## UNIT 3:

## Relational database structure

The database and the database structure are defined in the installation process. The structure of the database depends on whether the database is Oracle Database, IBM® Db2®, or Microsoft SQL Server.

A database that can be perceived as a set of tables and manipulated in accordance with the relational model of data. Each database includes:

- a set of system catalog tables that describe the logical and physical structure of the data
- a configuration file containing the parameter values allocated for the database
- a recovery log with ongoing transactions and archivable transactions

**Terminologies**:

| S.N | Component | Description |
|---|---|---|
| 1 | Data dictionary | A repository of information about the application programs, databases, logical data models, and authorizations for an organization. When you change the data dictionary, the change process includes edit checks that can prevent the data dictionary from being corrupted. The only way to recover a data dictionary is to restore it from a backup. |
| 2 | Container | A data storage location, for example, a file, directory, or device that is used to define a database. |
| 3 | Storage partition | A logical unit of storage in a database such as a collection of containers. Database storage partitions are called table spaces in Db2 and Oracle, and called file groups in SQL Server. |
| 4 | Business object | A tangible entity within an application that users create, access, and manipulates while performing a use case. Business objects within a system are typically stateful, persistent, and long-lived. Business objects contain business data and model the business behavior. |
| 5 | Database object | An object that exists in an installation of a database system, such as an instance, a database, a database partition group, a buffer pool, a table, or an index. A database object holds data and has no behavior. |
| 6 | Table | A database object that holds a collection of data for a specific topic. Tables consist of rows and columns. |
| 7 | Column | The vertical component of a database table. A column has a name and a particular data type for example, character, decimal, or integer. |
| 8 | Row | The horizontal component of a table, consisting of a sequence of values, one for each column of the table. |
| 9 | View | A logical table that is based on data stored in an underlying set of tables. The data returned by a view is determined by a SELECT statement that is run on the underlying tables. |
| 10 | Index | A set of pointers that is logically ordered by the values of a key. Indexes provide quick access to data and can enforce uniqueness of the key values for the rows in the table. |
| 11 | Relationship | A link between one or more objects that is created by specifying a join statement. |
| 12 | Join | An SQL relational operation in which data can be retrieved from two tables, typically based on a join condition specifying join columns. |

**Data dictionary tables**

The structure of a relational database is stored in the data dictionary tables of the database.

**Integrity checker**

The integrity checker is a database configuration utility that you can use to assesses the health of the base layer data dictionary. The tool compares the data dictionary with the underlying physical database schema. If errors are detected, the tool produces error messages detailing how to resolve the issues.

**Storage partitions**

A database storage partition is the location where a database object is stored on a disk. Database storage partitions are called table spaces in Db2 and Oracle, and called file groups in SQL Server.

**Business objects**

A business object is an object that has a set of attributes and values, operations, and relationships to other business objects. Business objects contain business data and model the business behaviour.

**User-defined objects**

Objects can be created in two ways: you can create an object in the database or an object can be natively defined in the database. User-defined objects are always created in the Database Configuration application.

**Configuration levels for objects**

Levels describe the scope of objects and must be applied to objects. Depending on the level that you assign to objects, you must create certain attributes. For users to access an object, an attribute value must exist at the level to which they have authority. The level that you assign to an object sometimes depends on the level of the record in the database.

**Database relationships**

Database relationships are associations between tables that are created using join statements to retrieve data.

**Business object attributes**

Attributes of business objects contain the data that is associated with a business object. A persistent attribute represents a database table column or a database view column. A nonpersistent attribute exists in memory only, because the data that is associated with the attribute is not stored in the database.

**Attribute data types**

Each database record contains multiple attributes. Every attribute has an associated data type.

**Database views**

A database view is a subset of a database and is based on a query that runs on one or more database tables. Database views are saved in the database as named queries and can be used to save frequently used, complex queries.

**Indexes**

You can use indexes to optimize performance for fetching data. Indexes provide pointers to locations of frequently accessed data. You can create an index on the columns in an object that you frequently query.

