Databases

Session Objectives

- **The Relational Data Model**
- **Codd's Rules**
- **Relational Algebra**
- **The SQL Language**
- **Postgres architectural concepts**
- **Creating and destroying databases using Postgres**

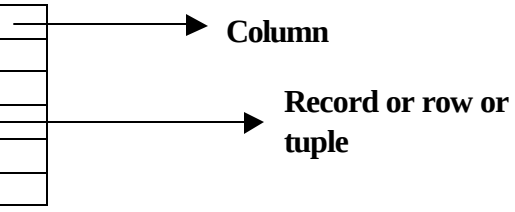
The Relational Data Model

A database is an integrated collection of data. This data is used and retrieved together for one or more application systems. Databases can include student details in a school or an employee's details in an organization.

A relational database is a database that is perceived by its users as a collection of tables. A table consists of rows and columns where each row represents a record and each column represents an attribute of the records contained in the table.

For example, we can have a table called student that contains sno, sname, and marks.

Sno	Sname	Marks
1	Rajesh	87
2	Ravi	69
3	Ramesh	98
4	Raja	100
5	Rakesh	57



Codd's Rules

A DBMS should obey Codd's twelve rules for it to be relational. These twelve rules are as discussed below:

The information rule

All information is explicitly and logically represented in tables as data values.

The rule of guaranteed access

Every item of data must be logically addressable with the help of table name, primary key value and column name. From this it is clear that any individual item or record can be retrieved with the use of a table name, primary key value of the row and the column name where it is to be found.

The systematic treatment of all null values

The DBMS must be able to support NULL values to represent missing or inapplicable information. They must be distinct from zeros and spaces. NULL values for all the data types must be the same.

One of the most important aspects that must be noted here is that there is a vast difference between a null value and zero or a space.

The database description rule

A description of database is maintained using the same logical structures with which data was defined by the DBMS. These are accessible to users with appropriate authority and are stored in the data dictionary.

Comprehensive data sub language

According to this rule, the system must support the following:

Data definition
View definition
Data manipulation
Integrity constraints
Authorization
Transaction management operations

The view updating rule

All views that are theoretically updateable must be updateable by the system

The insert and update rule

A single operand must hold good for all retrieval, update, delete and insert activities. This rule implies that all the data manipulation commands must be operational on set of rows in relation rather than on a single row.

The physical independence rule

Application programs must remain unimpaired when any changes are made in storage representation or access methods.

The logical data independence rule

The changes that are made should not affect the user's ability to work with the data. The change can be splitting the table into many more tables.

The integrity independence rule

The integrity constraints should be stored in the system catalog or in the database as a table.

The distribution rule

The system must be able to access or manipulate the data that is distributed in other systems. This is the distribution rule.

The non subversion rule

The non-subversion rule states that different levels of the language cannot subvert or bypass the integrity rules and constraints. To put in simple words, if any RDBMS supports a lower language then it should not bypass any integrity constraints defined in the higher level.

Relational Data Elements

Entity

Entity is a distinguishable object about which we are going to record data.

E.g. Student, Course

Attributes

It is a unique piece of information about an entity that is required in the scope of an application.

For example, the Customer entity might contain such attributes about a customer as the customer number, customer name, region and so on.

Table

A two-dimensional data structure in the "physical" database. A table is composed of rows and columns. Tables have unique names. There are no duplicate column names.

Relation

An association between two entities. The following types of relations are possible.

One-to-One

Eg., one student can be assigned only one roll number and one roll number can be given to only one student.

One-to-Many or Many-to-One

Eg., one student can register for multiple courses or one course can be offered to many students.

Many-to-Many

Many vendors can sell multiple items and many items can be sold by multiple vendors.

Relational Algebra

The Relational Algebra was introduced by E. F. Codd in 1972. It consists of a set of operations on relations:

- **UNION** (∪): Builds the union of two tables. Given the tables R and S (both must have the same entity), the union R U S is the set of tuples that are in R or S or both.

Consider 2 tables A and B. A contains roll_no and names of all students doing the PROGRESS course. B contains roll_no and names of all students doing the PROACT course. These two tables are union compatible.

A

Roll_no	Name
111	Ira
112	Jai

UNION

B

Roll_no	Name
115	Bob
111	Ira

AB

Roll_no	*Name*
111	*Ira*
112	*Jai*
115	*Bob*

- **INTERSECT** (∩): Builds the intersection of two tables. Given the tables R and S, R∩S is the set of tuples that are in R and in S. We again require that R and S have the same entity.

