SQL Server

Hexavarsity

Objective

- Data Storage
- Relational Data Model
- Introduction to SQL
- Database Keys

SQL Introduction

Data Storage

- Information storage and retrieval (data processing) is a major part of the software application development in the IT industry.
- It is mandatory for every software professional to be aware of the approach of data storage and retrieval systems.

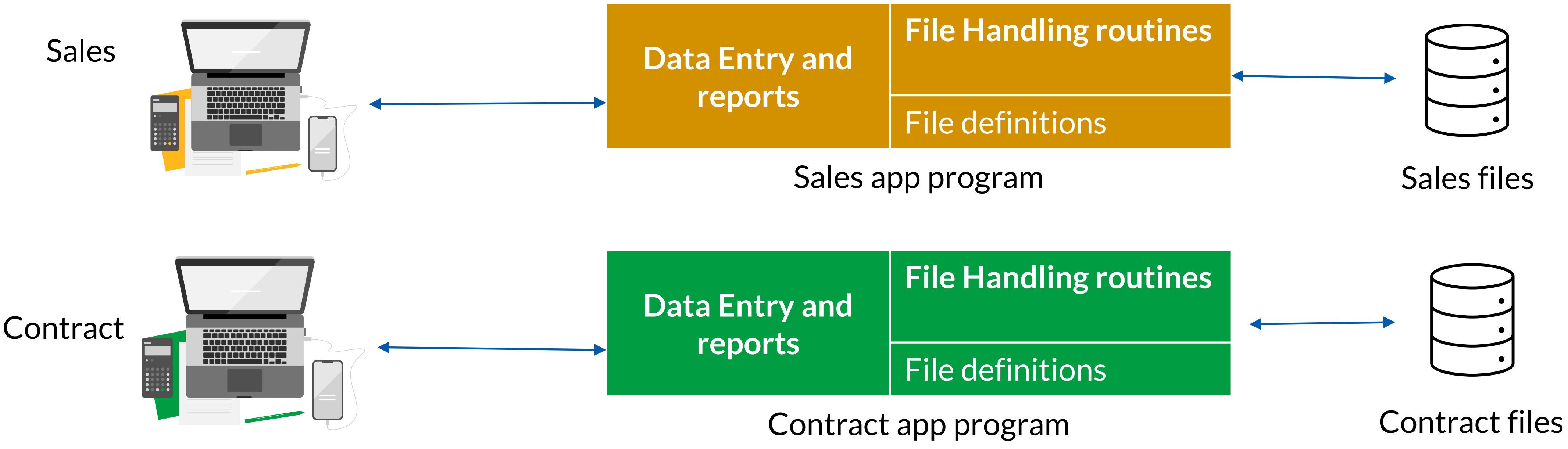
Data and information

- **Data**: Known facts, figures, objects and events which can be stored.
  - Structured: numbers, text, dates
  - Unstructured Data: images, video, documents
  - *Examples:*
    - ✓ RDBMS 02/01/2016 “It is raining”
- **Information**: Data that is processed to be useful
  - *Examples*:
    - ✓ Course Code is 1
    - ✓ The course name is RDBMS
    - ✓ The begin date of course is 02/01/2016
    - ✓ The temperature dropped 20 degrees and then it started raining.

Traditional approach

- **The traditional approach to store and access the data is file-based system.**
- File-based System
  - Data are stored as collection of records in flat-files (data files) on the disk
  - Collection of application programs that perform services for the end users (e.g. reports) access these data files
  - Each application defines and manages its own data

How traditional approach works?



Sales

Data Entry and reports

File Handling routines

File definitions

Sales app program

Sales files

Contract

Data Entry and reports

File Handling routines

File definitions

Contract app program

Contract files

- **Sales Files**
  - PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo)
  - PrivateOwner (ownerNo, Name, address, telNo) ,
  - Client (clientNoName, address, telNo, prefType, maxRent)
- **Contracts Files**
  - Lease (leaseNo, propertyNo, clientNo, rent, paymentMethod, deposit, paid, rentStart, rentFinish.
  - PropertyForRent (propertyNo, street, city, postcode, rent)
  - Client (clientNo, fName, lName, address, telNo)

Limitations of traditional approach

- Separation and isolation of data
  - Each program maintains its own set of data.
  - Users of one program may be unaware of potentially useful data held by other programs.
- Duplication of data
  - Same data is held by different applications.
  - Wasted space and potentially different values and/or different formats for the same item.
- No Concurrent access to data
- Poor security
- Lack of data sharing
- No simultaneous application access to data
- No data independence
  - File structure is defined in the program code

Database approach

- Data is stored in the database as a collection of data files.
- Database:
  - A collection of related data.
- Database Management System (DBMS):
  - A software package/ system to facilitate the creation and maintenance of a computerized database.
- Database System:
  - The DBMS software together with the database

Advantages of database approach

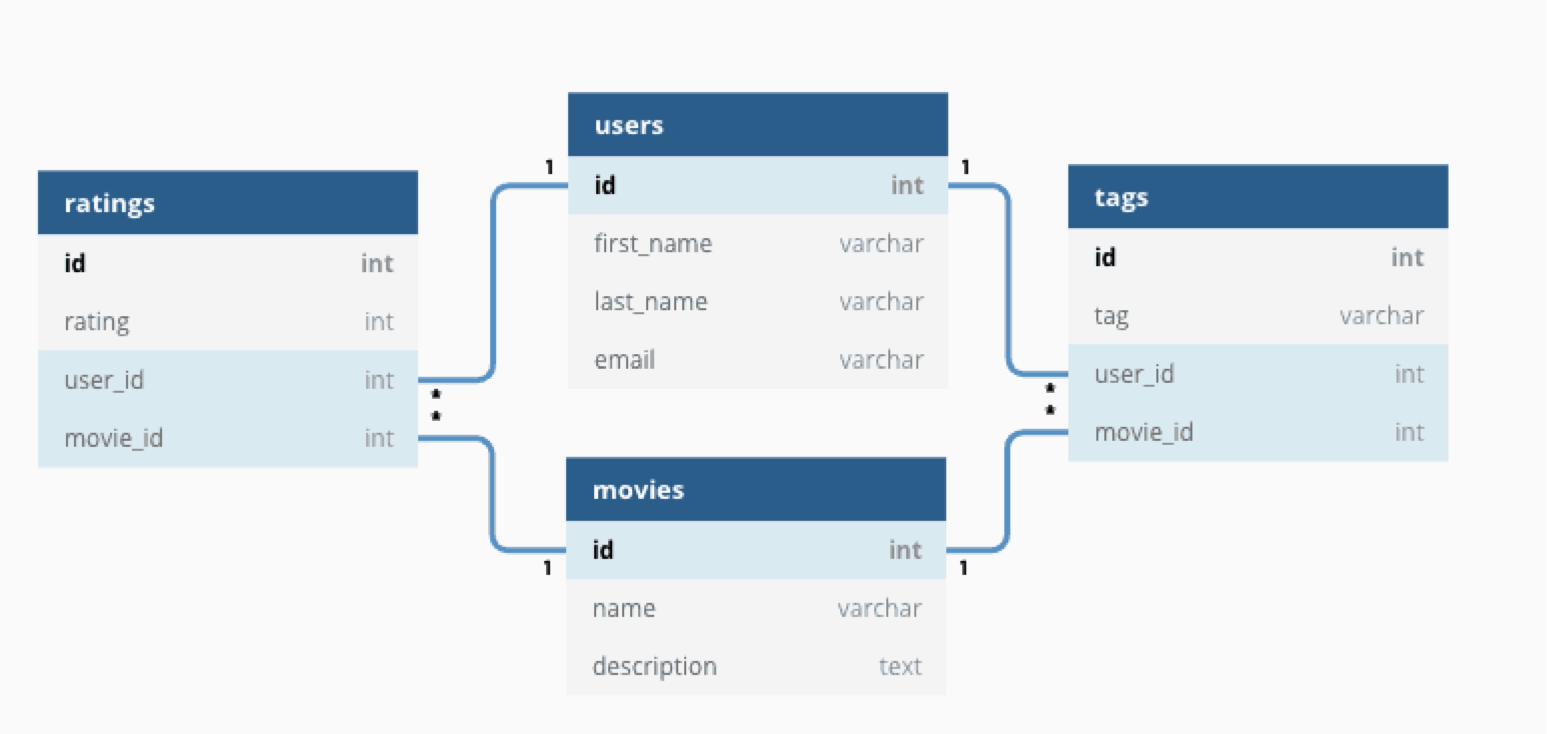
- Control of data redundancy
- Data consistency
- Program-Data independence
- More Secure
- Concurrent access to data through application programs
- Flexible for application development

Data Model

- What is data model?
  - Integrated collection of concepts (Tool) for describing data, relationships between data, and constraints on the data in a database
- Why data model?
  - To represent data in an understandable way.
- Types of data models include:
  - Object-Based Data Models
    - Entity-Relationship, Semantic, Functional, Object-Oriented.
  - Record-Based Data Models
    - Relational Data Model, Network Data Model, Hierarchical Data Model, Physical Data Models

Relational data model

- Proposed in 1970 by E.F. Codd (IBM), first commercial system in 1981-82.
- Now in several commercial products (e.g. DB2, ORACLE, MS SQL Server, SYBASE, INFORMIX).
- Several free open source implementations, e.g. MySQL, PostgreSQL



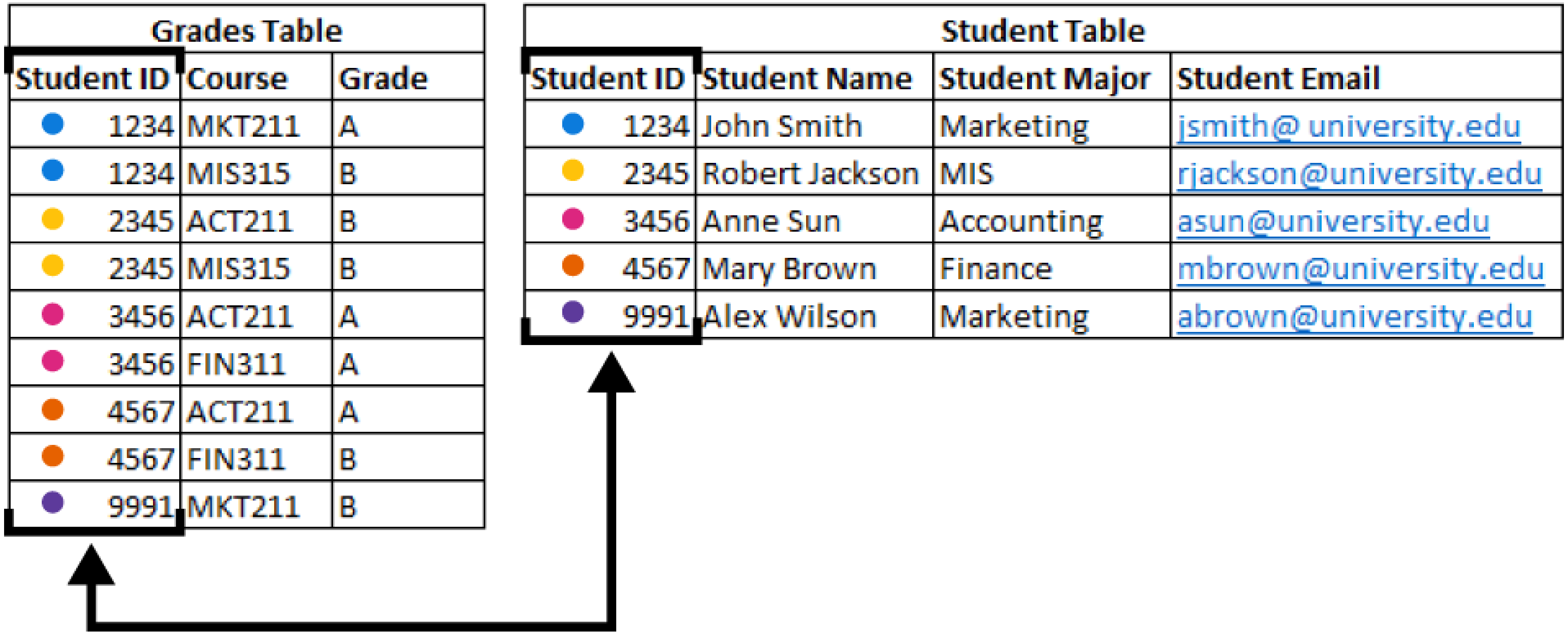
ratings	
id	int
rating	int
user_id	int
movie_id	int

users	
id	int
first_name	varchar
last_name	varchar
email	varchar

tags	
id	int
tag	varchar
user_id	int
movie_id	int

movies	
id	int
name	varchar
description	text

Relational data model

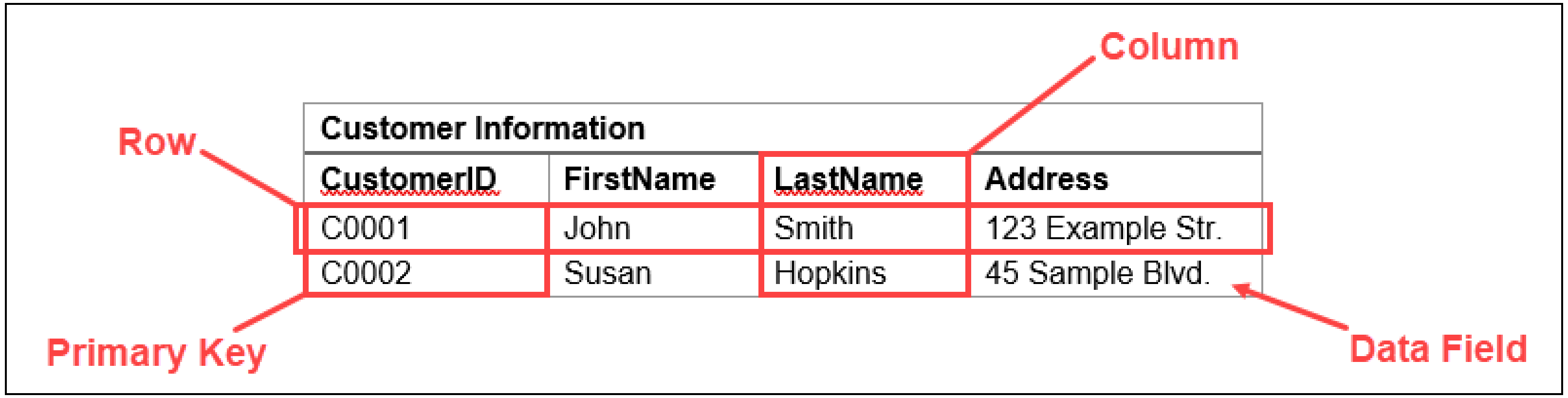


Grades Table		
Student ID	Course	Grade
● 1234	MKT211	A
● 1234	MIS315	B
● 2345	ACT211	B
● 2345	MIS315	B
● 3456	ACT211	A
● 3456	FIN311	A
● 4567	ACT211	A
● 4567	FIN311	B
● 9991	MKT211	B

Student Table			
Student ID	Student Name	Student Major	Student Email
● 1234	John Smith	Marketing	jsmith@ university.edu
● 2345	Robert Jackson	MIS	rjackson@university.edu
● 3456	Anne Sun	Accounting	asun@university.edu
● 4567	Mary Brown	Finance	mbrown@university.edu
● 9991	Alex Wilson	Marketing	abrown@university.edu

Relational Model Terminology

- A table with columns and rows.
  - Only applies to logical structure of the database, not the physical structure.
- Attribute is a named column of a relation.
- Domain is the set of allowable values for one or more attributes
- Tuple is a row of a relation.
- Degree is the number of attributes in a relation.
- Cardinality is the number of tuples in a relation.
- Relational Database is a collection of normalized relations with distinct relation names