**Primary keys**

When you assign a primary key to an attribute, the key uniquely identifies the object that is associated with that attribute. The value in the primary column determines which attributes are used to create the primary key.

## DBMS Schema

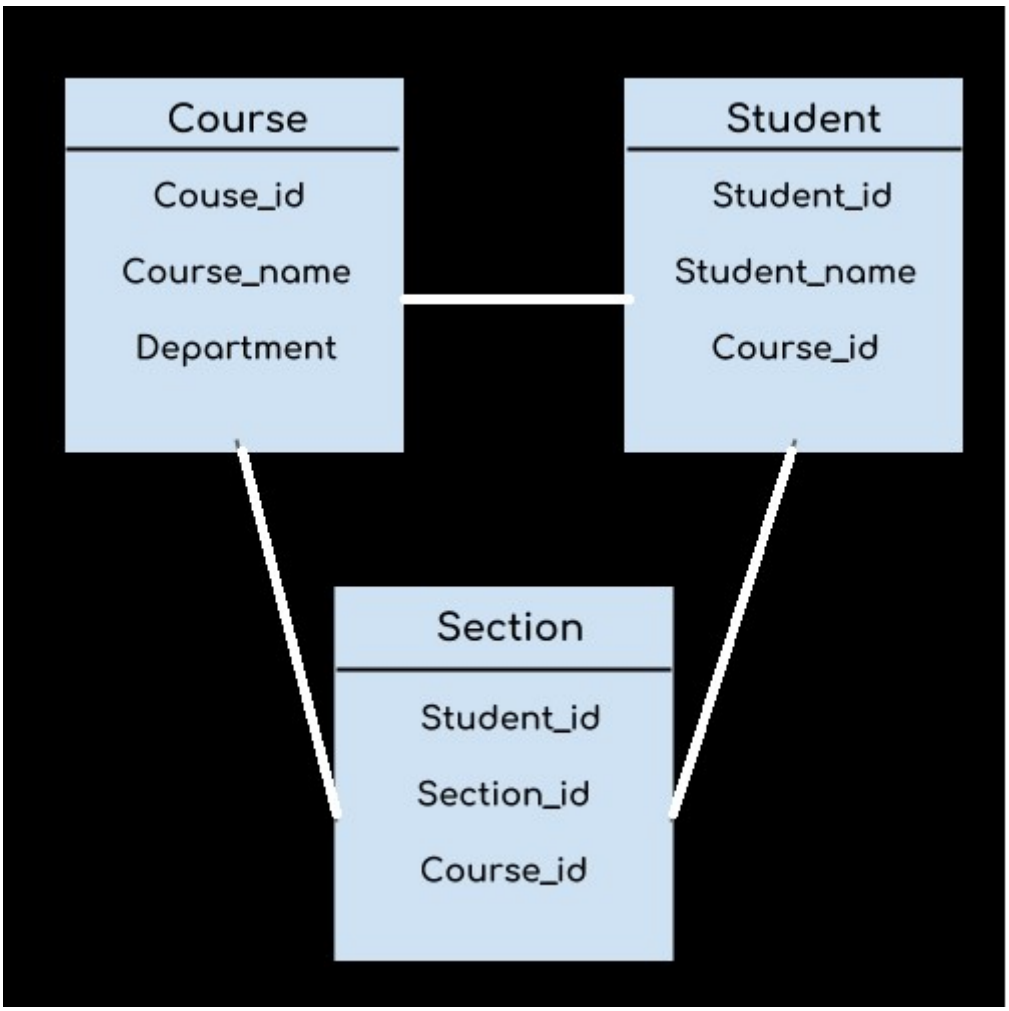
Definition of schema: Design of a database is called the schema. Schema is of three types: Physical schema, logical schema and view schema.

