is used to delete both the structure and record stored in the table.

Syntax

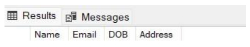
DROP TABLE table_name;

c. ALTER: It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

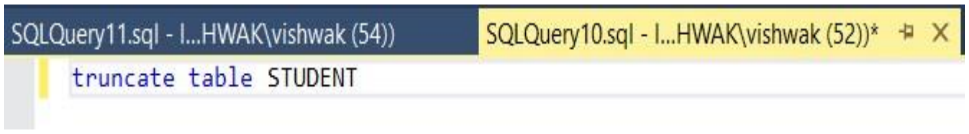

SQLQuery9.sql - lo...HWAK\vishwak (60))
SQLQuery8.sql - lo...HWAK\vishwal

```
alter table STUDENT add Address varchar(50)
```

d. TRUNCATE: It is used to delete all the rows from the table and free the space containing the table.



Name	Email	DOB	Address


SQLQuery11.sql - l...HWAK\vishwak (54))
SQLQuery10.sql - l...HWAK\vishwak (52))*

```
truncate table STUDENT
```

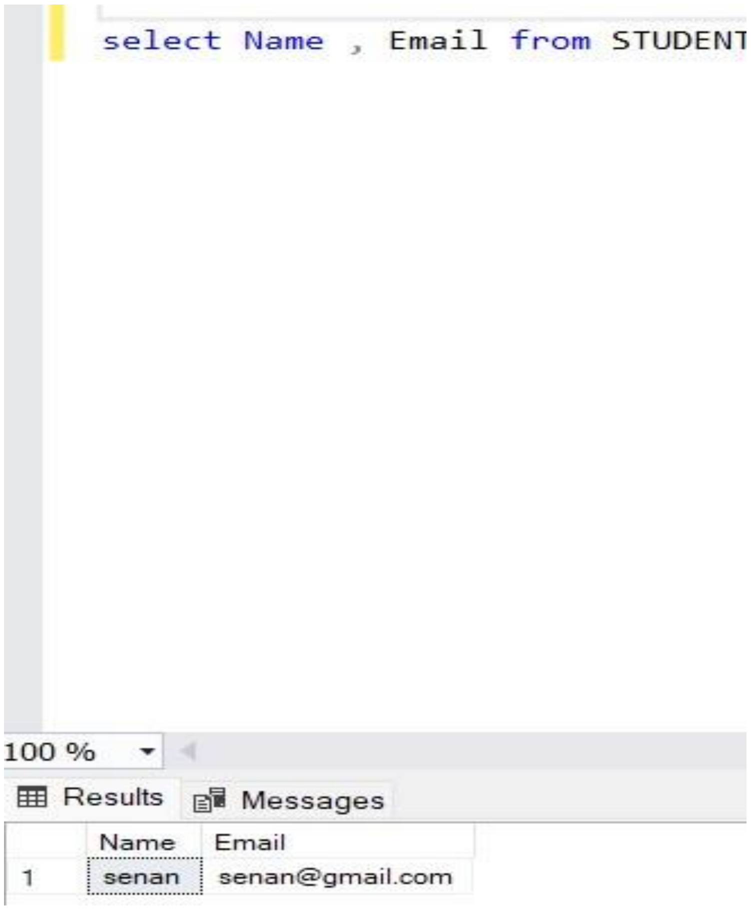
Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

a. SELECT: This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.



```
select Name , Email from STUDENT
```

```
    Name    Email
1   senan   senan@gmail.com
```

KEYS

Primary Key

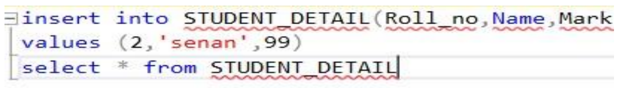
- A Primary Key is the minimal set of attributes of a table that has the task to uniquely identify the rows, or we can say the tuples of the given particular table.