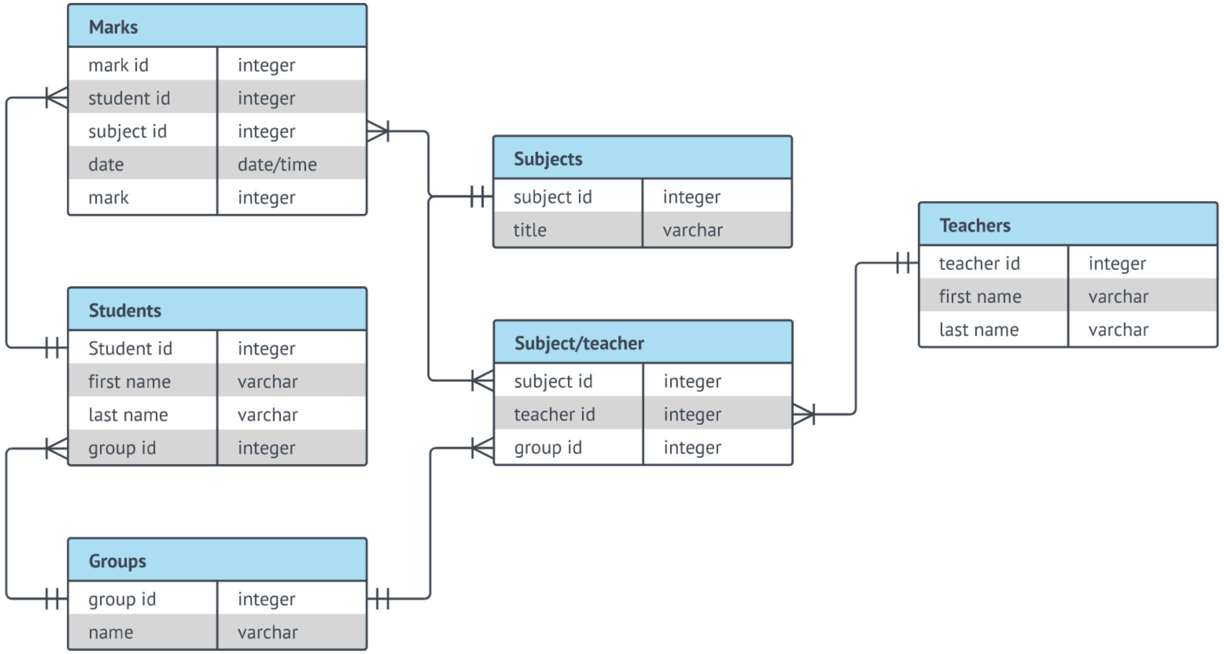


Customer Information

CustomerID	FirstName	LastName	Address
C0001	John	Smith	123 Example Str.
C0002	Susan	Hopkins	45 Sample Blvd.

Entity-Relationship (ER) Model

- ER model helps to capture conceptual database design
- Adopts top-down approach
- Describes the functional data requirements of a real-world problem in the form of ER diagrams
- Consists of Attributes, Entities, Relationships, Identifiers
- UML class diagrams is representative of another way of displaying ER concepts



Marks	
mark id	integer
student id	integer
subject id	integer
date	date/time
mark	integer

Subjects	
subject id	integer
title	varchar

Teachers	
teacher id	integer
first name	varchar
last name	varchar

Students	
Student id	integer
first name	varchar
last name	varchar
group id	integer

Subject/teacher	
subject id	integer
teacher id	integer
group id	integer

Groups	
group id	integer
name	varchar

Entity and Attribute

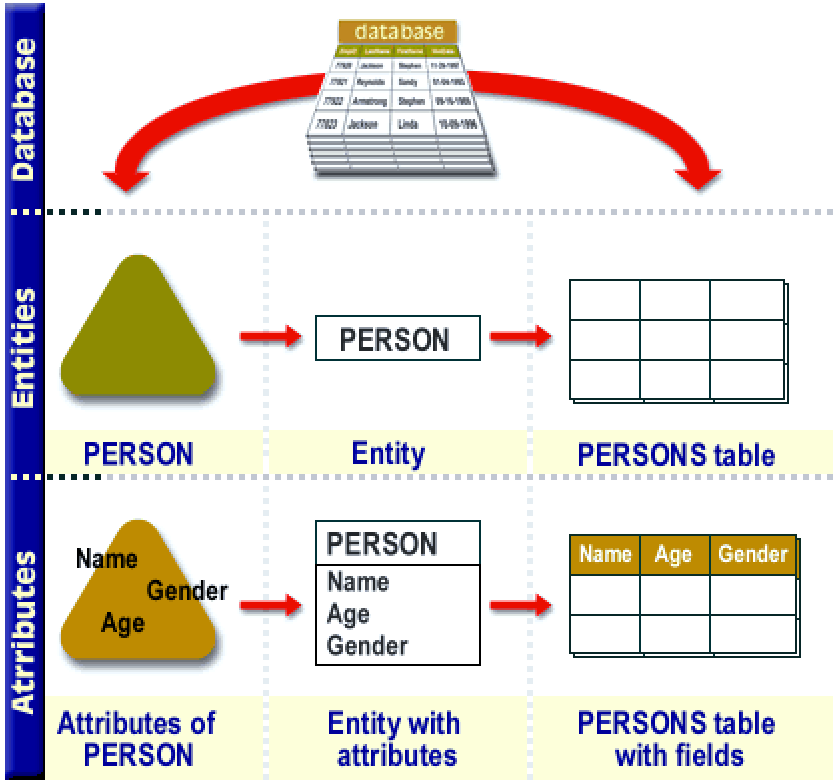
- Entities
  - Entities are specific objects or things that are represented in the database.
    - *Example*: The Person , the Book
- Attributes
  - Attributes are properties used to describe an entity.
    - *Example*:

      Person entity may have the attributes Name, Age, Gender

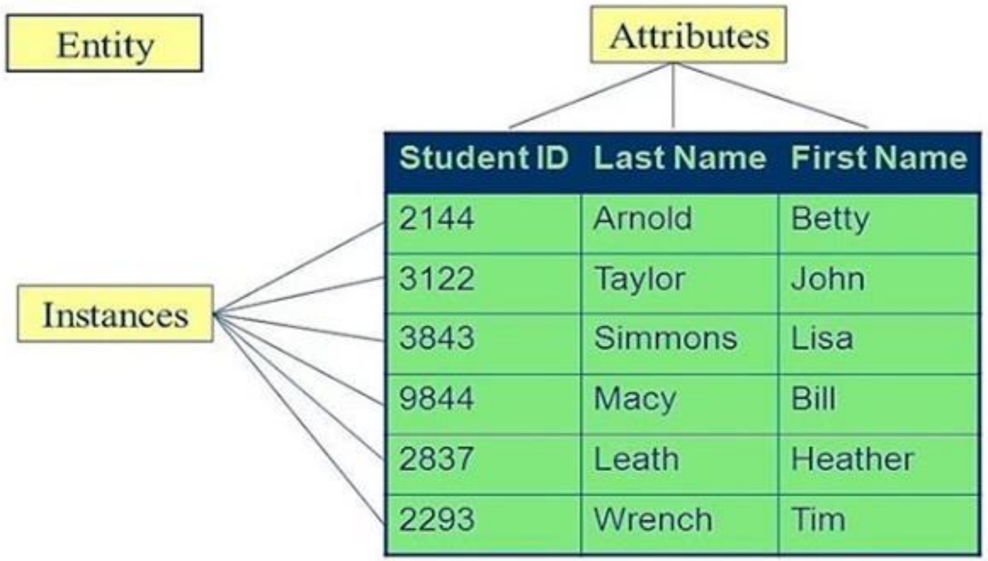
      Address, Degree, BirthDate
  - Each attribute has a value set associated with it.
    - *Example*:

      Attribute Age associated with value ranges from 18 to 52,

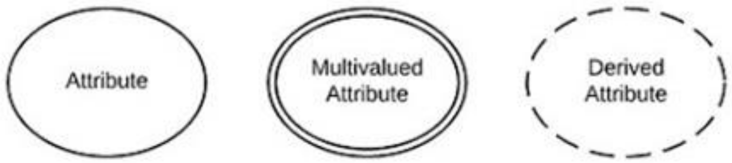
      attribute Gender should have values male, female



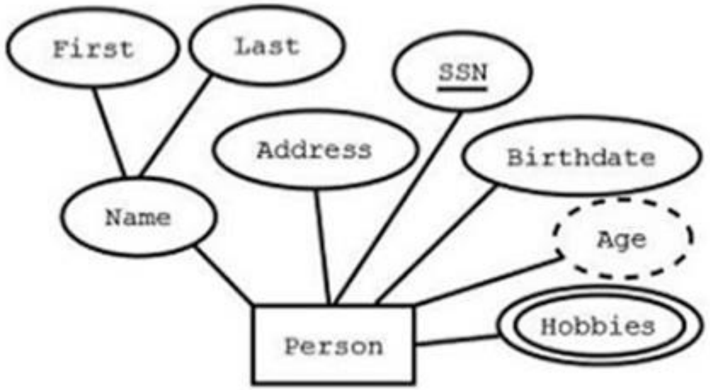
Entity and Attribute

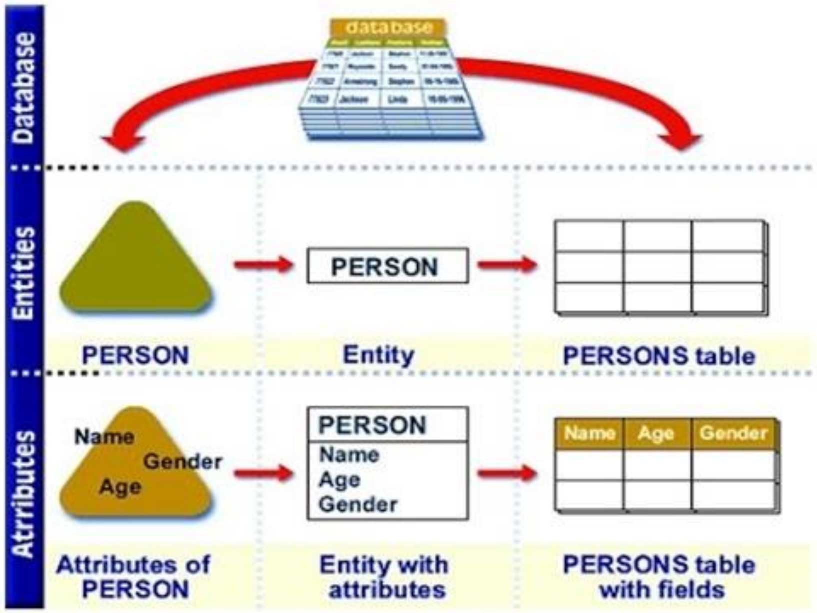


Student ID	Last Name	First Name
2144	Arnold	Betty
3122	Taylor	John
3843	Simmons	Lisa
9844	Macy	Bill
2837	Leath	Heather
2293	Wrench	Tim



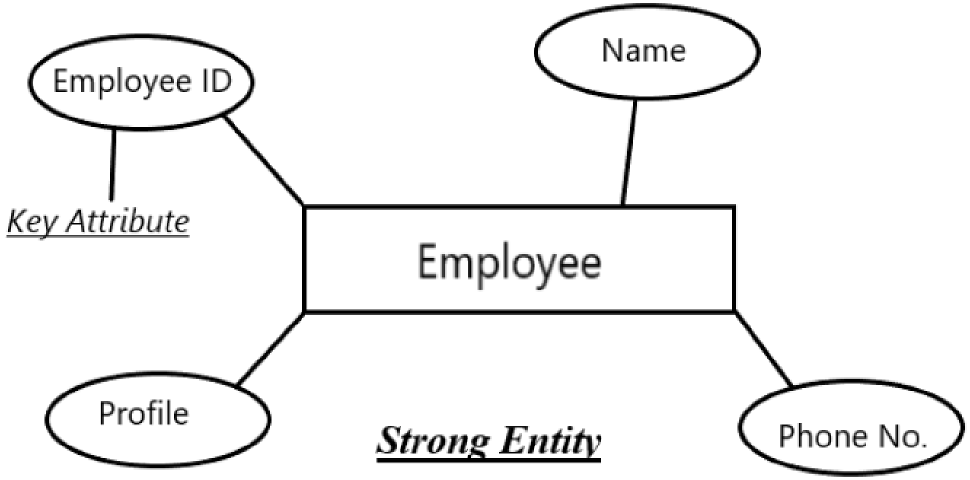
- Representing attributes

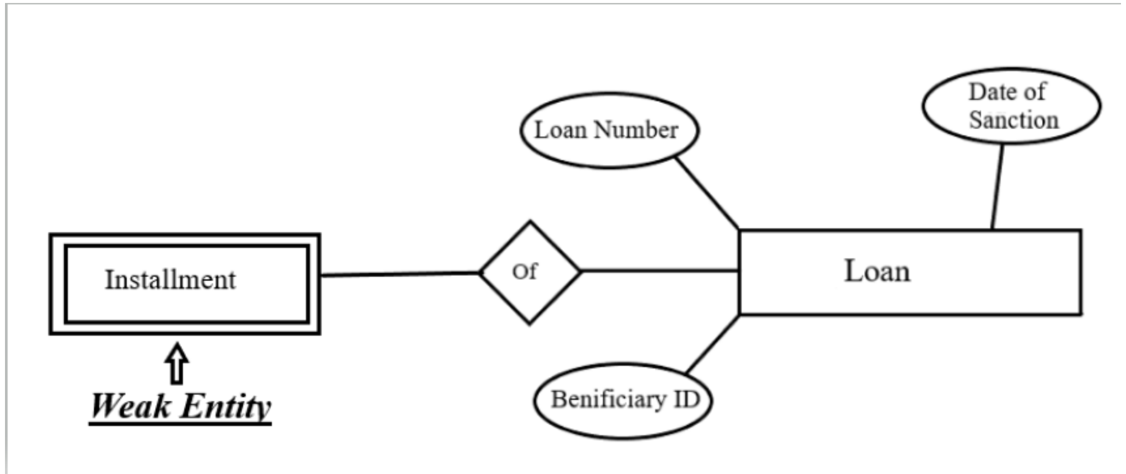




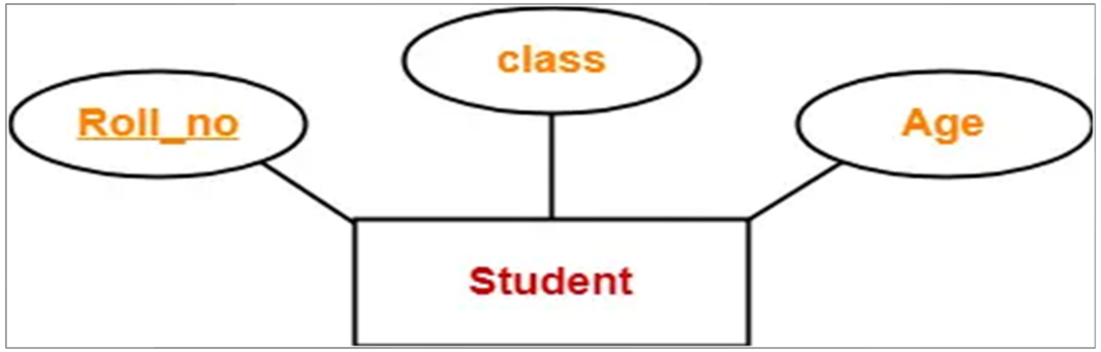
Types of Entities

- Strong/Regular Entity
  - If an Entity has a key attribute (uniquely identifiable feature), then it is called a Strong Entity.
  - Example of a Strong Entity is the Employee
- Weak Entity
  - It is dependent on a strong entity (identifying owner)...cannot exist on its own, It does not have a unique identifier
  - Example: Installment is an Entity, then it can exist only if a Loan exists as an Entity.