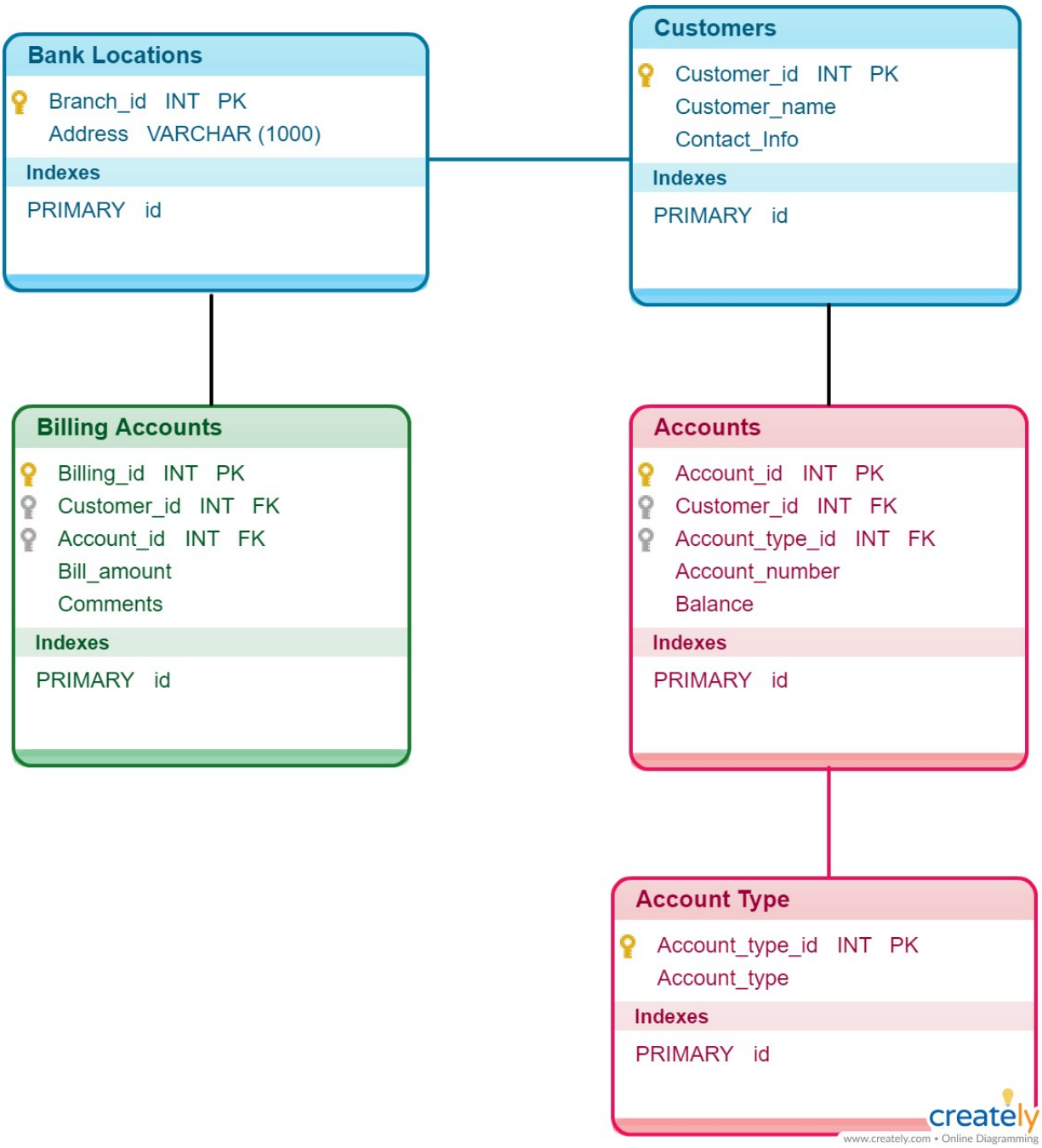
The design of a database at physical level is called physical schema, how the data stored in blocks of storage is described at this level.