Use of Primary Key

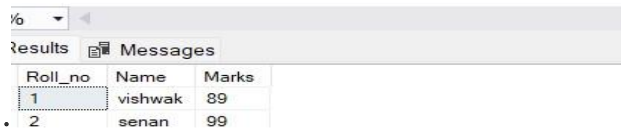
- As defined above, a primary key is used to uniquely identify the rows of a table. Thus, a row that needs to be uniquely identified, the key constraint is set as the Primary key to that particular field.

- A primary key can never have a **NULL** value because the use of the primary key is to identify a value uniquely, but if no value will be there, how could it sustain. Thus, the field set with the primary key constraint cannot be NULL.

Also, it all depends on the user that the user can add or delete the key if applied



```
insert into STUDENT_DETAIL(Roll_no,Name,Mark
values (2,'senan',99)
select * from STUDENT_DETAIL
```



Roll_no	Name	Marks
1	vishwak	89
2	senan	99

Removing Primary Key

It is also possible to delete the set primary key from an attribute using **ALTER** and **DROP** commands.

```
ALTER TABLE STUDENT_DETAIL DROP PRIMARY KEY ;
```

Adding Primary Key after creating the table

In order to set the primary key after creating a table, use the **ALTER** command and add the primary key constraint to do so.

The syntax is shown below:

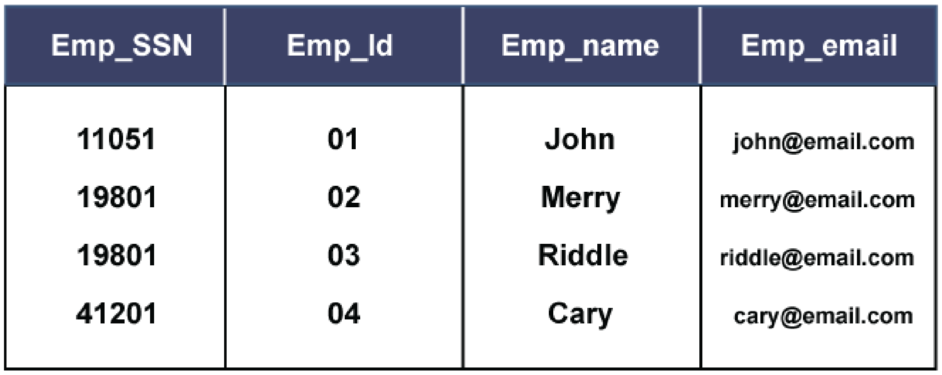
```
ALTER TABLE STUDENT_DETAIL
 ADD CONSTRAINT PK_STUDENT_DETAIL PRIMARY KEY (Roll_no, Name);
```

We have taken the Name attribute just for understanding the syntax.

So, in this way, we can use and set the primary key on a table. However, the syntax for defining the primary key may vary for different types of databases.

Super Key

- We can define a super key as a set of those keys that identify a row or a tuple uniquely. The word super denotes the superiority of a key. Thus, a super key is the superset of a key known as a **Candidate key** (discussed in the next section). It means a candidate key is obtained from a super key only.
- **Emp_SSN:** The SSN number is stored in this field.
- **Emp_Id:** An attribute that stores the value of the employee identification number.
- **Emp_name:** An attribute that stores the name of the employee holding the specified employee id.
- **Emp_email**: An attribute that stores the email id of the specified employees.
- The **EMPLOYEE_DETAIL** table is given below that will help you understand better:



Emp_SSN	Emp_Id	Emp_name	Emp_email
11051	01	John	john@email.com
19801	02	Merry	merry@email.com
19801	03	Riddle	riddle@email.com
41201	04	Cary	cary@email.com

- These all are the set of super keys which, together or combining with other prime attributes, can identify a table uniquely.
- Just like, if we set Super key on Emp_SSN, it will be able to identify all other tuples of the table very easily. Similarly, if we set the Super key on (Emp_Id, Emp_name}, we can easily get the value or details of the other remaining attributes of the employee. So, in this way, we can create and search out the super keys from a table.