Types of attributes

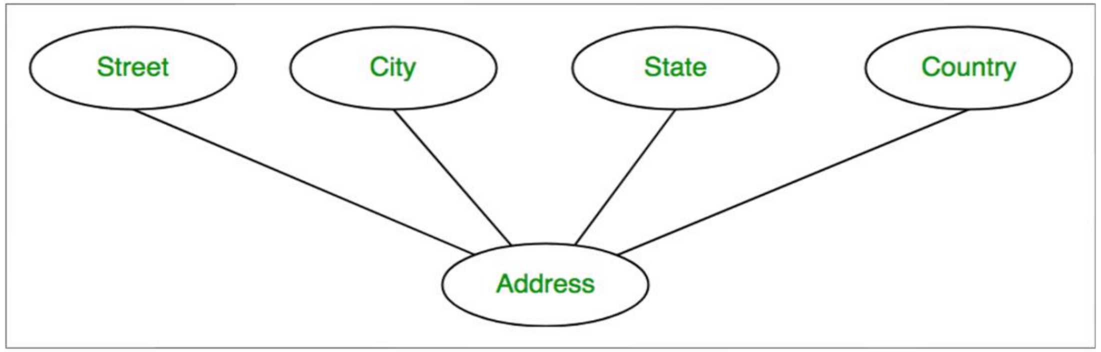


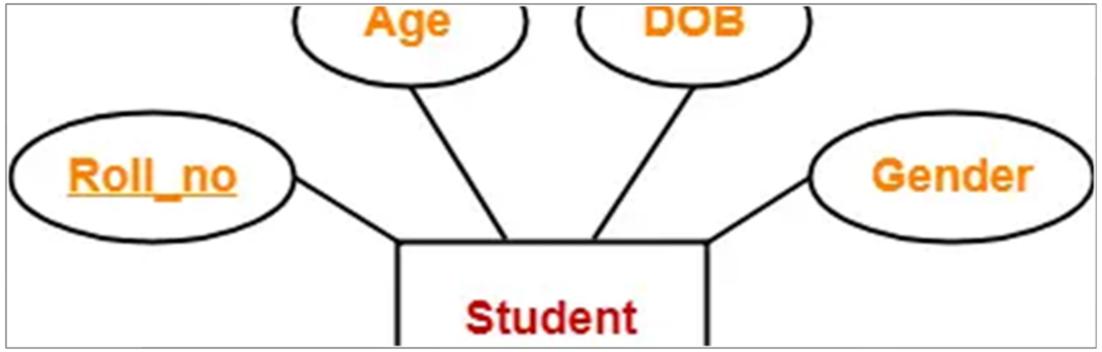
Simple Attributes

Simple attributes are those that cannot be further divided into sub-attributes.

Composite Attributes

Composite attributes are made up of two or more simple attributes.

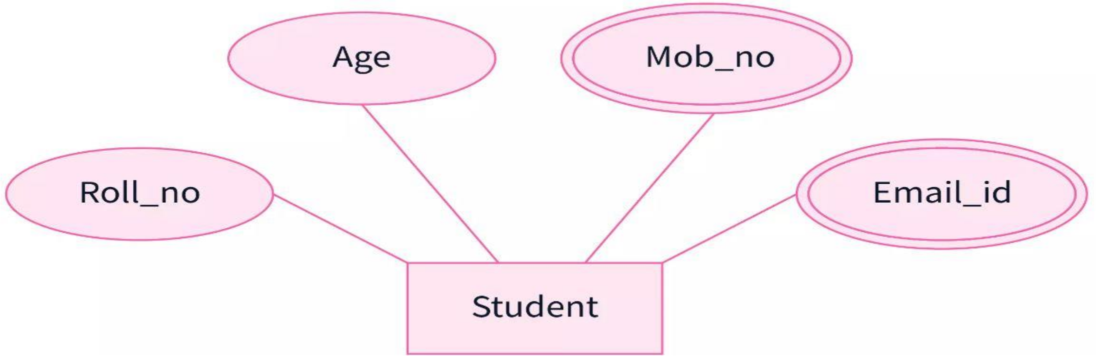




Single Valued Attributes

Single-valued attributes can only have one value.

Types of attributes

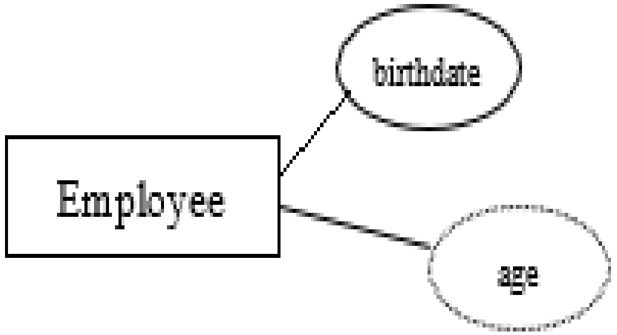


Multivalued Attributes

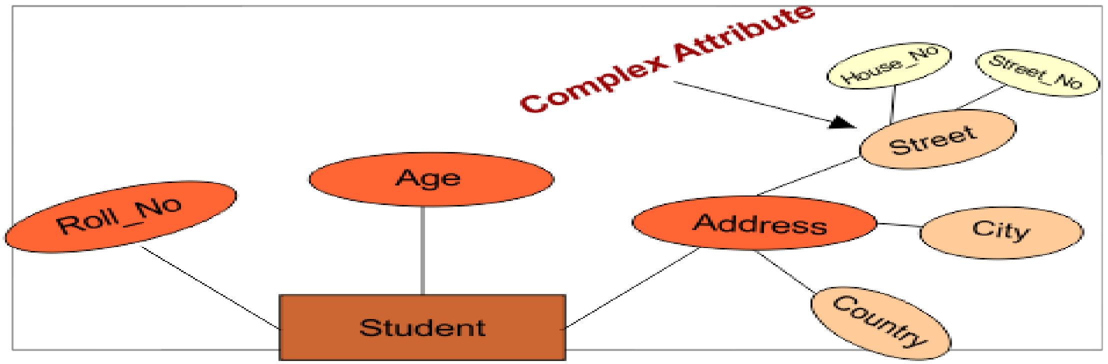
Multivalued attributes can have more than one value.

Derived Attributes

Derived attributes are based on other attributes and are not stored directly in the database.



The age of the employee is a derived attribute.

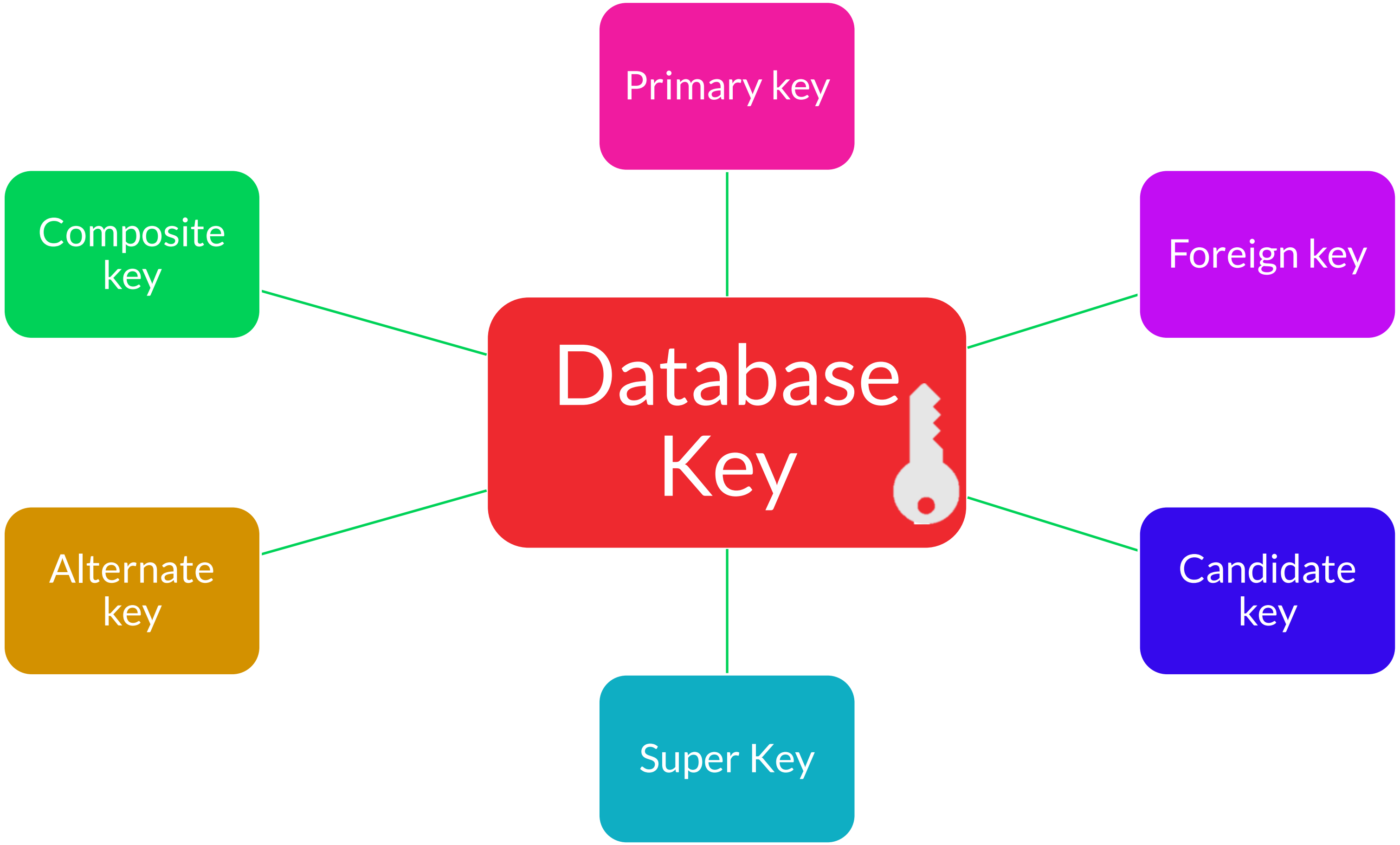


Complex Attributes

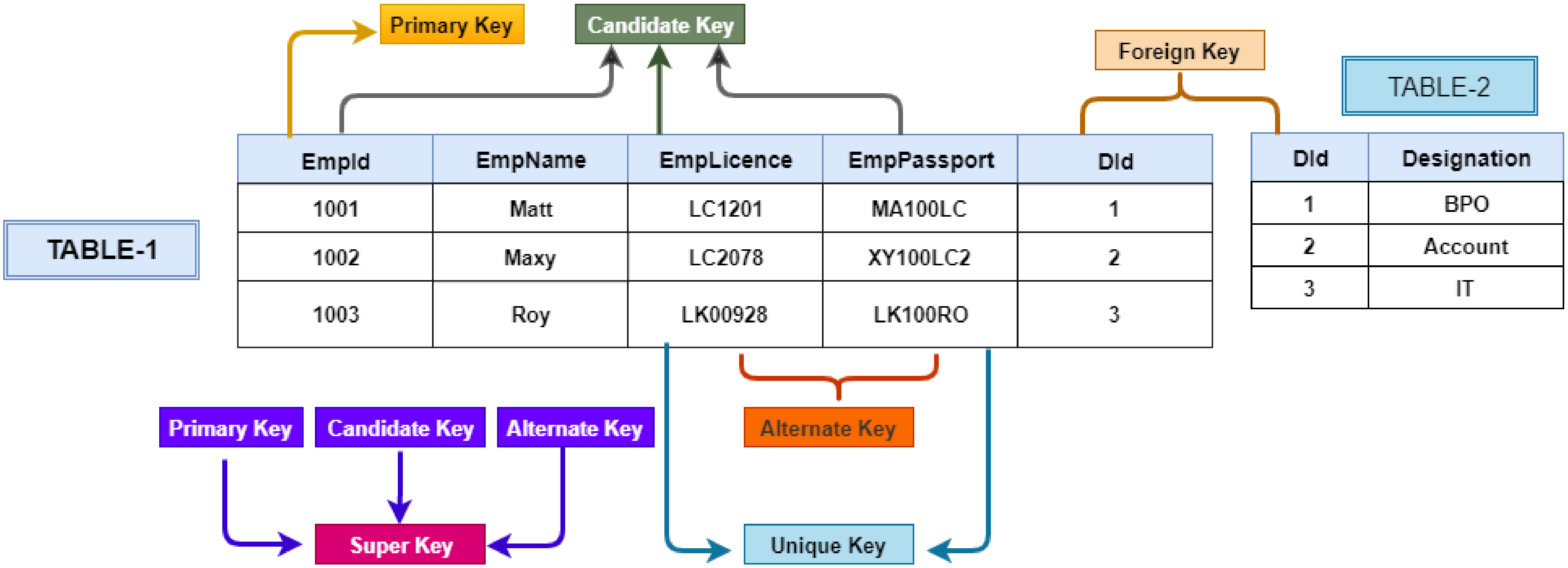
The complex attribute in DBMS involves both multivalued and composite attributes.

Key

- It is used to uniquely identify any record or row from the table.
- It is also used to establish and identify relationships between tables.



Key example

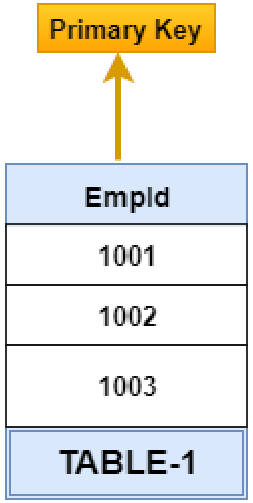


EmpId	EmpName	EmpLicence	EmpPassport	DId
1001	Matt	LC1201	MA100LC	1
1002	Maxy	LC2078	XY100LC2	2
1003	Roy	LK00928	LK100RO	3

DId	Designation
1	BPO
2	Account
3	IT

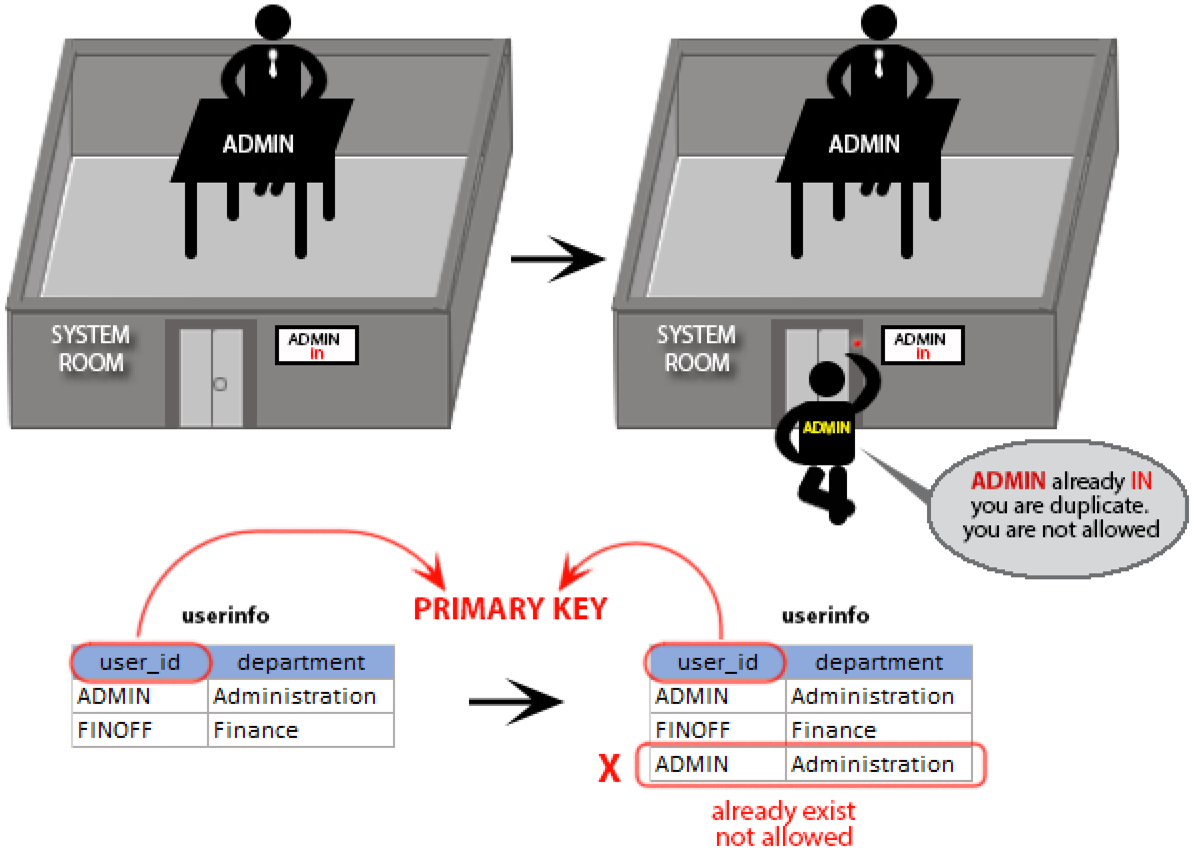
Primary key

- A primary key is a column -- or a group of columns -- in a table that uniquely identifies the rows of data in that table.
- A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).
- Primary keys must contain UNIQUE values
- It cannot contain NULL values.
- In Table-1, Empid is a Primary Key.