Design of database at logical level is called logical schema, programmers and database administrators work at this level, at this level data can be described as certain types of data records gets stored in data structures, however the internal details such as implementation of data structure is hidden at this level (available at physical level).

Design of database at view level is called view schema. This generally describes end user interaction with database systems.

For example: In the following diagram, we have a schema that shows the relationship between three tables: Course, Student and Section. The diagram only shows the design of the database, it doesn't show the data present in those tables. Schema is only a structural view(design) of a database as shown in the diagram below.


Course
Couse_id
Course_name
Department
Student
Student_id
Student_name
Course_id
Section
Student_id
Section_id
Course_id




## Keys:

Databases are used to store massive amounts of information which is stored across multiple tables. Each table might be running into thousands of rows. Needless to say, there will be many duplicate rows with redundant information. How do we deal with that? How do we manage records so that we are storing only unique data? And, how do we relate the multiple tables that are present in the database?

SQL keys are the answer to all these queries.

An SQL key is either a single column (or attribute) or a group of columns that can uniquely identify rows (or tuples) in a table.

SQL keys ensure that there are no rows with duplicate information. Not only that, but they also help in establishing a relationship between multiple tables in the database. Therefore, it becomes imperative to learn about the different keys in SQL.

### Super key:

Super key is a single key or a group of multiple keys that can uniquely identify tuples in a table.

Super Key can contain multiple attributes that might not be able to independently identify tuples in a table, but when grouped with certain keys, they can identify tuples uniquely.

Let me take an example to clarify the above statement. Have a look at the following table.

| Id | Name | Gender | City | Email | Dep_Id |
|---|---|---|---|---|---|
| 1 | Ajay | M | Delhi | ajay@gmail.com | 1 |
| 2 | Vijay | M | Mumbai | vijay@gmail.com | 2 |
| 3 | Radhika | F | Bhopal | radhika@gmail.com | 1 |
| 4 | Shikha | F | Jaipur | shikha@gmail.com | 2 |
| 5 | Hritik | M | Jaipur | hritik@gmail.com | 2 |

5 rows in set (0.00 sec)

Consider that Id attribute is unique to every employee. In that case, we can say that the Id attribute can uniquely identify the tuples of this table. So, Id is a Super key of this table. Note that we can have other Super Keys too in this table.

For instance – (Id, Name), (Id, Email), (Id, Name, Email), etc. can all be Super keys as they can all uniquely identify the tuples of the table. This is so because of the presence of the Id attribute which is able to uniquely identify the tuples. The other attributes in the keys are unnecessary. Nevertheless, they can still identify tuples.

**Candidate key:**

Candidate key is a single key or a group of multiple keys that uniquely identify rows in a table.

A Candidate key is a subset of Super keys and is devoid of any unnecessary attributes that are not important for uniquely identifying tuples.

The value for the Candidate key is unique and non-null for all tuples. And every table has to have at least one Candidate key. But there can be more than one Candidate Key too.

For example, in the example that we took earlier, both Id and Email can act as a Candidate for the table as they contain unique and non-null values.

```
+----+---------+--------+--------+--------------------+--------+
| Id | Name    | Gender | City   | Email              | Dep_Id |
+----+---------+--------+--------+--------------------+--------+
|  1 | Ajay    | M      | Delhi  | ajay@gmail.com     |      1 |
|  2 | Vijay   | M      | Mumbai | vijay@gmail.com    |      2 |
|  3 | Radhika | F      | Bhopal | radhika@gmail.com  |      1 |
|  4 | Shikha  | F      | Jaipur | shikha@gmail.com   |      2 |
|  5 | Hritik  | M      | Jaipur | hritik@gmail.com   |      2 |
+----+---------+--------+--------+--------------------+--------+
5 rows in set (0.00 sec)
```

### Primary Key

Primary key is the Candidate key selected by the database administrator to uniquely identify tuples in a table.

Out of all the Candidate keys that can be possible for a table, there can be only one key that will be used to retrieve unique tuples from the table. This Candidate key is called the Primary Key.

