DATABASE MANAGEMENT SYSTEM

DBMS CORE GROUP

Authors	Gaurav Kansal, B P Sharma, Pratibha Singh, Nidhi Singh, Anand Kr. Srivastava, Ritin Behl, Sachin Goel
Authorized by	
Creation/Revision Date	May 2021
Version	1.0

ABES

Estd. 2000

UNIT-3
INTRODUCTION TO RELATIONAL MODEL

Table of Contents

3.0 Introduction

After implementing the conceptual model of an application's database, the next step is to develop the logical model of the application database. The logical model is implemented using the Relational Model, which means the model is conceptualized around the mathematical concept of Relations.

The relational model provided a standard way of representing and querying data used by any application. The most fundamental elements in the relational model are **relations**, which users and modern RDBMSs recognize as **tables**. A relation is a set of **tuples or rows**, with each tuple sharing a set of **attributes or columns**.
The other strength of the relational model is to use structured query language (SQL) to write and query data in a relational database. SQL has been widely used as the language for relational database queries.

To ensure that data is always accurate and accessible, relational databases follow certain integrity rules. For example, an integrity rule can specify that duplicate rows are not allowed in a relation to eliminate the potential for erroneous information entering the database.
Organizations of all types and sizes use the simple yet powerful relational model for a wide variety of information needs. A relational database can be considered for any information need in which data points relate to each other and are managed in a secure, rules-based, consistent way.

This chapter will learn the concept of a relational model, terminology of the model, characteristics, constraints, integrity rules, and implementation of relations using SQL.

3.1 Structure of Relational Model

3.1.1 Database Schema

The database schema description is called the **database schema**, which is specified during database design and is not expected to change frequently.

The term "schema" refers to the organization of data as a blueprint of how the database is constructed (divided into database relations in the case of relational databases).

The concept of a relation schema corresponds to the programming-language notion of the type definition. It is convenient to give a name to a relation schema.

The relation schemas of the SMS database (the case study covered in Unit-2) can be shown as follows:
A database schema with primary key and foreign key dependencies can be depicted pictorially by schema diagrams. A schema diagram of the SMS database is shown in

Figure 3.**1**.

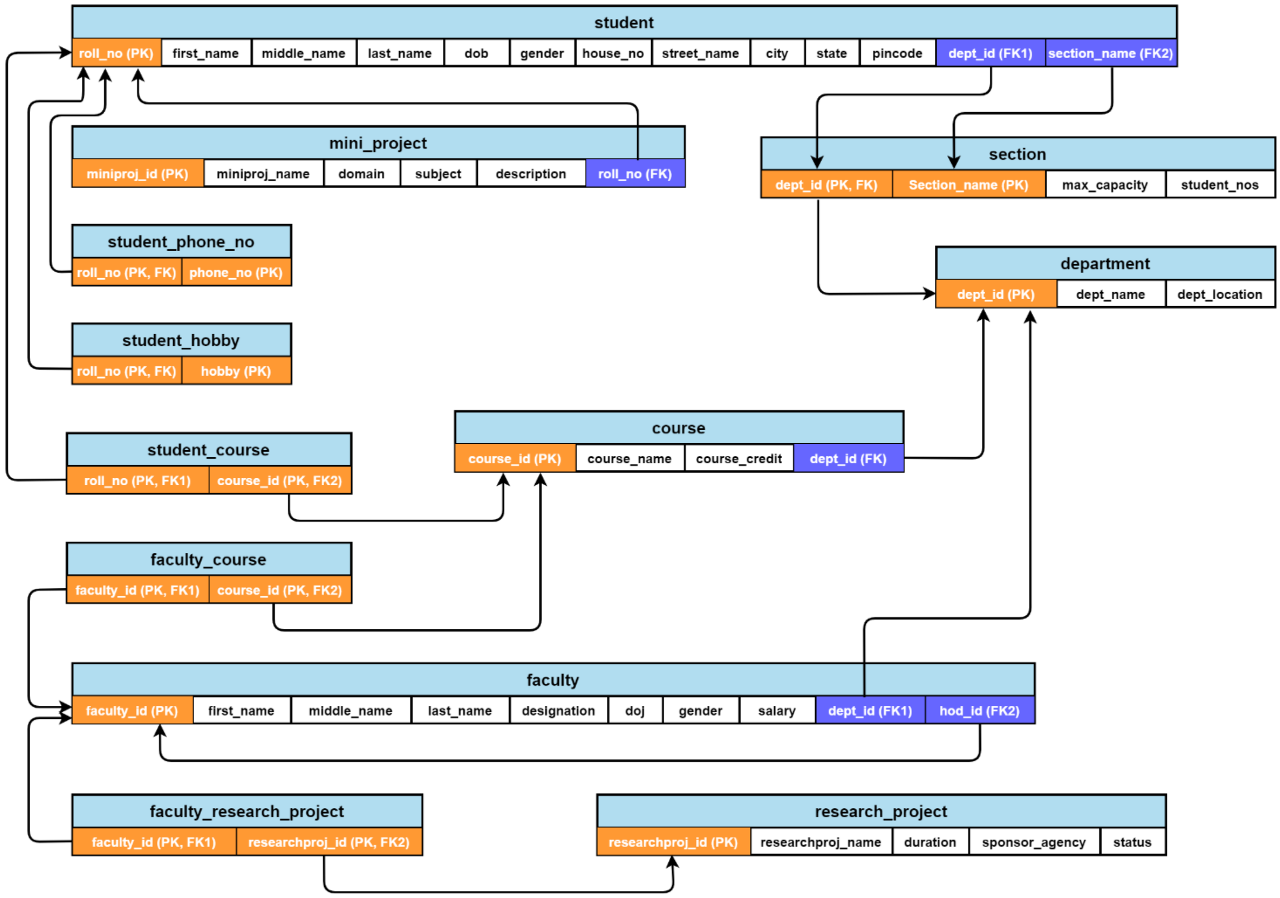


Figure 3.1 Schema diagram for the SMS database

Database Instance is a snapshot of the data in the database at a given instant in time. The data stored in a database at a particular moment is called an **Instance of a Database**.

3.1.2 Relational Model Objects

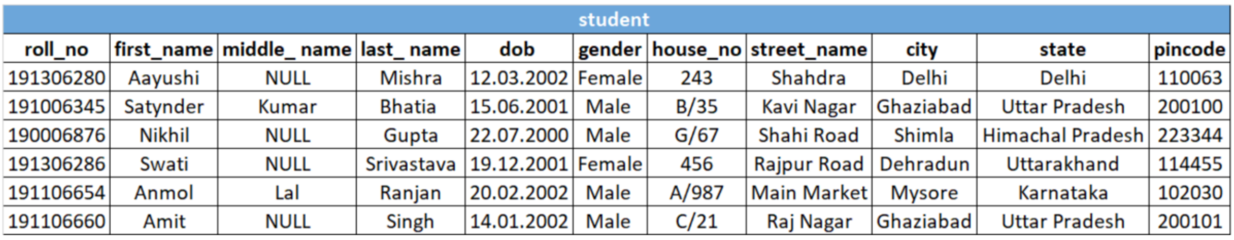
In this section, we will understand few terminologies related to relational model objects as mentioned below.

- Database
- Relation/Table
- Attribute/Field/Column
- Domain
- Record/Row/Tuple
- Degree /Arity

The relational model is best suitable for managing the structured data in a database. It represents the database as a collection of relations. Informally, each relation resembles a table of values. When a relation is thought of as a table of values, each row in the table represents a collection of related data values. A row represents a fact that typically corresponds to a real-world entity or relationship. The table name and column names are used to help to interpret the meaning of the values in each row.

For example, the relation is shown in

Figure 3.**2** is called STUDENT because each row represents facts about a particular student entity. The column names – roll_no, first_name, middle_name, last_name, etc. specify how to interpret the data values in each row.



student										
roll_no	**first_name**	**middle_ name**	**last_ name**	**dob**	**gender**	**house_no**	**street_name**	**city**	**state**	**pincode**
191306280	Aayushi	NULL	Mishra	12.03.2002	Female	243	Shahdra	Delhi	Delhi	110063
191006345	Satynder	Kumar	Bhatia	15.06.2001	Male	B/35	Kavi Nagar	Ghaziabad	Uttar Pradesh	200100
190006876	Nikhil	NULL	Gupta	22.07.2000	Male	G/67	Shahi Road	Shimla	Himachal Pradesh	223344
191306286	Swati	NULL	Srivastava	19.12.2001	Female	456	Rajpur Road	Dehradun	Uttarakhand	114455
191106654	Anmol	Lal	Ranjan	20.02.2002	Male	A/987	Main Market	Mysore	Karnataka	102030
191106660	Amit	NULL	Singh	14.01.2002	Male	C/21	Raj Nagar	Ghaziabad	Uttar Pradesh	200101

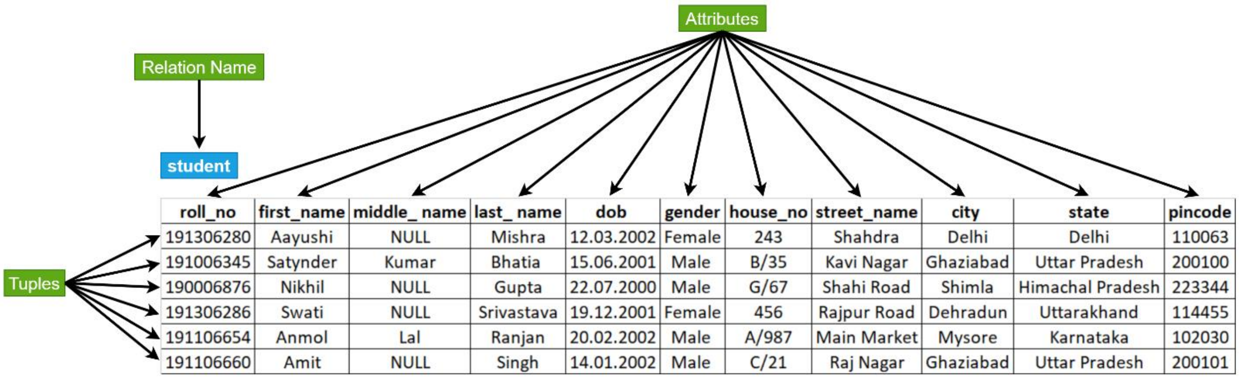
Figure 3.2: Representation of Student relation

In the formal relational model terminology, as shown in

Figure 3.3

Figure 3.**3**, a row is called a **tuple**, a column header is called an **attribute**, and the table is called a **relation**. A domain of possible values represents the data type describing the types of values that can appear in each column. The number of attributes in a relation is called **degree or arity.**

Here relation STUDENT has degree 11 and cardinality 6.



roll_no	first_name	middle_ name	last_ name	dob	gender	house_no	street_name	city	state	pincode
191306280	Aayushi	NULL	Mishra	12.03.2002	Female	243	Shahdra	Delhi	Delhi	110063
191006345	Satynder	Kumar	Bhatia	15.06.2001	Male	B/35	Kavi Nagar	Ghaziabad	Uttar Pradesh	200100
190006876	Nikhil	NULL	Gupta	22.07.2000	Male	G/67	Shahi Road	Shimla	Himachal Pradesh	223344
191306286	Swati	NULL	Srivastava	19.12.2001	Female	456	Rajpur Road	Dehradun	Uttarakhand	114455
191106654	Anmol	Lal	Ranjan	20.02.2002	Male	A/987	Main Market	Mysore	Karnataka	102030
191106660	Amit	NULL	Singh	14.01.2002	Male	C/21	Raj Nagar	Ghaziabad	Uttar Pradesh	200101

Figure 3.3: Representation of Student Table (with attribute and tuples)

In general terminology, the **tuple** is also called a row or record; an **attribute** is called a field or column. A **relation** is called a table. The **domain** defines the data type, format, and possible set of values for a column.

Let us discuss these terms—relation, tuple, domain, and attribute—formally.

A) Relation

A **relation** (or **relation state**) r of the relation schema R(A_1, A_2, … , A_n), also denoted by r(R), is a set of n-tuples r = {t_1, t_2, … , t_m}. Each n-tuple t is an ordered list of n values t =<v_1, v_2, …,v_n>, where each value v_i, 1 ≤ i ≤ n, is an element of dom(A_i) or special NULL value. The ith value in tuple t, which corresponds to the attribute A_i, is referred to as t[A_i] or t.A_i.

For example, as shown in

Figure 3.2, **relation** STUDENT of the relation schema STUDENT (roll_no, first_name, middle_ name, last_ name, dob, gender, house_no, street_name, city, state, pin code) is a set of 5-tuples r = {t_1, t_2, t_3, t_4, t_5}. Second tuple t_2 is an ordered list of 12 values t_2 = <191006345, Satynder, Kumar, Bhatia, 15.06.2001, Male, B/35, Kavi Nagar, Ghaziabad, Uttar Pradesh, 200100>, where a value suppose Kavi Nagar is an element from dom(street_name). The 6th value in tuple t_2 corresponds to the attribute (gender) and can be referred to as t_2 [gender] or t_2.gender.

We display the relation as a table, where each tuple is shown as a row. Each attribute corresponds to a column header indicating a role or interpretation of the values in that column. NULL values represent attributes whose values are unknown or do not exist for some individual STUDENT tuple.

B) Domains

A **domain** is the original set of atomic values used to model data. By **atomic**, we mean that each value in the domain is indivisible. A common method of specifying a domain is to specify a data type, format, and possible set of values from which the data values are drawn.
Some examples of domains and their logical definitions are mentioned in **Error! Reference source not found.**:

Table 3.1 Example of attribute domains

Attribute	Domain Name	Meaning	Domain definition
roll_no	Roll_Numbers	The set of all roll numbers of students	character: size 15
first_name	First_Names	The set of all first names of students	character: size 15
middle_ name	Middle_Names	The set of all middle names of students	character: size 15
last_ name	Last_Names	The set of all last names of students	character: size 15
dob	Dates_of_Birth	The set of all possible values of birth dates of students	date, format dd-mm-yyyy
gender	Genders	The set of all genders	character: size 15
			value: Male or Female
house_no	House_Numbers	The set of all valid house numbers on a street	character: size 25
street_name	Street_Names	The set of all valid street names in a city	character: size 25
city	City_Names	The set of all valid city names in a state	character: size 20
state	State_Names	The set of all valid state names in India	character: size 20
pincode	Pincode_Numbers	The set of all valid Pincode numbers in India	character: size 6

These domains get applied to the attributes to understand what type of data, set of values, and format attributes. Let us now discuss the attributes and how domains are applied to attributes.

C) Attribute

A **relation schema** R, denoted by R(A_1, A_2, …, A_n), comprises a relation name R and a list of attributes, A_1, A_2, …, A_n. Each **attribute** A_i is the name of a role played by some domain D in the relation schema R. D is called the domain of A_i and is denoted by **dom(A_i)**. A relation schema is used to describe a relation; R is called the **name** of this relation. The **degree** of a relation is the number of attributes n of its relation schema.

A relation of degree eleven, which stores information about the students, would contain twelve attributes describing each student as follows:

STUDENT(roll_no, first_name, middle_name, last_name, dob, gender, house_no, street_name, city, state, pincode)

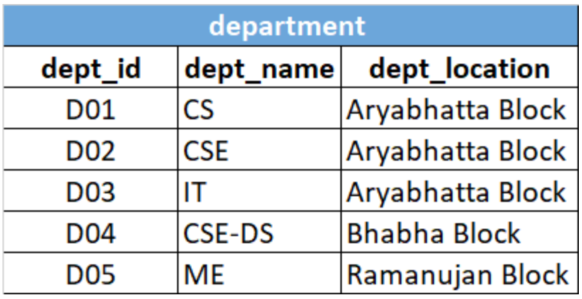
For this relation schema, STUDENT is the name of the relation, which has twelve attributes. In the preceding definition, we showed the assignment of generic types such as string or integer to the attributes. More precisely, we can specify the following previously defined domains for some of the attributes of the STUDENT relation: dom(roll_no): Roll_number; dom(first_name) :Names; dom(middle_name): Names; dom(last_name): Names; dom(dob) =Dates, dom(gender):Genders, dom(house_no): Address, dom(street_name): Address, dom(city): City_names, dom(state): State_names, dom(Pincode): Pincodes

3.1.3 Characteristics of Relations

The earlier definition of relation implies specific characteristics that make a relation different from a file. We will now discuss some of these characteristics.

- Each relation in a database must have a distinct or unique name that would separate it from the other relations in a database, e.g., the relations STUDENT and DEPARTMENT have distinct names.

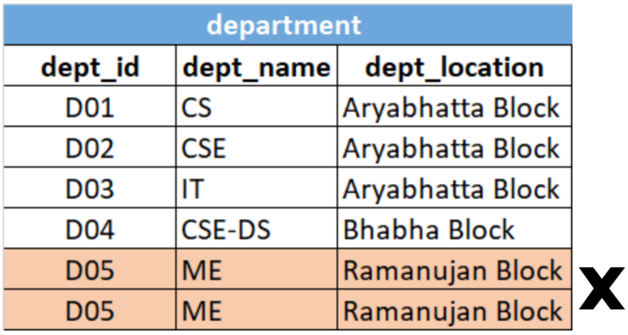
- A relation must not have two attributes with the same name. Each attribute must have a distinct name, e.g., the relation DEPARTMENT has three distinct attributes dept_id, dept_name, and dept_location



department		
dept_id	dept_name	dept_location
D01	CS	Aryabhatta Block
D02	CSE	Aryabhatta Block
D03	IT	Aryabhatta Block
D04	CSE-DS	Bhabha Block
D05	ME	Ramanujan Block

Figure 3.4: Representation of Department Relation (attributes with distinct name)

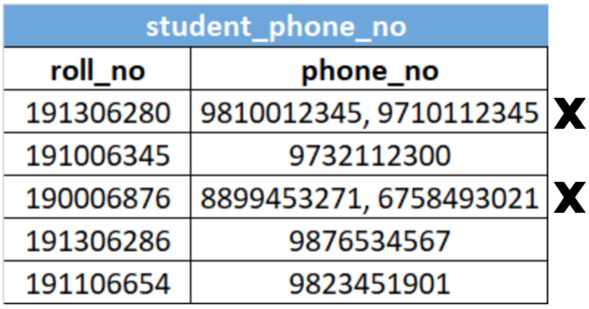
- Duplicate tuples must not be present in a relation.



department			
dept_id	dept_name	dept_location	
D01	CS	Aryabhatta Block	
D02	CSE	Aryabhatta Block	
D03	IT	Aryabhatta Block	
D04	CSE-DS	Bhabha Block	
D05	ME	Ramanujan Block	X
D05	ME	Ramanujan Block	

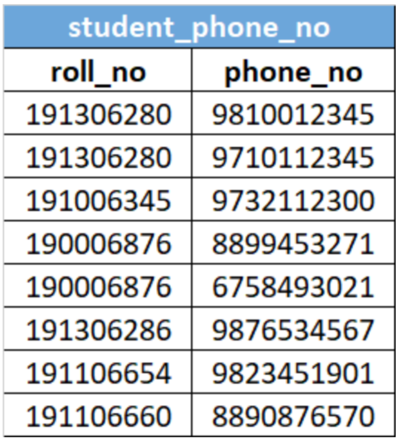
Figure 3.5: Representation of Department Relation (duplicate tuples not allowed)

- Each tuple must have exactly one data (atomic) value for an attribute.



student_phone_no		
roll_no	phone_no	
191306280	9810012345, 9710112345	X
191006345	9732112300	
190006876	8899453271, 6758493021	X
191306286	9876534567	
191106654	9823451901	

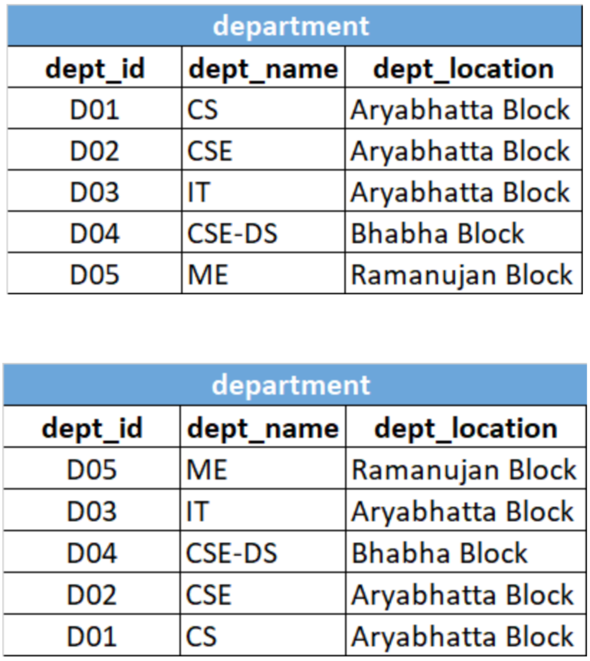
Figure 3. 6: Representation of student_phone_no relation having the multi-valued attribute (not allowed)



student_phone_no	
roll_no	phone_no
191306280	9810012345
191306280	9710112345
191006345	9732112300
190006876	8899453271
190006876	6758493021
191306286	9876534567
191106654	9823451901
191106660	8890876570

Figure 3.7: Representation of Student_phone_number relation having atomic value attribute (allowed)

- Tuples in a relation do not have to follow a significant order as the relation is not order-sensitive. Both of these DEPARTMENT relations are the same.



department		
dept_id	dept_name	dept_location
D01	CS	Aryabhatta Block
D02	CSE	Aryabhatta Block
D03	IT	Aryabhatta Block
D04	CSE-DS	Bhabha Block
D05	ME	Ramanujan Block

department		
dept_id	dept_name	dept_location
D05	ME	Ramanujan Block
D03	IT	Aryabhatta Block
D04	CSE-DS	Bhabha Block
D02	CSE	Aryabhatta Block
D01	CS	Aryabhatta Block

Figure 3.8: Different ways of representing same department relation

Relational Model Notation

We will use the following notation throughout the chapter

- A relation schema R of degree n is denoted by R(A_1, A_2, …, A_n)
- A relation name is denoted by the uppercase letters Q, R, S
- A relation state is denoted by the lowercase letters q, r, s
- The tuples are denoted by the letters t, u, v
- The name of a relation schema such as STUDENT also indicates the current set of tuples in that relation—the current relation state—whereas STUDENT(roll_no, first_name, …) refers only to the relation schema.
- An attribute *A* can be qualified with the relation name *R* to which it belongs by using the dot notation R.A—for example, STUDENT.first_name
- An n-tuple t in a relation r(R) is denoted by t = <v_1, v_2, … ,v_n>, where v_i is the value corresponding to attribute A_i. The following notation refers to component values of tuples:
    - Both t[A_i] and t.A_i refer to the value v_i in t for attribute A_i.
    - Both t[A_u, A_w, …, A_z] and t.(A_u, A_w, …, A_z), where A_u, A_w, …, A_z is a list of attributes from relation R, refer to the sub-tuple of values <v_u, v_w, …, v_z> from t corresponding to the attributes specified in the list.

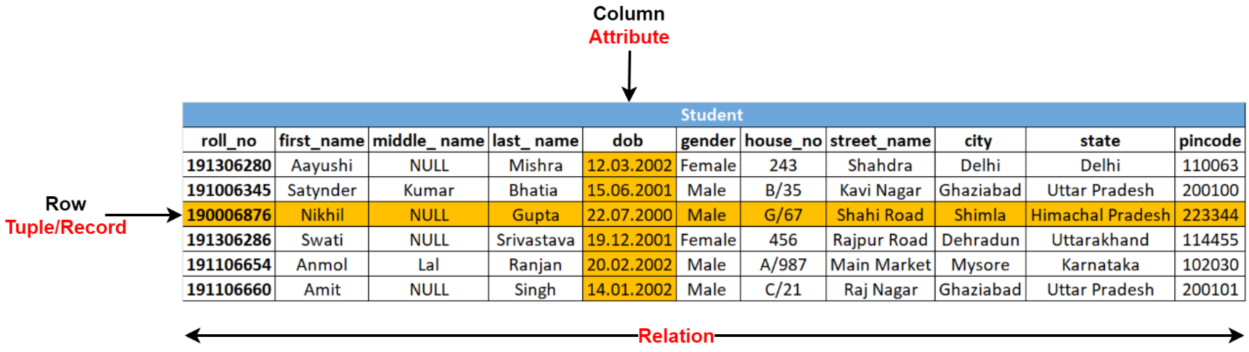
As an example, consider the tuple t = <'Anmol','Lal','Ranjan','20-02-2002','Male','A/987', 'Main Market','Mysore','Karnataka','102030'> from the STUDENT relation in **Error! Reference source not found.**; we have t[first_name] = <‘Anmol’>, and t[first_name, middle_name, last_name] = <'Anmol','Lal','Ranjan'>

3.2 Keys in a Relational Database

The need for database keys

Each relation in the database has rows (tuples) and columns. Columns represent the attributes of the relation, and each row (tuple) represents an instance of the relation containing a valid combination of the attributes of the particular relation.

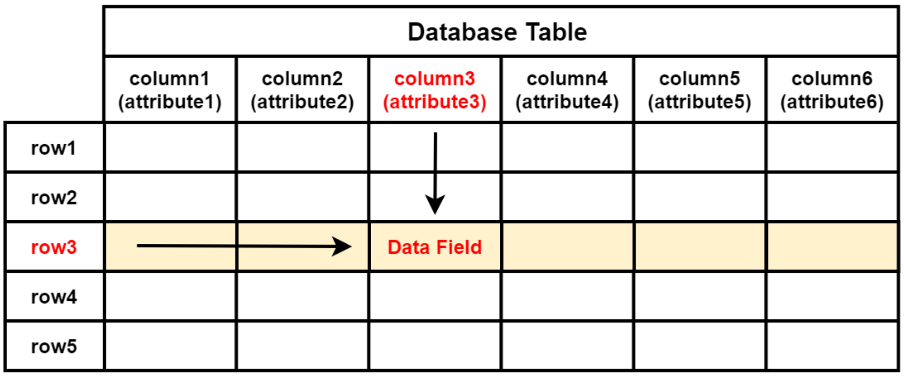
For example, each row (tuple) contains the value of attributes (columns) for one specific student in the student relation.


Column
Attribute
Row
Tuple/Record
Student
Relation

roll_no	first_name	middle_ name	last_ name	dob	gender	house_no	street_name	city	state	pincode
191306280	Aayushi	NULL	Mishra	12.03.2002	Female	243	Shahdra	Delhi	Delhi	110063
191006345	Satynder	Kumar	Bhatia	15.06.2001	Male	B/35	Kavi Nagar	Ghaziabad	Uttar Pradesh	200100
190006876	Nikhil	NULL	Gupta	22.07.2000	Male	G/67	Shahi Road	Shimla	Himachal Pradesh	223344
191306286	Swati	NULL	Srivastava	19.12.2001	Female	456	Rajpur Road	Dehradun	Uttarakhand	114455
191106654	Anmol	Lal	Ranjan	20.02.2002	Male	A/987	Main Market	Mysore	Karnataka	102030
191106660	Amit	NULL	Singh	14.01.2002	Male	C/21	Raj Nagar	Ghaziabad	Uttar Pradesh	200101

Figure 3.9: Representation of a Relation Table (Student)

In a database relation, the data is stored (logically) into the data field, an intersection of relation row and column. Each data element in a relation can be accessed with reference to the specific row and column.



	column1 (attribute1)	column2 (attribute2)	column3 (attribute3)	column4 (attribute4)	column5 (attribute5)	column6 (attribute6)
row1						
row2						
row3			Data Field			
row4						
row5						

Figure 3.10: Logical representation of data stored in a relation

In a database relation, it is easy to identify a particular (specific) column; however, we need a mechanism (way) to identify a particular (specific) row in a database relation. For this, the concept of database keys is used so that we can identify each row uniquely.

A database key (database relation key) is an attribute or a set of attributes in a particular relation that can uniquely identify each row (tuple) in that relation.

Database keys have three primary objectives:

1. Ensure each row (tuple) in the database relation is uniquely identified.
2. Establish and enforce various types of integrity within the database.
3. Help establish the relationship between relations.

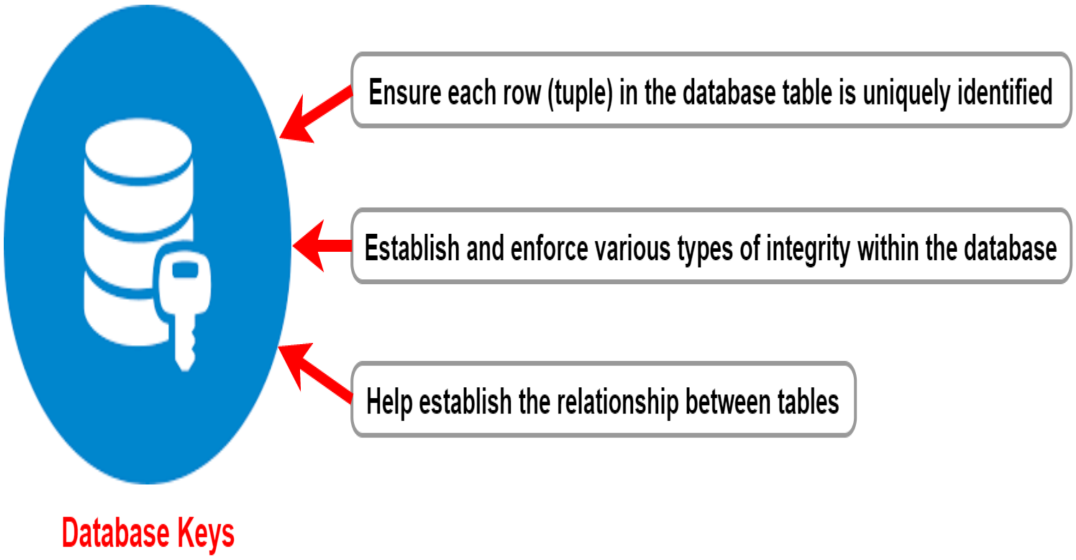

Ensure each row (tuple) in the database table is uniquely identified
Establish and enforce various types of integrity within the database
Help establish the relationship between tables
Database Keys

Figure 3.11: Significance of Database Keys

Type of database keys

There are mainly seven different types of Keys in DBMS and each key has its different functionality:

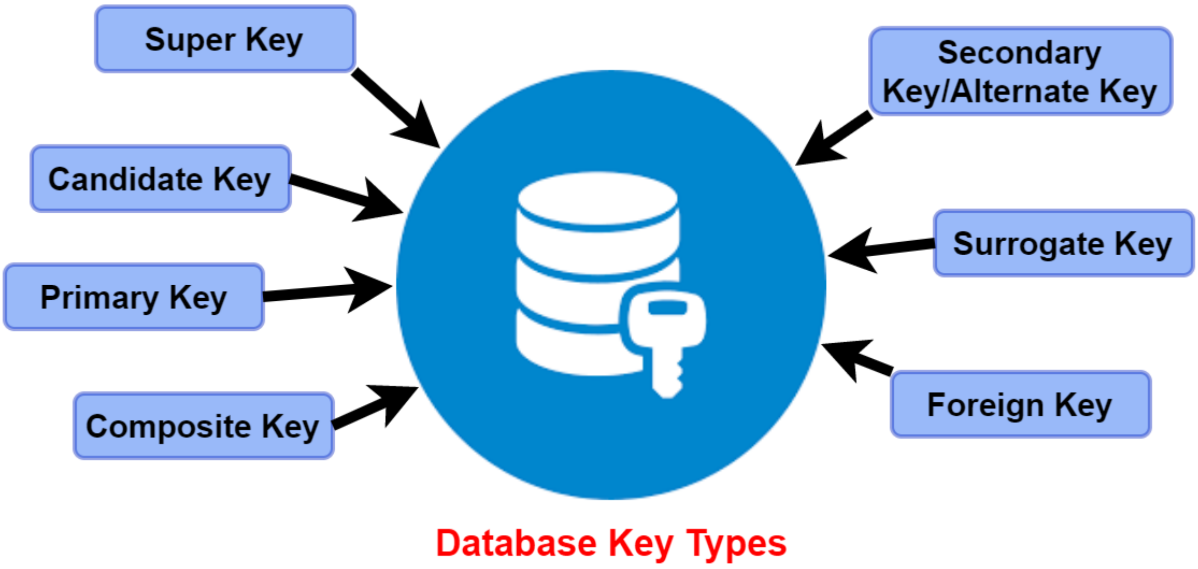


Figure 3.12: Different types of Database Keys

3.2.1 Super Key

A super key (SK) is a set of one or more attributes that allows us to uniquely identify a row (tuple) in the relation.

SK is a set of attributes of a relation schema R with the property that no two tuples t_1 and t_2 in any relation state r of R should have the same combination of values for these attributes. This is known as uniqueness property.
Thus, we can state that:

$$t_1[SK] \neq t_2[SK] \text{ or } t_1.SK \neq t_2.SK$$

For example, two attribute set (roll_no, first_name, last_name)[1] and (roll_no)[2] are both super keys of STUDENT relation because no two student tuples can have the same value for these set of attributes, or we can say that t_1[roll_no, first_name, last_name] ≠ t_2[roll_no, first_name, last_name] similarly, t_1[roll_no] ≠ t_2[roll_no].

A relation can have many super keys. Consider the STUDENT schema below to understand this:
STUDENT (adhaar_no, roll_no, first_name, last_name, address, dob, gender, mobile_no, email_id).

Below are some sets of super keys for the above STUDENT schema since each set, i.e., SK(i), uniquely identifies a unique student in the Student relation.
SK1 = (adhaar_no)
SK2 = (roll_no)
SK3 = (mobile_no)
SK4 = (email_id)
SK5 = (adhaar_no, first_name, last_name)
SK6 = (roll_no, first_name, last_name)
SK7 = (first_name, last_name, address, mobile_no)
SK8 = (roll_no, first_name, gender)
SK9 = (first_name, last_name, address, email_id)
SK10 = (first_name, mobile_no, email_id, dob) etc.

A combination of all the attributes in a super key can identify each tuple uniquely in the given relation; however, all of them may not be necessary to establish the uniqueness of a row (tuple).
Maximum no. of possible super keys for a relation with n attributes = $\mathbf{2^n-1}$

Some facts to remember about super keys:

- A super key specifies uniqueness property for a row (tuple) i.e. t_1 [SK] ≠ t_2 [SK] .

- Every relation has one default super key, i.e., the set of all attributes of that relation. In STUDENT relation { adhaar_no, roll_no, first_name, last_name, address,dob, gender, mobile_no, email_id } is a default super key.
- A super key can have extraneous attribute/s. For super key SK2 i.e. {roll_no, first_name, last_name}, first_name and last_name are extraneous attributes. If we remove these attributes, then also uniqueness property holds.
- If SK is a super key, then any superset of SK will also be a super key. For example, Roll_no is the super key (SK) of the STUDENT relation. If we add some extra attributes to make a superset of SK as (roll_no, first_name, gender), then no two tuples can have the same value with this new set of attributes. Hence, a superset of (roll_no), i.e. (roll_no, first_name, gender), is also a super key.

3.2.2 Candidate Key

A candidate key is a minimal super key that can identify each row (tuple) of a given relation uniquely. It means that a candidate key is a super key from which we cannot remove any attributes and still have the uniqueness property hold.

In other words, a candidate key is a super key without any redundant attribute. A candidate key has a minimum number of attributes that are definitely needed to establish a row (tuple) uniqueness in a given relation.
We can also say that the candidate key is a super key of which no proper subset exists as a super key.

What are Subsets and Proper Subset?
If A and B are two sets, and every element of set A is also an element of set B, then A is called a subset of B. Subset is denoted by '⊆', so A ⊆ B. The definition of "subset" allows the possibility that the first set is the same as (equal to) the second set.

However, A proper subset of a set A is a subset of A that is not equal to A. In other words, if B is a proper subset of A, then all elements of B are in A, but A contains at least one element that is not in B.

For example, if A={1,3,5} then B={1,5} is a proper subset of A. The set C={1,3,5} is a subset of A, but it is not a proper subset of Asince C=A. Proper Subset is denoted by the symbol '⊂'. So as per the above example, B ⊂ A and C ⊆ A.

Given below are the examples of candidate keys or minimal super keys since each set consists of minimal attributes required to identify each student uniquely in the Student relation:
CK1 = (adhaar_no)
CK2 = (roll_no)

CK3 = (first_name, last_name, address)
CK4 = (email_id)

Therefore, the candidate keys obtained above examples are the minimal set of super keys. **The set of all possible candidate keys is known as the candidate key set.**
Hence **{roll_no, adhaar_no, email_id, (first_name, last_name, address)}** is a possible set of candidate keys.

Some facts to remember about candidate keys:-

- All the attributes in a candidate key are sufficient as well as necessary to identify each tuple uniquely.
- Removing any attribute from the candidate key fails in identifying each tuple uniquely.
- The value of the candidate key must always be unique.
- The value of the candidate key can never be NULL.
- It is possible to have multiple candidate keys in a relation.
- Those attributes which appear in any candidate key are called **prime attributes**.
- A candidate key is a super key but not vice versa.

3.2.3 Primary Key

We now know that a relation can have many candidate keys, which can uniquely identify each row (tuple) of a given relation. Any one of these candidate keys can be chosen as a Primary key (PK) of a given relation.
The database designer chooses any one of the candidate keys as the primary key of the relation.

The primary key should be chosen such that:

1. Its attributes are never, or very rarely, changed , i.e., time-invariant. For instance, the adhaar_no, roll_no, email_id, and (first_name, last_name, address) are the candidate keys of the student relation, out of which email_id and address are time-variant. On the other hand, adhaar_no & roll_no are guaranteed never to change.
2. It should consist of either a single attribute or a small number of attributes.

Hence, a database designer can choose either **adhaar_no** or **roll_no** as a primary key for the student relation satisfying the above two conditions.

Some facts to remember about primary keys:-

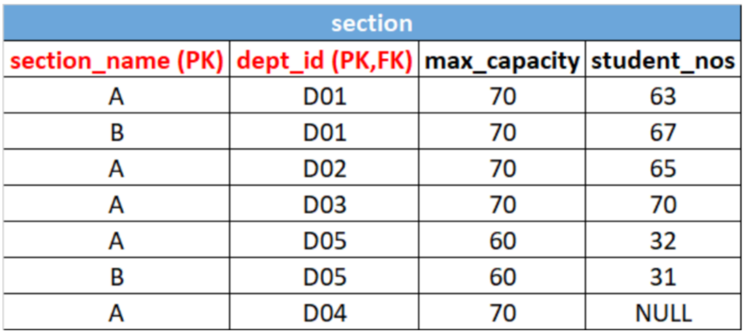
- The value of the primary key field will always be unique
- The value of a primary key field can never be NULL

- A relation is allowed to have only one primary key

3.2.4 Composite key

A composite key is a primary key consisting of two or more columns that uniquely identify rows in a relation. The combination of columns guarantees uniqueness, though individually uniqueness is not guaranteed. Hence, they are combined to identify records in a relation uniquely.

Example:



section			
section_name (PK)	dept_id (PK,FK)	max_capacity	student_nos
A	D01	70	63
B	D01	70	67
A	D02	70	65
A	D03	70	70
A	D05	60	32
B	D05	60	31
A	D04	70	NULL

Figure 3.13: Representation of Section Table

Figure 3.**13**, (section_name) and (dept_id) individually cannot be a primary key as it does not uniquely identify a record. However, a composite key consisting of (section_name) and (dept_id) could be used to identify each row (tuple) uniquely.

3.2.5 Secondary Key or Alternate key

Secondary or Alternate key (s) are the candidate keys that are left unimplemented or unused after implementing the primary key.

If in the student relation, (roll_no) is designated as a primary key, the remaining candidate keys – (adhaar_no), (email_id), and (first_name, last_name, address) are the secondary keys or alternate keys of the student relation.

The secondary keys are used for data retrieval purposes and indexing. Indexing is done for better and faster data searching and retrieval from the database.

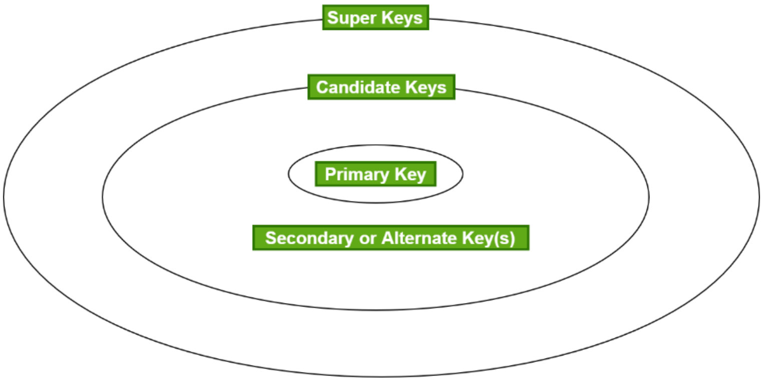


Figure 3.14: Representation of different keys in a Venn diagram

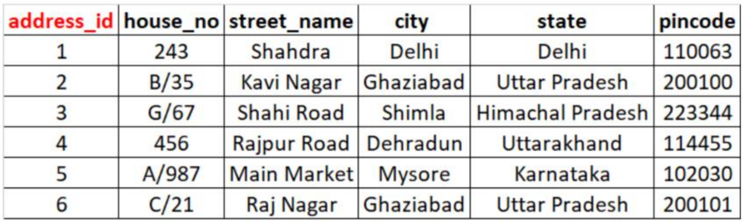
3.2.6 Surrogate Keys

A surrogate key is a system-generated value with no business meaning that is used to identify a record in a relation uniquely. This key is used as a primary key in a given relation when a natural primary key is not available. The surrogate key is usually an integer and thus does not lend any meaning to the data in the relation. A surrogate key is a value generated right before the record is inserted into a relation.

When the primary key is too big or complicated, Surrogate keys are preferred. A surrogate key is generated for each unique combination of the primary key.

Example:

Suppose, in a relation; we have a key comprising of a combination of various components of address (house_no, street_name, city, state, Pincode). We might want to introduce a surrogate key address_id for each unique combination of address (house_no, street_name, city, state, Pincode).



address_id	house_no	street_name	city	state	pincode
1	243	Shahdra	Delhi	Delhi	110063
2	B/35	Kavi Nagar	Ghaziabad	Uttar Pradesh	200100
3	G/67	Shahi Road	Shimla	Himachal Pradesh	223344
4	456	Rajpur Road	Dehradun	Uttarakhand	114455
5	A/987	Main Market	Mysore	Karnataka	102030
6	C/21	Raj Nagar	Ghaziabad	Uttar Pradesh	200101

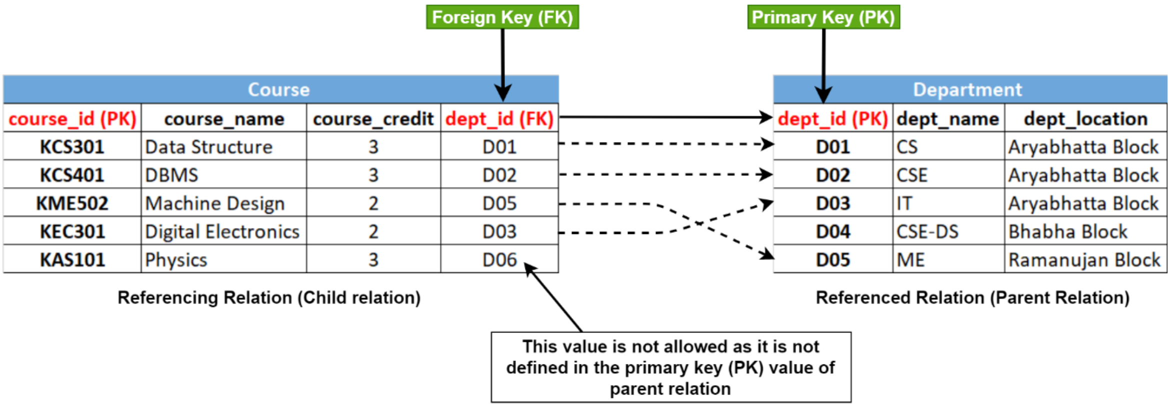
Figure 3.15: Usage of Surrogate keys in a relation

3.2.7 Foreign Key

The primary purpose of the foreign key (FK) is to define and create a relationship between the two relations. When the primary key (PK) of one relation (also referred to as parent relation) is included as a non-unique attribute in another relation (also referred to as child relation), then such a database key is called a foreign key (FK). Please note that there is no uniqueness constraint for a foreign key (FK).

The relation which is being referenced is called referenced relation, and the corresponding attribute is called a referenced attribute. The relation which refers to the referenced relation is called referencing relation, and the corresponding attribute is called referencing attribute.

For example, attribute (dept_id) in course relation, as shown in Figure 3.16 is a foreign key (FK), and all values for this attribute will match the values of the primary key (PK) attribute (dept_id) in the department relation.



Course

course_id (PK)	course_name	course_credit	dept_id (FK)
KCS301	Data Structure	3	D01
KCS401	DBMS	3	D02
KME502	Machine Design	2	D05
KEC301	Digital Electronics	2	D03
KAS101	Physics	3	D06

Referencing Relation (Child relation)

Department

dept_id (PK)	dept_name	dept_location
D01	CS	Aryabhatta Block
D02	CSE	Aryabhatta Block
D03	IT	Aryabhatta Block
D04	CSE-DS	Bhabha Block
D05	ME	Ramanujan Block

Referenced Relation (Parent Relation)

This value is not allowed as it is not defined in the primary key (PK) value of parent relation

Figure 3.16: Representation of Foreign Key in a Course Table with referenced to Department Table

Some facts to remember about foreign keys:-

- The foreign key of a relation references the primary key of the other relation.
- A foreign key can take only those values which are present in the primary key of the referenced relation.
- A foreign key may have a name other than that of a primary key.
- A foreign key can take the NULL value.
- There is no restriction on a foreign key to be unique. In fact, the foreign key is not unique most of the time.
- Referenced relation may also be called the parent relation or primary relation. Referencing relation may also be called the child relation.
- There may be more than one foreign key in a relation.

3.2.8 Unique Key

Unique key constraints also identifies an individual tuple uniquely in a relation or table. Multiple unique keys can present in a table. NULL values are allowed in case of a unique key. These can also be used as foreign keys for another table. It can be used when someone wants to enforce unique constraints on a column and a group of columns which is not a primary key.

For an example in a Department relation dept_name is a unique key. In another example of Course relation, course_name is a unique key.

Difference between Primary Key and Unique Key

Primary Key	**Unique Key**
Unique identifier for rows of a table	Unique identifier for rows of a table when primary key is not present
Cannot accept NULL values	Can accept NULL
Only one primary key can be present in a table	Multiple Unique Keys can be present in a table
Selection using primary key creates clustered index (the order of the rows does not match the physical order of the actual data)	Selection using unique key creates non-clustered index (the order of the rows does not match the physical order of the actual data)

3.3 Constraints

A relational database is a collection of relations. These relations store data about a business or a business process. Functional users or business managers define the rules that must be applied to this data before storing it in the database. For example, in our SMS case study,

- Each student must have a roll number which must be unique in the records maintaining student data.
- No student in a department can work on more than one mini-project.
- A student can belong to only one department, etc.

There will typically be many relations in a relational database, and the tuples in those relations are usually related in various ways. The state of the whole database will correspond to the states of all its relations at a particular point in time. There are generally many rules or restrictions on the actual values in a database state. We need to enforce these rules on the data values stored in tuples for a particular column. The data that satisfies these rules will be stored in the respective column of a relation; otherwise, it will be rejected.

The business rules or restrictions which are enforced on data being stored in a relation are called Constraints. These constraints ensure that data stored in relations will be valid.

Note: Even if a single column of the record being entered into the relation fails a constraint, then the entire record is rejected and not stored in a relation.

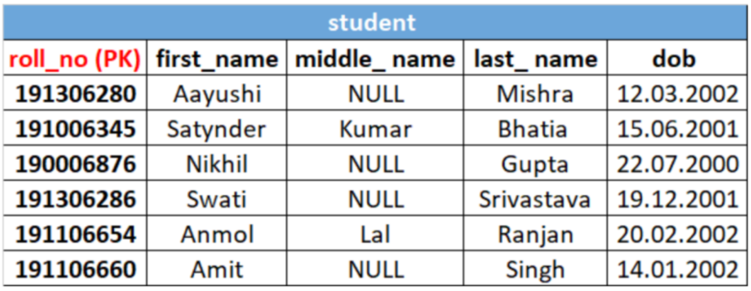
We will now discuss the various restrictions on data specified on a relational database in the form of constraints. Constraints on databases are divided into three main categories:

1. **Inherent model-based constraints or implicit constraints**

   The constraints that are implicit in a data model are called inherent model-based constraints. These constraints are:

   - **Ordering of tuples in a relation**: A relation is defined as a set of tuples. As mathematically, elements of a set have no order among them; hence, tuples in a relation do not have any particular order. In other words, a relation is not sensitive to the ordering of tuples.

   - **Values and NULLs in the Tuples**: Each value in a tuple is an atomic value; that is, it is not divisible into components within the framework of the basic relational model. Hence, multi-valued attributes are not allowed. An important concept is that of NULL values, which represent the values of attributes that may be unknown or may not apply to a tuple. A special value, called NULL, is used in these cases. For example, if a student does not have a middle_name in student relation, it will be represented by NULL.



student				
roll_no (PK)	first_name	middle_ name	last_ name	dob
191306280	Aayushi	NULL	Mishra	12.03.2002
191006345	Satynder	Kumar	Bhatia	15.06.2001
190006876	Nikhil	NULL	Gupta	22.07.2000
191306286	Swati	NULL	Srivastava	19.12.2001
191106654	Anmol	Lal	Ranjan	20.02.2002
191106660	Amit	NULL	Singh	14.01.2002

Figure 3.17: Representation of NULL value in student relation

2. **Schema-based constraints or explicit constraints or integrity constraints:**

   The constraints that can be directly expressed in the schemas of the relational model by specifying them in the Data Definition Language (DDL) are schema-based. These are further categorized as:

   - Entity integrity constraint or Key constraint
   - Referential integrity constraint
   - Domain Integrity constraints
     - Check constraint
     - Unique constraint
     - Null constraint
     - Default constraint

3. **Application-based constraints:**

   Constraints that cannot be directly expressed in the schemas of the data model and hence must be expressed and enforced by the application programs are called application-based constraints. For example, a student can take a maximum of 25 credits in a semester.

The constraints that we will discuss in this section are schema-based constraints that can be expressed in a schema of the relational model via the DDL statements.

3.3.1 Schema Based Constraints or Integrity Constraints

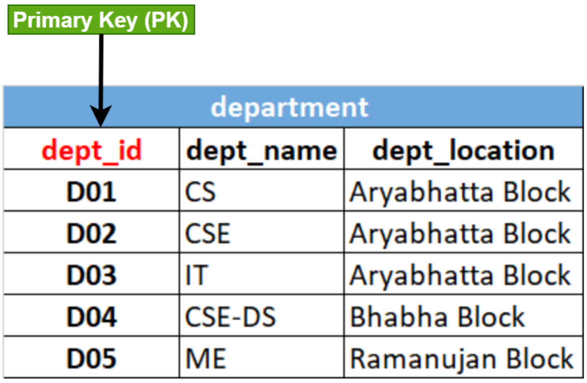
Integrity constraints ensure that changes made to the database by authorized users do not result in a loss of data consistency. Thus, integrity constraints guard against accidental damage to the database. Let us discuss various types of integrity constraints.

3.3.1.1 Entity Integrity Constraint or Key Constraint

Entity integrity constraint is a major component of overall data integrity, and it ensures that there are no duplicate tuples in a relation. Entity integrity constraint is based on the primary key (PK). The primary key of a relation ensures there are no duplicate tuples in a relation. The value of a primary key is unique and not null.

For example, in

Figure 3.**18** below, (dept_id) is the primary key of department relation. (Dept_id) will be unique and not null for all tuples of the department relation.


Primary Key (PK)

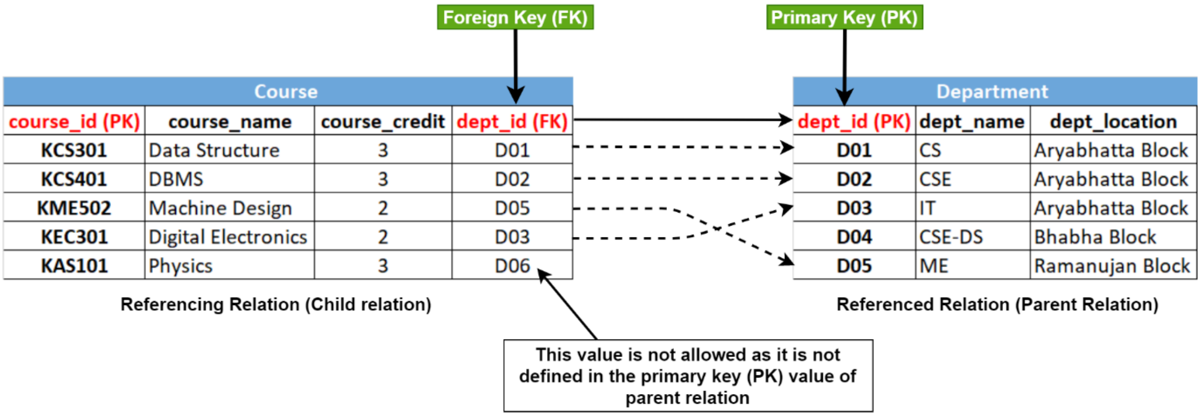
department		
dept_id	dept_name	dept_location
D01	CS	Aryabhatta Block
D02	CSE	Aryabhatta Block
D03	IT	Aryabhatta Block
D04	CSE-DS	Bhabha Block
D05	ME	Ramanujan Block

Figure 3.18: Representation of Primary Key (dept_id) in Department Relation (Entity Integrity Constraint)

3.3.1.2 Referential Integrity Constraints

Referential Integrity constraints in DBMS are based on the concept of Foreign Keys. The referential integrity constraint says, that if a relation R2 has a foreign key attribute (FK) matching the primary key attribute (PK) of other relation R1, then every value of FK in R2 must either be equal to the value of PK in some tuple of R1 or the FK value must be null. R1 and R2 are not necessarily distinct (in the case of a recursive relationship).

For example, as shown in Figure 3.19, the department id attribute in the Course and department must be the same. Attribute (dept_id) in the course relation is referring to the primary key attribute (dept_id) of department relation. Thus the Course is called the referencing relation, and the department is the referenced relation. All values in (dept_id) of Course, relation satisfy the referential integrity constraint except the value D06 as it is not present in (dept_id) attribute in department relation. **Referential integrity constraint will not allow D06 as a value for the foreign key (dept_id) in the course relation.**



Course			
course_id (PK)	course_name	course_credit	dept_id (FK)
KCS301	Data Structure	3	D01
KCS401	DBMS	3	D02
KME502	Machine Design	2	D05
KEC301	Digital Electronics	2	D03
KAS101	Physics	3	D06

Referencing Relation (Child relation)

Department		
dept_id (PK)	dept_name	dept_location
D01	CS	Aryabhatta Block
D02	CSE	Aryabhatta Block
D03	IT	Aryabhatta Block
D04	CSE-DS	Bhabha Block
D05	ME	Ramanujan Block

Referenced Relation (Parent Relation)

This value is not allowed as it is not defined in the primary key (PK) value of parent relation

Figure 3.19: Representation of Foreign Key (dept_id) in Course Relation (Referential Integrity Constraint)

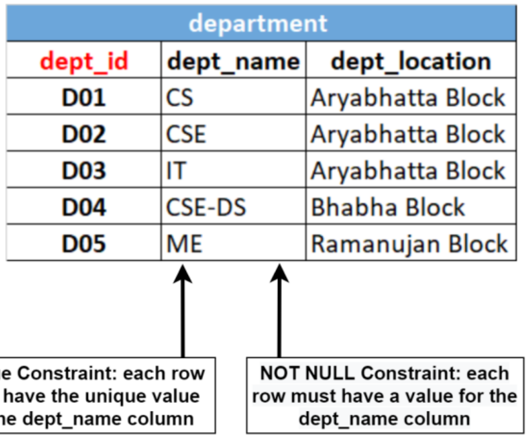
3.3.1.3 Domain constraint

Domain constraints specify the set of possible values that may be associated with an attribute. **A common method of specifying a domain for an attribute is to specify a data type from which the data values forming the domain are drawn**. Below are some of the most commonly used domain constraints.

A. Null constraint

It specifies whether null values are permitted for an attribute. NOT NULL constraint makes sure that a column does not hold NULL value. When we do not provide a value for a particular attribute (column) while inserting a record into a relation, it takes NULL value by default. By specifying the NOT NULL constraint, we can ensure that a particular column cannot have NULL values.

For example, if every department must have a valid, non-null value for the (dept_name) attribute, then the (dept_name) attribute for the department relation is constrained to be NOT NULL. As shown in Figure 3.20, each tuple of department relation must have the value for the (dept_name) column as it cannot be left empty.



department		
dept_id	dept_name	dept_location
D01	CS	Aryabhatta Block
D02	CSE	Aryabhatta Block
D03	IT	Aryabhatta Block
D04	CSE-DS	Bhabha Block
D05	ME	Ramanujan Block

Unique Constraint: each row must have the unique value for the dept_name column

NOT NULL Constraint: each row must have a value for the dept_name column

Figure 3.20: Representation of NULL constraint in Department Relation

B. Unique constraint

A *unique constraint* (also referred to as a *unique key constraint*) is a rule that forbids duplicate values in one or more columns within a relation. This constraint is used to implement secondary or alternate keys.
However, attributes declared as unique are permitted to have null values unless they have been explicitly declared NOT NULL.
For example, as shown in

Figure 3.**20**, each row of Department relation must have the unique value for the (dept_name) column.

C. Check constraint

A check constraint is defined on a column for specifying the range of values that can be inserted into it, using a predefined condition. When this constraint is set on a column, it ensures that the specified column must have the value falling in the specified range.

Example 1 - We understand that the salary of the faculty cannot be a negative value; it must always be greater than 0. So, in faculty relation (salary), the salary of the faculty will have a check constraint, i.e., the constraint will ensure that the value of (salary) will be greater than zero always.

Example 2 - In student relation, the gender can be Female, Male, or Trans. So, (gender) will have a check constraint that will ensure that the value inserted in each row must be any one of (Female, Male, or Trans).

D. Default constraint

The default constraint is defined to provide a default value to a column if no other value is provided while inserting a new record.

For example, the default value for the (status) column of Research Project relation can be taken as "Ongoing".

3.4 Case Study Design with Complete Constraints

3.4.1 Data Dictionary

3.4.1.1 Introduction to Data Dictionary

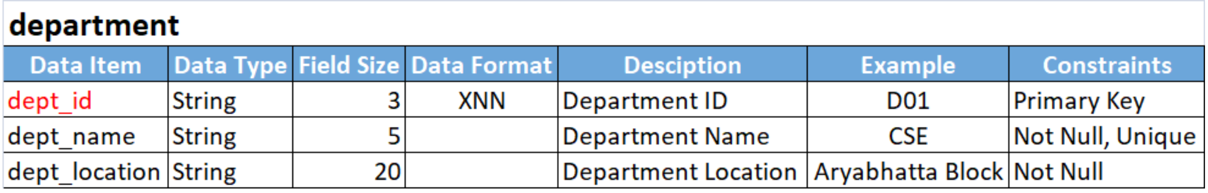
A data dictionary is a set of information used by database administrators to describe the structure of a relational database by defining the data items (attributes/column name), their types, format, constraints, and relationship between the relational tables.
A data dictionary contains ***data about the relational tables,*** which actually contains the data. It is also called "data about data" or metadata.

Data Dictionary provides the metadata about each relational table in the database, e.g., data item or column name, suggested data type for that column, maximum allowed column width, data format to be stored in on the disc, constraints applied on column level and relational table level, a brief description of the column and example representing actual data item to be filled in the column.

Since the composite keys are implemented on table level rather than column level, they are described as separate column UniqueID.

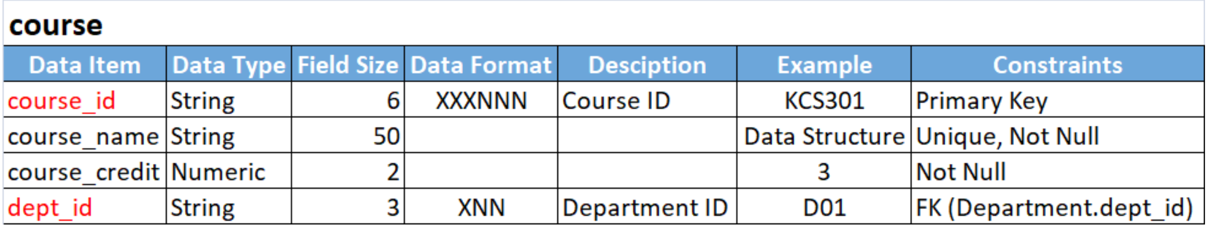
The data dictionary of all the relational tables in the case study is provided below, which will help us while implementing the actual database operations.

3.4.1.2 Creating a Data Dictionary



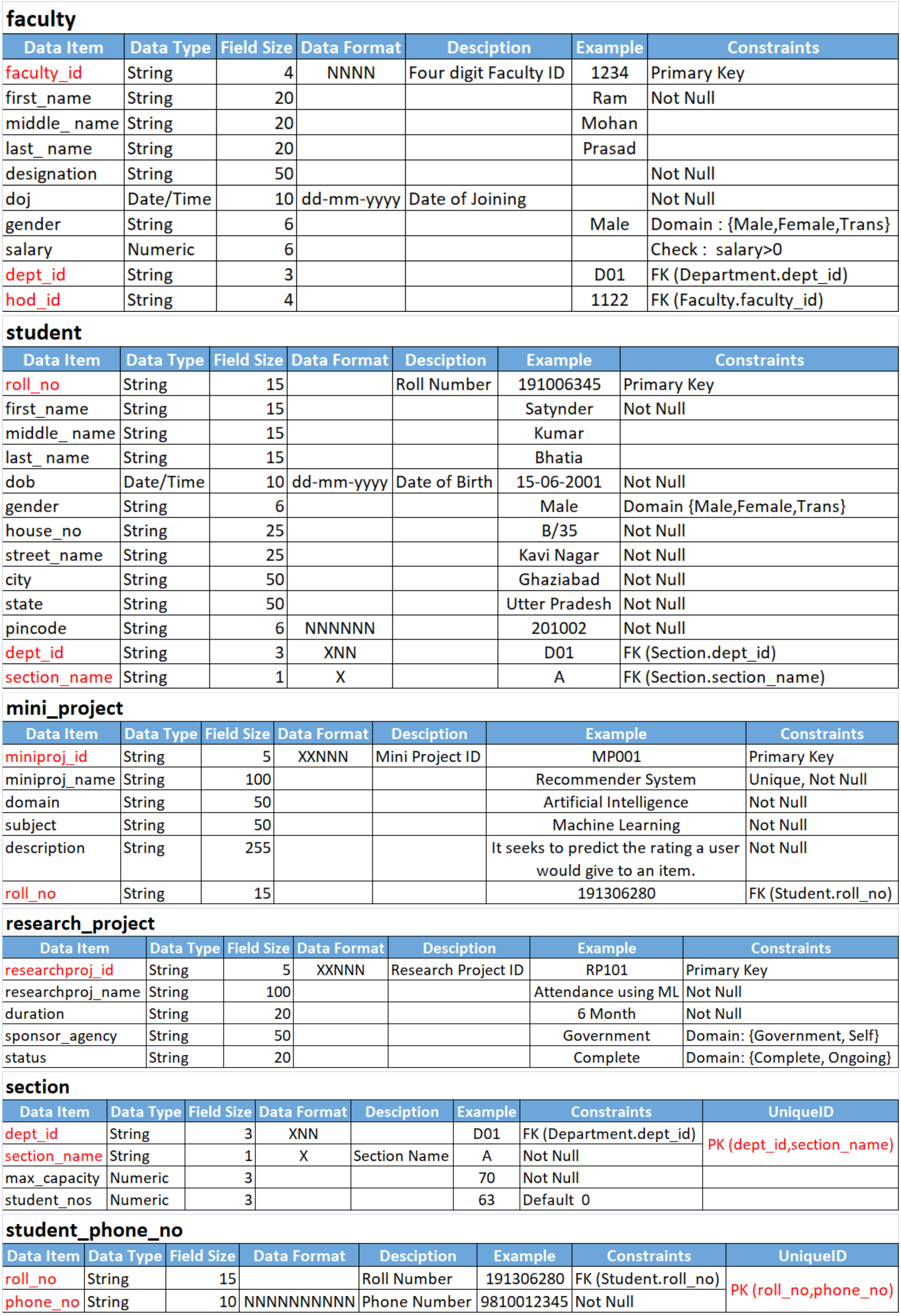
department

Data Item	Data Type	Field Size	Data Format	Desciption	Example	Constraints
dept_id	String	3	XNN	Department ID	D01	Primary Key
dept_name	String	5		Department Name	CSE	Not Null, Unique
dept_location	String	20		Department Location	Aryabhatta Block	Not Null



course

Data Item	Data Type	Field Size	Data Format	Desciption	Example	Constraints
course_id	String	6	XXXNNN	Course ID	KCS301	Primary Key
course_name	String	50			Data Structure	Unique, Not Null
course_credit	Numeric	2			3	Not Null
dept_id	String	3	XNN	Department ID	D01	FK (Department.dept_id)



faculty

Data Item	Data Type	Field Size	Data Format	Desciption	Example	Constraints
faculty_id	String	4	NNNN	Four digit Faculty ID	1234	Primary Key
first_name	String	20			Ram	Not Null
middle_ name	String	20			Mohan	
last_ name	String	20			Prasad	
designation	String	50				Not Null
doj	Date/Time	10	dd-mm-yyyy	Date of Joining		Not Null
gender	String	6			Male	Domain : {Male,Female,Trans}
salary	Numeric	6				Check : salary>0
dept_id	String	3			D01	FK (Department.dept_id)
hod_id	String	4			1122	FK (Faculty.faculty_id)

student

Data Item	Data Type	Field Size	Data Format	Desciption	Example	Constraints
roll_no	String	15		Roll Number	191006345	Primary Key
first_name	String	15			Satynder	Not Null
middle_ name	String	15			Kumar	
last_ name	String	15			Bhatia	
dob	Date/Time	10	dd-mm-yyyy	Date of Birth	15-06-2001	Not Null
gender	String	6			Male	Domain {Male,Female,Trans}
house_no	String	25			B/35	Not Null
street_name	String	25			Kavi Nagar	Not Null
city	String	50			Ghaziabad	Not Null
state	String	50			Utter Pradesh	Not Null
pincode	String	6	NNNNNN		201002	Not Null
dept_id	String	3	XNN		D01	FK (Section.dept_id)
section_name	String	1	X		A	FK (Section.section_name)

mini_project

Data Item	Data Type	Field Size	Data Format	Desciption	Example	Constraints
miniproj_id	String	5	XXNNN	Mini Project ID	MP001	Primary Key
miniproj_name	String	100			Recommender System	Unique, Not Null
domain	String	50			Artificial Intelligence	Not Null
subject	String	50			Machine Learning	Not Null
description	String	255			It seeks to predict the rating a user would give to an item.	Not Null
roll_no	String	15			191306280	FK (Student.roll_no)

research_project

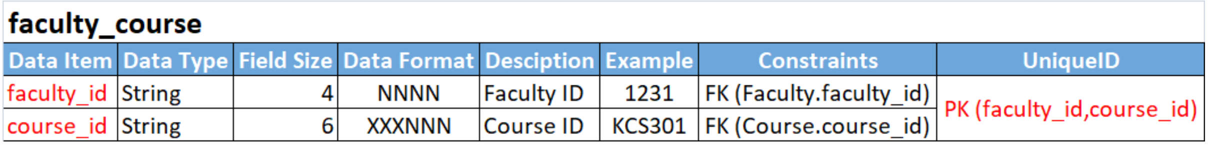
Data Item	Data Type	Field Size	Data Format	Desciption	Example	Constraints
researchproj_id	String	5	XXNNN	Research Project ID	RP101	Primary Key
researchproj_name	String	100			Attendance using ML	Not Null
duration	String	20			6 Month	Not Null
sponsor_agency	String	50			Government	Domain: {Government, Self}
status	String	20			Complete	Domain: {Complete, Ongoing}

section

Data Item	Data Type	Field Size	Data Format	Desciption	Example	Constraints	UniqueID
dept_id	String	3	XNN		D01	FK (Department.dept_id)	PK (dept_id,section_name)
section_name	String	1	X	Section Name	A	Not Null	
max_capacity	Numeric	3			70	Not Null	
student_nos	Numeric	3			63	Default 0	

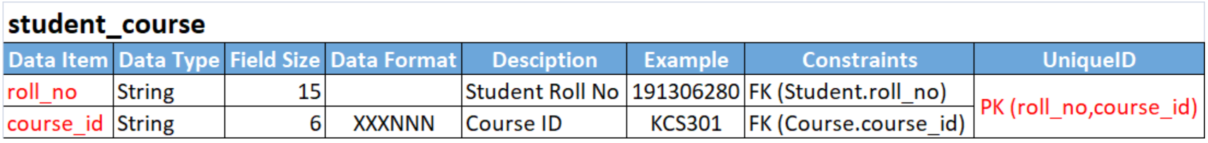
student_phone_no

Data Item	Data Type	Field Size	Data Format	Desciption	Example	Constraints	UniqueID
roll_no	String	15		Roll Number	191306280	FK (Student.roll_no)	PK (roll_no,phone_no)
phone_no	String	10	NNNNNNNNNN	Phone Number	9810012345	Not Null	



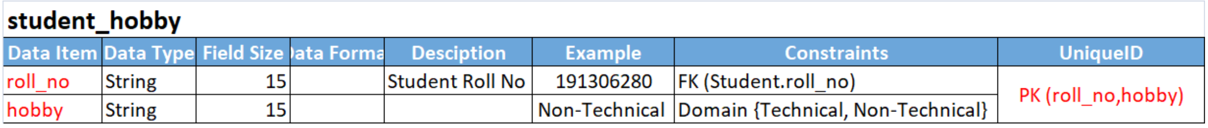
faculty_course

Data Item	Data Type	Field Size	Data Format	Desciption	Example	Constraints	UniqueID
faculty_id	String	4	NNNN	Faculty ID	1231	FK (Faculty.faculty_id)	PK (faculty_id,course_id)
course_id	String	6	XXXNNN	Course ID	KCS301	FK (Course.course_id)	



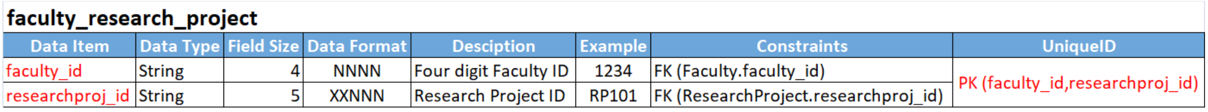
student_course

Data Item	Data Type	Field Size	Data Format	Desciption	Example	Constraints	UniqueID
roll_no	String	15		Student Roll No	191306280	FK (Student.roll_no)	PK (roll_no,course_id)
course_id	String	6	XXXNNN	Course ID	KCS301	FK (Course.course_id)	



student_hobby

Data Item	Data Type	Field Size	ata Forma	Desciption	Example	Constraints	UniqueID
roll_no	String	15		Student Roll No	191306280	FK (Student.roll_no)	PK (roll_no,hobby)
hobby	String	15			Non-Technical	Domain {Technical, Non-Technical}	



faculty_research_project

Data Item	Data Type	Field Size	Data Format	Desciption	Example	Constraints	UniqueID
faculty_id	String	4	NNNN	Four digit Faculty ID	1234	FK (Faculty.faculty_id)	PK (faculty_id,researchproj_id)
researchproj_id	String	5	XXNNN	Research Project ID	RP101	FK (ResearchProject.researchproj_id)	

Now we will use this data dictionary to create database schema using Structured Query Language (SQL) with some existing RDBMS software like Oracle, MySQL, SQL Server, etc.

3.4.2 Introduction to SQL

3.4.2.1 SQL Basics

Structured Query Language (SQL) is designed for managing data held in a relational database management system (RDBMS) like Oracle, MySQL, SQL Server, etc.

It is based on the relational model described by Dr. E.F. Codd in 1970 in the paper, "A Relational Model of Data for Large Shared Data Banks."

Structured English Query Language (SEQUEL) was developed by IBM Corporation, Inc., to use Codd's model. SEQUEL later became SQL (still pronounced "sequel").

In 1979, Relational Software, Inc. (now Oracle) introduced the first commercially available implementation of SQL. It was standardized by American National Standards Institute (ANSI) in 1986 and International Organization for Standardization (ISO) in 1987.

Today, SQL is accepted as the standard RDBMS language.

Features of SQL

- The ANSI standard for operating relational databases
- Open-Source
- Efficient
- Functionally complete
- Easy to learn and use
- Case insensitive

3.4.2.2 Data Types in SQL

Data types are used to represent the nature of the data that can be stored in the database tables. For every database, the data types are categorized into three major categories:

- String
- Numeric
- Date and Time

Data types might have different names in different RDBMS. Moreover, even if the name is the same, the size and other details may be different.

Here we have described commonly used data types of MySQL, SQL Server, and Oracle. Always check the documentation of corresponding RDBMS for the latest updates.

MySQL Data Types

Below is the list of data types used in MySQL database. This is based on MySQL 8.0.

String Data Types

Data Type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The size parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The size parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The size parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The size parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large OBjects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters

LONGBLOB	For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data
ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. It can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list; a blank value will be inserted. The values submitted are sorted in order.
SET(val1, val2, val3, ...)	A string object that can have 0 or more values, chosen from a list of possible values. Maximum 64 values can be set in a SET list.

Numeric Data Types

Data Type	**Description**
BIT(size)	A bit-value type. The number of bits per value is specified in size. The size parameter can hold a value from 1 to 64. The default value for size is 1.
TINYINT(size)	A very small integer. The signed range is from -128 to 127. The unsigned range is from 0 to 255. The size parameter specifies the maximum display width (which is 255)
BOOL	Zero is considered false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL
SMALLINT(size)	A small integer. The signed range is from -32768 to 32767. The unsigned range is from 0 to 65535. The size parameter specifies the maximum display width (which is 255)
MEDIUMINT(size)	A medium integer. The signed range is from -8388608 to 8388607. The unsigned range is from 0 to 16777215. The size parameter specifies the maximum display width (which is 255)
INT(size)	A medium integer. The signed range is from -2147483648 to 2147483647. The unsigned range is from 0 to 4294967295. The size parameter specifies the maximum display width (which is 255)
INTEGER(size)	Equal to INT(size)
BIGINT(size)	A large integer. The signed range is from -9223372036854775808 to 9223372036854775807. The unsigned range is from 0 to 18446744073709551615. The

	size parameter specifies the maximum display width (which is 255)
FLOAT(size, d)	A floating-point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions.
FLOAT(p)	A floating-point number. MySQL uses the p value to determine whether to use FLOAT or DOUBLE for the resulting data type. If p is from 0 to 24, the data type becomes FLOAT(). If p is from 25 to 53, the data type becomes DOUBLE()
DOUBLE(size, d)	A normal-size floating-point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter.
DECIMAL(size, d)	An exact fixed-point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. The maximum number for size is 65 and d is 30. The default value for size is 10 and d is 0.

Date and Time Data Types

Data Type	**Description**
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31.'
DATETIME(fsp)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP(fsp)	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP

	and ON UPDATE CURRENT_TIMESTAMP in the column definition
TIME(fsp)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in the four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

SQL Server Data Types

String Data Types

Data Type	**Description**
char(n)	Fixed width character string upto 8,000 characters
varchar(n)	Variable width character string upto 8,000 characters. It takes 2 extra bytes while storing data on the disc.
varchar(max)	Variable width character string upto 1,073,741,824 characters. It takes 2 extra bytes while storing data on the disc.
Text	Variable width character string upto 2GB of text data. It takes 4 extra bytes while storing data on the disc.
Nchar	Fixed width Unicode string upto 4,000 characters
Nvarchar	Variable width Unicode string upto 4,000 characters
nvarchar(max)	Variable width Unicode string upto 536,870,912 characters
Ntext	Variable width Unicode string upto 2 GB
binary(n)	Fixed width binary string upto 8,000 bytes
Varbinary	Variable width binary string upto 8,000 bytes
varbinary(max)	Variable width binary string upto 2 GB
Image	Variable width binary string upto 2 GB

Numeric Data Types

Data Type	**Description**
Bit	Integer that can be 0, 1, or NULL
tinyint	Allows whole numbers from 0 to 255
smallint	Allows whole numbers between -32,768 and 32,767

Int	Allows whole numbers between -2,147,483,648 and 2,147,483,647
Bigint	Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807
decimal(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1.
numeric(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1.
smallmoney	Monetary data from -214,748.3648 to 214,748.3647
money	Monetary data from -922,337,203,685,477.5808 to 922,337,203,685,477.5807
float(n)	Floating precision number data from -1.79E + 308 to 1.79E + 308
Real	Floating precision number data from -3.40E + 38 to 3.40E + 38

Date and Time Data Types

Data Type	Description
Datetime	From January 1, 1753 to December 31, 9999, with an accuracy of 3.33 milliseconds
datetime2	From January 1, 0001 to December 31, 9999, with an accuracy of 100 nanoseconds.
Smalldatetime	From January 1, 1900, to June 6, 2079, with an accuracy of 1 minute
Date	Store a date only. From January 1, 0001 to December 31, 9999
Time	Store a time only to an accuracy of 100 nanoseconds
datetimeoffset	The same as datetime2 with the addition of a time zone offset
Timestamp	It stores a unique number that gets updated every time a row gets created or modified. The timestamp value is based upon an internal clock and does not correspond to real-time. Each table may have only one timestamp variable.

Oracle Data Types

Data Type	Description
CHAR(size)	Fixed-length character data of length size bytes. The maximum size is 2000 bytes or characters. Default and the minimum size is 1 byte.

NCHAR(size)	Fixed-length character data of length size characters. The number of bytes can be up to two times the size for AL16UTF16 encoding and three times the size for UTF8 encoding.
VARCHAR2(size)	Variable-length character string having maximum length size bytes or characters. The maximum size is 4000 bytes or characters, and the minimum is 1 byte or 1 character. The size for VARCHAR2 must be specified.
NVARCHAR2(size)	Variable-length Unicode character string having maximum length size characters. The number of bytes can be up to two times the size for AL16UTF16 encoding and three times the size for UTF8 encoding. Maximum size is determined by the national character set definition, with an upper limit of 4000 bytes. Size for NVARCHAR2 must be specified.
NUMBER[(precision[, scale]])	Number having precision p and scale s. The precision p can range from 1 to 38. The scale s can range from -84 to 127.
LONG	Character data of variable length up to 2 gigabytes, or 2^{31} -1 bytes. It is provided for backward compatibility.
DATE	Valid date range from January 1, 4712 BC to December 31, 9999 AD. The default format is determined explicitly by the NLS_DATE_FORMAT parameter or implicitly by the NLS_TERRITORY parameter. The size is fixed at 7 bytes. This datatype contains the datetime fields YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND. It does not have fractional seconds or a time zone.
BINARY_FLOAT	32-bit floating-point number. This datatype requires 5 bytes, including the length byte.
BINARY_DOUBLE	64-bit floating-point number. This datatype requires 9 bytes, including the length byte.
TIMESTAMP	Year, month, and day values of date, as well as hour, minute, and second values of time
RAW(size)	Raw binary data of length size bytes. The maximum size is 2000 bytes. Size for a RAW value must be specified.
LONG RAW	Raw binary data of variable length up to 2 gigabytes.
CLOB	A character large object is containing single-byte or multibyte characters. Both fixed-width and variable-width character sets are supported, both using the database character set. Maximum size is (4 gigabytes - 1) * (database block size).
BLOB	A binary large object. Maximum size is (4 gigabytes - 1) * (database block size).

BFILE	Contains a locator to a large binary file stored outside the database. Enables byte stream I/O access to external LOBs residing on the database server. The maximum size is 4 gigabytes.

3.4.2.3 Types of SQL Statements

The commands used in SQL are generally termed as SQL statements. SQL statements are categorized into four major categories:

- Data Definition Language (DDL)
    - CREATE
    - ALTER
    - DROP
    - RENAME
    - TRUNCATE
- Data Manipulation Language (DML)
    - INSERT
    - UPDATE
    - DELETE
    - SELECT (Also considered as Data Query Language (DQL) statement)
- Data Control Language (DCL)
    - GRANT
    - REVOKE
- Transaction Control Language (TCL)
    - COMMIT
    - ROLLBACK
    - SAVEPOINT

Points to remember while writing SQL Statements:

- SQL statements are non-case-sensitive
- SQL statements can be entered on one or more lines
- Indentation is not required but used to enhance the readability
- Some command/keywords can be specific to an RDBMS Software
- Some RDBMS software may require semicolon (;) to terminate the statement
- RDBMS keywords cannot be used as identifiers, e.g., table name, view name, etc.

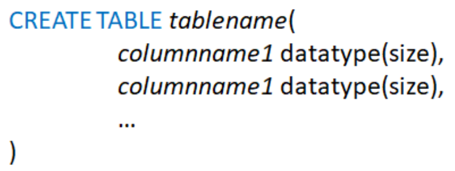
3.4.3 Working with SQL Commands

3.4.3.1 Tables Creation

When we start working on implementing the relational model, the first thing starts with creating the database schema. A database schema contains structures of tables along with other related items to be discussed in subsequent chapters.

To create a table, SQL provides the CREATE TABLE statement. The CREATE TABLE statement requires the table name, its column names, data types, and size to be allocated. Some data types do not require size.

Syntax of CREATE TABLE statement



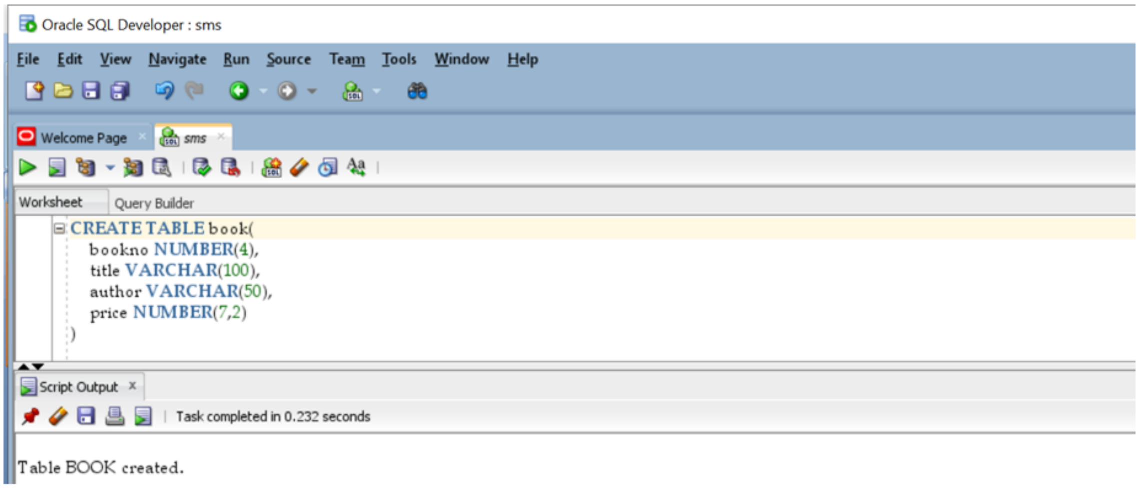
```
CREATE TABLE tablename(
            columnname1 datatype(size),
            columnname1 datatype(size),
            ...
)
```

Before moving further you need to install the required software using
“**Appendix A: Installation of Oracle**”

Example:

Let us create a sample database table using Oracle RDBMS to manage the books in a library. We can define the table name as ***Book*** with column names (also referred to as attributes or data items or field names) as ***bookno***, ***title***, ***author,*** and ***price***. Assuming the maximum number of books to be stored is a numeric value 9999, the maximum size of the book title is 100 characters long string. The maximum size of the author's name is 50 characters long string, and the maximum price of the book is a numeric value of 9999.99.

Oracle provides the NUMBER data type to store numeric data and store dynamic length string; it provides VARCHAR2 data type. Most of the other RDBMS gives VARCHAR as the data type for dynamic size strings, but Oracle gives us the flexibility to use VARCHAR or VARCHAR2 as the data type.



Oracle SQL Developer : sms
File Edit View Navigate Run Source Team Tools Window Help
Welcome Page
sms
Worksheet
Query Builder
Script Output
Task completed in 0.232 seconds

```
CREATE TABLE book(
    bookno NUMBER(4),
    title VARCHAR(100),
    author VARCHAR(50),
    price NUMBER(7,2)
)
```

```
Table BOOK created.
```

Figure 3.21: Representation of table creation in Oracle SQL Developer

3.4.3.2 Inserting Records

Once the table structure is created, we can insert data of books generally referred to as records or rows or tuples.

SQL provides an INSERT statement to insert the records in the database table.

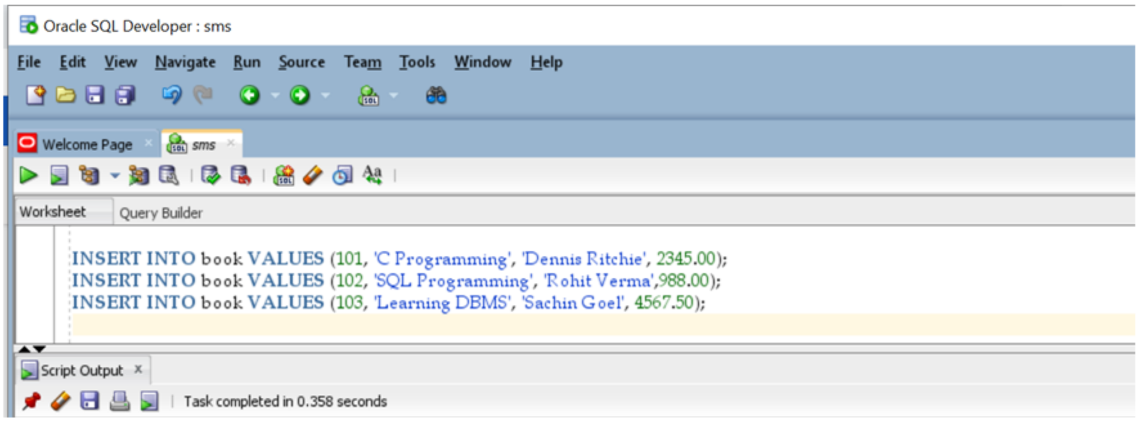
Syntax of INSERT Statement

INSERT INTO *tablename* VALUES (*value1, value2,...*)

While inserting data in a database table, we need to remember that the string type and date/time values are enclosed in a single quote.

Let us insert few records of the books.

While writing multiple SQL statements, use a semicolon (;) to terminate a statement.



Oracle SQL Developer : sms
File Edit View Navigate Run Source Team Tools Window Help
Welcome Page
sms
Worksheet
Query Builder
Script Output
Task completed in 0.358 seconds

```
INSERT INTO book VALUES (101, 'C Programming', 'Dennis Ritchie', 2345.00);
INSERT INTO book VALUES (102, 'SQL Programming', 'Rohit Verma',988.00);
INSERT INTO book VALUES (103, 'Learning DBMS', 'Sachin Goel', 4567.50);
```

Figure 3.22: Representation of Inserting records in a table in Oracle SQL Developer

3.4.3.3 Displaying Records

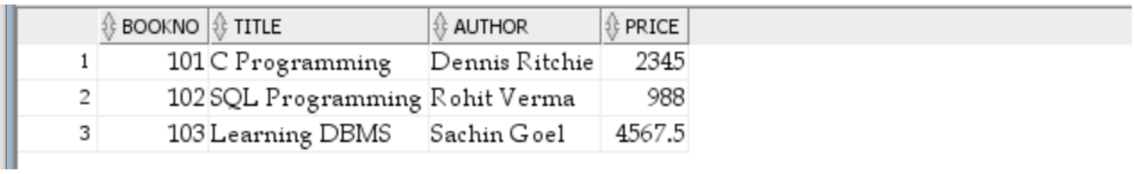
Once we insert the record in the database table, we can display the records.
SQL provides SELECT statements to view the records as per choice. We can view all records with all their columns using the following syntax.

```
SELECT * FROM tablename;
```

Example:

To view all the records, present the book table, we can write the following SQL statement.

```
SELECT * FROM book;
```

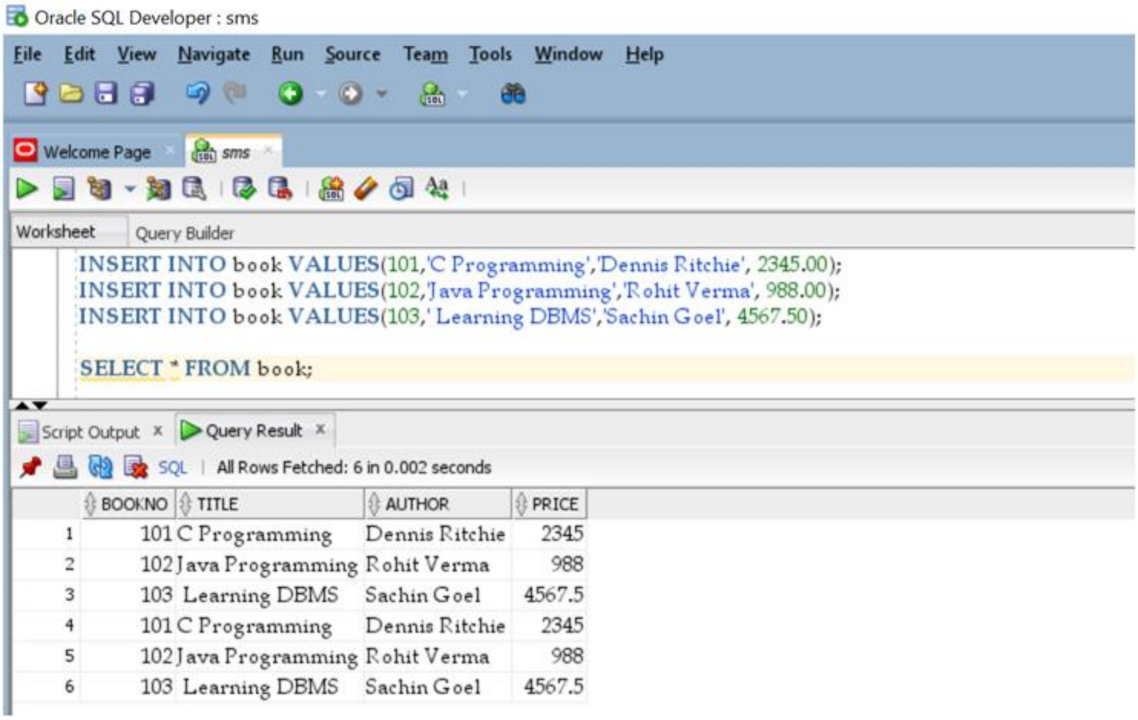


	BOOKNO	TITLE	AUTHOR	PRICE
1	101	C Programming	Dennis Ritchie	2345
2	102	SQL Programming	Rohit Verma	988
3	103	Learning DBMS	Sachin Goel	4567.5

Figure 3.23: Displaying of records from table book in Oracle SQL Developer

3.4.3.4 Specifying Constraints

While inserting records, as shown above, if we re-run the Insert commands, it will execute the commands again and insert three records again, creating duplicate records of books creating ambiguity and inconsistency in the database since we have not specified any restrictions or constraints on the table book. Now we need to specify the constraints on the table.



Oracle SQL Developer : sms
File Edit View Navigate Run Source Team Tools Window Help
Welcome Page
sms
Worksheet
Query Builder
Script Output
Query Result
SQL

```
INSERT INTO book VALUES(101,'C Programming','Dennis Ritchie', 2345.00);
INSERT INTO book VALUES(102,'Java Programming','Rohit Verma', 988.00);
INSERT INTO book VALUES(103,' Learning DBMS','Sachin Goel', 4567.50);

SELECT * FROM book;
```

All Rows Fetched: 6 in 0.002 seconds

	BOOKNO	TITLE	AUTHOR	PRICE
1	101	C Programming	Dennis Ritchie	2345
2	102	Java Programming	Rohit Verma	988
3	103	Learning DBMS	Sachin Goel	4567.5
4	101	C Programming	Dennis Ritchie	2345
5	102	Java Programming	Rohit Verma	988
6	103	Learning DBMS	Sachin Goel	4567.5

Figure 3.24: Displaying of inconsistent Records in Table (Book)

Constraints are the rules enforced on the table. These are used to restrict the type or value of data inserted into it or updated or deleted later on.

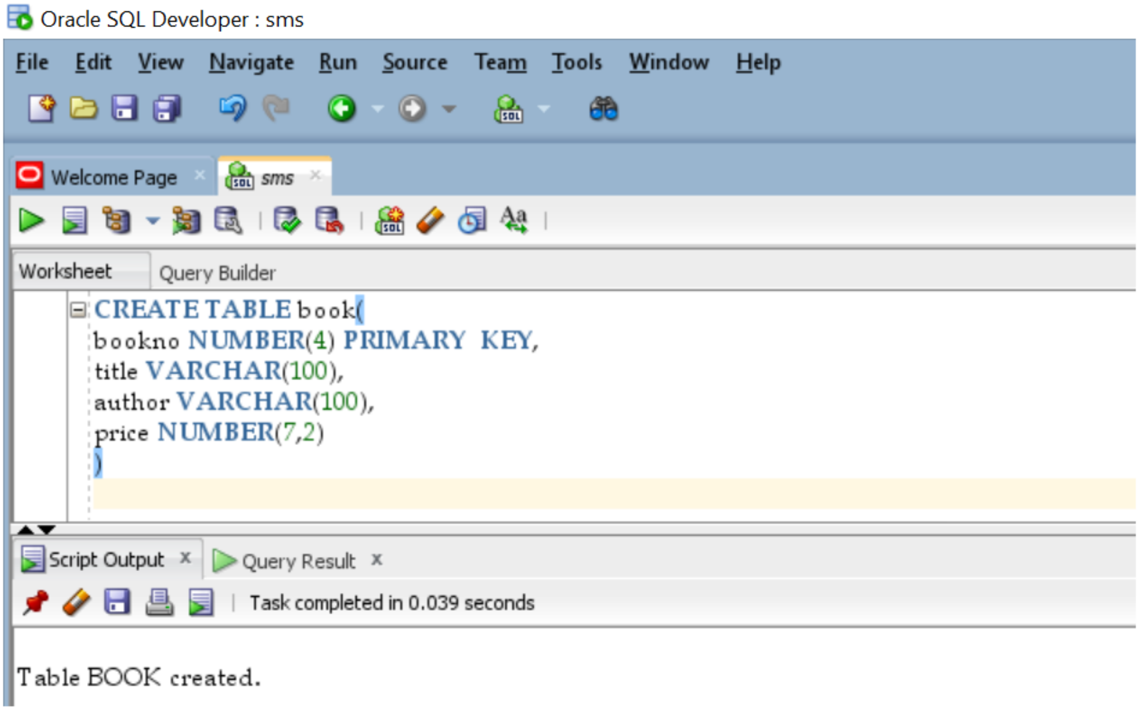
The database system checks the specified constraint each time the database is updated. Any updation that causes modification to the database but **violates** an integrity constraint is **rejected**.

For example, suppose we apply the Key constraint **"PRIMARY KEY"** on column ***bookno***. In that case, this kind of error will not be possible since the RDBMS will take care that once a bookno is present in the database table, the duplicate value will not be allowed, and even it cannot be left blank.

However, before continuing, we need to remove the existing database table using the DROP TABLE statement.

```
DROP TABLE book;
```

Now re-create the table with the PRIMARY KEY constraint on the column bookno.


Oracle SQL Developer : sms
File Edit View Navigate Run Source Team Tools Window Help
Welcome Page
sms
Worksheet
Query Builder
Script Output
Query Result

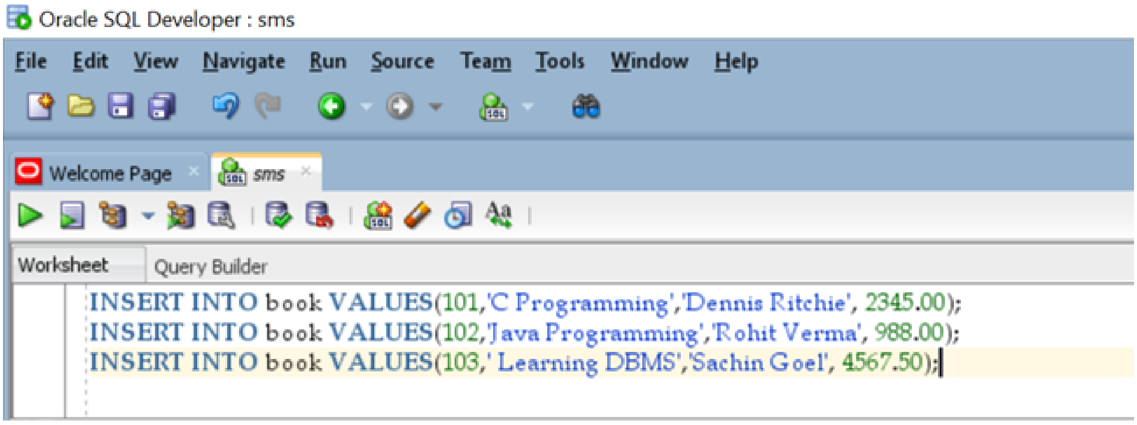
```
CREATE TABLE book(
bookno NUMBER(4) PRIMARY  KEY,
title VARCHAR(100),
author VARCHAR(100),
price NUMBER(7,2)
)
```

Task completed in 0.039 seconds

```
Table BOOK created.
```

Figure 3.25: Creating Table (BOOK) with constraints (Primary Key)

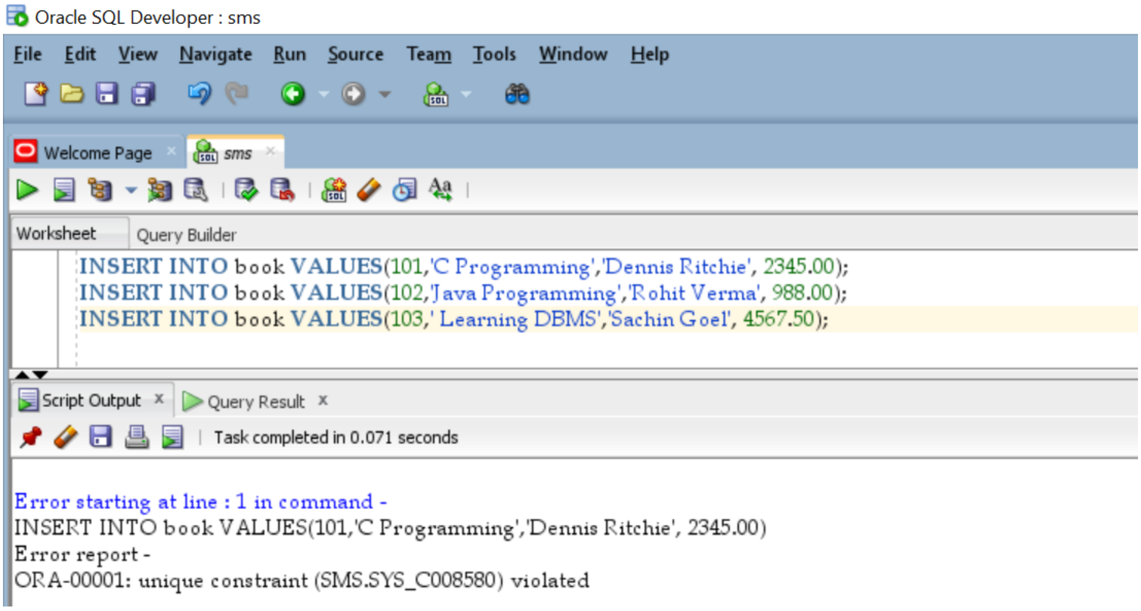
Now again, insert the three records into the database table by executing the following statements.


Oracle SQL Developer : sms
File Edit View Navigate Run Source Team Tools Window Help
Welcome Page
sms
Worksheet
Query Builder

```
INSERT INTO book VALUES(101,'C Programming','Dennis Ritchie', 2345.00);
INSERT INTO book VALUES(102,'Java Programming','Rohit Verma', 988.00);
INSERT INTO book VALUES(103,' Learning DBMS','Sachin Goel', 4567.50);
```

Figure 3.26: Inserting records in BOOK Tables

Records get inserted property. However, if we repeat the same statement by mistake, an error will appear from RDBMS due to the PRIMARY KEY constraint applied on the column bookno. Try to re-run the INSERT statements.


Oracle SQL Developer : sms
File Edit View Navigate Run Source Team Tools Window Help
Welcome Page
sms
Worksheet
Query Builder
Script Output
Query Result

```
INSERT INTO book VALUES(101,'C Programming','Dennis Ritchie', 2345.00);
INSERT INTO book VALUES(102,'Java Programming','Rohit Verma', 988.00);
INSERT INTO book VALUES(103,' Learning DBMS','Sachin Goel', 4567.50);
```

Task completed in 0.071 seconds

```
Error starting at line : 1 in command -
INSERT INTO book VALUES(101,'C Programming','Dennis Ritchie', 2345.00)
Error report -
ORA-00001: unique constraint (SMS.SYS_C008580) violated
```

Figure 3.27: Display of Error (Unique Key Constraint)

Understanding the Implementation of Constraints in SQL

In SQL, constraints could be either on a column level or a table level.

1. **Column-level constraints**– These constraints are applied to a single column
2. **Table-level constraints** – These constraints are the application to the complete table.

They can be applied when a table is created or when its structure is altered. The constraints are specified within the SQL DDL command, like the 'CREATE TABLE' command.

The general syntax of 'CREATE TABLE' command specifying the integrity constraint:

Column-level:

```
CREATE TABLE tablename (
        columnname1 datatype(size) constraint_name,
        columnname2 datatype(size) constraint_name,
        ...
 );
```

Table-level:

```
CREATE TABLE tablename(
        columnname1 datatype(size) ,
        columnname2 datatype(size) ,
        ...,
        constraint_name (column_name1, column name2, ...)
         );
```

Entity Integrity Constraints

Entity integrity constraint implemented using SQL by defining any column as the PRIMARY KEY. By defining the primary key attribute in a relation, we ensure that the attribute will not accept a null value. Also, the values will be unique for the tuple.

Syntax for defining Primary Key using SQL:

Column –Wise:

```
CREATE TABLE tablename (
        columnname1 datatype(size) PRIMARY KEY,
        columnname2 datatype(size),
        ...
);
```

Example:

The following example identifies the ***dept_id*** column as the PRIMARY KEY for the Department table:

```
CREATE TABLE department (
        dept_id CHAR(3) PRIMARY KEY,
        dept_name VARCHAR(5),
        dept_location VARCHAR(20)
);
```

Table-Wise:

```
CREATE TABLE tablename (
        columnname1 datatype(size),
        columnname2 datatype(size) ,
        ...,
        PRIMARY KEY (columnname1 ,columnname2, . . . )
);
```

Example

The primary key constraint in this example is defined after the column comma list in the CREATE TABLE statement.

```
CREATE TABLE department (
        dept_id CHAR(3),
        dept_name VARCHAR(5),
        dept_location VARCHAR(20),
        PRIMARY KEY(dept_id)
);
```

Whenever we declare any attribute in relation as the Primary Key attribute, it is not necessary to specify it **explicitly** to be not null in addition to the primary key constraint.

Referential Integrity Constraints

A *foreign key* is a column in a child table that references a primary key in the parent table. A *foreign key constraint* is a primary mechanism used to enforce referential integrity between tables in a relational database. A column defined as a foreign key refers to a column defined as a primary key in another table.

Syntax for defining foreign key using SQL:

At column-level:

```
CREATE TABLE tablename (
        columnname1 datatype(size),
        columnname2 datatype(size) REFERENCESreferenced_table(columnname) ,
        ...
);
```

Example:

```
CREATE TABLE course (
          course_id CHAR(6) PRIMARY KEY,
          course_name VARCHAR(50),
          course_credit NUMBER(2),
          dept_id CHAR(3) REFERENCES department(dept_id)
  ) ;
```

In this example, the dept_id column of the course table has been designated as the foreign key for the course table. This foreign key references the dept_id column in the department table. This foreign key ensures that for every dept_id in the course table, there is a corresponding dept_id in the department table.

At table-level:

```
CREATE TABLE tablename (
          columnname1 datatype(size),
          columnname2 datatype(size),
          ...,
          FOREIGN KEY (columnname) REFERENCES referenced_table(columnname)
);
```

Example:

```
CREATE TABLE course (
          course_id CHAR(6) PRIMARY KEY,
          course_name VARCHAR(50),
          course_credit NUMBER(2),
          dept_id CHAR(3),
          FOREIGN KEY (dept_id) REFERENCES department(dept_id)
  ) ;
```

NOT NULL Constraint

NOT NULL is a constraint that can place on a table's column. This constraint disallows the entrance of NULL values into a column; in other words, data is required in a NOT NULL column for each row of data in the table. NULL is generally the default column if NOT NULL is not specified, allowing NULL values in a column.

Syntax:

```
CREATE TABLE tablename (
          columnname1 datatype(size) NOT NULL,
          columnname2 datatype(size),
        ...
);
```

Example:

```
CREATE TABLE department (
        dept_id CHAR(3) PRIMARY KEY,
        dept_nameVARCHAR(5) NOT NULL,
        dept_locationVARCHAR(20) NOT NULL
);
```

In this example, the NOT NULL constraint is applied on the dept_name column and dept_location column. The columns will not allow NULL values.

UNIQUE Constraint

This constraint ensures that a column or a group of columns in each row has a distinct value. A column(s) can have a null value, but the values cannot be duplicated.

Column level:

```
CREATE TABLE tablename (
        columnname1 datatype(size) UNIQUE,
        columnname2 datatype(size),
        ...
);
```

Example:

Column Level:

```
CREATE TABLE department (
        dept_id CHAR(3) PRIMARY KEY,
        dept_name VARCHAR(10) UNIQUENOT NULL,
        dept_location VARCHAR(20) NOT NULL
);
```

Table Level:

```
CREATE TABLEtablename (
        columnname1 datatype(size),
        columnname2 datatype(size),
        ...,
        UNIQUE (columnname1)
);
```

Example:

```
CREATE TABLE department (
	dept_id CHAR(3) PRIMARY KEY,
	dept_name VARCHAR(5) NOT NULL,
	dept_location VARCHAR(20) NOT NULL,
	UNIQUE (dept_name)
);
```

In the above example, the UNIQUE keyword has been used with the dept_name column. This column will have distinct values.

CHECK Constraint

This constraint defines a business rule on a column. The check constraint specifies conditions for the data inserted into a column. Each row inserted into a table or each value updating the column's value must meet these conditions. The CHECK clause is used to specify check constraints. The constraint can be applied to a single column or a group of columns.
Syntax at column level:

Column level:

```
CREATE TABLEtablename (
	columnname1 datatype(size) CHECK (condition),
	columnname2 datatype(size),
	...
);
```

Table level:

```
CREATE TABLE tablename (
	columnname1 data_type(size),
	columnname2 data_type(size) ,
	...,
	CHECK (condition)
);
```

Example:

In the faculty table, the salary of the employee will always be greater than zero.

Column Level:

```
CREATE TABLE faculty(
        faculty_id CHAR(4) PRIMARY KEY,
        first_name VARCHAR(20) NOT NULL,
        middle_name VARCHAR(20),
        last_name VARCHAR(20),
        designation VARCHAR(50) NOT NULL,
        doj DATE NOT NULL,
        gender CHAR(6) NOT NULL CHECK (gender in (‘Male’,’Female’,’Trans’)),
        salary NUMBER(6) NOT NULL CHECK(salary>0),
        dept_id CHAR(3) NOT NULL,
        hod_id CHAR(4),
        FOREIGN KEY (dept_id) REFERENCES department(dept_id),
        FOREIGN KEY (hod_id) REFERENCES faculty(faculty_id)
)
```

Table Level:

```
CREATE TABLE faculty(
        faculty_id CHAR(4) PRIMARY KEY,
        first_name VARCHAR(20) NOT NULL,
        middle_name VARCHAR(20),
        last_name VARCHAR(20),
        designation VARCHAR(50) NOT NULL,
        doj DATE NOT NULL,
        gender CHAR(6) NOT NULL,
        salary NUMBER(6) NOT NULL,
        dept_id CHAR(3) NOT NULL,
        hod_id CHAR(4),
        FOREIGN KEY (dept_id) REFERENCES department(dept_id),
        FOREIGN KEY (hod_id) REFERENCES faculty(faculty_id),
        CHECK(salary>0),
        CHECK (gender in (‘Male’,’Female’,’Trans’)
)
```

DEFAULT Constraint

DEFAULT is a constraint that you can place on a table's column. This constraint inserts the given default value in the record column if no value is provided for that column while inserting a record.

Syntax:

```
CREATE TABLE tablename (
        columnname1 datatype(size) DEFAULT value,
        columnname2 datatype(size),
        ...
);
```

Example:

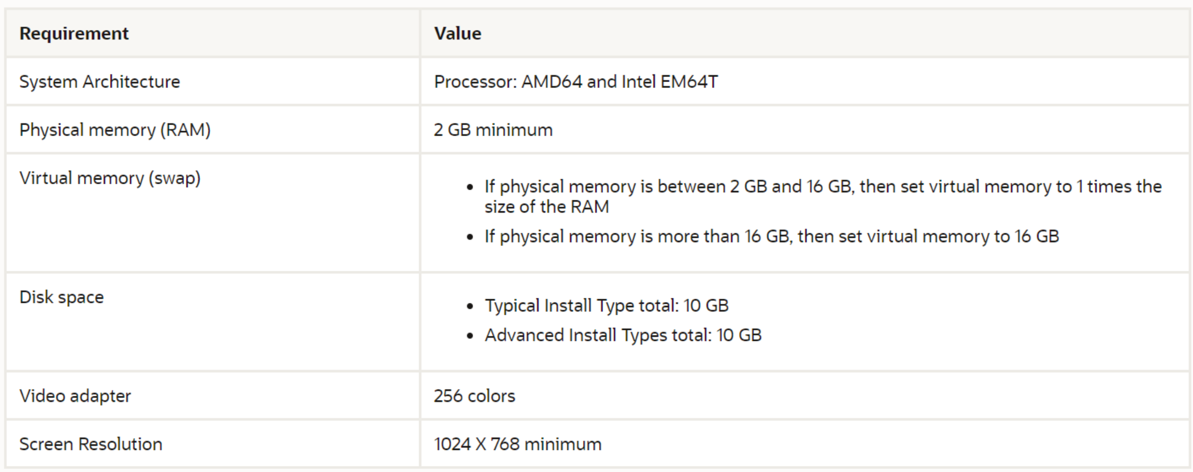
```
CREATE TABLE section(
   dept_id CHAR(3) NOT NULL,
   section_name CHAR(1) NOT NULL,
   max_capacity NUMBER(3) NOT NULL,
   student_nos NUMBER(3) DEFAULT 0,
   PRIMARY KEY (dept_id,section_name)
);
```

In this example, the DEFAULT constraint is applied to the student_nos column. The student_nos columns will have zero (0) if we do not specify any value while inserting the record in the section table.

Appendix A: Installation of Oracle

Here we will learn how to install the required software to implement the database using Oracle XE RDBMS Software. It is the free version of Oracle RDBMS software with limited capabilities. In Oracle XE, the XE stands for Express Edition.

Before installing the software, you need to check the minimum hardware specifications required



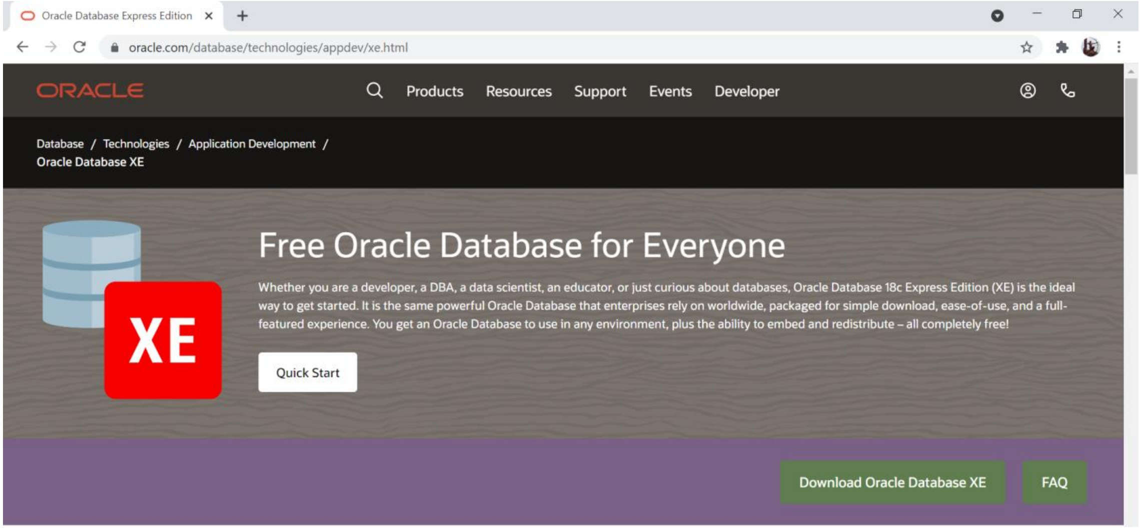
Requirement	Value
System Architecture	Processor: AMD64 and Intel EM64T
Physical memory (RAM)	2 GB minimum
Virtual memory (swap)	• If physical memory is between 2 GB and 16 GB, then set virtual memory to 1 times the size of the RAM • If physical memory is more than 16 GB, then set virtual memory to 16 GB
Disk space	• Typical Install Type total: 10 GB • Advanced Install Types total: 10 GB
Video adapter	256 colors
Screen Resolution	1024 X 768 minimum

Figure 3. 28: Display of System Requirements for Installation of Oracle XE

Download the latest available software from the link given below
https://www.oracle.com/database/technologies/appdev/xe.html

You will get the following webpage (accessed on 21st May 2021) or a similar page.
Click on the Green button Download Oracle Database XE to download the software.

It will ask you to select your operating system and agree with terms and conditions. Download the software as a ZIP file and save it somewhere on your disc.


Oracle Database Express Edition
oracle.com/database/technologies/appdev/xe.html
ORACLE
Products
Resources
Support
Events
Developer
Database / Technologies / Application Development /
Oracle Database XE
XE
Quick Start
Download Oracle Database XE
FAQ

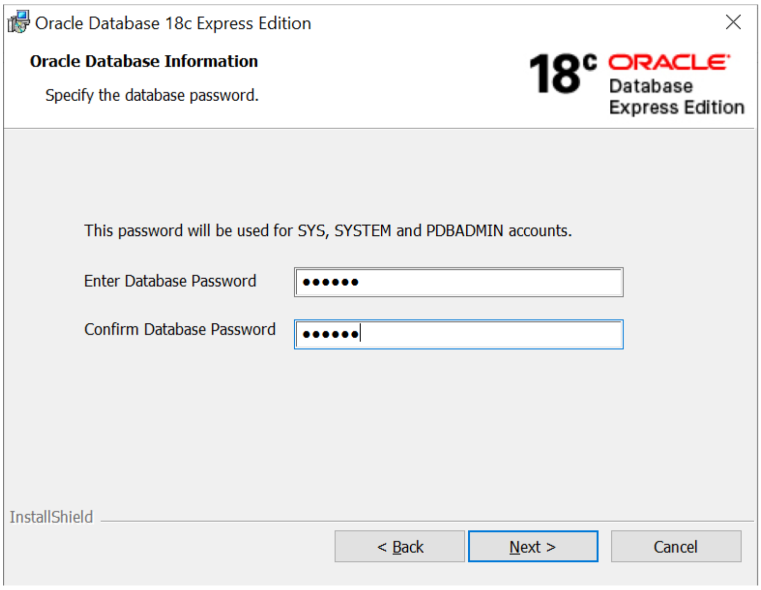
Free Oracle Database for Everyone

Whether you are a developer, a DBA, a data scientist, an educator, or just curious about databases, Oracle Database 18c Express Edition (XE) is the ideal way to get started. It is the same powerful Oracle Database that enterprises rely on worldwide, packaged for simple download, ease-of-use, and a full-featured experience. You get an Oracle Database to use in any environment, plus the ability to embed and redistribute – all completely free!

Figure 3.29: Display of Oracle XE homepage

Installing the Oracle XE RDBMS Software

Once the download is complete, you can install the software on your laptop or desktop. Unzip the downloaded ZIP file and click on the Setup.exe file to start the setup.
While installing the software, it will ask you for a password. Enter the password and remember that password. It will be required while performing database operations and schema creation. It is the password for three built-in users SYS, SYSTEM, and PDBADMIN.


Oracle Database 18c Express Edition
18c ORACLE Database Express Edition
InstallShield
< Back
Next >
Cancel

Oracle Database Information

Specify the database password.

This password will be used for SYS, SYSTEM and PDBADMIN accounts.

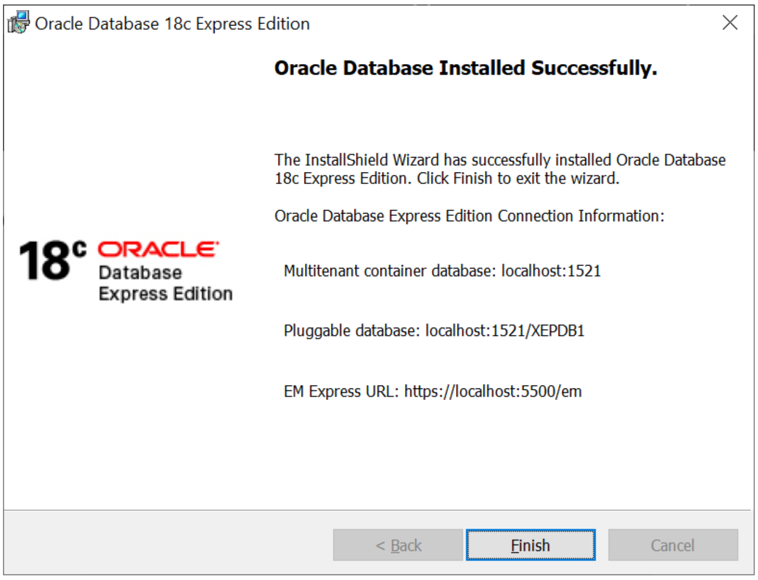
Enter Database Password ••••••

Confirm Database Password ••••••

Figure 3. 30: Display of Password page for Oracle installation

Once installation is complete, you will get some more details about the Oracle XE RDBMS software. Default database name as **XEPDB1.**

You have to keep this information with you. This information will be required later.



Oracle Database 18c Express Edition
18c ORACLE Database Express Edition
< Back
Finish
Cancel

Oracle Database Installed Successfully.

The InstallShield Wizard has successfully installed Oracle Database 18c Express Edition. Click Finish to exit the wizard.

Oracle Database Express Edition Connection Information:

Multitenant container database: localhost:1521

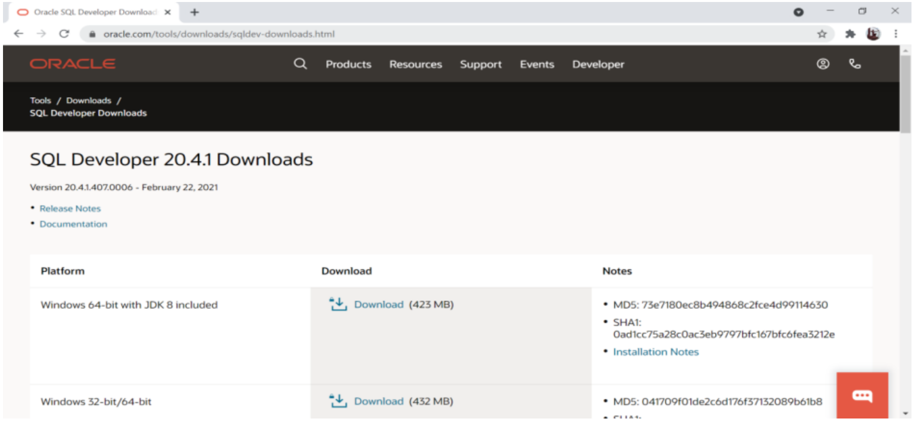
Pluggable database: localhost:1521/XEPDB1

EM Express URL: https://localhost:5500/em

Figure 3.31: Display of successful message for Oracle installation

Downloading and Installing Oracle SQL Developer Software

Now download the latest available Oracle SQL Developer software from the Oracle website using the following link https://www.oracle.com/tools/downloads/sqldev-downloads.html



SQL Developer 20.4.1 Downloads

Version 20.4.1.407.0006 - February 22, 2021

- Release Notes
- Documentation

Platform	Download	Notes
Windows 64-bit with JDK 8 included	Download (423 MB)	• MD5: 73e7180ec8b494868c2fce4d99114630 • SHA1: 0ad1cc75a28c0ac3eb9797bfc167bfc6fea3212e • Installation Notes
Windows 32-bit/64-bit	Download (432 MB)	• MD5: 041709f01de2c6d176f37132089b61b8

Figure 3.31: Display of webpage for downloading Zip file of Oracle SQL Developer

Unzip the ZIP file to start the software. No installation is required.

Create a connection between Oracle XE RDBMS and Oracle SQL Developer

Start the Oracle SQL Developer Software from the folder downloaded

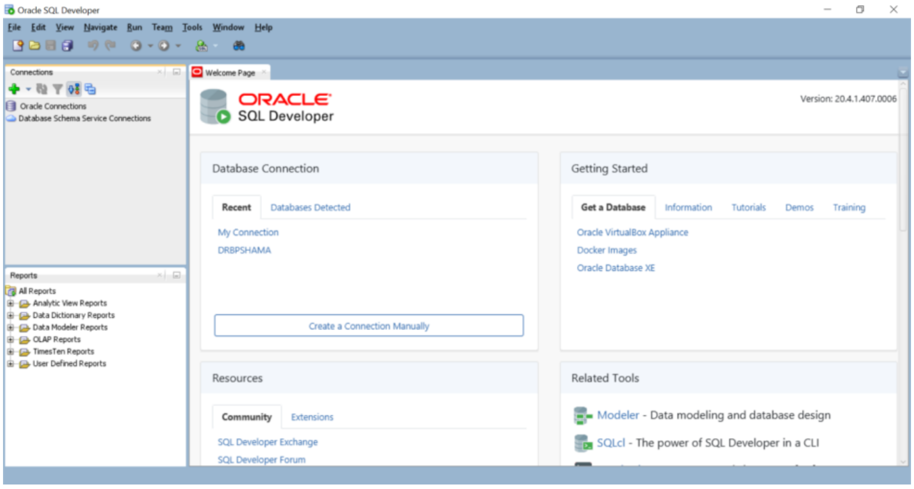


Figure 3.32: Display of Oracle SQL Developer Software

Now click on ✚ in the top left corner to create a new connection with Oracle XE RDBMS software.

Define the following information

Name: Any Name of your choice e.g. ABESEC

User Name as SYS. It is the built-in superuser.

Password as given at the time of installation

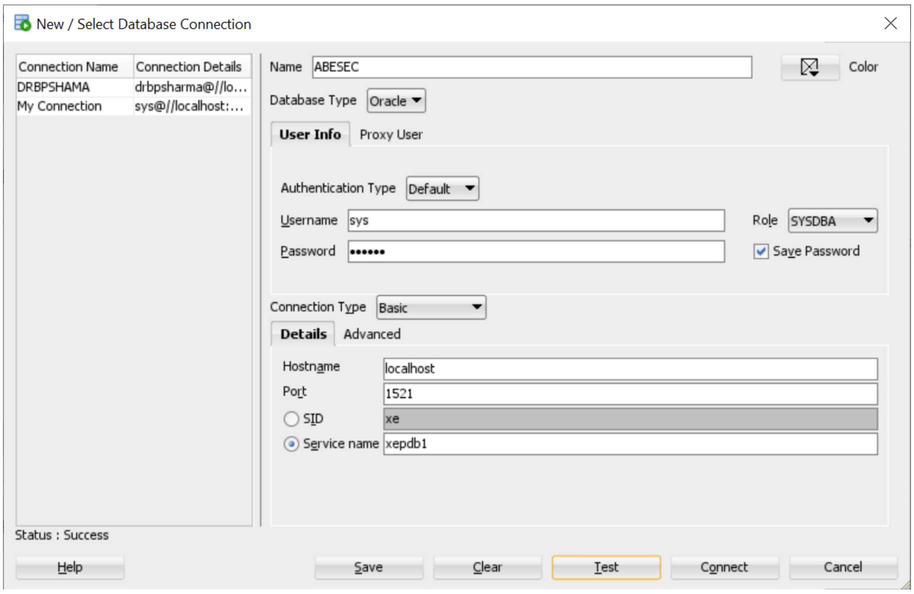
Role as SYSDBA for Database Administration

Check the check box [ ] Save Password

Service Name as XEPDB1 (default database name)

Now Click on the **Test** button to check the connection **status**

If status us **Success**, that means everything is perfect. Click on **Connect** button to complete the connection creation process.



Connection Name	Connection Details
DRBPSHAMA	drbpsharma@//lo...
My Connection	sys@//localhost:...

Name ABESEC

Database Type Oracle

Authentication Type Default

Username sys

Password ••••••

Role SYSDBA

☑ Save Password

Connection Type Basic

Hostname localhost

Port 1521

○ SID xe

● Service name xepdb1

Status : Success

Figure 3.33: Display of Oracle SQL Database Connection

Now your connection ABESEC is ready to work for database operations

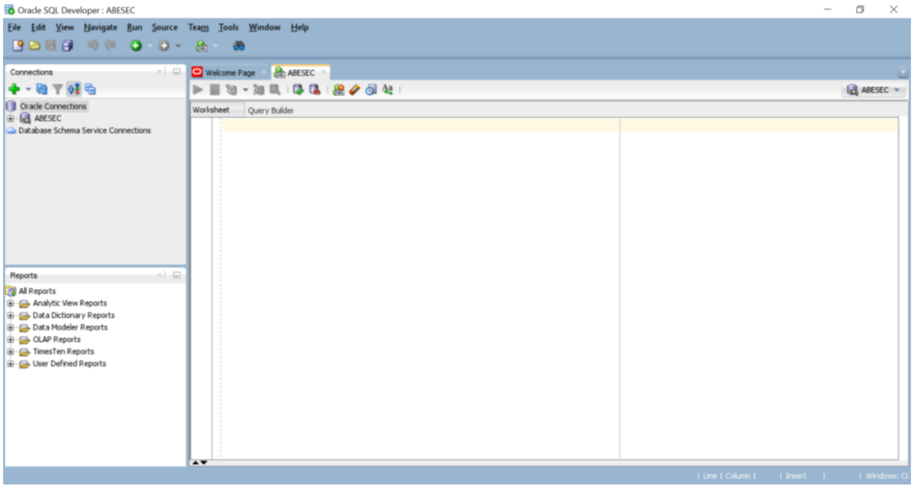


Figure 3.34: Display of Oracle SQL Developer Worksheet

Now we can execute SQL statements using Query Builder.

Appendix B: SQL Script for SMS Case Study using Oracle

```
CREATE TABLE department(
    dept_id CHAR(3) PRIMARY KEY,
    dept_name VARCHAR(10) UNIQUE NOT NULL,
    dept_location VARCHAR(20) NOT NULL
);

CREATE TABLE course (
    course_id CHAR(6) PRIMARY KEY,
course_name VARCHAR(50) NOT NULL,
course_credit NUMBER(2) NOT NULL,
dept_id CHAR(3) NOT NULL,
    CONSTRAINT course_deptid_fk FOREIGN KEY (dept_id) REFERENCES department(dept_id)
   );

CREATE TABLE section(
    dept_id CHAR(3) NOT NULL ,
    section_name CHAR(1) NOT NULL,
    max_capacity NUMBER(3) NOT NULL,
    student_nos NUMBER(3) DEFAULT 0,
    CONSTRAINT course_deptid_fk FOREIGN KEY (dept_id) REFERENCES department(dept_id),
    CONSTRAINT section_deptid_sectionname_cpk PRIMARY KEY (dept_id,section_name)
);

CREATE TABLE student(
    roll_no CHAR(15) PRIMARY KEY,
    first_name VARCHAR(15) NOT NULL,
    middle_name VARCHAR(15),
    last_name VARCHAR(15),
    dob DATE NOT NULL,
    gender CHAR(6) NOT NULL CHECK(gender IN ('Male','Female','Trans')),
    house_no VARCHAR(25) NOT NULL,
    street_name VARCHAR(25) NOT NULL,
    city VARCHAR(50) NOT NULL,
    state VARCHAR(50) NOT NULL,
    pincode CHAR(6) NOT NULL,
    dept_id CHAR(3) NOT NULL,
    section_name CHAR(1) NOT  NULL,
    CONSTRAINT student_deptid_fk FOREIGN KEY (dept_id) REFERENCES department(dept_id),
    CONSTRAINT student_deptid_sectionname_cfk FOREIGN KEY (dept_id,section_name)
REFERENCES section(dept_id,section_name)
);

CREATE TABLE faculty(
    faculty_id CHAR(4) PRIMARY KEY,
    first_name VARCHAR(20) NOT NULL,
    middle_name VARCHAR(20),
    last_name VARCHAR(20),
    designation VARCHAR(50) NOT NULL,
    doj DATE NOT NULL,
    gender CHAR(6) NOT NULL CHECK(gender IN ('Male','Female','Trans')),
    salary NUMBER(6) NOT NULL CHECK (salary>0),
    dept_id CHAR(3) NOT NULL,
    hod_id CHAR(4),
    CONSTRAINT faculty_deptid_fk FOREIGN KEY (dept_id) REFERENCES department(dept_id),
    CONSTRAINT faculty_hodid_rfk FOREIGN KEY (hod_id) REFERENCES faculty(faculty_id)
);

CREATE  TABLE mini_project(
    miniproj_id CHAR(5) PRIMARY KEY,
    miniproj_name VARCHAR(100) UNIQUE NOT NULL,
    domain VARCHAR(50) NOT NULL,
    subject VARCHAR(50) NOT NULL,
    description VARCHAR(255) NOT NULL,
    roll_no CHAR(15) REFERENCES student(roll_no)
);

CREATE TABLE research_project(
    researchproj_id CHAR(5) PRIMARY KEY,
    researchproj_name VARCHAR(100) NOT NULL,
    duration VARCHAR(20) NOT NULL,
    sponsor_agency VARCHAR(50) NOT NULL CHECK (sponsor_agency IN 
('Government','Self')),
    status VARCHAR(20) NOT NULL CHECK(status IN ('Complete','Ongoing'))
);

CREATE TABLE student_phone_no(
    roll_no CHAR(15) NOT NULL REFERENCES student(roll_no),
    phone_no CHAR(10) NOT NULL,
    CONSTRAINT studentphoneno_rollno_phoneno_cpk PRIMARY KEY (roll_no,phone_no)
);

CREATE TABLE student_hobby(
    roll_no CHAR(15) NOT NULL REFERENCES student(roll_no),
    hobby VARCHAR(15) NOT NULL CHECK (hobby IN ('Technical','Non-Technical')),
    CONSTRAINT studenthobby_rollno_hobby_cpk PRIMARY KEY (roll_no,hobby)
);


CREATE TABLE faculty_course(
    faculty_id CHAR(4) NOT NULL,
    course_id CHAR(6) NOT NULL,
    CONSTRAINT facultycourse_facultyid_fk FOREIGN KEY (faculty_id) REFERENCES 
faculty(faculty_id),
    CONSTRAINT facultycourse_courseid_fk FOREIGN KEY (course_id) REFERENCES 
course(course_id),
    CONSTRAINT facultycourse_facultyid_courseid_cpk PRIMARY KEY (faculty_id,course_id)
);


CREATE TABLE student_course(
    roll_no CHAR(15) NOT NULL,
    course_id CHAR(6) NOT NULL,
    CONSTRAINT studentcourse_rollno_fk FOREIGN KEY (roll_no) REFERENCES
student(roll_no),
    CONSTRAINT studentcourse_courseid_fk FOREIGN KEY (course_id) REFERENCES
course(course_id),
    CONSTRAINT studentcourse_rollno_courseid_cpk PRIMARY KEY (roll_no,course_id)
);

CREATE TABLE faculty_research_project(
    faculty_id CHAR(4) NOT NULL,
    researchproj_id CHAR(5) NOT NULL,
    CONSTRAINT facultyresearchproject_facultyid_fk FOREIGN KEY (faculty_id) REFERENCES
faculty(faculty_id),
    CONSTRAINT facultyresearchproject_researchprojectid_fk FOREIGN KEY
(researchproj_id) REFERENCES research_project(researchproj_id),
    CONSTRAINT facultyresearchproject_facultyid_reseachprojectid_cpk PRIMARY KEY
(faculty_id,researchproj_id)
);

INSERT INTO department VALUES('D01','CS','Aryabhatta Block');
INSERT INTO department VALUES('D02','CSE','Aryabhatta Block');
INSERT INTO department VALUES('D03','IT','Aryabhatta Block');
INSERT INTO department VALUES('D04','CSE-DS','Bhabha Block');
INSERT INTO department VALUES('D05','ME','Ramanujan Block');

INSERT INTO SECTION (section_name,max_capacity,student_nos,dept_id) VALUES('A',70,
       63,'D01');
INSERT INTO SECTION (section_name,max_capacity,student_nos,dept_id) VALUES('B',70,
       67,'D01');
INSERT INTO SECTION (section_name,max_capacity,student_nos,dept_id) VALUES('A',70,
       65,'D02');
INSERT INTO SECTION (section_name,max_capacity,student_nos,dept_id) VALUES('A',70,
       70,'D03');
INSERT INTO SECTION (section_name,max_capacity,student_nos,dept_id) VALUES('A',60,
       32,'D05');
INSERT INTO SECTION (section_name,max_capacity,student_nos,dept_id) VALUES('B',60,
       31,'D05');
INSERT INTO SECTION (section_name,max_capacity,dept_id) VALUES('A',70,'D04');


INSERT INTO course (course_id,course_name,course_credit,dept_id) VALUES('KCS301','Data
Structure',   3,     'D01');
INSERT INTO course (course_id,course_name,course_credit,dept_id)
VALUES('KCS401','DBMS',     3,     'D02');
INSERT INTO course (course_id,course_name,course_credit,dept_id)
VALUES('KME502','Machine Design',  2,     'D05');
INSERT INTO course (course_id,course_name,course_credit,dept_id)
VALUES('KEC301','Digital Electronics',    2,     'D03');
INSERT INTO course (course_id,course_name,course_credit,dept_id)
VALUES('KAS101','Physics',  3,     'D02');

INSERT INTO faculty(faculty_id,first_name,middle_name, last_name,designation,doj,
       gender,salary, hod_id,dept_id) VALUES('1231',   'Ram', 'Mohan',      'Prasad',
       'Assistant Professor',      '17-12-2011', 'Male',       54000, '1231',
       'D01');
```

```
INSERT INTO faculty(faculty_id,first_name,      last_name,designation,doj,
       gender,salary, hod_id,dept_id) VALUES('9765',    'Laxman',     'Naryan',
       'Associate Professor',      '19-09-2013', 'Male',       72000, '9765',
       'D02');
INSERT INTO faculty(faculty_id,first_name,      last_name,designation,doj,
       gender,salary, dept_id) VALUES('1987',    'Ayush',      'Giri',
       'Professor',  '13-07-2010', 'Male' ,'100125','D03');
INSERT INTO faculty(faculty_id,first_name,middle_name, last_name,designation,doj,
       gender,salary, hod_id,dept_id) VALUES('2765',    'Grish',      'Kumar',
       'Sharma',     'Assistant Professor',      '01-02-2015', 'Male',       62000,
       '2765',       'D05');
INSERT INTO faculty(faculty_id,first_name,      last_name,designation,doj,
       gender,salary, hod_id,dept_id) VALUES('1734',    'Nidhi',      'Bhatia',
       'Assistant Professor',      '06-07-2016', 'Female',     56000, '1987',
       'D03');

INSERT INTO student(roll_no,first_name,last_name,dob,
gender,house_no,street_name,city, state,pincode,dept_id,      section_name)
VALUES('191306280',  'Aayushi',           'Mishra',     '12-03-2002', 'Female',
       '243', 'Shahdra',    'Delhi',      'Delhi',      '110063',     'D01', 'A');
INSERT INTO
student(roll_no,first_name,middle_name,last_name,dob,gender,house_no,street_name,city,
state,pincode,dept_id,      section_name) VALUES('191006345','Satynder',     'Kumar',
       'Bhatia',     '15-06-2001', 'Male',       'B/35',       'Kavi Nagar',
       'Ghaziabad',  'Uttar Pradesh',     '200100',     'D01', 'B');
INSERT INTO student(roll_no,first_name,last_name,dob,
gender,house_no,street_name,city, state,pincode,dept_id,      section_name)
VALUES('190006876',  'Nikhil','Gupta',    '22-07-2000',  'Male',      'G/67',
       'Shahi Road', 'Shimla',     'Himachal Pradesh',  '223344',     'D02', 'A');
INSERT INTO student(roll_no,first_name,last_name,dob,
gender,house_no,street_name,city, state,pincode,dept_id,      section_name)
VALUES('191306286',  'Swati',      'Srivastava', '19-12-2001', 'Female',     '456',
       'Rajpur Road',       'Dehradun',   'Uttarakhand',       '114455',     'D02',
       'A');
INSERT INTO student(roll_no,first_name,middle_name,last_name,dob,
gender,house_no,street_name,city, state,pincode,dept_id,      section_name)
VALUES('191106654',  'Anmol',      'Lal', 'Ranjan',     '20-02-2002', 'Male',
       'A/987',      'Main Market',       'Mysore',     'Karnataka','102030','D05',
       'A');

INSERT INTO student_phone_no VALUES('191306280','9810012345');
INSERT INTO student_phone_no VALUES('191306280','9710112345');
INSERT INTO student_phone_no VALUES('191006345','9732112345');
INSERT INTO student_phone_no VALUES('190006876','8899453271');
INSERT INTO student_phone_no VALUES('190006876','6758493021');
INSERT INTO student_phone_no VALUES('190006876','9876534567');
INSERT INTO student_phone_no VALUES('191106654','9823451901');

INSERT INTO student_hobby VALUES('191306280','Technical');
INSERT INTO student_hobby VALUES('191006345','Non-Technical');
INSERT INTO student_hobby VALUES('190006876','Technical');
INSERT INTO student_hobby VALUES('191306286','Non-Technical');
INSERT INTO student_hobby VALUES('191106654','Technical');
INSERT INTO research_project(researchproj_id,researchproj_name,duration,
       sponsor_agency,status) VALUES('RP101','Attendance using ML',  '6
Months','Government',      'Complete');
INSERT INTO research_project(researchproj_id,researchproj_name,duration,
       sponsor_agency,status) VALUES('RP102',    'Electrical Vehical Design  ','9
Months',      'Government', 'Ongoing');
INSERT INTO research_project(researchproj_id,researchproj_name,duration,
       sponsor_agency,status) VALUES('RP103','Car Electric Battery Storage System',
       '12 Months',  'Self',       'Complete');
INSERT INTO research_project(researchproj_id,researchproj_name,duration,
       sponsor_agency,status) VALUES('RP104',    'Road Crack identification System',
       '15 Months',  'Government', 'Ongoing');
INSERT INTO research_project(researchproj_id,researchproj_name,duration,
       sponsor_agency,status) VALUES('RP105',    'Chip design for wireless system', '10
Months',      'Self',       'Ongoing');

INSERT INTO mini_project(miniproj_id,miniproj_name,domain,subject,description,roll_no)
VALUES('MP001','Recommender System',      'Artificial Intelligence',  'Machine
Learning',    'It seeks to predict the rating a user would give to an
item.','191306280');
INSERT INTO mini_project(miniproj_id,miniproj_name,domain,subject,description,roll_no)
VALUES('MP002',      'Malware detection System', 'Network Security',  'Cryptography',
       ' Malware has become a big risk in todays time','191006345');
INSERT INTO mini_project(miniproj_id,miniproj_name,domain,subject,description,roll_no)
VALUES('MP003','Tumor detection System',  'Artificial Intelligence',  'Deep Learning',
       'AI is used to construct models to predict any kind of tumor','190006876');
INSERT INTO mini_project(miniproj_id,miniproj_name,domain,subject,description,roll_no)
VALUES('MP004',      'Supply chain management',  'Blockchain', 'HyperLedger',
       'The management of the flow of goods and services','191306286');
INSERT INTO mini_project(miniproj_id,miniproj_name,domain,subject,description,roll_no)
VALUES('MP005',      'Opinion Mining',    'Artificial Intelligence',  'Machine
Learning',    'It is a text analysis technique that uses computational linguistics and
natural language processing','191106654');

INSERT INTO student_course VALUES('191306280',   'KCS301');
INSERT INTO student_course VALUES('191006345',   'KCS301');
INSERT INTO student_course VALUES('191006345',   'KCS401');
INSERT INTO student_course VALUES('191306286',   'KEC301');
INSERT INTO student_course VALUES('190006876',   'KEC301');
INSERT INTO student_course VALUES('191306286',   'KAS101');
INSERT INTO student_course VALUES('191106654',   'KME502');
INSERT INTO student_course VALUES('191106654',   'KAS101');

INSERT INTO faculty_course VALUES('1231','KCS301');
INSERT INTO faculty_course VALUES('1231','KCS401');
INSERT INTO faculty_course VALUES('9765','KCS401');
INSERT INTO faculty_course VALUES('1987','KEC301');
INSERT INTO faculty_course VALUES('1987','KAS101');
INSERT INTO faculty_course VALUES('2765','KME502');
INSERT INTO faculty_course VALUES('1734','KAS101');

INSERT INTO faculty_research_project      VALUES('1231',      'RP101');
INSERT INTO faculty_research_project      VALUES('1231',      'RP102');
INSERT INTO faculty_research_project      VALUES('1987',      'RP103');
INSERT INTO faculty_research_project      VALUES('1734',      'RP104');
INSERT INTO faculty_research_project      VALUES('1734',      'RP105');
INSERT INTO faculty_research_project      VALUES('1734',      'RP101');
```