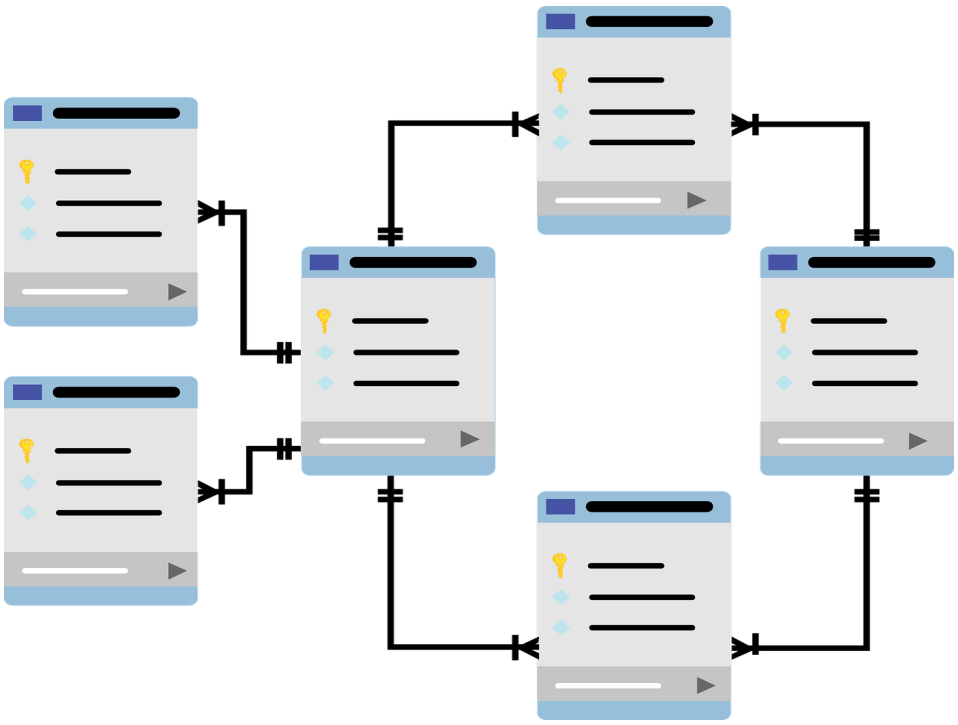


# LAZY ANALYTIC HOME

**Pandit Deendayal Petroleum University**

**Team Members:**

**Vatsal Parsaniya (17BIT028)**

**Parth Chauhan (17BIT029)**

## Introduction

This project aims to create a “LAZY” home in which we can control various electrical appliances such as fan, light, etc and also provide analytic information of various parameters such as Temperature, Humidity using sensors.

# List of Experiments:

## Experiment 1:

Advantages of DBMS over file processing system in ABC on the following points. Write meaning and example for each.

- Data redundancy and inconsistency
- Difficulty in accessing the data
- Data isolation
- Integrity problems
- Atomicity problems
- Concurrent-access anomalies
- Security problems

**Data redundancy**: Data redundancy refers to the duplication of data, let's say we are managing the data of a company where a customer is taking a product from a different location, the same customer details in such case will be stored twice, which will take more storage than needed. Data redundancy often leads to higher storage costs and poor access time.

**Data inconsistency**: Data redundancy leads to data inconsistency, let's take the same example that we have taken above, a customer is taking product from different location and we have customer address stored twice, now let's say customer requests to change his address, if the address is changed at one place and not on all the records then this can lead to data inconsistency.