There can be only one Primary key for a table. Depending on how the Candidate Key is constructed the primary key can be a single attribute or a group of attributes. But the important point to remember is that the Primary key should be a unique and non-null attribute(s). Add the Primary Key constraint at the end after defining all the attributes in the table.

```
mysql> Create table Student(S_Id int Not Null,
    -> Name varchar(20),
    -> Gender varchar(3),
    -> Class int,
    -> Primary Key(S_Id));
Query OK, 0 rows affected (0.75 sec)
```

To define a Primary Key constraint on multiple attributes, you can list all the attributes in the parenthesis as shown below.

```
mysql> Create table Person(Id int Not Null,
    -> Name varchar(20),
    -> City varchar(20),
    -> Email varchar(20) Not Null,
    -> Phone varchar(10) Not Null,
    -> Primary Key(Id, Email, Phone));
Query OK, 0 rows affected (0.79 sec)
```

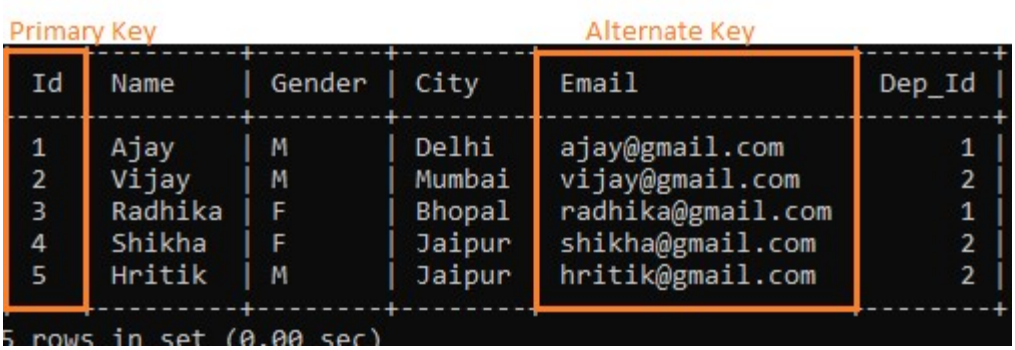
But remember that these attributes should be defined as non-null values otherwise the whole purpose of using the Primary key to identify tuples uniquely gets defeated.

### Alternate or Secondary keys

Alternate keys are those candidate keys which are not the Primary key.

There can be only one Primary key for a table. Therefore, all the remaining Candidate keys are known as Alternate or Secondary keys. They can also uniquely identify tuples in a table, but the database administrator chose a different key as the Primary key.

If we look at the Employee table once again, since I have chosen Id as the Primary key, the other Candidate Key (Email), becomes the Alternate key for the table.




### Foreign key

Foreign key is an attribute which is a Primary key in its parent table, but is included as an attribute in another host table.

A Foreign key generates a relationship between the parent table and the host table. For example, in addition to the Employee table containing the personal details of the employees, we might have another table Department containing information related to the department of the employee.

```
mysql> Select * from Department;
+----+-----------+----------+
| Id | Name      | Location |
+----+-----------+----------+
|  1 | Marketing | Kolkata  |
|  2 | Finance   | Mumbai   |
+----+-----------+----------+
2 rows in set (0.14 sec)
```

The Primary key in this table is the Department Id. We can add this attribute to the Employee by making it the Foreign key in the table. We can either do this when we are creating the table or we can alter the table later to add the Foreign Key constraint. Here I have altered the table, but creating Foreign Key during table creation is similar to that for Primary Key.

```
mysql> Alter table Employee
    -> Add Foreign Key(Dep_Id) References Department(Id);
Query OK, 5 rows affected (3.37 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Here, Dep_Id is now the Foreign Key in table Employee while it is a Primary Key in the Department table.

The Foreign key allows you to create a relationship between two tables in the database. Each of these tables describes data related to a particular field (employee and department here). Using the Foreign key we can easily retrieve data from both the tables.