Consider the same tables A and B. The student Ira is doing two courses simultaneously. Hence her name appears in both the tables. An intersect operation on the tables A and B extract the row common to both the relation. The intersect operation works on union compatible tables.

A B

Roll_no	Name
111	Ira

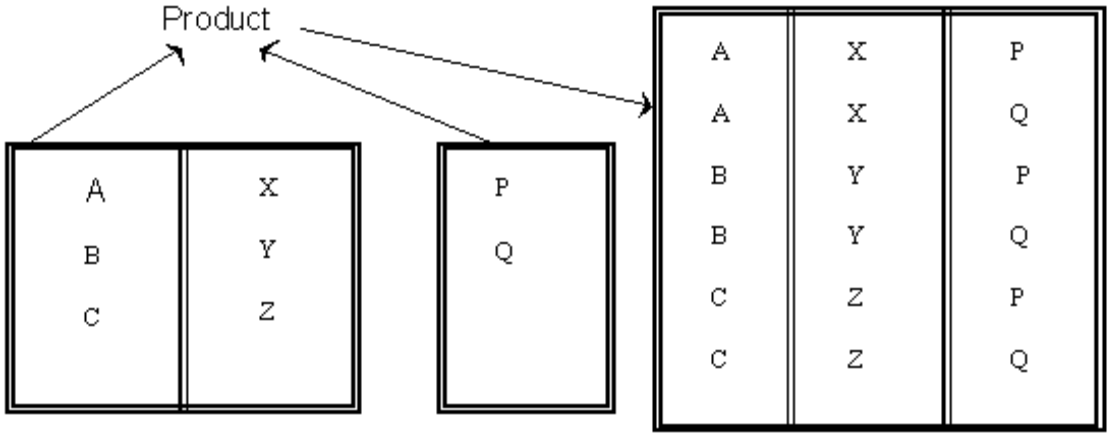
- **DIFFERENCE** (-): Builds the set difference of two tables. Let R and S again be two tables with the same entity. R - S is the set of tuples in R but not in S.

In case of tables A and B the operation 'A difference B" will extract all those rows that belong to A but not to B.

A difference B

Roll_no	Name
112	Jai

- **PRODUCT**: Builds the Cartesian product of two relations. It concatenates every row in one relation with every row in another.



- **JOIN** : Connects two tables by their common attributes.

Eg.

Student

Roll_no	Name	Course_id
111	Ram	P1
112	Sam	MCSD
113	Joy	P1

JOIN

Course

Course_id	Course_nm
P1	PROACT
P2	PROGRESS
MCSD	MS Cert. Soln. Devl.

Roll_no	Course_id	Course_nm
111	P1	PROACT
112	MCSD	MS Cert. Soln. Devl.
113	P1	PROACT

In the above case there are two tables – student and course.

Student table contains the columns roll_no,student_nm and course_id .Course_id is the course for which the student has registered. Course Table contains the columns course_id,course_nm. The common column between the two tables is course_id.

To find out the course name for which the student has enrolled perform a join operation based on the common column course_id.

- **SELECT** : Extracts tuples from a relation that satisfy a given restriction.

Eg. Consider a student table with the attributes Roll Number (Roll_no), Student Name (Stud_nm) and Course.
The condition specified is to extract the tuples of only those students doing the Linux course.

Roll_no	Student_nm	Course
111	Raj	LINUX
112	Anu	LINUX

- **PROJECT** : Extracts specified attributes (columns) from a relation.

Eg. If only the student_nm and course is to be extracted the resultant output will be as follows:

Student_nm	Course
Raj	LINUX
Anu	LINUX
Uma	C++
Jo	LINUX

Data Types

When we define a table we have to decide which attributes to include. Additionally we have to decide which kind of data is going to be stored as attribute values. For example the values of **sname** from the table **student** will be character strings, whereas **sno** will store integers. We define this by assigning a data type to each attribute. The type of **sname** will be **varchar2(20)** (this is the SQL type for character strings of length <= 20), the type of **sno** will be **integer**. With the assignment of a data type we also have selected a domain for an attribute. The domain of **sname** is the set of all character strings of length <= 20, the domain of **sno** is the set of all integer numbers.

The SQL Language

SQL is a relational language. It is based on the relational data model first published by E.F. Codd in 1970. Here is a list of some features provided by SQL:

- Non-procedural language, enabling to concentrate more on the data Presentation.
- Common language for all relational databases.
- Interacts with the Database using SQL Commands
- Operate on Records
- Print and store calculated fields
- Produce formatted Reports.