**Data Isolation**: Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

**Dependency on application programs**: Changing files would lead to a change in application programs.

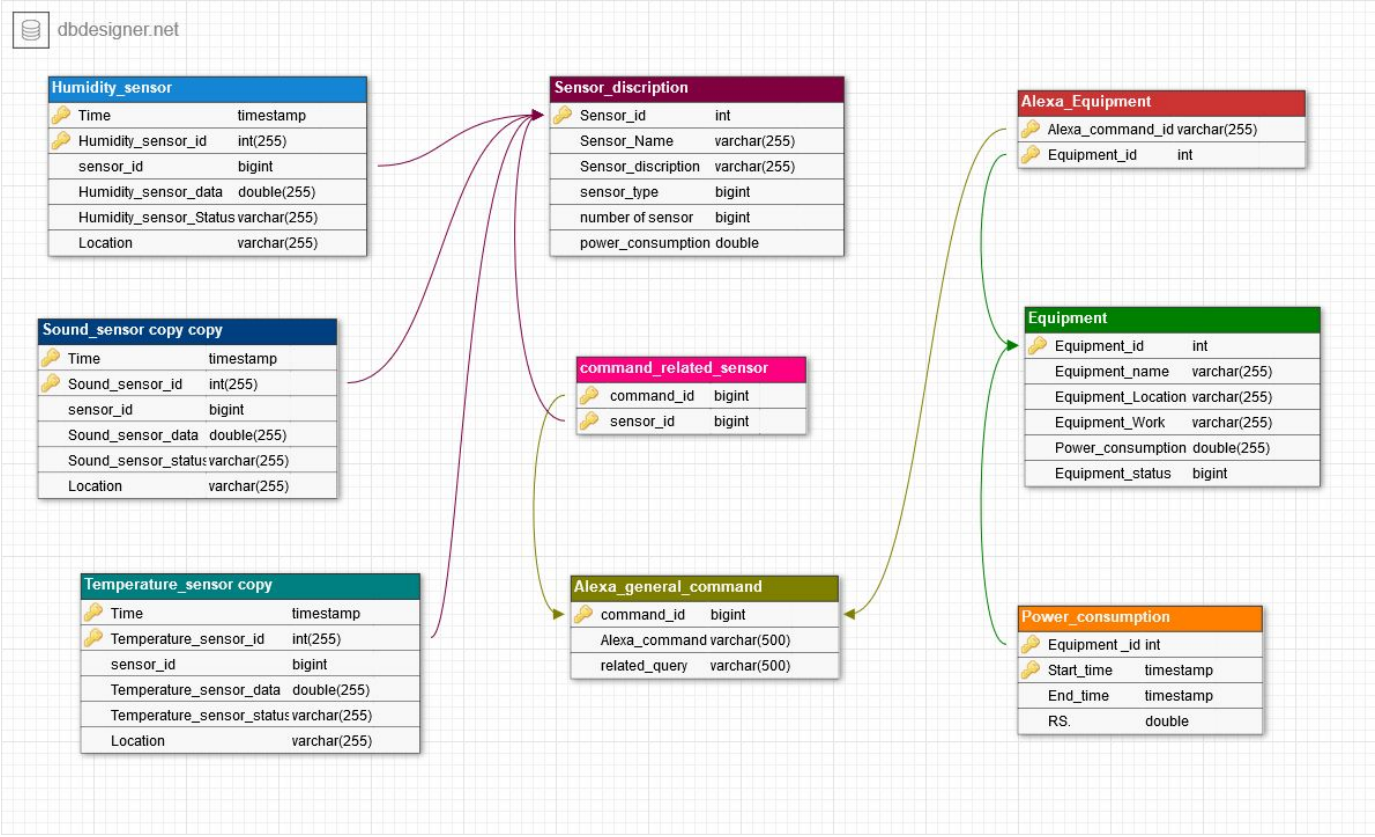
**Atomicity issues**: Atomicity of a transaction refers to “All or nothing”, which means either all the operations in a transaction executes or none. For example, let’s say Vatsal transfers 100$ to Parth’s account. This transaction consists of multiple operations such as debit 100$ from Vatsal’s account, credit 100$ to Parth’s account. Like any other device, a computer system can fail let’s say it fails after the first operation then, in that case, Vatsal’s account would have been debited by 100$ but the amount was not credited to Parth’s account, in such case the rollback of operation should occur to maintain the atomicity of transaction. It is difficult to achieve atomicity in file processing systems.

**Data Security**: Data should be secured from unauthorised access, for example, an Employee in a Company should not be able to see the payroll details of the Manager, such kind of security constraints are difficult to apply in file processing systems.