```
mysql> select employee.name, employee.city, department.name
    -> from employee inner join department
    -> on employee.dep_id = department.id;
+---------+--------+-----------+
| name    | city   | name      |
+---------+--------+-----------+
| Ajay    | Delhi  | Marketing |
| Radhika | Bhopal | Marketing |
| Vijay   | Mumbai | Finance   |
| Shikha  | Jaipur | Finance   |
| Hritik  | Jaipur | Finance   |
+---------+--------+-----------+
5 rows in set (0.00 sec)
```

Note: To operate on Foreign keys, you need to know about Joins.

Using Foreign keys makes it easier to update the database when required. This is so because we only have to make the necessary changes in limited rows. For example, if the Marketing department shifts from Kolkata to Pune, instead of updating it for all the relevant rows in the Employee table, we can simply update the location in the Department table. This ensures that there are only a few places to update and less risk of having different data in different places.

### Composite keys

A Composite key is a Candidate key or Primary key that consists of more than one attribute.

Sometimes it is possible that no single attribute will have the property to uniquely identify tuples in a table. In such cases, we can use a group of attributes to guarantee uniqueness. Combining these attributes will uniquely identify tuples in the table.

Consider the following table:

```
mysql> select * from product;
+----------------+------------+-------------+--------------+----------+
| Transaction_Id | Product_Id | Customer_Id | Product      | Quantity |
+----------------+------------+-------------+--------------+----------+
| A1001          | P1005      | C9001       | Smartphone   |        1 |
| A1001          | P2010      | C9001       | Screen guard |        1 |
| A1002          | P2013      | C9003       | Smartwatch   |        1 |
| A1003          | P2010      | C9010       | Screen guard |        2 |
+----------------+------------+-------------+--------------+----------+
4 rows in set (0.00 sec)
```

Here, neither of the attributes contains unique values to identify the tuples. Therefore, we can combine two or more attributes to create a key that can uniquely identify the tuples. For example, we can group Transaction_Id and Product_Id to create a key that can uniquely identify the tuples. These are called composite keys.

**Key difference between SQL Keys**

- SQL keys are used to uniquely identify rows in a table.
- SQL keys can either be a single column or a group of columns.
- Super key is a single key or a group of multiple keys that can uniquely identify tuples in a table.
- Super keys can contain redundant attributes that might not be important for identifying tuples.
- Candidate keys are a subset of Super keys. They contain only those attributes which are required to uniquely identify tuples.
- All Candidate keys are Super keys. But the vice-versa is not true.
- Primary key is a Candidate key chosen to uniquely identify tuples in the table.
- Primary key values should be unique and non-null.
- There can be multiple Super keys and Candidate keys in a table, but there can be only one Primary key in a table.
- Alternate keys are those Candidate keys that were not chosen to be the Primary key of the table.
- Composite key is a Candidate key that consists of more than one attribute.
- Foreign key is an attribute which is a Primary key in its parent table but is included as an attribute in the host table.
- Foreign keys may accept non-unique and null values.

## Relational Algebra

Relational algebra is a procedural query language that works on relational model. The purpose of a query language is to retrieve data from database or perform various operations such as insert, update, delete on the data. When I say that relational algebra is a procedural query language, it means that it tells what data to be retrieved and how to be retrieved.

(What to do and How to do).

On the other hand relational calculus is a non-procedural query language, which means it tells what data to be retrieved but doesn't tell how to retrieve it.

**Types of operations in relational algebra**

We have divided these operations in two categories:

**1. Basic Operations**

**2. Derived Operations**

Basic/Fundamental Operations:

1. Select (σ)
2. Project (∏)
3. Union (∪)

4. Set Difference (-)
5. Cartesian product (X)
6. Rename (ρ)

Derived Operations:

1. Natural Join (⋈)
2. Left, Right, Full outer join (⟕, ⟖, ⟗)
3. Intersection (∩)
4. Division (÷)