EmpId
1001
1002
1003

TABLE-1


ADMIN
SYSTEM ROOM
ADMIN in
ADMIN
SYSTEM ROOM
ADMIN in
ADMIN
ADMIN already IN you are duplicate. you are not allowed
PRIMARY KEY
X
already exist not allowed

userinfo

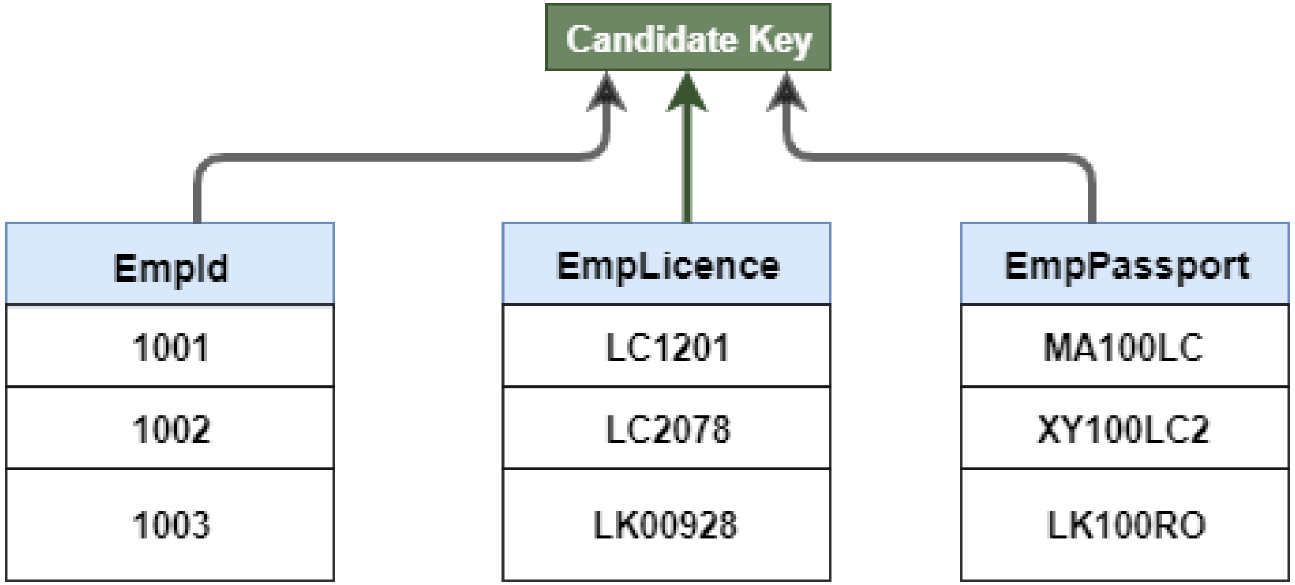
user_id	department
ADMIN	Administration
FINOFF	Finance

userinfo

user_id	department
ADMIN	Administration
FINOFF	Finance
ADMIN	Administration

Candidate Key

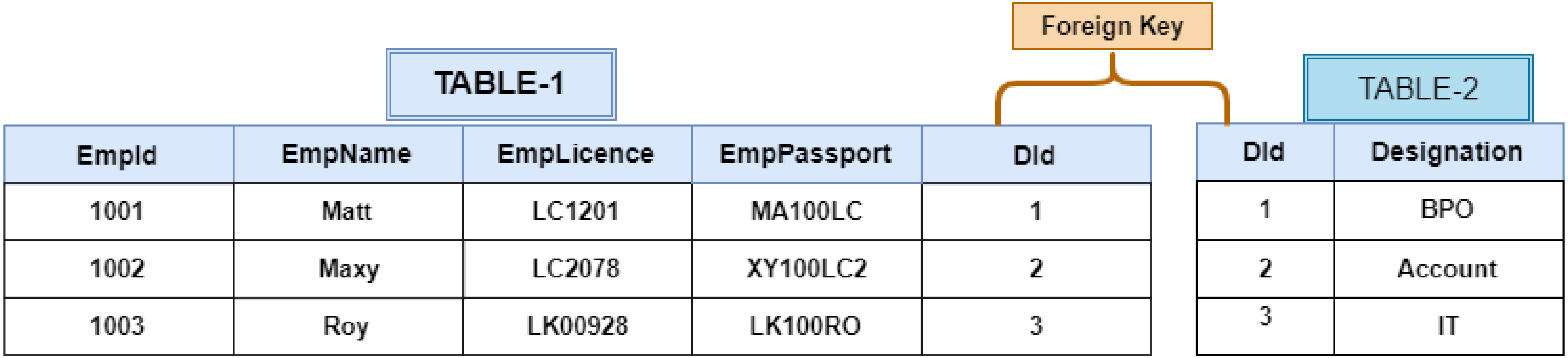
- Candidate Key can be defined as a set of one or more columns that can identify a record uniquely in a table and which can be selected as a primary key of the table.
- It contains UNIQUE values in column, and does not allows NULL values.
- In Table-1, Empid, EmpLicence and EmpPassport are candidate keys.



EmpId	EmpLicence	EmpPassport
1001	LC1201	MA100LC
1002	LC2078	XY100LC2
1003	LK00928	LK100RO

Foreign key

- Foreign creates a relationship between two or more tables, a primary key of one table is referred as a foreign key in another table.
- It can also accept multiple null values and duplicate values.



Foreign Key

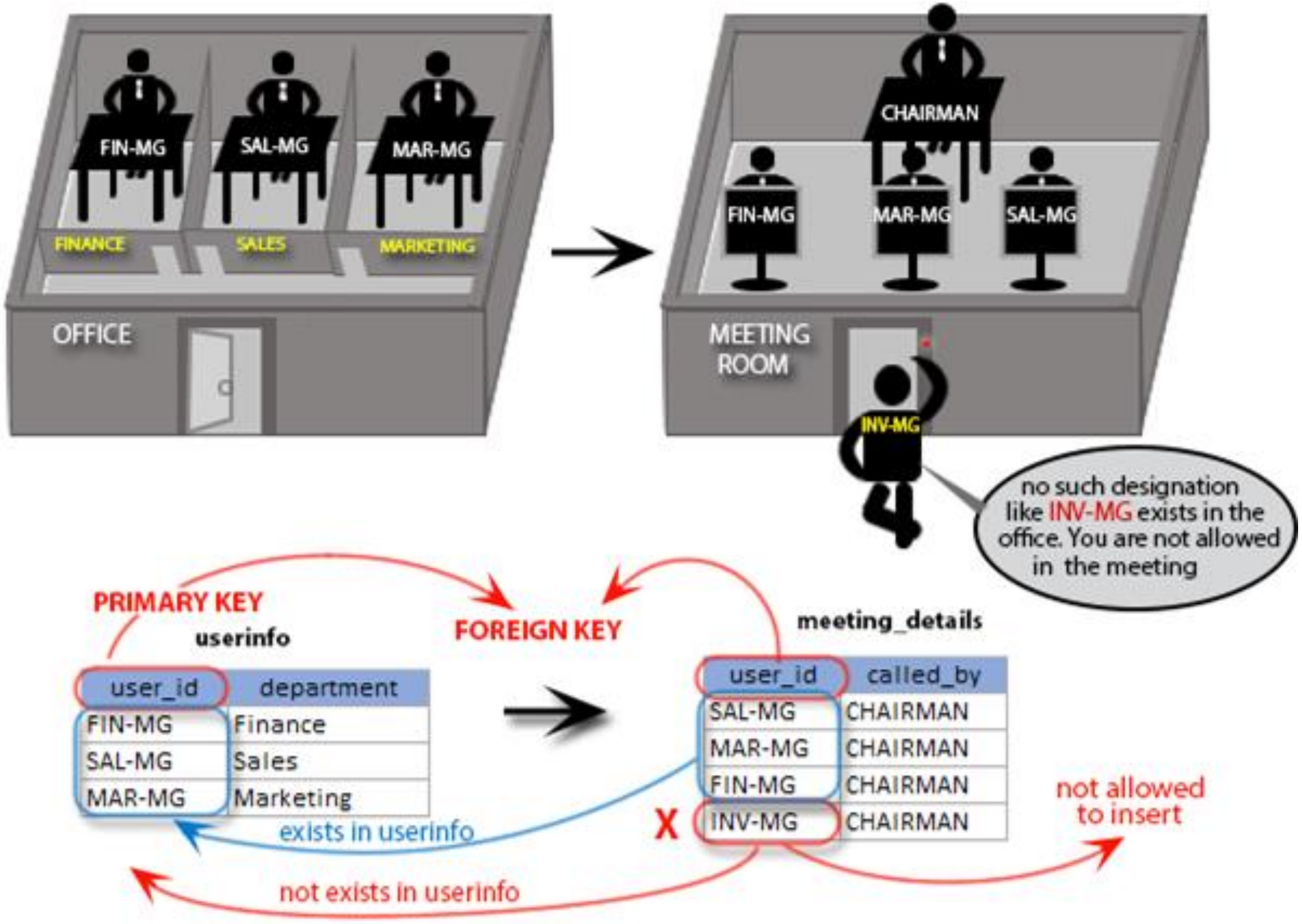
TABLE-1

EmpId	EmpName	EmpLicence	EmpPassport	DId
1001	Matt	LC1201	MA100LC	1
1002	Maxy	LC2078	XY100LC2	2
1003	Roy	LK00928	LK100RO	3

TABLE-2

DId	Designation
1	BPO
2	Account
3	IT

Foreign key



userinfo

user_id	department
FIN-MG	Finance
SAL-MG	Sales
MAR-MG	Marketing

meeting_details

user_id	called_by
SAL-MG	CHAIRMAN
MAR-MG	CHAIRMAN
FIN-MG	CHAIRMAN
INV-MG	CHAIRMAN

Alternate Key

- Alternate key can be defined as a key that can be work as a primary key if required but right now it is not Primary key.
- *Example:*

  In Table-1, Empid is primary key but we can use EmpLicence & EmpPassport as a primary key to get unique record from table, That's why EmpLicence & EmpPassport are Alternate keys but right now it is not primary keys.

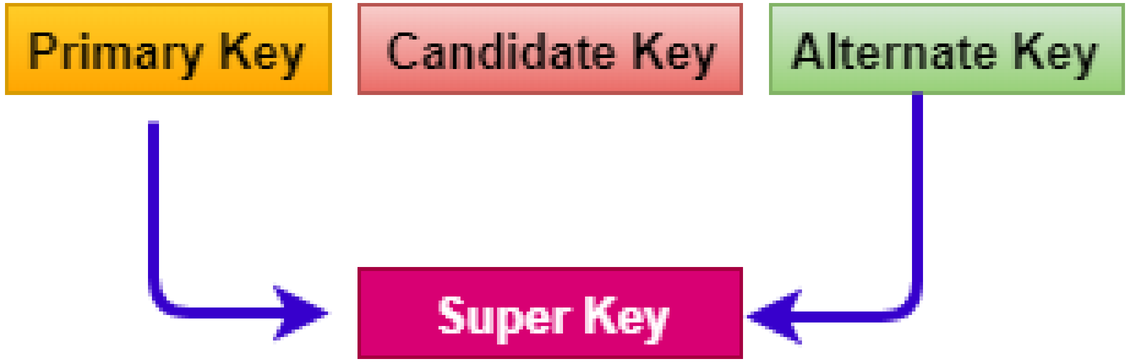
Composite Key

- Composite Key is a combination of more than one columns of a table. It can be a Candidate key and Primary key.
- *Example*:

  In Table-1, we can combine Empid & EmpLicence columns to fetch the data from table.

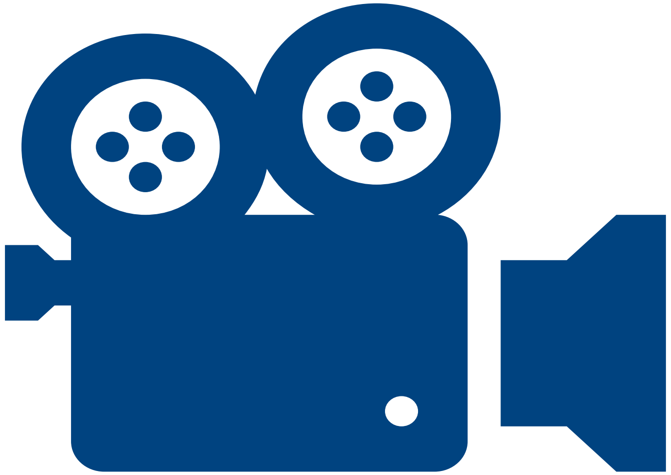
Super Key

- A super key is a group of single or multiple keys which identifies rows in a table.
- *Example*:
  - In Table-1, Primary key, Unique key, Alternate key are a subset of Super Keys.
  - {Empid, Empname}, {Empid, EmpPassport, Empname}, {EmpLicence, Empname}
  - Any set of column which contains EmpLicence or EmpPassport or Empid is a super key of the table.