## Experiment 2:

Development of a relational model for the **Lazy Analytic Home**

- Identify the various relations that exist in the **Lazy Analytic Home**
- Identify the super-key, candidate-key, primary key and foreign key for the identified relations.
- Develop the relational model for **Lazy Analytic Home**



Humidity sensor :

- Candidate key : Time and humidity_sensor_id
- Forign key : sensor_id -> Sensor_discription.sensor_id

Sound_sensor:

- Candidate key : Time and sound_sensor_id
- Forign key : sensor_id -> Sensor_discription.sensor_id

Temperature_sensor:

- Candidate key : Time and Temperature_sensor_id
- Forign key : sensor_id -> Sensor_discription.sensor_id

Sensor_discripction:

- Primary Key : sensor_id

Command_related_sensor:

- Primary_key: {Command_id,sensor_id}
- Forign key : command_id->Sensor_discripction.command_id
- Forign key : sensor_id->Sensor_discription.sensor_id

Alexa_equipment:

- Primary_key: {Command_id,Equipment_id}
- Forign key : Equipment_id -> Equipment.Equipment_id

Equipment:

- Primary_key:Equipment_id

Power_consumption:

- Primary_key : {equipment_id,start_time}
- Forign key :equipment_id-> equipment_id.equipment_id

## Experiment 3:

Relational Algebra queries for the **Lazy Analytic Home**. Clearly write the definition as well as relational algebra queries for each. Develop at least 3 queries using each of the following operators

- Selection
- Projection
- Cartesian product
- Union
- Set difference
- Natural join
- Composition of any two from (1-6) operators

## Selection

- **Selection is used to select required tuples of the relations.**

1) **σ (Sensor_Id)(**Sensor_discription**)**
2) **σ (Sensor_Id,Sensor_Name)(**Sensor_discription**)**
3) **σ (Alexa_command,Equipment_id)(**Alexa_Equipment**)**
4) **σ (Equipment_id,Equipment_name)(**Equipment**)**
5) **σ (Equipment_id,RS)(**Power_consumption**)**

## Projection

- **Projection is used to project required column data from a relation. By Default projection removes duplicate data.**

1. **∏(Sensor_Id)(σ (Sensor_Id)(**Sensor_discription**))**
2. **∏(Equipment_id,RS)(σ (Equipment_id,RS)(**Power_consumption**))**
3. **∏(Time,Temperature_sensor_data)**
    i. **(σ (Time,Temperature_sensor_data)(**Temperature_sensor**))**
4. **∏(Alexa_command,Equipment_id)(**
    **σ (Alexa_command,Equipment_id)(**Alexa_Equipment**))**

## Cartesian Product:

- **Cross product is used to join two relations. For every row of Relation1, each row of Relation2 is concatenated. If Relation1 has m tuples and Relation-2 has n tuples, cross product of Relation-1 and Relation-2 will have m X n tuples.**

1. **σ** (**Sensor_Id=1**)(Sensor_discription **X** Power_consumption)
2. **σ** (**Temperature_sensor_data>=30**)(Temperature_sensor **X** Humidity_sensor)
3. **σ** (**Temperature_sensor_status=1**)(Temperature_sensor **X** Sound_sensor)

## Union:

- **Union on two relations R1 and R2 can only be computed if R1 and R2 are union compatible (These two relations should have same number of attributes and corresponding attributes in two relations have the same domain) . Union operator when applied on two relations R1 and R2 will give a relation with tuples which are either in R1 or R2.**

1. ∏(**Sensor_Id**)(Sensor_discription) **U** ∏(**Sensor_Id**)(Number_of_sensor)
2. ∏(**Time**)(Current_status) **U** ∏(**Time**)(Temperature_sensor_id)
3. ∏(**Temperature_sensor_status**)(Current_status) **U** **(**∏(**Temperature_sensor_status**)(Temperature_sensor)**)**

## Set Difference:

 **Minus on two relations R1 and R2 can only be computed if R1 and R2 are union compatible. Minus operator when applied on two relations as R1-R2 will give a relation with tuples which are in R1 but not in R2.**

1. ∏(**Sensor_Id**)(Sensor_discription) **-** ∏(**Sensor_Id**)(Current_status)
2. ∏(**Time**)(Current_status) **-** ∏(**Time**)(Humidity_sensor_id)
3. ∏(**Temperature_sensor_status**)(Current_status) **-** **(**∏(**Temperature_sensor_status**)(Temperature_sensor)**)**

## Natural Join:

 **Natural join is a binary operator. Natural join between two or more relations will result set of all combinations of tuples where they have equal common attribute.**

1. ∏(Sensor_discription) ⋈ ∏(Current_status)
2. ∏(Current_status) ⋈ ∏ (Humidity_sensor_id)
3. ∏(Current_status) ⋈ ∏(Temperature_sensor)
4. ∏(Temperature_sensor) ⋈ ∏ (Temperature_sensor)

## Experiment 4:

Development of the E-R model for the **Lazy Analytic Home**

- Identify the various entities and entity sets that exist in the **Lazy Analytic Home**
- Identify the super-key, candidate-key and primary key for the identified entity sets.
- Identify the all possible relations that exist between entity sets
- Develop the E-R model for **Lazy Analytic Home**

## Experiment 5:

Installation of the relational database management system

- sudo apt-get update
- sudo apt-get install mysql-server
- sudo mysql_secure_installation systemctl start mysql
- sudo mysql -u root -p

## Experiment 6:

Introduction to SQL, DDL, DML, DCL, database and table creation, alteration, defining Constraints, primary key, foreign key, unique, not null, check, IN operator.

**Introduction to SQL**:

- The most widely used commercial language
- SQL is NOT a Turing machine equivalent language
- To be able to compute complex functions SQL is usually embedded in some higher-level language
- Application programs generally access databases through one of
- Language extensions to allow embedded SQL
- Application program interface (e.g., ODBC/JDBC) which allow SQL queries to be sent to a database

**DDL (Data Definition Language)**:

DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

**Examples of DDL commands**:

- CREATE – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
- DROP – is used to delete objects from the database.
- ALTER-is used to alter the structure of the database.
- TRUNCATE–is used to remove all records from a table, including all spaces allocated for the records are removed.
- COMMENT –is used to add comments to the data dictionary.
- RENAME –is used to rename an object existing in the database.

**DML (Data Manipulation Language)**: The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

**Examples of DML**:

- INSERT – is used to insert data into a table.
- UPDATE – is used to update existing data within a table.
- DELETE – is used to delete records from a database table.

**DCL (Data Control Language)**: - DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

**Examples of DCL commands**

- GRANT- gives user's access privileges to the database.
- REVOKE- withdraw user's access privileges given by using the GRANT command.

**Database and table creation**:

CREATE DATABASE **database name**;

**database name**: Name of the database.

CREATE TABLE **table name**

(

column1 datatype(size),

column2 datatype(size),

column3 datatype(size),

....

);

**table name**: name of the table

**column1**: Name of the first column

**datatype**: Type of data we want to store in a particular column. For example, int for integer data.

**size**: Size of the data we can store in a particular column. For example, if for a column we specify the data type as int and size as 10 then this column can store an integer number of maximum 10 digits.

**ALTERATION**:-

- **Alter Table ADD**:

  ALTER TABLE table-name

  ADD

  (Columnname_1 datatype,

  Columnname_2 datatype,

  …

  Columnname_n datatype);

- **Alter Table DROP**:

  ALTER TABLE table-name

  DROP COLUMN column name;

- **Alter Table MODIFY**:

  ALTER TABLE table name

  MODIFY column_name column_type;

- **Alter Table RENAME**:

  ALTER TABLE table-name RENAME TO new_table_name;

**CHECK:**

Check Constraint is used to specify a predicate that every tuple must satisfy in a given relation. It limits the values that a column can hold in a relation.

The predicate in check constraint can hold a subquery.

Check constraint defined on an attribute restricts the range of values for that attribute.

If the value is added to an attribute of a tuple violates the check constraint, the check constraint evaluates to false and the corresponding update is aborted.

Check constraint is generally specified with the CREATE TABLE command in SQL.

**Syntax**:

```
CREATE TABLE pets
(
ID INT NOT NULL,
Name VARCHAR(30) NOT NULL,
Breed VARCHAR(20) NOT NULL,
Age INT,
GENDER VARCHAR(9),
PRIMARY KEY(ID),
check(GENDER in ('Male', 'Female', 'Unknown'))
);
```

**Note**: The check constraint in the above SQL command restricts the GENDER to belong to only the categories specified. If a new tuple is added or an existing tuple in the relation is updated with a GENDER that doesn't belong to any of the three categories mentioned, then the corresponding database update is aborted.

**IN**:

The SQL IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

IN Syntax

SELECT column name(s)

FROM table-name

WHERE column-name IN (value1, value2, ...);

## Experiment 7:

**Study and use of inbuilt SQL functions - aggregate functions, Built-in functions**

**Numeric, date, string functions**

**Summary:** in this tutorial, we will introduce you to aggregate function concepts and common SQL aggregate functions.

**Introduction to SQL aggregate functions**

An aggregate function allows you to perform a calculation on a set of values to return a single scalar value. We often use aggregate functions with the GROUP BY and HAVING clauses of the SELECT statement.

The following are the most commonly used SQL aggregate functions:

• AVG – calculates the average of a set of values. • COUNT – counts rows in a specified table or view. • MIN – gets the minimum value in a set of values. • MAX – gets the maximum value in a set of values. • SUM – calculates the sum of values.

Notice that all aggregate functions above ignore NULL values except for the COUNT function.

Numeric Functions are used to perform operations on numbers and return numbers.

Following are the numeric functions defined in SQL:

ABS (): It returns the absolute value of a number.

Syntax: SELECT ABS (-243.5);

Output: 243.5

SQL  SELECT ABS (-10);

+-----------------------------------+

| ABS (10)

+-----------------------------------+

| 10

+-----------------------------------+

ACOS (): It returns the cosine of a number.

Syntax:  SELECT ACOS (0.25);

Output: 1.318116071652818

ASIN (): It returns the arc sine of a number.

Syntax: SELECT ASIN (0.25);

Output: 0.25268025514207865

ATAN (): It returns the arc tangent of a number.

Syntax: SELECT ATAN (2.5);

Output: 1.1902899496825317

CEIL (): It returns the smallest integer value that is greater than or equal to a number.

Syntax: SELECT CEIL (25.75);

Output: 26

CEILING (): It returns the smallest integer value that is greater than or equal to a number.

Syntax: SELECT CEILING (25.75);

Output: 26

COS (): It returns the cosine of a number.

Syntax: SELECT COS (30);

Output: 0.15425144988758405

COT (): It returns the cotangent of a number.

Syntax: SELECT COT (6);

Output: -3.436353004180128

DEGREES (): It converts a radian value into degrees.

Syntax: SELECT DEGREES (1.5);

Output: 85.94366926962348

SQL SELECT DEGREES(PI ());

+--------------------------------------+

| DEGREES (PI ())

+--------------------------------------+

| 180.000000

+--------------------------------------+

DIV (): It is used for integer division.

Syntax: SELECT 10 DIV 5;

Output: 2

EXP (): It returns be raised to the power of number.

Syntax: SELECT EXP (1);

Output: 2.718281828459045

FLOOR (): It returns the largest integer value that is less than or equal to a number.

Syntax: SELECT FLOOR (25.75);

Output: 25

GREATEST (): It returns the greatest value in a list of expressions.

Syntax: SELECT GREATEST (30, 2, 36, 81, 125);

Output: 125

LEAST (): It returns the smallest value in a list of expressions.

Syntax: SELECT LEAST (30, 2, 36, 81, 125);

Output: 2

LN (): It returns the natural logarithm of a number.

Syntax: SELECT LN (2);

Output: 0.6931471805599453

LOG10(): It returns the base-10 logarithm of a number.

Syntax: SELECT LOG (2);

Output: 0.6931471805599453

LOG2(): It returns the base-2 logarithm of a number.

Syntax: SELECT LOG2(6);

Output: 2.584962500721156

MOD (): It returns the remainder of n divided by m.

Syntax: SELECT MOD (18, 4);

Output: 2

PI (): It returns the value of PI displayed with 6 decimal places.

Syntax: SELECT PI ();

Output: 3.141593

POW (): It returns m raised to the nth power.

Syntax: SELECT POW (4, 2);

Output: 16

RADIANS (): It converts a value in degrees to radians.

Syntax: SELECT RADIANS (180);

Output: 3.141592653589793

RAND (): It returns a random number.

Syntax: SELECT RAND ();

Output: 0.33623238684258644

ROUND (): It returns a number rounded to a certain number of decimal places.

Syntax: SELECT ROUND (5.553);

Output: 6

SIGN (): It returns a value indicating the sign of a number.

Syntax: SELECT SIGN (255.5);

Output: 1

SIN (): It returns the sine of a number.

Syntax: SELECT SIN (2);

Output: 0.9092974268256817

SQRT (): It returns the square root of a number.

Syntax: SELECT SQRT (25);

Output: 5

TAN (): It returns the tangent of a number.

Syntax: SELECT TAN (1.75);

Output: -5.52037992250933

ATAN2(): It returns the arctangent of the x and y coordinates, as an angle and expressed in radians.

Syntax: SELECT ATAN2(7);

Output: 1.42889927219073

TRUNCATE (): It returns 7.53635 truncated to 2 places right of the decimal point.

Syntax: SELECT TRUNCATE (7.53635, 2);

Output: 7.53

**DATE FUNCTIONS**

NOW (): - It Returns the current date and time

CURDATE (): - It Returns the current date

CURTIME (): - It Returns the current time

DATE (): - It Extracts the date part of a date or date/time expression

EXTRACT (): - It Returns a single part of a date/time

DATE_ADD (): -It Adds a specified time interval to a date

DATE_SUB (): - It Subtracts a specified time interval from a date

DATEDIFF (): - It Returns the number of days between two dates

DATE_FORMAT (): -It Displays date/time data in different formats

**STRING FUNCTIONS**

ASCII: - It Returns the ASCII value for the specific character

CHAR: -It Returns the character based on the ASCII code

CHARINDEX: -It Returns the position of a substring in a string

CONCAT: - It Adds two or more strings together

Concat with +: -Adds two or more strings together

CONCAT_WS: -Adds two or more strings together with a separator

DATALENGTH: -It Returns the number of bytes used to represent an expression

DIFFERENCE: -It Compares two SOUNDEX values, and returns an integer value

FORMAT: -It Formats a value with the specified format

LEFT: -It Extracts a number of characters from a string (starting from left)

LEN: -It Returns the length of a string

LOWER: -It Converts a string to lower-case

LTRIM: -It Removes leading spaces from a string

NCHAR: -It Returns the Unicode character based on the number code

PATINDEX: -It Returns the position of a pattern in a string

QUOTENAME: - It Returns a Unicode string with delimiters added to make the string a valid SQL Server delimited identifier

REPLACE: - It Replaces all occurrences of a substring within a string, with a new substring

REPLICATE: -It Repeats a string a specified number of times

REVERSE: -It Reverses a string and returns the result

RIGHT: -It Extracts a number of characters from a string (starting from right)

RTRIM: -It Removes trailing spaces from a string

SOUNDEX: -Returns a four-character code to evaluate the similarity of two strings

SPACE: -Returns a string of the specified number of space characters

STR: - It Returns a number as string

STUFF: -Deletes a part of a string and then inserts another part into the string, starting at a specified position

SUBSTRING: -It Extracts some characters from a string

TRANSLATE: - It Returns the string from the first argument after the characters specified in the second argument are translated into the characters specified in the third argument.

TRIM: -Removes leading and trailing spaces (or other specified characters) from a string

UNICODE: -Returns the Unicode value for the first character of the input expression

UPPER: -Converts a string to upper-case

## Experiment 8 :

Study, write and use the set operations, sub-queries, correlated sub-queries in SQL

**SET Operations in SQL**

SQL supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions.

In this tutorial, we will cover 4 different types of SET operations:

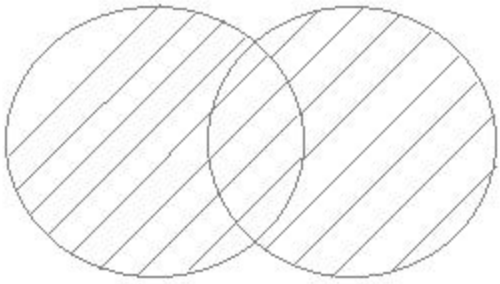
**UNION**

**UNION ALL**

**INTERSECT**

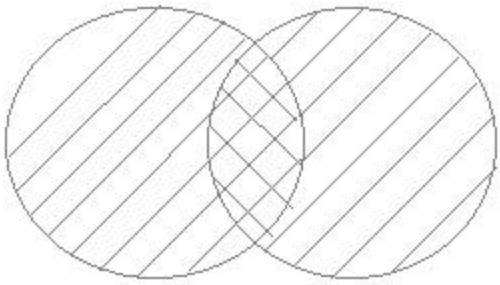
**MINUS**

UNION Operation:



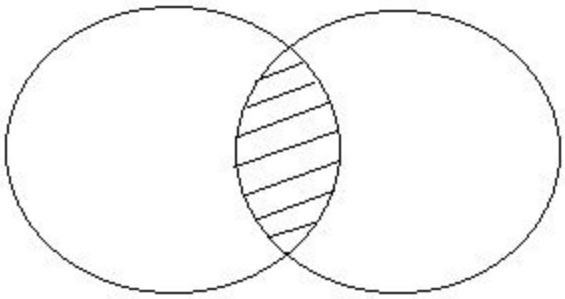
UNION is used to combine the results of two or more SELECT statements. However it will eliminate duplicate rows from its resultset. In case of union, number of columns and datatype must be same in both the tables, on which UNION operation is being applied.

UNION ALL:



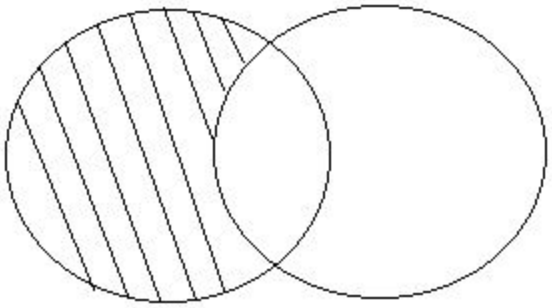
This operation is similar to Union. But it also shows the duplicate rows.

INTERSECT:



Intersect operation is used to combine two SELECT statements, but it only retuns the records which are common from both SELECT statements. In case of Intersect the number of columns and datatype must be same.

MINUS:



The Minus operation combines results of two SELECT statements and return only those in the final result, which belongs to the first set of the result.

**subquery in SQL:**

A subquery is a SQL query nested inside a larger query.

A subquery may occur in :

- A SELECT clause

- A FROM clause

- A WHERE clause

The subquery can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery.

A subquery is usually added within the WHERE Clause of another SQL SELECT statement.

You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, or ALL.

A subquery is also called an inner query or inner select, while the statement containing a subquery is also called an outer query or outer select.

The inner query executes first before its parent query so that the results of an inner query can be passed to the outer query.

**SQL Correlated Subqueries:**

w3resource

menu

Inviting useful, relevant, well-written and unique guest posts.

Custom Search

**SQL Correlated Subqueries**:

Correlated Subqueries

SQL Correlated Subqueries are used to select data from a table referenced in the outer query. The subquery is known as a correlated because the subquery is related to the outer query. In this type of queries, a table alias (also called a correlation name) must be used to specify which table reference is to be used.

The alias is the pet name of a table which is brought about by putting directly after the table name in the FROM clause. This is suitable when anybody wants to obtain information from two separate tables.

## Experiment 9 :

Study and use of group by, having, order by features of SQL

**The SQL GROUP BY Statement:**

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

***GROUP BY Syntax***

SELECT column_name(s)

FROM table_name

WHERE condition

GROUP BY column_name(s)

ORDER BY column_name(s);

**The SQL HAVING Clause:**

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

**HAVING Syntax**

SELECT column_name(s)

FROM table_name

WHERE condition

GROUP BY column_name(s)

HAVING condition

ORDER BY column_name(s);

**The SQL ORDER BY Keyword:**

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

**ORDER BY Syntax**

SELECT column1, column2, ...

FROM table_name

ORDER BY column1, column2, ... ASC|DESC;

## Experiment 10:

Study different types of join operations, Exist, Any, All and relevant features of SQL.

**SQL Joins:**

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Here are the different types of the JOINs in SQL:

(INNER) JOIN: Returns records that have matching values in both tables

LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table

RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table

FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table

**The SQL EXISTS Operator:**

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns true if the subquery returns one or more records.

**EXISTS Syntax**

SELECT column_name(s)

FROM table_name

WHERE EXISTS

(SELECT column_name FROM table_name WHERE condition);

**The SQL ANY and ALL Operators:**

The ANY and ALL operators are used with a WHERE or HAVING clause.

The ANY operator returns true if any of the subquery values meet the condition.

The ALL operator returns true if all of the subquery values meet the condition.

**ANY Syntax**

SELECT column_name(s)

FROM table_name

WHERE column_name operator ANY

(SELECT column_name FROM table_name WHERE condition);

**ALL Syntax**

SELECT column_name(s)

FROM table_name

WHERE column_name operator ALL

(SELECT column_name FROM table_name WHERE condition);

## Experiment 11:

Study and apply Database Normalization techniques

**Database Normalization** is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy(repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

Normalization is used for mainly two purposes,

- Eliminating redundant(useless) data.
- Ensuring data dependencies make sense i.e data is logically stored.