Candidate Key

- A candidate key is a subset of a super key set where the key which contains no redundant attribute is none other than a **Candidate Key**. In order to select the candidate keys from the set of super key, we need to look at the super key set.

How a Candidate key is different from a Primary Key

- Although the purpose of both candidate and the primary key is the same, that is to uniquely identify the tuples, and then also they are different from each other. It is because, in a table, we can have one or more than one candidate key, but we can create only one primary key for a table. Thus, from the number of obtained candidate keys, we can identify the appropriate primary key. However, if there is only one candidate key in a table, then it can be considered for both key constraints.

Example of Candidate Key

- Let's look at the same example took while discussing Super Key to understand the working of a candidate key.

- We have an **EMPLOYEE_DETAIL** table where we have the following attributes:

Emp_SSN: The SSN number is stored in this field.

Emp_Id: An attribute that stores the value of the employee identification number.

Emp_name: An attribute that stores the name of the employee holding the specified employee id.

Emp_email: An attribute that stores the email id of the specified employees.

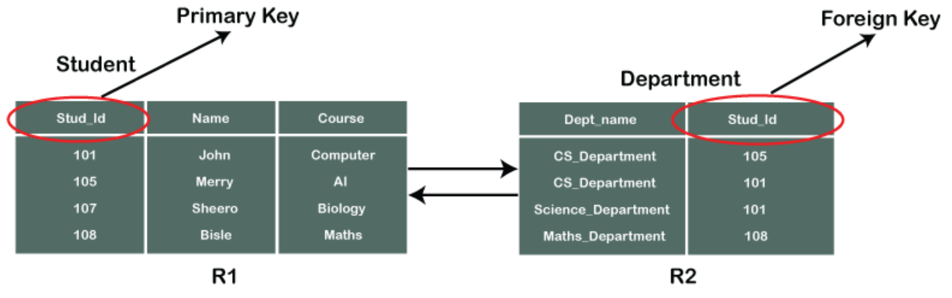
The **EMPLOYEE_DETAIL** table is given below that will help you understand better

Foreign Key

- A foreign key is different from a super key, candidate key or primary key because a foreign key is the one that is used to link two tables together or create connectivity between the two.

- A foreign key is the one that is used to link two tables together via the primary key. It means the columns of one table points to the primary key attribute of the other table. It further means that if any attribute is set as a primary key attribute will work in another table as a foreign key attribute. But one should know that a foreign key has nothing to do with the primary key.

In the below-shown figure, you can view the following structure of the relationship between the two tables.



Stud_Id	Name	Course
101	John	Computer
105	Merry	AI
107	Sheero	Biology
108	Bisle	Maths

Dept_name	Stud_Id
CS_Department	105
CS_Department	101
Science_Department	101
Maths_Department	108

SQL Arithmetic Operators

Operator	Description	Example
+	Add	SELECT 30 + 20;
-	Subtract	SELECT 30 - 20;
*	Multiply	SELECT 30 * 20;
/	Divide	SELECT 30 / 10;
%	Modulo	SELECT 17 % 5;

SQL Bitwise Operators

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

SQL Comparison Operators

Operator	Description	Example
=	Equal to	SELECT * FROM Products WHERE Price = 18;

>	Greater than	SELECT * FROM Products WHERE Price > 30;
<	Less than	SELECT * FROM Products WHERE Price < 30;
>=	Greater than or equal to	SELECT * FROM Products WHERE Price >= 30;
<=	Less than or equal to	SELECT * FROM Products WHERE Price <= 30;
<>	Not equal to	`SELECT * FROM Products WHERE Price <> 18;`

SQL Compound Operators

Operator	**Description**
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals

%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise OR equals

SQL Logical Operators

Operator	**Description**
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition