Introduction to relational databases

Introduction to relational databases

- A relational database is based on the relational model and uses a collection of tables to represent both data and the relationship among those data.
- It also includes DDL and DML
- Relational database was originally defined by Edger Codd at IBM Research center in 1970.
- In relational database, user only needs to understand logical structure of data, not how it is physically stored.
- Data is represented using tables → consists of rows and columns.
- A relational database simply a collection of tables.

Relational database basic concepts

i) Relations or table
- A relation is defined as set of tuples that have the same attribute
- A relation is usually described as table, which is organized into rows and columns.

ii) Base and derived relation
- In relational database, all data are stored and accessed using relation.
- Relation / table which store data are called base relations.
- Relations which do not store data, but are computed by applying relational operator are called Derived relation.

iii) Tuple / Row / Record
- It holds all information about one item
- Example: all information like roll, name, age, address, age, mark etc., of a particular student.

iv) Field / Column
- A field holds one piece of information about an item.
- Field is column in database table.
- Example: age of all the student.

v) Constraints
- Condition specified for a particular data.
- Constraints restrict data that can be stored in relations.
- Example: User can set constraints that a given integer attribute should be between 1 & 10.

vi) Data type
- Every field in a database table is assigned a data types, which describe the kind of data that can be stored in the field.

vii) Stored procedure

- A stored procedure is a high-end database tool that adds programming power into database.
- Stored procedure is executable code generally stored in database.
- DBA will often create stored procedures to handle insert, edit and update of records.
- Front end programmer calls the stored procedure to utilize its functions.
- It makes programming code easier.

viii) Indices

- An index is one way of providing quicker access to data.
- It can be created on any combination of attributes on a relation.
- Relational database typically support multiple indexing technique.
- Indexing technique used are B Tree, B+ Tree.

ix) Normalization

- Normalization is used to eliminate the duplication of data.
- It is an integral part of the relational model.
- It prevents data manipulation anomalies and loss of data integrity.

Keys

1. Primary key

- A primary key is a field that uniquely identifies each record in the table.
- Primary key are essential in the relational database
- Example: "rollno" can be considered as primary key.
- The PRIMARY KEY constraint uniquely identifies each record in a table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A table can have only one primary key, which may consist of single or multiple fields.
- A table can contain only one primary key constraint.
- All columns defined within a primary key constraint must be defined as not null. If nullability is not specified, all columns participating in a primary key constraint have their nullability set to not null.

2. Foreign Key

- A foreign key is a field (or collection of fields) in one table that uniquely identifies a row of another table or the same table.
- A foreign key is a reference to a key in another relation.
- Primary key of one table is used as foreign key in another table.
- Example: Consider two tables(subject & timetable)
  - Primary key of "subject" table (subject_id) is used as foreign key in "timetable" table.
- A FOREIGN KEY is a key used to link two tables together.

- A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.
- The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

Look at the following two tables:
"Persons" table:

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

"Orders" table:

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

- Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.
- The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.
- The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.
- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
- The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

3. Super key
- A super key is a set of one or more attributes (columns), which can uniquely identify a row in a table.
- Example:
    - o customer_id → is super key because this attribute is sufficient to distinguish one customer tuple to another.
    - o Customer_id & customer_name → is also a super key of the relation.
    - o Customer_name → is not super key because there may the one or more customers with same name.

4. Candidate Key
- A super key with no redundant attribute is known as candidate key.

- Candidate keys are selected from the set of super keys, the only thing we take care while selecting candidate key is that the candidate key should not have any redundant attributes.

Example of super & candidate key

Lets take an example of table "Employee". This table has three attributes: Emp_Id, Emp_Number & Emp_Name. Here Emp_Id & Emp_Number will be having unique values and Emp_Name can have duplicate values as more than one employees can have same name.

Emp_Id	Emp_Number	Emp_Name
E01	2264	Tamil
E22	2278	Selva
E23	2288	Kavi
E45	2290	Tamil

Super key

1. {Emp_Id}
2. {Emp_Number}
3. {Emp_Id, Emp_Number}
4. {Emp_Id, Emp_Name}
5. {Emp_Id, Emp_Number, Emp_Name}
6. {Emp_Number, Emp_Name}

Identify Candidate key

1. {Emp_Id} – No redundant attributes
2. {Emp_Number} – No redundant attributes
3. {Emp_Id, Emp_Number} – Redundant attribute. Either of those attributes can be a minimal super key as both of these columns have unique values.
4. {Emp_Id, Emp_Name} – Redundant attribute Emp_Name.
5. {Emp_Id, Emp_Number, Emp_Name} – Redundant attributes. Emp_Id or Emp_Number alone are sufficient enough to uniquely identify a row of Employee table.
6. {Emp_Number, Emp_Name} – Redundant attribute Emp_Name.

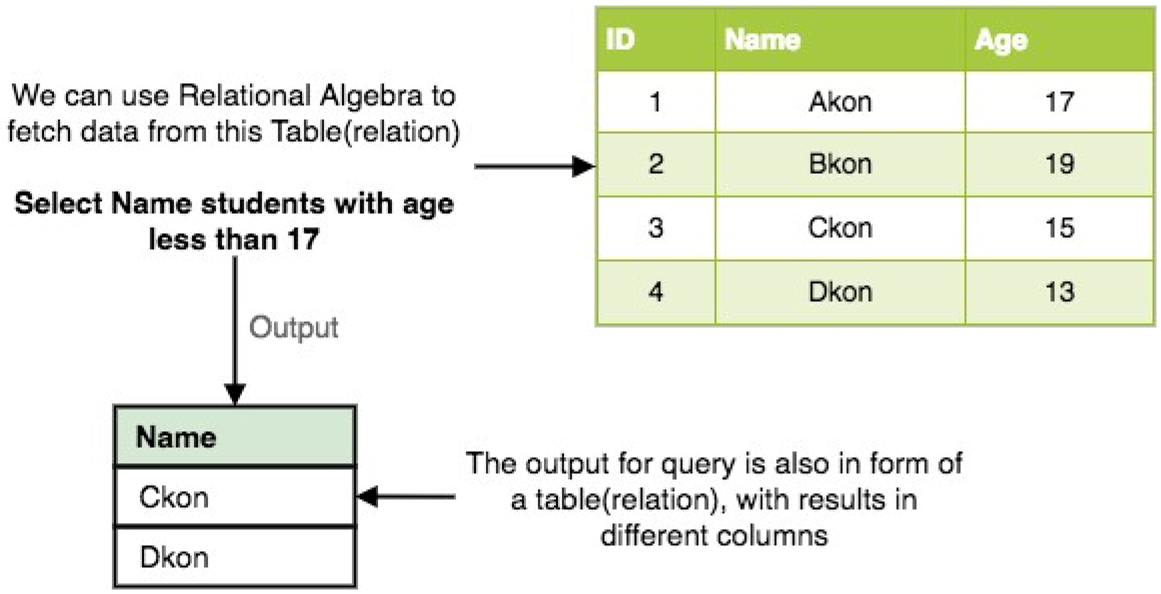
The **candidate keys** we have selected are:
{Emp_Id}
{Emp_Number}

Relational Algebra

- A query language is a language in which a user requests information from the database,
- These languages are usually on the level higher than that of standard language.
- Query language can be categorized into two types
  - Procedural Language
    - User instructs the system by giving a specific procedure to perform a sequence of operations on the database to compute the desired result.
  - Non-procedural language
    - User describes the desired information without giving a specific procedure for obtaining that information.
- Relational Algebra is a procedural query language.
- It consists of a set of operations that take one or two operations as input and produce new relation as their result.
- Relational algebra is a widely used procedural query language. It collects instances of relations as input and gives occurrences of relations as output. It uses various operation to perform this action.
- Relational algebra operations are performed recursively on a relation. The output of these operations is a new relation, which might be formed from one or more input relations
- Definition of relational algebra
  - A basic expression in a relational algebra consists of either a relation in the database or a constant relation is written by listing its tuples within { }.
  - Let E1 & E2 be relational algebra expression
  - Relational algebra expression are,

E1 U E2
E1 ∩ E2
E1 - E2
E1 x E2

- **Relational Algebra** is a procedural query language used to query the database tables to access data in different ways.
- In relational algebra, input is a relation(table from which data has to be accessed) and output is also a relation(a temporary table holding the data asked for by the user).


We can use Relational Algebra to fetch data from this Table(relation)
Select Name students with age less than 17
Output
The output for query is also in form of a table(relation), with results in different columns

ID	Name	Age
1	Akon	17
2	Bkon	19
3	Ckon	15
4	Dkon	13

Name
Ckon
Dkon

- Relational Algebra works on the whole table at once, so we do not have to use loops etc to iterate over all the rows(tuples) of data one by one. All we have to do is specify the table name from which we need the data, and in a single line of command, relational algebra will traverse the entire given table to fetch data for you.

- Relational algebraic operations are divided into 2 groups
- **Set operations**
  1) Union
  2) Intersection
  3) Set difference
  4) Cartesian Product
- **Relational algebraic operation**
  1) Select
  2) Project
  3) Rename
- **Additional algebraic operation**
  1) Join
  2) Division

I) Fundamental operations of relational algebra

- Conditions to be satisfied to perform set operation (union, intersection, difference) are,
  - Two relation must contain same number of columns
  - Columns of each table must be same data types.

- Consider two table (Depositor & Borrower)

Depositor

cust_name	**city**
Tamil	Erode
Kavitha	Gopi
Selva	Chml
Durai	Chennai
Kutty	Erode
Samy	Gopi

Borrower

cust_name	**city**
Bala	Gopi
Abi	Erode
Tamil	Erode
Sathya	Chml
Selva	Chml
Siva	Salem

- The above tables are compatible and contain same number of field with common datatype.

1) Union

- Union is a relation that includes all tuples that are either in depositor or borrower or in both.
- Duplicates will be eliminated

Depositor U Borrower

cust_name	**city**
Tamil	Erode
Kavitha	Gopi
Selva	Chml
Durai	Chennai
Kutty	Erode
Samy	Gopi
Bala	Gopi
Abi	Erode
Sathya	Chml
Siva	Salem

2) Intersection

- Intersection is a relation which includes all tuples that are in both depositor and borrower

Depositor ∩ Borrower

cust_name	city
Tamil	Erode
Selva	Chml

3) Difference

- Difference operator form relation that contain all tuples in depositor but not in borrower.

Depositor - Borrower

cust_name	city
Kavitha	Gopi
Durai	Chennai
Kutty	Erode
Samy	Gopi

4) Cartesian Product

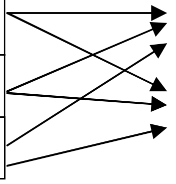
- Cartesian product is also known as cross product or cross join.
- It is denoted by "x"

Relation X x Relation Y

- If Relation X, has 2 columns (3 rows)
- Relation Y, has 2 columns (2 rows)
- Resultant Relation, has X+Y columns → 4 columns
  X * Y rows → 6 rows

Publisher

Pub_code	Pub_name
P1	McGraw
P2	Pearson
P3	PHI



Book

Book_id	Title
B1	DBMS
B2	Python

Pub_code	Pub_name	Book_id	Title
P1	McGraw	B1	DBMS
P2	Pearson	B1	DBMS
P3	PHI	B1	DBMS
P1	McGraw	B2	Python

P2	Pearson	B2	Python
P3	PHI	B2	Python

Relational algebraic operation

1) Select

- Select operation is used selects tuples that satisfy the given predicate from a relation
- This is used to fetch rows (tuples) from table(relation) which satisfies a given condition.
- It is represented as,

$$\boxed{\sigma_{\text{<select\_condition>}} (R)}$$

- σ → Symbol of select operation
- <select_condition> → Expression of condition
- R → Relation / Table

Example (book)

ID	TITLE	PRICE	YEAR
1	DBMS	250	2000
2	CP	350	2015
3	PYTHON	450	2009
4	DS	500	2010

1. Display book having price 500

$$\boxed{\sigma_{\text{price=500}} (\text{book})}$$

ID	**TITLE**	**PRICE**	**YEAR**
4	**DS**	**500**	**2010**

2. Display all books having price greater than 300

$$\boxed{\sigma_{\text{price>300}} (\text{book})}$$

ID	**TITLE**	**PRICE**	**YEAR**
2	**CP**	**350**	**2015**

3	PYTHON	450	2009
4	DS	500	2010

3. Display all books having price greater than 300 and year before 2010

$$\sigma_{\text{price>300 and year>2010}}(\text{book})$$

ID	TITLE	PRICE	YEAR
3	PYTHON	450	2009

2) Project

- Project operation selects certain columns from the table while discarding others.
- It projects column(s) that satisfy a given predicate.
- Project operation is used to project only a certain set of attributes of a relation. In simple words, If you want to see only the **names** all of the students in the **Student** table, then you can use Project Operation.
- It will only project or show the columns or attributes asked for, and will also remove duplicate data from the columns.
- It is represented as,

$$\prod_{\text{<attribute\_list>}}(R)$$

- ∏ → Symbol of project operation
- <attribute_list> → List of attributes of relation R to be projected
- R → Relation / Table

1. Display all titles of book available in relation “book”

$$\prod_{\text{title}}(\text{book})$$

TITLE
DBMS
CP
PYTHON
DS

2. Display all titles of book with its price

$$\prod_{\text{title,price}}(\text{book})$$

TITLE	PRICE
DBMS	250
CP	350
PYTHON	450
DS	500

Composite of select and project operation

- Relational operator "select" and "project" can be combined to form a complicated query.

$$\boxed{\Pi_{\text{title,price}}(\sigma_{\text{price}>350}(\text{book}))}$$

TITLE	PRICE
PYTHON	450
DS	500

3) Rename

- In relational algebra user can rename a relation/attributes or both.

$\rho_{s(\text{new\_attribute\_name})}(R)$
$\rho_{s}(R)$
$\rho_{(\text{new\_attribute\_name})}(R)$

- ρ → Symbol of rename operation
- s → name of the new relation to be renamed
- < *new_attribute_name* > → new attributes name
- R → Relation / Table

Example

Consider a relation "book" with attribute (ID, title, author, price)

id	title	author	price

1. Rename both relation and attribute name

$\boxed{\rho_{\text{temp(id1,title1,author1,price1)}} \text{(book)}}$

Temp

id1	title1	author1	price1

2. Rename table name

$\boxed{\rho_{\text{table}} \text{(book)}}$

table

id	title	author	price

3. Rename only attributes

$\boxed{\rho_{\text{(token,title,name,amount)}} \text{(book)}}$

Book

token	Title	name	amount

II) Additional Relation operators

- Fundamental operation of relational algebra are,
  - Select, Project, rename
  - Union, difference, Cartesian product etc.,
- These operations are sufficient to express any query in relational algebra.
- But, certain queries are lengthy to express,
  - Hence to simplify common queries, we use additional operations.
- Additional operations are,
  - Natural Join
  - Assignment operation

a) Natural Join

- Natural join is a binary operation that allows us to combine certain
  - Selection &
  - Cartesian product into one operation

It is denoted by Join Symbol (⋈)

- Natural join operation
  - Forms Cartesian product of two arguments

- Perform selection forcing equality on those attributes that appear in both selection
- Finally remove duplicates

Example

Employee

Code	Name
E1	Tamil
E2	Selva
E3	Kavi
E4	Kutty

Salary

Code	Salary
E1	25000
E2	23000
E3	20000
E4	30000

To display names of all employee with salary

$$\boxed{\prod_{\text{name,salary}}(\sigma_{\text{employee.code = salary.code}}(\text{employee x salary}))}$$

Above query can be rewritten using Natural Join

$$\boxed{\prod_{\text{name,salary}}(\text{employee} \bowtie \text{salary}))}$$

- Query using natural join is simpler than Cartesian product
- Natural join considers only those pairs of tuples that have same value on code attribute in both relations.
- Output

Name	Salary
Tamil	25000
Selva	23000
Kavi	20000
Kutty	30000

b) Assignment

- Assignment operator is denoted by $(\leftarrow)$
- It works like assignment in a programming language
- Example

$$\boxed{\textbf{temp} \leftarrow \sigma_{\text{roll=1}}(\text{student})}$$

- Result of right side will be assigned to the variable at left side

III) Extended Relational Algebra operations

- Arithmetic Operation
- Aggregate operation
- Outer Join

a) Arithmetic operation

- Generalized projection operation extends projection operation by allowing arithmetic functions to be used in the projection list.
- Syntax

$$\boxed{\Pi\ F1,F2,\ldots,Fn\ (E))}$$

Here,

F1,F2,…Fn → arithmetic expressions

E → any relational algebraic expression

- Example

Student

Roll	Name	Total
1	Tamil	350
2	Selva	400
3	Kavi	375
4	Kutty	425

$$\boxed{\Pi_{\text{name,total/5 as percentage}}\ (\text{student})}$$

Name	Percentage
Tamil	70
Selva	80
Kavi	75
Kutty	85

b) Aggregate functions

- Aggregate function takes a collection of values and returns a single value as a result.
- Common aggregate functions are
  - Sum
  - Avg
  - count_distinct
  - max
  - min etc.,

- Example (emp)

Emp_code	Name	Salary	Dept
E1	Tamil	25000	CSE
E2	Selva	23000	IT
E3	Kavi	20000	IT
E4	Kutty	30000	CSE

1) Calculate total salary paid to employees

$$\boxed{G_{sum(salary)}(emp)}$$

G → Group By symbol

Salary
98000

2) Calculate average salary paid to employees

$$\boxed{G_{avg(salary)}(emp)}$$

Salary
24500

3) Count number of distinct dept in the emp table

$$\boxed{G_{count\_distinct(dept)}(emp)}$$

2

4) Display total salary of employee department wise

$$\boxed{dept\ G_{sum(salary)}(emp)}$$

Dept	Salary
CSE	55000
IT	43000

c) Outer Join

- Outer join operation is an extension of join operation
- It is used to deal with mission information

- Types of outer join
  - Left outer join
  - Right outer join
  - Full outer join

Example

Emp

Name	City
Tamil	Kvp
Selva	Chml
Kavi	Gopi
Kutty	Chennai

Emp_salary

Name	Dept	Salary
Tamil	CSE	25000
Selva	IT	23000
Durai	IT	20000
Kutty	CSE	30000

Natural Join

Emp ⋈ Emp_Salary

Name	City	Dept	Salary
Tamil	Kvp	CSE	25000
Selva	Chml	IT	23000
Kutty	Chennai	CSE	30000

Here, details of "kavi & durai" are lost. To avoid the loss of information, we can use outer join

1) Left outer Join

- It takes all the tuples in the left relation
- Pad tuples with NULL values the do not available in the right relation

Emp ⟕ Emp_Salary

Name	City	Dept	Salary
Tamil	Kvp	CSE	25000
Selva	Chml	IT	23000
Kavi	Gopi	NULL	NULL
Kutty	Chennai	CSE	30000

2) Right outer Join

- It takes all the tuples in the right relation
- Pad tuples with NULL values the do not available in the left relation

Emp ⟖ Emp_Salary

Name	City	Dept	Salary
Tamil	Kvp	CSE	25000
Selva	Chml	IT	23000
Durai	NULL	IT	20000
Kutty	Chennai	CSE	30000

3) Full outer Join

- It takes all the tuples from both left and right relation
- Pad tuples with NULL values the do not available.