Video on Keys

- Objective:
  - ✓ To make the Trainee understand the concept of Database keys.
- Video Path:
  - ✓ https://www.youtube.com/watch?v=JkwbhFUftSc



Relationships

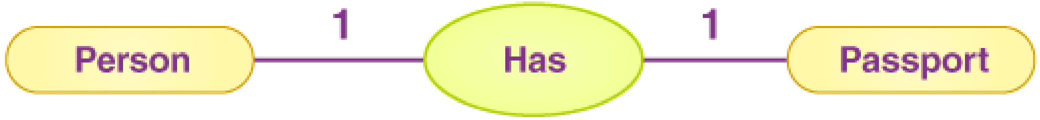
- A relationship relates two or more distinct entities with a specific meaning
- Relationships of the same type are grouped or typed into a relationship type
  - Example:

    the MANAGES relationship type in which EMPLOYEEs and DEPARTMENTs participate

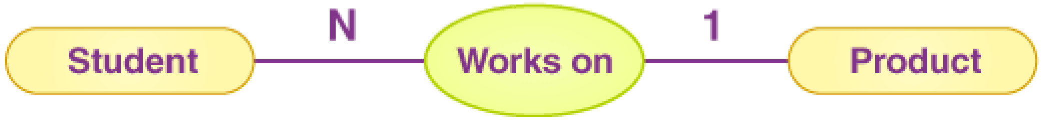
    the WORKSON relationship type in which EMPLOYEEs and PROJECTs participate
- Relationships can have attributes, which describe features pertaining to the association between the entities in the relationship
- Degree of a relationship is the number of entity types that participate in it.
  1. Unary relationship
  2. Binary relationship
  3. Ternary relationship

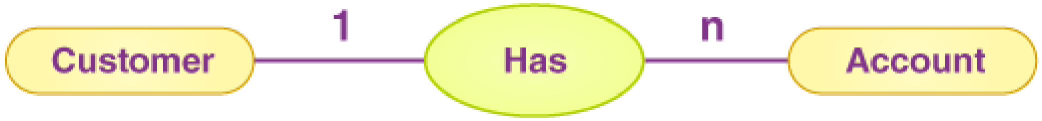
Cardinality of Relationships

- The number of entity instances that may participate in a relationship instance.
- **One-to-one (1:1)**
  - Each entity in the relationship will have exactly one related entity

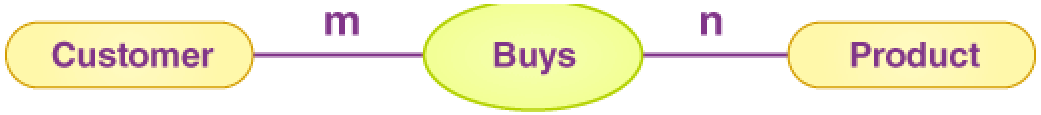


- **One-to-many (1:N) or Many-to-one (N:1)**
  - An entity on one side of the relationship can have many related entities, but an entity on the other side will have a maximum of one related entity

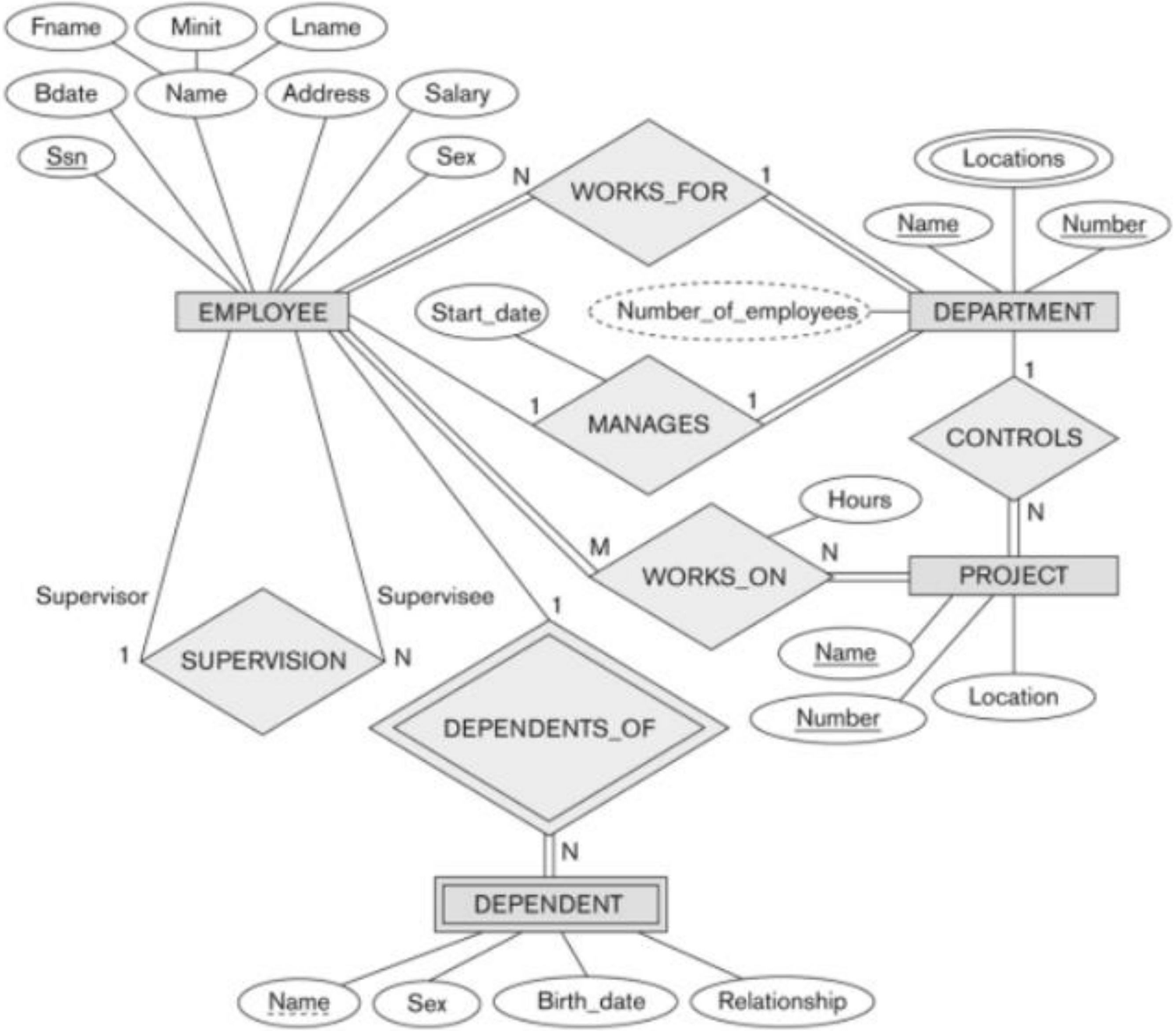




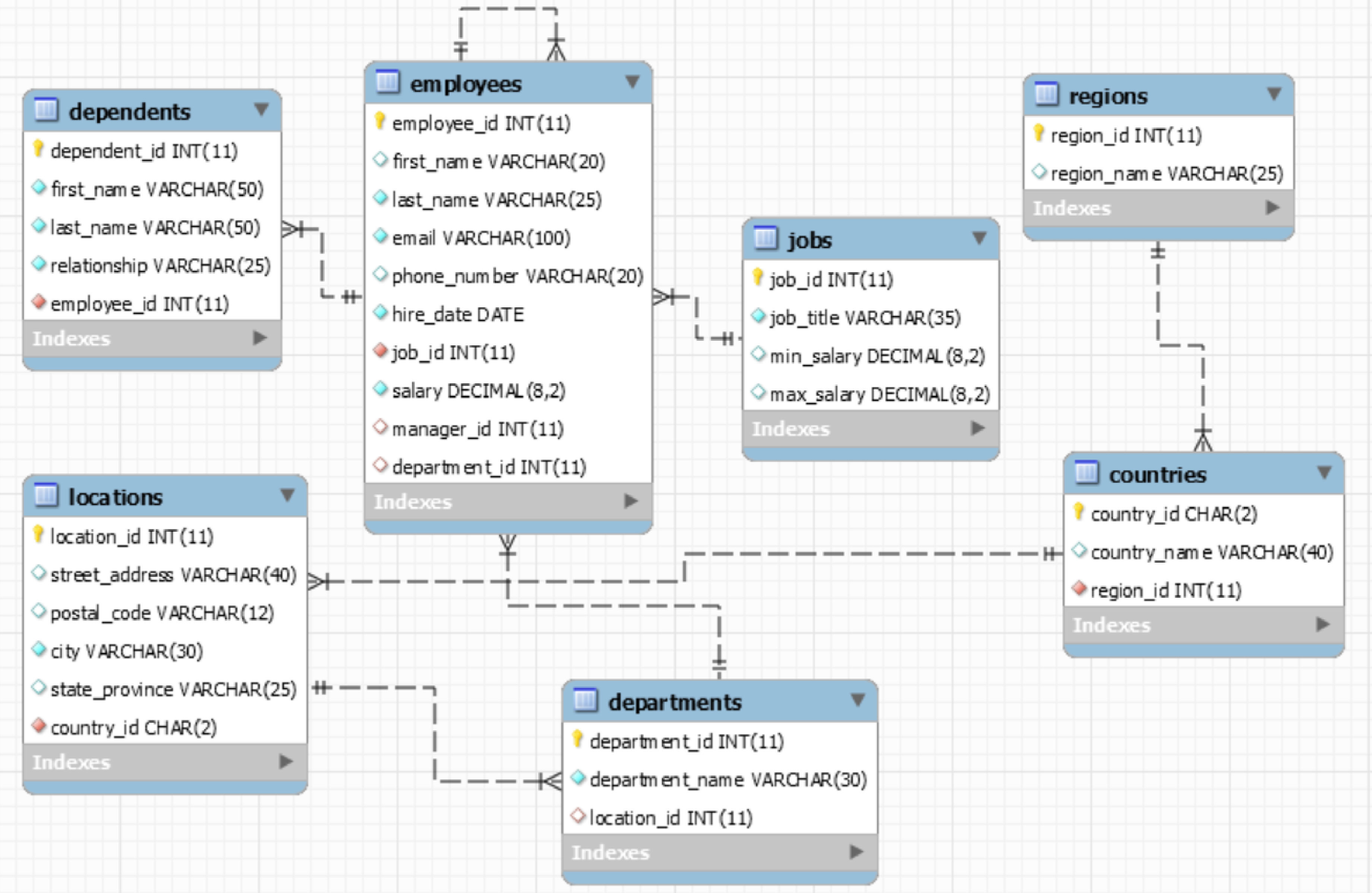
- **Many-to-many (M:N)**
  - Entities on both sides of the relationship can have many related entities on the other side



ER Diagram Example



ER Diagram



dependents

- dependent_id INT(11)
- first_name VARCHAR(50)
- last_name VARCHAR(50)
- relationship VARCHAR(25)
- employee_id INT(11)
- Indexes

employees

- employee_id INT(11)
- first_name VARCHAR(20)
- last_name VARCHAR(25)
- email VARCHAR(100)
- phone_number VARCHAR(20)
- hire_date DATE
- job_id INT(11)
- salary DECIMAL(8,2)
- manager_id INT(11)
- department_id INT(11)
- Indexes

regions

- region_id INT(11)
- region_name VARCHAR(25)
- Indexes

jobs

- job_id INT(11)
- job_title VARCHAR(35)
- min_salary DECIMAL(8,2)
- max_salary DECIMAL(8,2)
- Indexes

countries

- country_id CHAR(2)
- country_name VARCHAR(40)
- region_id INT(11)
- Indexes

locations

- location_id INT(11)
- street_address VARCHAR(40)
- postal_code VARCHAR(12)
- city VARCHAR(30)
- state_province VARCHAR(25)
- country_id CHAR(2)
- Indexes

departments

- department_id INT(11)
- department_name VARCHAR(30)
- location_id INT(11)
- Indexes

NORMALIZATION

What is Normalization?

- Overall objective on normalization is to reduce redundancy
- Redundancy – Data repeatedly stored
- Normalization recommends to divide the data across multiple tables to avoid redundancy
- When data is added, altered or deleted in one table, it helps to maintain the data consistency.
- Three important forms of normalization
    - First normal form (1NF)
    - Second normal form (2NF)
    - Third normal form (3NF)

First Normal Form (1NF)

- "The domain of each attribute contains only atomic values, and the value of each attribute contains only a single value from that domain"
- We Cannot have multiple values for a particular attribute (field) in a single record
- No two records should be exactly simillar

Against 1NF

Employee

Name	Skill
Raj	C++, C#, MS SQL
Arun	Java, Oracle

Compliance with 1NF

Employee

Name	Skill
Raj	C++
Raj	C#
Raj	MS SQL
Arun	Java
Arun	Oracle

Second Normal Form (2NF)

- "a table is in 2NF if and only if it is in 1NF and no non-prime attribute is dependent on any proper subset of any candidate key of the table"
- Prime – Key field used to identify the entire record (eg. Emp ID)
- Non – Prime – Field that depends on Prime Key (eg. DOB)
- Composite Key – When two fields combine to form primary key

Against 2NF

Employee

Name	Skill	Location
Raj	C++	Chennai
Raj	C#	Chennai
Arun	Java	Bangalore
Arun	Oracl e	Bangalore

Compliance with 2NF

Employee Skill

Name	Skill
Raj	C++
Raj	C#
Arun	Java
Arun	Oracle

Employee Location

Name	Location
Raj	Chennai
Arun	Bangalore

Third Normal Form (3NF)

- "the entity is in second normal form and all the attributes in a table are dependent on the primary key and only the primary key"

Against 3NF

Employee

Name	City	PIN
Raj	Chennai	600033
Arun	Bangalore	400028
Arjun	Chennai	600033

Compliance with 3NF

Employee Pin

Name	PIN
Raj	600033
Arun	400028
Arjun	600033

Compliance with 3NF

Pin

PIN	City
600033	Chennai
400028	Bangalore

Characteristics of good database system

- Good database is identified by ACID properties
- A – Atomicity
- C – Consistency
- I – Isolation
- D - Durability

Atomicity

- Atomicity refers to combining multiple transaction into single transaction
- A group of transaction can be considered as a atomic (single) transaction
- So the database system have to do all or do nothing

Example:

  - A is transferring Rs.1000 to B"s account

Steps:

  - Deduct 1000 from A"s account balance
  - Add 1000 to B"s account balance.
  - If first step is done and due to some factors 2nd is not done, will lead to serious error. So we should do all steps or do nothing

Consistency

- System will provide the user to define some rules regarding the data. (eg. Unique Key)
- Once rules defined, they are consistently maintained until the database is deleted

Isolation

- Database systems are accessed by multiple users simultaneously.
- Every request is isolated from each other.
- Eg: -
  - Single Credit card account having two credit cards.
  - Two cards are swiped simultaneously for amount equal to the credit limit.
  - Two requests reach the server simultaneously.
  - But the requests are processed one by one (each request is isolated from another)

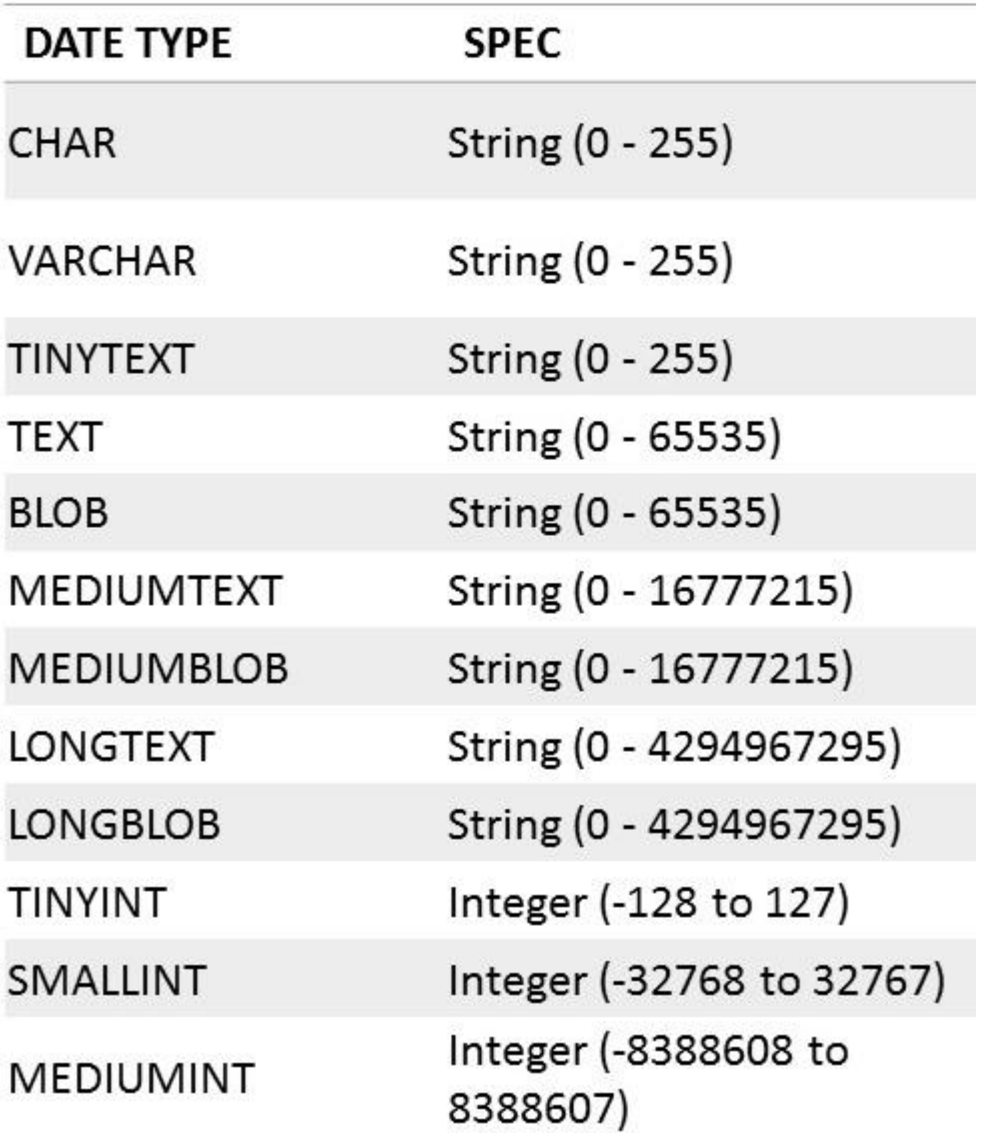
Durability

- Once data is stored and committed. And it was retrieved at a later time, durability confirms that we will get the same data which was stored earlier.

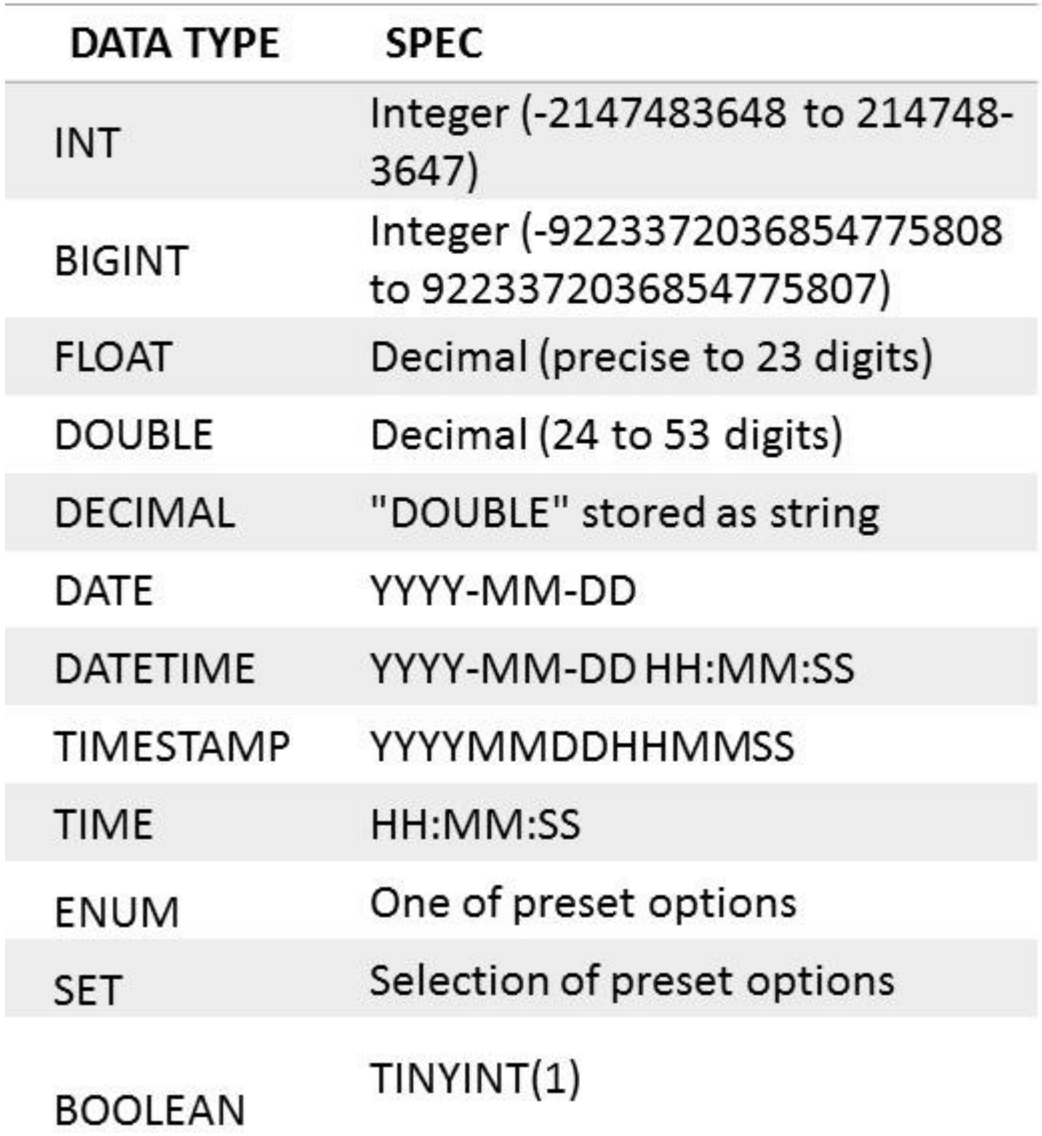
What is SQL?

- Structure Query Language(SQL) is a database query language used for storing and managing data in Relational DBMS.
- SQL was the first commercial language introduced for E.F Codd's Relational model of database.

MySql Datatype



DATE TYPE	SPEC
CHAR	String (0 - 255)
VARCHAR	String (0 - 255)
TINYTEXT	String (0 - 255)
TEXT	String (0 - 65535)
BLOB	String (0 - 65535)
MEDIUMTEXT	String (0 - 16777215)
MEDIUMBLOB	String (0 - 16777215)
LONGTEXT	String (0 - 4294967295)
LONGBLOB	String (0 - 4294967295)
TINYINT	Integer (-128 to 127)
SMALLINT	Integer (-32768 to 32767)
MEDIUMINT	Integer (-8388608 to 8388607)



DATA TYPE	SPEC
INT	Integer (-2147483648 to 214748-3647)
BIGINT	Integer (-9223372036854775808 to 9223372036854775807)
FLOAT	Decimal (precise to 23 digits)
DOUBLE	Decimal (24 to 53 digits)
DECIMAL	"DOUBLE" stored as string
DATE	YYYY-MM-DD
DATETIME	YYYY-MM-DD HH:MM:SS
TIMESTAMP	YYYYMMDDHHMMSS
TIME	HH:MM:SS
ENUM	One of preset options
SET	Selection of preset options
BOOLEAN	TINYINT(1)

DDL Statements

DDL

- DDL is short name of Data Definition Language.
- DDL deals with database schemas like table.
- DDL Commands
  - CREATE – create the structure of a data base object (ex: table).
  - ALTER – alters the structure of the existing database.
  - DROP – delete objects from the database.
  - TRUNCATE – remove all records from a table, including all spaces allocated for the records are removed.

CREATE Database

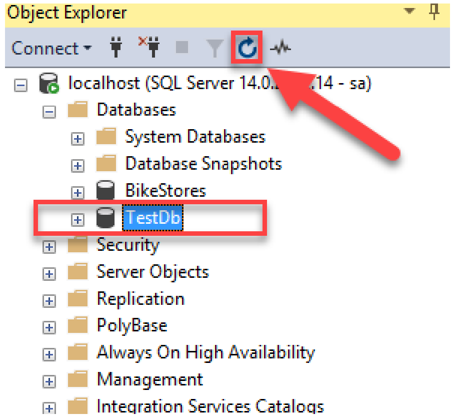
Syntax:

```
CREATE DATABASE database_name;
```

Example:

```
CREATE DATABASE TestDb;
```

- Once the statement executes successfully, you can view the newly created database in the Object Explorer.



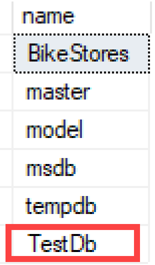
Object Explorer

Connect

```
localhost (SQL Server 14.0.  14 - sa)
  Databases
    System Databases
    Database Snapshots
    BikeStores
    TestDb
  Security
  Server Objects
  Replication
  PolyBase
  Always On High Availability
  Management
  Integration Services Catalogs
```

- lists all databases in the SQL Server:

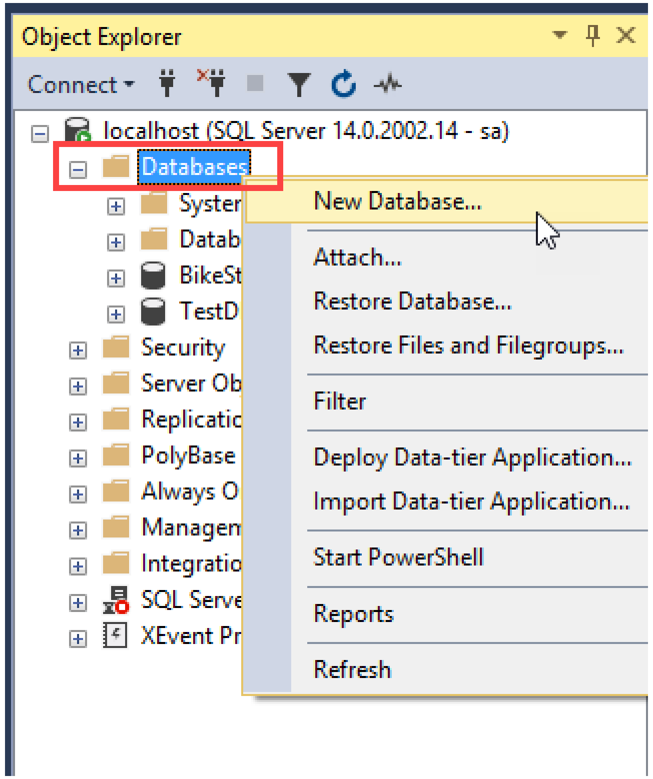
```
SELECT name FROM master.sys.databases
ORDER BY name;

Or

EXEC sp_databases;
```



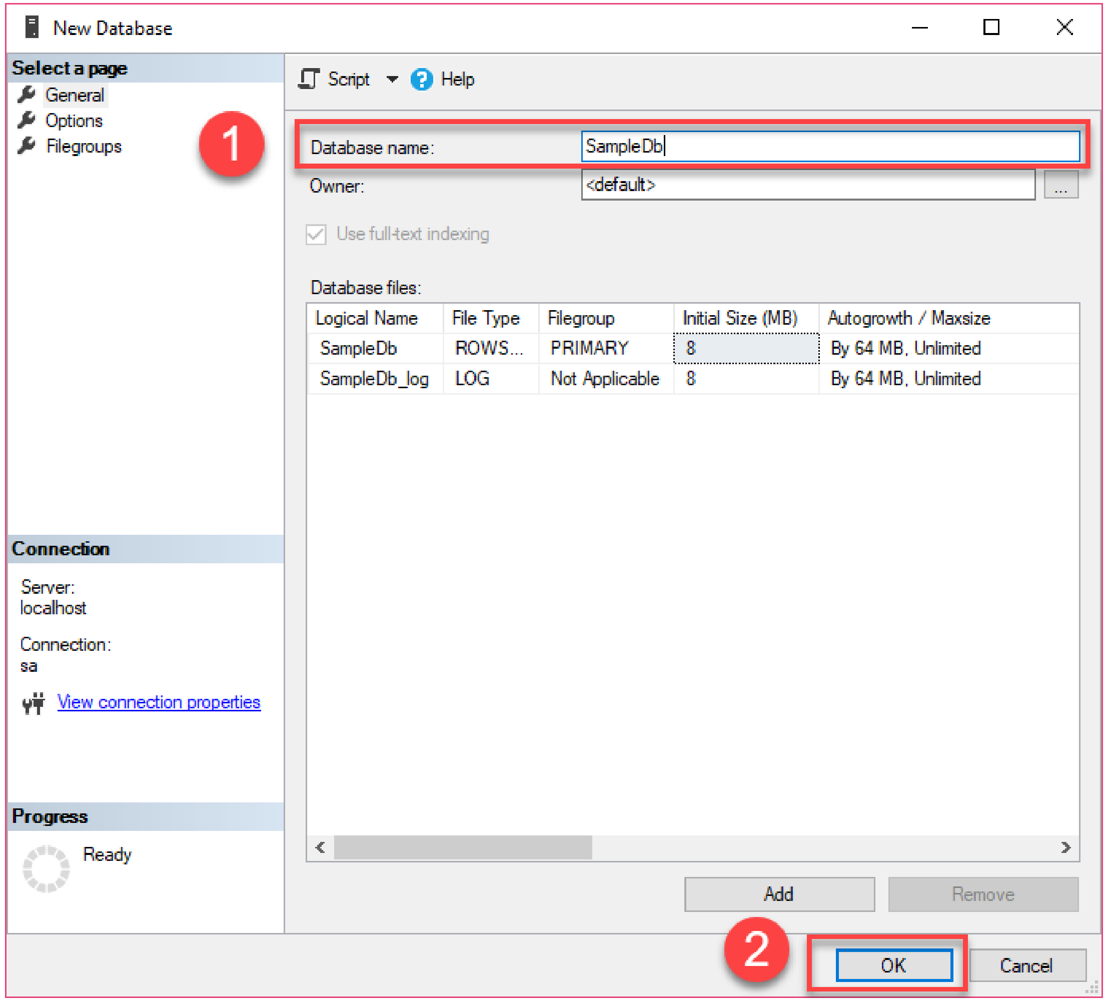
name
BikeStores
master
model
msdb
tempdb
TestDb

CREATE Database using SQL Server Management Studio

First, right-click the Database and choose New Database… menu item.



Second, enter the name of the database e.g., SampleDb and click the OK button.