The SQL statements are broadly classified as

- Data Definition Language **(DDL)** - These commands are used to Create, Alter and Drop objects.
- Data Query Language **(DQL)** – It is used for Querying and Viewing data in a desired Format.
- Data Manipulation Language **(DML)** - These Commands are used to Add, Modify, and Delete data in the database.
- Data Control Language **(DCL)** – These SQL Commands provide security to Database Objects such as Tables, Views etc.

What is Postgres?

Traditional relational database management systems (DBMSs) support a data model consisting of a collection of named relations, containing attributes of a specific type. In current commercial systems, possible types include floating point numbers, integers, character strings, money, and dates. It is commonly recognized that this model is inadequate for future data processing applications. The relational model successfully replaced previous models in part because of its simplicity.

However, as mentioned, this simplicity often makes the implementation of certain applications very difficult. Postgres offers substantial additional power by incorporating the following four additional basic concepts in such a way that users can easily extend the system:

- Classes
- Inheritance
- Types
- Functions

Other features provide additional power and flexibility:

- Constraints
- Triggers
- Rules
- Transaction integrity

These features of Postgres make it object-relational. This is distinct from object-oriented, which in general is not well suited to supporting the traditional relational database languages. So, although Postgres has some object-oriented features, it is firmly in the relational database world.

The Object-Relational Database Management System now known as PostgreSQL (and briefly called Postgres95) is derived from the Postgres package written at Berkeley. PostgreSQL is the most advanced open-source database available, supporting almost all SQL constructs (including subselects, transactions, and user-defined types and functions), and having a wide range of language bindings available (including C, C++, Java, perl, tcl, and python).

Major enhancements in PostgreSQL include:

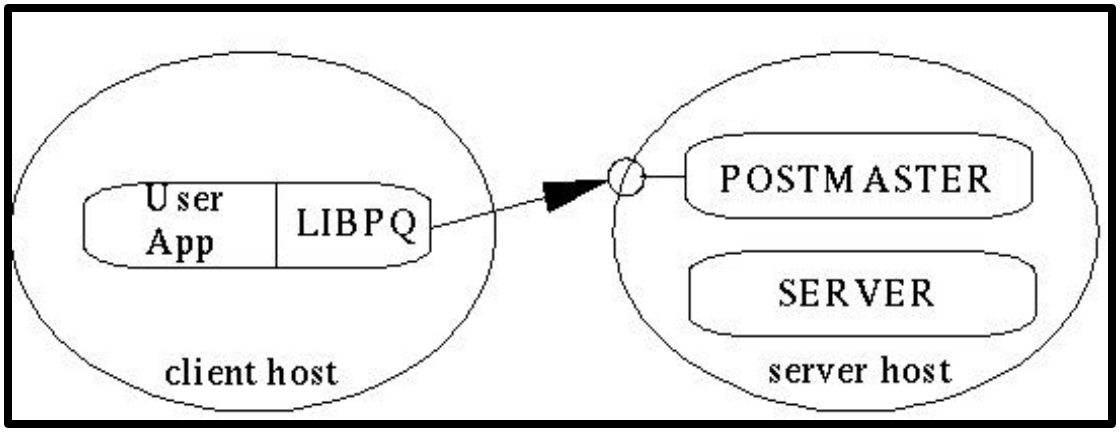
- Table-level locking has been replaced with multi-version concurrency control, which allows readers to continue reading consistent data during writer activity and enables hot backups from pg_dump while the database stays available for queries.
- Important backend features, including subselects, defaults, constraints, and triggers, have been implemented.
- Additional features have been added, including primary keys, quoted identifiers, type casting, and binary and hexadecimal integer input.
- Built-in types have been improved, including new wide-range date/time types.
- Overall backend code speed has been increased by approximately 20-40%, and backend startup time has decreased 80%

Postgres Architectural Concepts

Postgres uses a simple "process per-user" client/server model. A Postgres session consists of the following cooperating Unix processes (programs):

- A supervisory daemon process (postmaster),
- The user's front-end application (e.g., the psql program), and
- One or more backend database servers (the Postgres process itself).

A single postmaster manages a given collection of databases on a single host. Such a collection of databases is called an installation or site. Front-end applications that wish to access a given database within an installation make calls to the library. The library sends user requests over the network to the postmaster, which in turn starts a new backend server process and connects the front-end process to the new server. From that point on, the front-end process and the backend server communicate without intervention by the postmaster. Hence, the postmaster is always running, waiting for requests, whereas front-end and backend processes come and go.



Establishing a connection

The libpq library allows a single front-end to make multiple connections to backend processes. However, the front-end application is still a single-threaded process. Multithreaded front-end/backend connections are not currently supported in libpq. One implication of this architecture is that the postmaster and the backend always run on the same machine (the database server), while the front-end application may run anywhere. This should be kept in mind, because the files that can be accessed on a client machine may not be accessible (or may only be accessed using a different filename) on the database server machine.

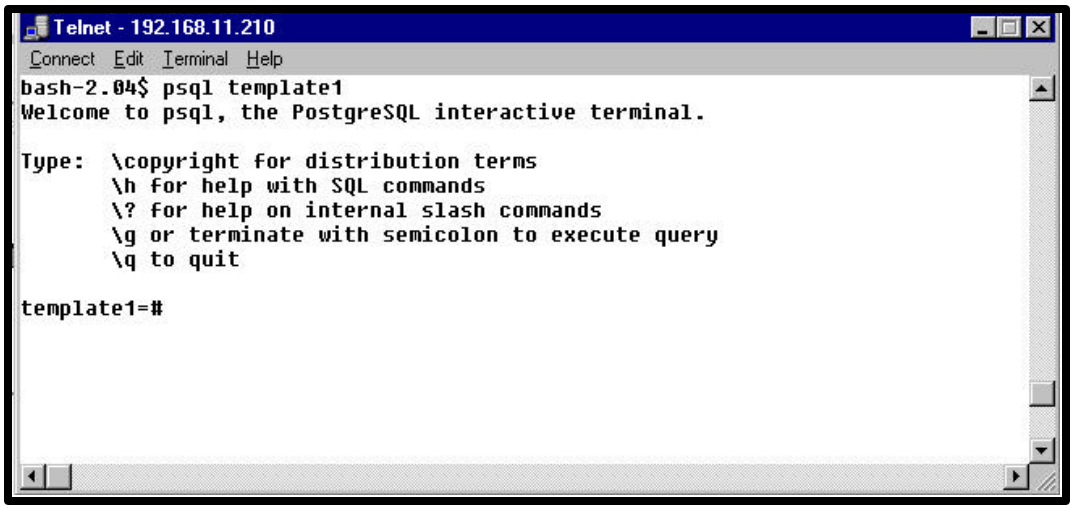
The postmaster and Postgres servers run with the user-id of the Postgres "superuser." Note that the Postgres superuser does not have to be a special user (e.g., a user named "postgres"). Furthermore, the Postgres superuser should definitely not be the Unix superuser ("root")! In any case, all files relating to a database should belong to this Postgres superuser.

Getting started with Postgres

After logging into Linux using the user name and password, type the following:

```
$ psql template1
```

template1 is the default database that is available. The output is as shown below:



psql is a terminal-based front-end to Postgres. It enables the user to type in queries interactively, issue them to Postgres, and see the query results.

In normal operation, psql provides a prompt with the name of the database to which psql is currently connected, followed by the string "=#". (in the above case, template1 is the name of the default database).

At the prompt, the user may type in SQL queries. Ordinarily, input lines are sent to the backend when a query-terminating semicolon is reached. An end of line does not terminate a query! Thus queries can be spread over several lines for clarity. If the query was sent and without error, the query results are displayed on the screen.

Creating a database

Postgres allows the user to create any number of databases at a given site and the user automatically becomes the database administrator of the database he just created. Database names must have an alphabetic first character and are limited to 32 characters in length. Not every user has authorization to become a database administrator. If Postgres refuses to create databases for the user, then the site administrator needs to grant him permission to create databases. Consult the site administrator if this occurs.

To create your own database, type

```
template1=# create database student;
CREATE DATABASE
template1=#
```

The database **student** has now been created. To come back to the **$** prompt type **\q** at the **#** prompt.. Now you can work inside the student database. Once you have constructed a database, you can access it by running the Postgres terminal monitor programs (e.g. psql) which allows you to interactively enter, edit, and execute SQL commands.

To get into the student database, type

```
$ psql student
Welcome to psql, the PostgreSQL interactive terminal.
```

```
Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help on internal slash commands
       \g or terminate with semicolon to execute query
       \q to quit
student=#
```

Now SQL commands can be typed at the prompt.

Destroying a Database

If you are the database administrator for a database, you can destroy it using the following command:

```
template1=# drop database student;
DROP DATABASE
template1=#
```

This action physically removes all of the Linux files associated with the database and cannot be undone, so this should only be done with a great deal of forethought.

If you try to drop the current database in which you are working, the following error occurs:

```
student=# drop database student;
ERROR:  DROP DATABASE: Cannot be executed on the currently open database
student=#
```

To get help on the commands available, type **\h** at the prompt.