Emp ⟗ Emp_Salary

Name	City	Dept	Salary
Tamil	Kvp	CSE	25000
Selva	Chml	IT	23000
Kavi	Gopi	NULL	NULL
Kutty	Chennai	CSE	30000
Durai	NULL	IT	20000

SQL fundamentals

- SQL is the standard command set used to communicate with the relational database system
- SQL → Structured Query Language
- **Characteristics**
  - SQL usage is extremely flexible
  - SQL optimize the result
  - SQL query can be written in a variety of ways.
- **Advantage**
  - SQL is a high level language that provides a greater degree of abstraction.
  - SQL enables the end users to deal with number of database where it is available.
  - SQL is simple and easy to learn
  - SQL can handle complex situation.
- **SQL Literals**
  - **String literals** are always surrounded by single quotes (').

For example:

'Tamil Selvan'
'This is a literal'
'123'

- These string literal examples contain of strings enclosed in single quotes.
- **Integer literals** can be either positive numbers or negative numbers, but do not contain decimals. If you do not specify a sign, then a positive number is assumed.
- Here are some examples of valid integer literals:

  536
  +536
  -536

- **Decimal literals** can be either positive numbers or negative numbers and contain decimals. If you do not specify a sign, then a positive number is assumed.
- Here are some examples of valid decimal literals:

  24.7
  +24.7
  -24.7

- **Datetime literals** are character representations of datetime values that are enclosed in single quotes.
- Here are some examples of valid datetime literals:

  'April 30, 2015'
  '2015/04/30'
  '2015/04/30 08:34:25'

- SQL Commands
  - DDL
  - DML
  - DCL
  - TCL

- **SQL Data type**
  - Data types are a classification of a particular type of information.
  - It is easy for humans to distinguish between different types of data.
  - SQL support following data types

1) varchar

- represents a varying length string whose maximum length is 'n' characters
- Example:

  name varchar(n) → name varchar(10) → tamil / kavitha etc.,

2) **character**
   - represents a fixed length string whose exact length should be specified 'n'
   - Example

     name character(6) → 'Selvan"

3) **Number**
   - Represent numbers i.e., integer values.
   - Number can also includes decimal point values
   - Example

     age number(3)

4) **Integer**
   - Represents integer values
   - Example

     roll int(3)

5) **Float**
   - Represent floating point numbers
   - Example

     height float(5,2) → floating point with two decimal values.

6) **Date**
   - Calendar date containing
     - 4 digit year
     - Month
     - Day of the month
   - Example

     dob date

7) **Time**
   - Time of the day in hours, minutes and seconds
   - Example

     arrival_time time

8) **Timestamp**
   - Combination of date and time
   - Example

     departure timestamp

Advanced SQL features
Embedded SQL

- The SQL standard defines ***embeddings of SQL in a variety of programming*** languages such as C, Java, and Cobol
- A language to which SQL queries are embedded is referred to as a ***host language***, and the SQL structures permitted in the host language comprise embedded SQL.
- EXEC SQL statement is used to ***identify embedded SQL*** request to the preprocessor

```
EXEC SQL <embedded SQL statement > END_EXEC
```

- Exact syntax for embedded SQL request depends on the language in which SQL is embedded
- Example in JAVA embedding of SQL uses the Syntax

```
#SQL { <embedded SQL Statements > };
```

- Before executing SQL, program must connect to database

```
EXEC SQL connect to server user username using password;
```

- Here
  - Server → identifies the server to which connection to be established.
  - Database implementation requires (username and password)

- Syntax for declaring variable. It can be used within Embedded SQL but it must be proceeded with colon (:), which distinguish from SQL variable.

```
EXEC SQL BEGIN DECLARE SECTION
        int credit_amount;
EXEC SQL END DECLARE SECTION
```

- To write relational query, we user ***declare cursor*** statement.
- Query to find ID & Name of all the students who have taken more than the specified "credit_amount"

```
EXEC SQL
        declare c cursor for
        select ID, name
        from student
        where tot_credit > :credit_amount;
END-EXEC
```

- Here
    - Variable c → is called cursor for the query. This variable is used to identify the query in the ***open*** statement.