Database name: SampleDb

Owner: <default>

☑ Use full-text indexing

Server:
localhost

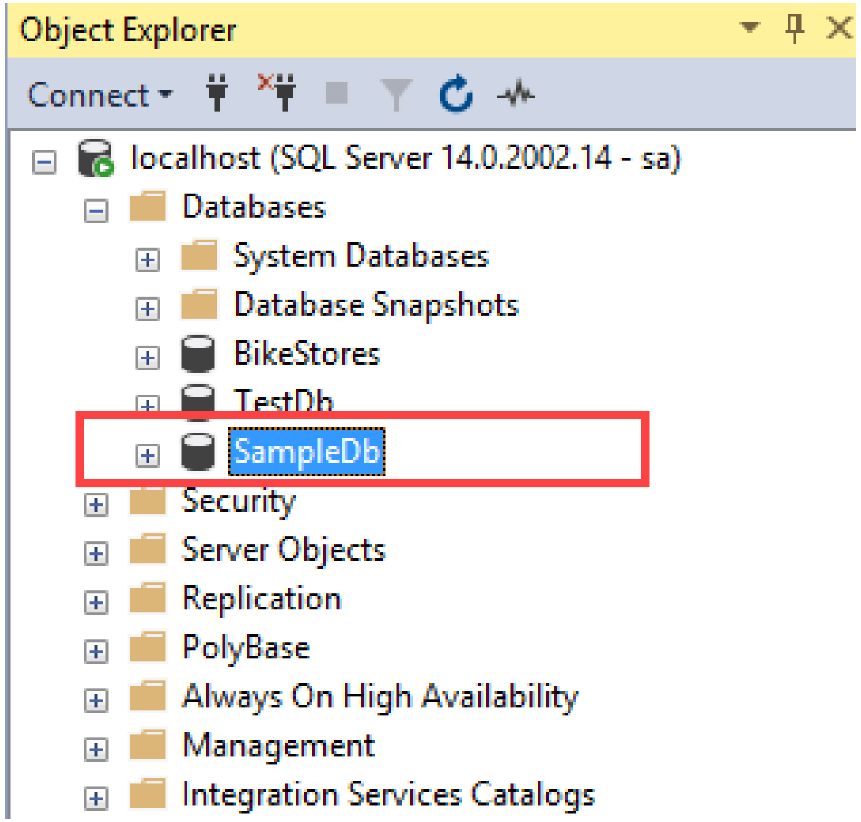
Connection:
sa

Database files:

Logical Name	File Type	Filegroup	Initial Size (MB)	Autogrowth / Maxsize
SampleDb	ROWS...	PRIMARY	8	By 64 MB, Unlimited
SampleDb_log	LOG	Not Applicable	8	By 64 MB, Unlimited

CREATE Database using SQL Server Management Studio

Third, view the newly created database from the Object Explorer:



Delete Database

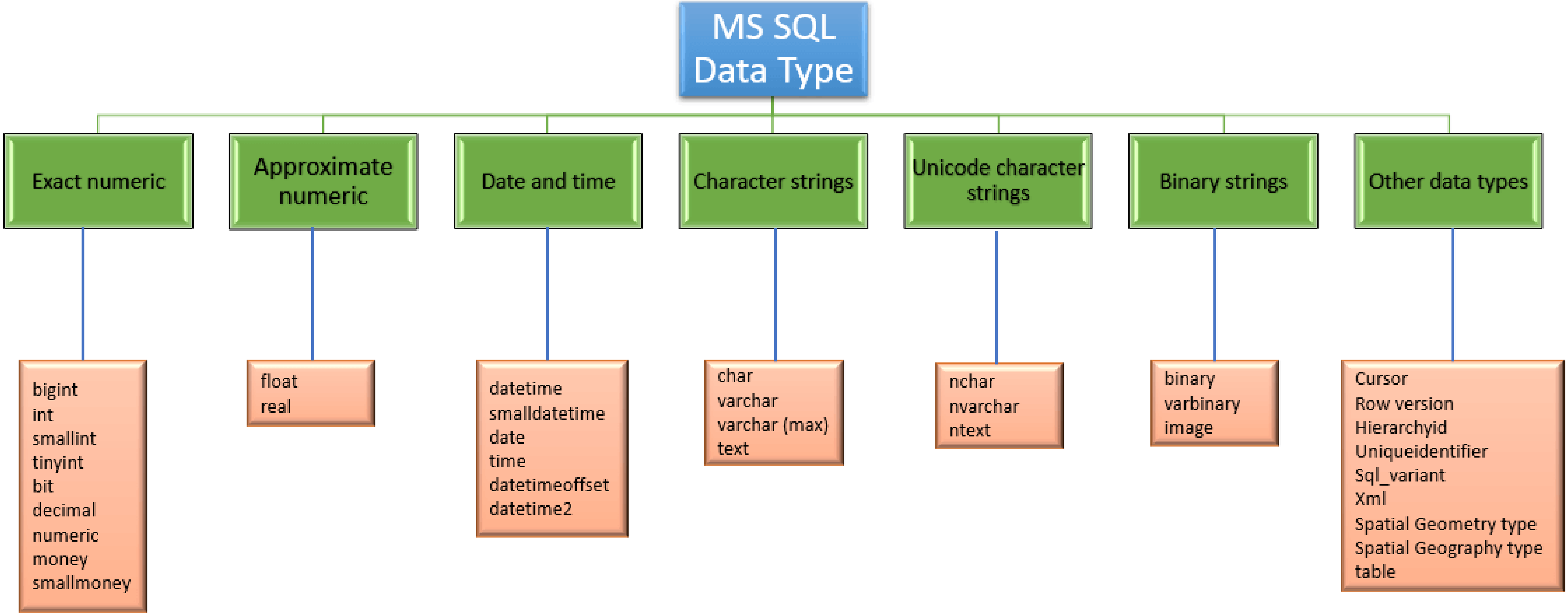
Syntax:

```
DROP DATABASE  [ IF EXISTS ]
   database_name
   [,database_name2,...];
```

Example:

```
DROP DATABASE IF EXISTS TestDb;
```

SQL Server data type



MS SQL Data Type

Exact numeric

bigint
int
smallint
tinyint
bit
decimal
numeric
money
smallmoney

Approximate numeric

float
real

Date and time

datetime
smalldatetime
date
time
datetimeoffset
datetime2

Character strings

char
varchar
varchar (max)
text

Unicode character strings

nchar
nvarchar
ntext

Binary strings

binary
varbinary
image

Other data types

Cursor
Row version
Hierarchyid
Uniqueidentifier
Sql_variant
Xml
Spatial Geometry type
Spatial Geography type
table

CREATE TABLE

Syntax:

```
CREATE TABLE [database_name.][schema_name.]table_name (
   column_definition1,
   column_definition2,
   ........,
   table_constraints
);
```

Example:

```
CREATE TABLE TestDB.dbo.Student(
   Id INT IDENTITY PRIMARY KEY,
   Name VARCHAR(65) NOT NULL,
   Gender VARCHAR(20),
   Age INT,
   Marks INT
 )
```

- command display the structure of the table
  - ✓ Exec sp_help tablename

ALTER TABLE

- ALTER TABLE ADD COLUMN command. It will always add the new column at the last position in the table.

Syntax:

```
ALTER TABLE table_name
ADD
   column_name_1 data_type_1 column_constraint_1,
   column_name_2 data_type_2 column_constraint_2,
   ...,
   column_name_n data_type_n column_constraint_n;
```

Example:

```
ALTER TABLE Student ADD Phone_number VARCHAR(20) NULL;
```

- ALTER TABLE ADD COLUMN command. It will always add the new column at the last position in the table.

Syntax:

```
ALTER TABLE table_name
ADD
   column_name_1 data_type_1 column_constraint_1,
   column_name_2 data_type_2 column_constraint_2,
   ...,
   column_name_n data_type_n column_constraint_n;
```

Example:

```
ALTER TABLE Student ADD Phone_number VARCHAR(20) NULL;
```

ALTER TABLE

- ALTER TABLE ADD COLUMN command. It will always add the new column at the last position in the table.

Syntax:

```
ALTER TABLE table_name
ADD
   column_name_1 data_type_1 column_constraint_1,
   column_name_2 data_type_2 column_constraint_2,
   ...,
   column_name_n data_type_n column_constraint_n;
```

Example:

```
ALTER TABLE Student ADD Phone_number VARCHAR(20) NULL;
```

- ALTER TABLE ALTER COLUMN statement to modify the column data type.

Syntax:

```
ALTER TABLE table_name
ALTER COLUMN column_name new_data_type(size);
```

Example:

```
ALTER TABLE [Student] ALTER COLUMN Gender NVARCHAR(10);
```

ALTER TABLE

- If we want to delete more than one column we can use the following syntax:

Syntax:

```
ALTER TABLE table_name
DROP COLUMN column_name1, DROP COLUMN column_name2...
```

Example:

```
ALTER TABLE Student DROP COLUMN Phone_number;
```

- Add Constraint on the Column

Syntax:

```
ALTER TABLE table_name
ADD CONSTRAINT [constraint_name] PRIMARY KEY ([column_name])
```

Example:

```
ALTER TABLE Student ADD CONSTRAINT PrimaryKey PRIMARY KEY (Id);
```

ALTER TABLE

- Drop Constraint on the Column

Syntax:

```
ALTER TABLE table_name DROP CONSTRAINT [constraint_name]
```

Example:

```
ALTER TABLE Student DROP CONSTRAINT PrimaryKey;
```

INSERT INTO TABLE

- Insert Data into the table

Syntax:

```
INSERT INTO [database_name].[dbo].[table_name]
(column_name1, column_name2, ... )
VALUES
(value1, value2, ... );
```

- store single records for all fields,

Example:

```
INSERT INTO Student (Name, Gender, Age, Marks)
VALUES ('Peter Huges', 'Male', 32, 450);
```

- store multiple records for all fields, use SELECT * FROM student to display records in table

```
INSERT INTO Student
VALUES ('Jolly Evans', 'Female', 28, 475),
('Alan Simmons', 'Male', 32, 405),
('Laura Bennet', 'Female', 30, 435);
```

Example:

INSERT with SELECT statement:

- SQL Server also allows us to insert records from one table into another table using the INSERT INTO SELECT statement. Suppose we want to insert 'Student' table data into 'Student_info'.

Example:

```
INSERT INTO Student_info
SELECT Name, Gender, Marks FROM Student;
```

- insert and return inserted values?
    - SQL Server provides the OUTPUT clause for capturing the inserted values into a defined table. We can explain this concept by using the below statement that inserts a new record into the 'Student' table and returns the inserted value of the 'Marks' column

Example:

```
INSERT INTO Student (Name, Gender, Age, Marks )
OUTPUT inserted.Marks
VALUES ('J P Dumini', 'Male', 32, 450);
```

Update Data in table

- The UPDATE query is always recommended to use with the SET and WHERE clause. We can modify or update the single or multiple columns at a time.

Syntax:

```
UPDATE [database_name].[ schema_name].table_name
SET column1 = new_value1,
    column2 = new_value2, ...
[WHERE Clause]
```

- Update Single Column

Example:

```
UPDATE Student
SET Marks = 492
WHERE Name = 'Alan Simmons';
```

- Update Multiple Column

Example:

```
UPDATE Student
SET Age = 28, Marks = 492
WHERE Name = 'Diego Bennet';
```

Delete table

- Use the DELETE statement to delete data from the existing table in the current schema or tables of the schema on which you have the DELETE privilege.

Syntax:

```
DELETE FROM table_name [WHERE Condition];
```

- delete Single record

Example:

```
DELETE FROM student WHERE id = 1;
```

- delete Delete All Rows which results in Now, the Select * from Employee query will display the empty table.

Example:

```
DELETE FROM student;
```

- Drop a table that does not exist

Example:

```
DROP TABLE IF EXISTS sales.revenues;
```

Demo

DML Statements

SQL WHERE clause

- To select specific rows from a table, you use a WHERE clause in the SELECT statement.
- The WHERE clause contains one or more logical expressions that evaluate each row in the table.
- If a row that causes the condition evaluates to true, it will be included in the result set; otherwise, it will be excluded.

```
SELECT column1, column2, ...

FROM table_name

WHERE condition;
```

Comparison operators

Operator	Meaning
=	Equal to
<> (!=)	Not equal to
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal

SQL WHERE clause Logical Operators

- A logical operator allows you to test for the truth of a condition.

Logical Operators

S.No	Operator	Meaning
1	ALL	Return true if all comparisons are true
2	AND	Return true if both expressions are true
3	ANY	Return true if any one of the comparisons is true.
4	BETWEEN	Return true if the operand is within a range
5	EXISTS	Return true if a subquery contains any rows
6	IN	Return true if the operand is equal to one of the value in a list
7	LIKE	Return true if the operand matches a pattern
8	NOT	Reverse the result of any other Boolean operator.
9	OR	Return true if either expression is true
10	SOME	Return true if some of the expressions are true

SQL WHERE clause

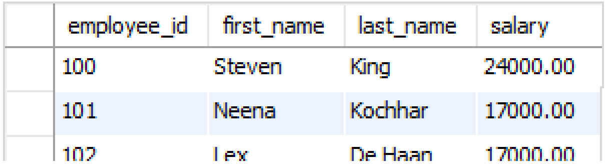
Example

- *SQL WHERE clause with numeric comparison*

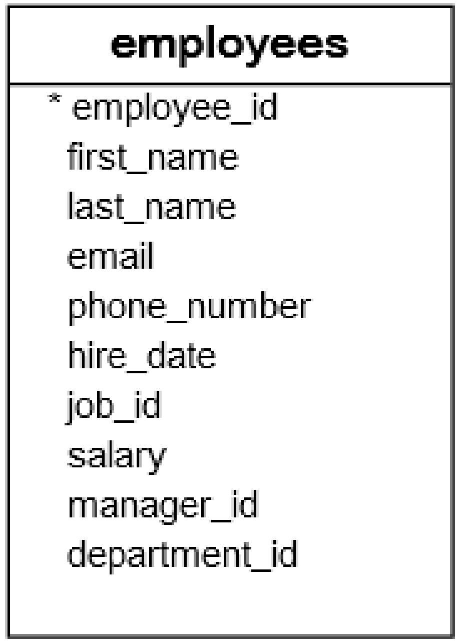
  SELECT employee_id, first_name, last_name, salary

  FROM [dbo].[Employees]

  WHERE salary > 14000



employee_id	first_name	last_name	salary
100	Steven	King	24000.00
101	Neena	Kochhar	17000.00
102	Lex	De Haan	17000.00



employees

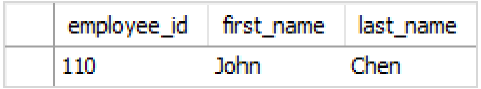
* employee_id
first_name
last_name
email
phone_number
hire_date
job_id
salary
manager_id
department_id

- *SQL WHERE clause with characters comparison*

  SELECT employee_id, first_name, last_name

  FROM [dbo].[Employees]

  WHERE last_name = 'Chen';



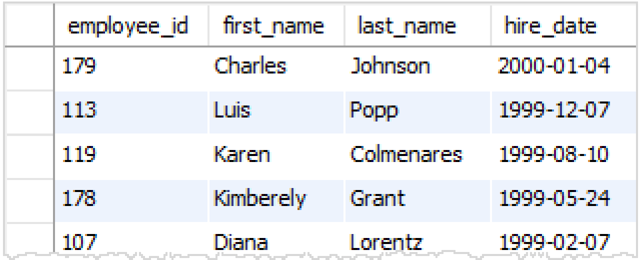
employee_id	first_name	last_name
110	John	Chen

- *SQL WHERE clause with date comparison*

  SELECT employee_id, first_name, last_name, hire_date

  FROM [dbo].[Employees]

  WHERE hire_date >= '1999-01-01';



employee_id	first_name	last_name	hire_date
179	Charles	Johnson	2000-01-04
113	Luis	Popp	1999-12-07
119	Karen	Colmenares	1999-08-10
178	Kimberely	Grant	1999-05-24
107	Diana	Lorentz	1999-02-07

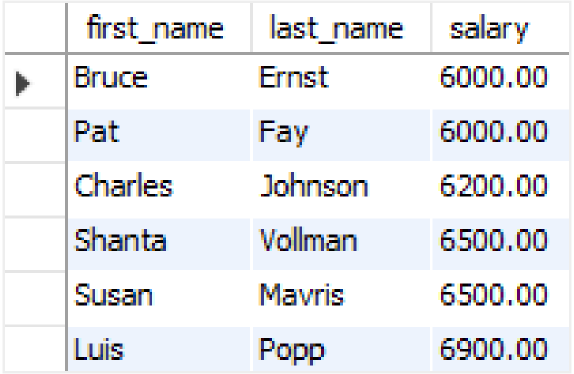
Note: Use YEAR function to get the year from hire_date column
WHERE YEAR (hire_date) = 1999