- Open statement causes the query to be evaluated and save the result within temporary relation.
- Fetch statement causes the values of one tuple to be placed in host-language variables.

```
EXEC SQL
        open c
END-EXEC
EXEC SQL
        fetch c into :si,:sn;
END-EXEC
```

- Here
    - si → hold student ID value
    - sn → holds student name
- Fetch statement produces a tuple of the result relation.
- It fetch single tuple from the database.
- To obtain all the tuples of the result, loop can be used.

- User must use close statement to tell the database to delete the temporary relation that held the result of the query.

```
EXEC SQL
        close c
END-EXEC
```

Dynamic SQL

- Dynamic SQL component of SQL allows programs to construct and submit SQL queries at runtime.
- Using Dynamic SQL, program can create SQL queries as strings at runtime and it can be executed immediately or kept prepared for subsequent use.
- Preparing a dynamic SQL statement compiles it, and it will be used in future as compiled version.
- **Dynamic** SQL is SQL statements that are constructed at runtime; for example, the application may allow users to enter their own queries.

- **Dynamic** SQL is a programming technique that enables you to build SQL statements dynamically at runtime
- SQL defines standard embedding dynamic SQL calls in a host language, such as C, C++, Java, VB etc.,

```
char *sqlprg = "update account set balance=balance*1.05
                    where account_number=?;
EXEC SQL prepare dynprog from :sqlprg;
char account[10]="A101";
EXEC SQL execute dynprog using :account;
```

- Here,
  - Dynamic SQL program contain "?", which is a placeholder for a value that is provided when the SQL program is executed.
- ODBC and JDBC are used to connect application program to database.

Difference between static and dynamic SQL

Static (Embedded) SQL	Dynamic (Interactive) SQL
In Static SQL, how database will be accessed is predetermined in the embedded SQL statement.	In Dynamic SQL, how database will be accessed is determined at run time.
SQL statements are compiled at compile time.	SQL statements are compiled at run time.
Parsing, Validation, Optimization and Generation of application plan are done at compile time.	Parsing, Validation, Optimization and Generation of application plan are done at run time.
It is generally used for situations where data is distributed uniformly.	It is generally used for situations where data is distributed non uniformly.
EXECUTE IMMEDIATE, EXECUTE and PREPARE statements are not used.	EXECUTE IMMEDIATE, EXECUTE and PREPARE statements are used.
It is less flexible.	It is more flexible.

ODBC

- Open Database Connectivity (ODBC) standard defines a way for an application program to communicate with database server.
- ODBC defines an Application Program Interface (API) that application can use to,
  - Open a connection with a database
  - Send queries and updates
  - Get back results.

- Applications such as
    - Graphical interfaces
    - Spreadsheet
    - MS Access etc, make use of the same ODBC API to connect to any database server that supports ODBC.

JDBC

- Java Database Connectivity (JDBC) standard defines an API that Java programs can use to connect to database servers.
- JDBC code contains
    - Class.forName → loads the appropriate drivers for the database.
    - getConnection → specifies the machine name where the server runs.
        - getConnection includes protocol to be used to communicate with the database, username and password.
    - Then program creates a statement handle on the connection and uses it to execute an SQL statement
        - Stmt.executeQuery
    - It can retrieve the set of rows in the result into "ResultSet" and fetch them one tuple at a time using the next() function on the result set.

Sample JDBC Code

```
public static void JDBCExample(String dbid,String userid, String passwd)
{
        try
        {
                Class.forName("oracle.jdbc.driver.OracleDriver");
                Connection con=DriverManager.getConnection("jdbc:oracle",userid,passwd);
                Statement stmt = con.createStatement();
                try
                {
                        stmt.executeUpdate("insert into account values(1,'Tamil',10000)");
                }
                catch(SQLException e)
                {
                        System.out.println("Could not insert"+e);
                }
                ResultSet rs=stmt.executeQuery("select id,name from account
                                                        where balance>5000");
                while(rs.next())
                {
```

```
                                System.out.println(rs.getInt("id")+" " + rs.getString("name"));
                      }
                      stmt.close();
                      con.close();
            }
            catch(SQLException e)
            {
                      System.out.println("SQLExeception: "+e);
            }
}
```