SQL WHERE clause Logical Operators

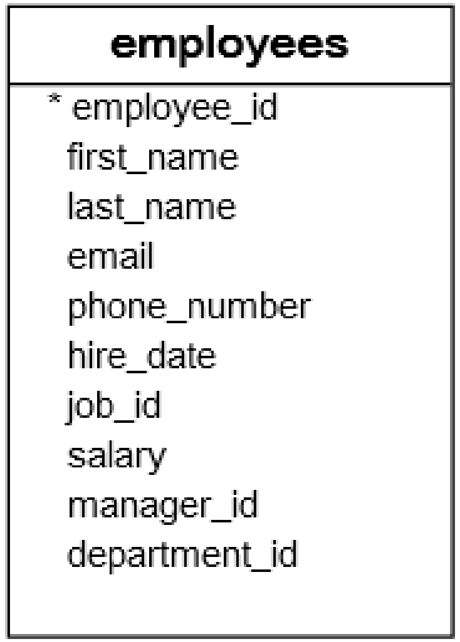
Example

- *SQL WHERE clause with **AND** operator*

```
SELECT first_name, last_name, salary
FROM [dbo].[Employees]
WHERE salary > 5000 AND salary < 7000
```



	first_name	last_name	salary
▶	Bruce	Ernst	6000.00
	Pat	Fay	6000.00
	Charles	Johnson	6200.00
	Shanta	Vollman	6500.00
	Susan	Mavris	6500.00
	Luis	Popp	6900.00

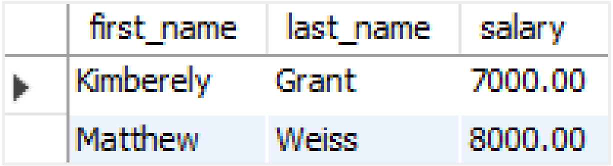


employees

* employee_id
first_name
last_name
email
phone_number
hire_date
job_id
salary
manager_id
department_id

- *SQL WHERE clause with **OR** operator*

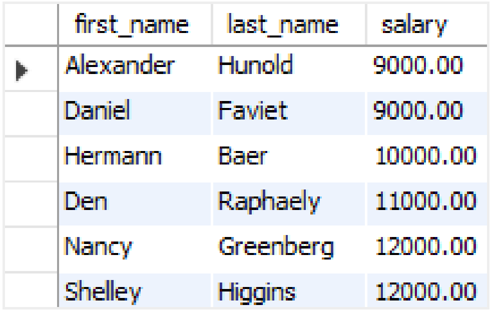
```
SELECT employee_id, first_name, last_name
FROM [dbo].[Employees]
WHERE salary = 7000 OR salary = 8000;
```



	first_name	last_name	salary
▶	Kimberely	Grant	7000.00
	Matthew	Weiss	8000.00

- *SQL WHERE clause with **BETWEEN** operator*

```
SELECT employee_id, first_name, last_name, hire_date
FROM [dbo].[Employees]
WHERE salary BETWEEN 9000 AND 12000
```



	first_name	last_name	salary
▶	Alexander	Hunold	9000.00
	Daniel	Faviet	9000.00
	Hermann	Baer	10000.00
	Den	Raphaely	11000.00
	Nancy	Greenberg	12000.00
	Shelley	Higgins	12000.00

SQL WHERE clause Logical Operators

- The IN operator compares a value to a list of specified values. The IN operator returns true if the compared value matches at least one value in the list; otherwise, it returns false.

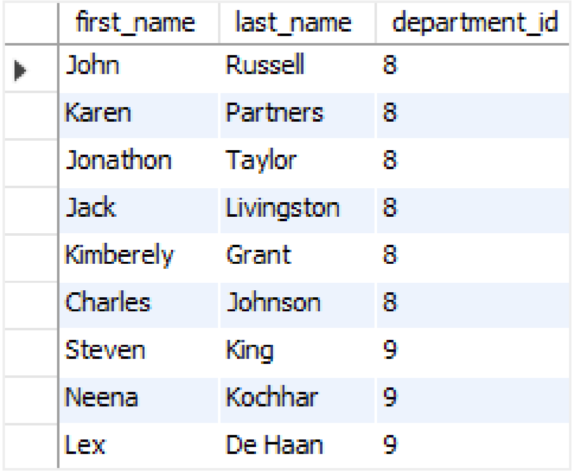
Example

- *SQL WHERE clause with AND operator*

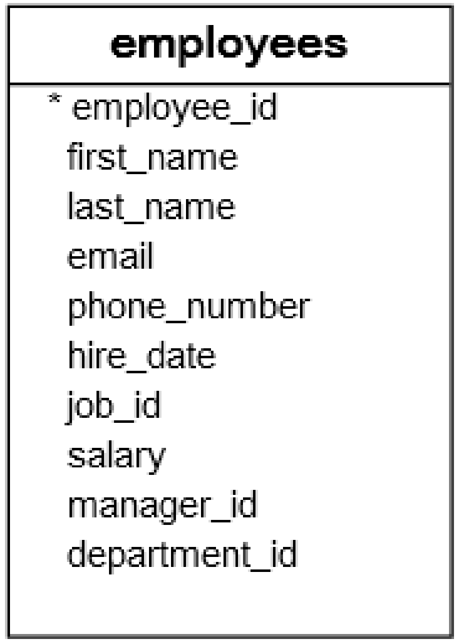
SELECT first_name, last_name, department_id

FROM [dbo].[Employees]

WHERE department_id IN (8, 9)



first_name	last_name	department_id
John	Russell	8
Karen	Partners	8
Jonathon	Taylor	8
Jack	Livingston	8
Kimberely	Grant	8
Charles	Johnson	8
Steven	King	9
Neena	Kochhar	9
Lex	De Haan	9



employees

* employee_id
first_name
last_name
email
phone_number
hire_date
job_id
salary
manager_id
department_id

SQL WHERE clause Logical Operators

- The LIKE operator compares a value to similar values using a wildcard operator.
  - The percent sign (%) represents zero, one, or multiple characters.
  - The underscore sign (_) represents a single character.

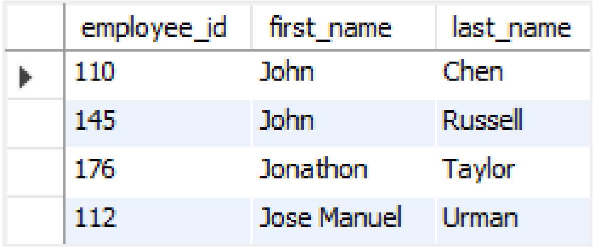
Example

- *SQL WHERE clause with LIKE operator*

SELECT employee_id, first_name, last_name

FROM [dbo].[Employees]

WHERE first_name LIKE 'jo%’;



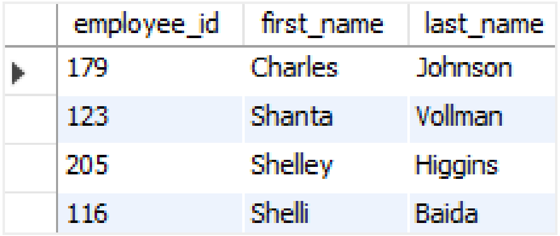
employee_id	first_name	last_name
110	John	Chen
145	John	Russell
176	Jonathon	Taylor
112	Jose Manuel	Urman

- *SQL WHERE clause with LIKE operator*

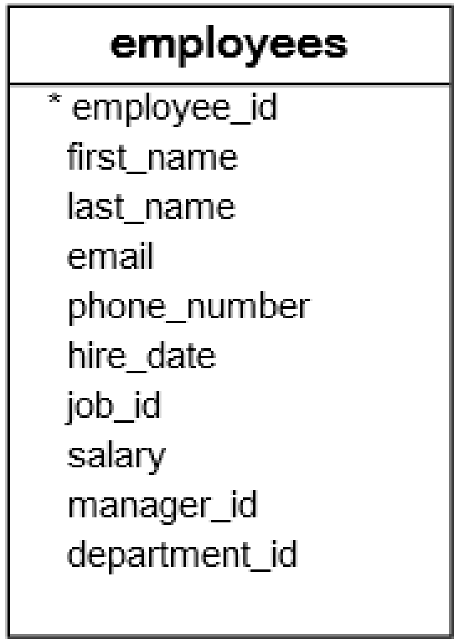
SELECT employee_id, first_name, last_name

FROM [dbo].[Employees]

WHERE first_name LIKE '_h%'



employee_id	first_name	last_name
179	Charles	Johnson
123	Shanta	Vollman
205	Shelley	Higgins
116	Shelli	Baida



employees

* employee_id
first_name
last_name
email
phone_number
hire_date
job_id
salary
manager_id
department_id

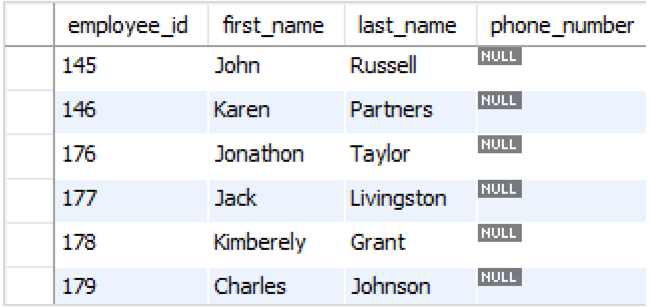
SQL WHERE clause with IS NULL

- To determine whether an expression or column is NULL or not, you use the IS NULL operator.
- To check if an expression or column is not NULL, you use the IS NOT operator:

Example

➢ *SQL WHERE clause with IS NULL operator*

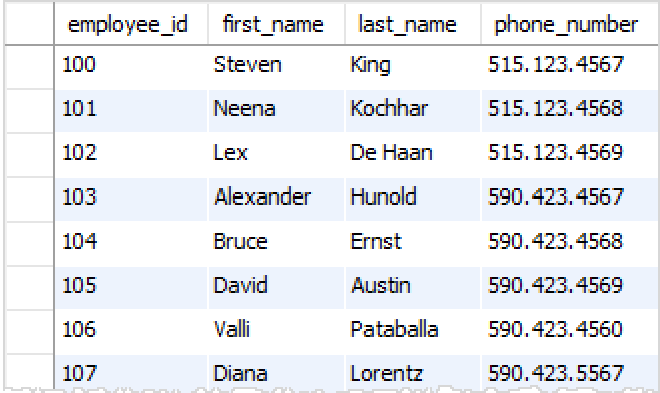
```
SELECT employee_id, first_name, last_name,    phone_number
FROM [dbo].[Employees]
WHERE phone_number IS NULL;
```



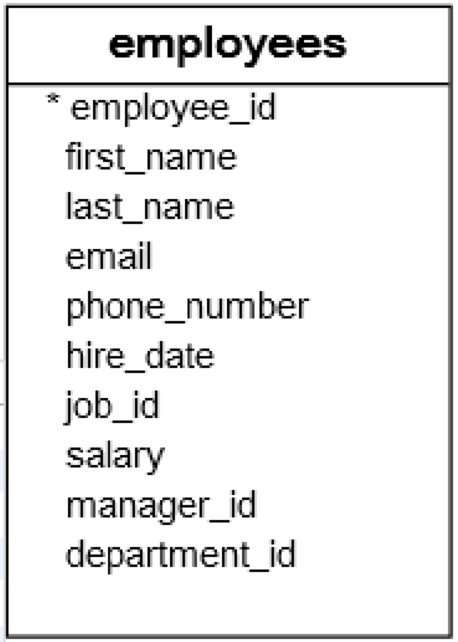
employee_id	first_name	last_name	phone_number
145	John	Russell	NULL
146	Karen	Partners	NULL
176	Jonathon	Taylor	NULL
177	Jack	Livingston	NULL
178	Kimberely	Grant	NULL
179	Charles	Johnson	NULL

➢ *SQL WHERE clause with IS NOT NULL operator*

```
SELECT employee_id, first_name, last_name,    phone_number
FROM [dbo].[Employees]
WHERE phone_number IS NOT NULL;
```



employee_id	first_name	last_name	phone_number
100	Steven	King	515.123.4567
101	Neena	Kochhar	515.123.4568
102	Lex	De Haan	515.123.4569
103	Alexander	Hunold	590.423.4567
104	Bruce	Ernst	590.423.4568
105	David	Austin	590.423.4569
106	Valli	Pataballa	590.423.4560
107	Diana	Lorentz	590.423.5567



SQL ORDER BY clause

- The ORDER BY is an optional clause of the SELECT statement.
- The ORDER BY clause allows you to sort the rows returned by the SELECT clause by one or more sort expressions in ascending or descending order.

Syntax:

```
    SELECT column_list
    FROM table1
    ORDER BY sort_expression [ASC | DESC];
```

SQL ORDER BY clause

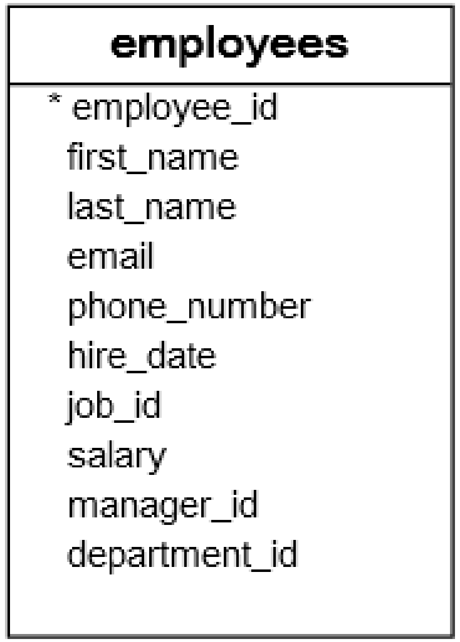
Example

- *SQL ORDER BY DESC clause*

```
SELECT employee_id, first_name, last_name, hire_date, salary

FROM [dbo].[Employees]

ORDER BY hire_date DESC;
```

- *SQL ORDER BY clause*

```
SELECT employee_id, first_name, last_name, hire_date, salary

FROM [dbo].[Employees]

ORDER BY first_name;
```



employees

* employee_id
first_name
last_name
email
phone_number
hire_date
job_id
salary
manager_id
department_id

Note:- ORDER BY clause sort first_name column in ascending order or use ASC Keyword

SQL LIMIT and OFFSET clause

- To limit the number of rows returned by a select statement, you use the FETCH and OFFSET clauses.

Syntax:

ORDER BY column_list **[ASC |DESC]**

OFFSET offset_row_count **{ROW | ROWS}**

FETCH {FIRST | NEXT} fetch_row_count **{ROW | ROWS} ONLY;**

- The FETCH row_count determines the number of rows (row_count) returned by the query.
- The OFFSET offset clause skips the offset rows before beginning to return the rows. The OFFSET clause is optional.

SQL LIMIT and OFFSET clause

Example

- *SQL FETCH and OFFSET clause*

```
SELECT product_name, list_price
FROM production.products
ORDER BY list_price, product_name
OFFSET 10 ROWS
FETCH NEXT 10 ROWS ONLY;
```

Demo

Quiz

3

Fill the below given query to select all the records where the value of the Price column is range 10 to 20

SELECT * FROM Products
WHERE Price ___________;

A. BETWEEN 10 AND 20
B. BETWEEN 10 OR 20
C. BETWEEN 10 >= 20
D. None of the above

BETWEEN 10 AND 20

4

Select all records where the first letter of the City starts with anything from an "a" to "f".

SELECT * FROM Customers
WHERE City _____________;

A. LIKE '%[a to f]’
B. LIKE '[a to f] %'
C. LIKE '[a-f] %’
D. LIKE '%[a-f]'

Ans: LIKE '[a-f] %'

5

Which of the following is considered as virtual table and does not necessarily exist in physical form?

A. Trigger B. View

B. C. Stored Procedure D. Table

View

6

A function that has no partial functional dependencies is in _____ form

A 3NF

B 2NF

C 4NF

D BCNF

B 2NF

7

Third normal form is based on the concept of _______

A Closure Dependency B Transitive Dependency

C Normal Dependency D Functional Dependency

B Transitive Dependency

Queries

References

1. https://powerbidocs.com/2019/12/25/sql-keys/
2. https://www.boardinfinity.com/blog/a-quick-guide-to-entities-in-dbms/

Thank you

Innovative Services

Passionate Employees

Delighted Customers