**Problems Without Normalization**

If a table is not properly normalized and has data redundancy then it will not only eat up extra memory space but will also make it difficult to handle and update the database, without facing data loss. Insertion, Updation and Deletion Anomalies are very frequent if the database is not normalized. To understand these anomalies let us take an example of a Student table.

| rollno | name | branch | hod | office_tel |
|---|---|---|---|---|
| 401 | Akon | CSE | Mr. X | 53337 |
| 402 | Bkon | CSE | Mr. X | 53337 |
| 403 | Ckon | CSE | Mr. X | 53337 |
| 404 | Dkon | CSE | Mr. X | 53337 |

In the table above, we have data of 4 Computer Sci. students. As we can see, data for the fields branch, hod(Head of Department) and office_tel is repeated for the students who are in the same branch in the college, this is Data Redundancy.

### Insertion Anomaly

Suppose for a new admission, until and unless a student opts for a branch, data of the student cannot be inserted, or else we will have to set the branch information is NULL.

Also, if we have to insert data of 100 students of the same branch, then the branch information will be repeated for all those 100 students.

These scenarios are nothing but Insertion anomalies.

### Update Anomal

What if Mr X leaves college? or is no longer the HOD of the computer science department? In that case, all the student records will have to be updated, and if by mistake we miss any record, it will lead to data inconsistency. This is Updation anomaly.

### Deletion Anomaly

In our Student table, two different pieces of information are kept together, Student information and Branch information. Hence, at the end of the academic year, if student records are deleted, we will also lose the branch information. This is Deletion anomaly.

### Normalization Rule

Normalization rules are divided into the following normal forms

- First Normal Form
- Second Normal Form
- Third Normal Form
- BCNF
- Fourth Normal Form

**First Normal Form (1NF)**

For a table to be in the First Normal Form, it should follow the following 4 rules:

It should only have single(atomic) valued attributes/columns.

Values stored in a column should be of the same domain

All the columns in a table should have unique names.

And the order in which data is stored does not matter.

**Second Normal Form (2NF)**

For a table to be in the Second Normal Form,

It should be in the First Normal form.

And, it should not have Partial Dependency.

**Third Normal Form (3NF)**

A table is said to be in the Third Normal Form when,

It is in the Second Normal form.

And, it doesn't have Transitive Dependency.

**Boyce and Codd Normal Form (BCNF)**

Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with certain type of anomaly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, following conditions must be satisfied:

R must be in 3rd Normal Form and, for each functional dependency ( $X \rightarrow Y$ ), X should be a super Key.

**Fourth Normal Form (4NF)**

A table is said to be in the Fourth Normal Form when,

It is in the Boyce-Codd Normal Form.  And, it doesn't have Multi-Valued Dependency.

# Find Minimal Cover

Step 1: Rewrite the FD into those with only one attribute on RHS. We obtain:

Step 2: Remove trivial FDs (those where the RHS is also in the LHS). We obtain:

Step 3: Minimize LHS of each FD. We obtain:

Step 4: Remove redundant FDs (those that are implied by others). We obtain:

Time Humidity_sensor_id → sensor_id

Time Humidity_sensor_id → Humidity_sensor_data

Time Humidity_sensor_id → Humidity_sensor_Status

Time Humidity_sensor_id → Location

Time Sound_sensor_id → sensor_id

Time Sound_sensor_id → Sound_sensor_data

Time Sound_sensor_id → Sound_sensor_status

Time Sound_sensor_id → Location

Time Temperature_sensor_id → sensor_id

Time Temperature_sensor_id → Location

Time Temperature_sensor_id → Temperature_sensor_data

Time Temperature_sensor_id → Temperature_sensor_status

sensor_id → Sensor_Name

sensor_id → Sensor_discription

sensor_id → Sensor_type

sensor_id → number_of_sensor

sensor_id → power_consumption

command_id → related_query

command_id → Alexa_command

Equipment_id → Equipment_Name

Equipment_id → Equipment_location

Equipment_id → Equipment_Work

Equipment_id → power_consumption

Equipment_id → Equipment_status

Equipment_id Start_time → End_time

Equipment_id Start_time → RS.

## Find Candidate Keys

**Step 1: Find the minimal cover of FDs, which contains**

**Step 2. Find the set of attributes not on the RHS of any FD, which is NotOnRHS = {Time,Humidity_sensor_id,Sound_sensor_id,Temperature_sensor_id,related_sensor_name,command_id,Equipment_id,Equipment_name,Power_consumption,Start_time}. Every CK must contain these attributes.**

**Step 3: NotOnRHS is a superkey, so it is the only candidate key**

Time

Humidity_sensor_id

Sound_sensor_id

Temperature_sensor_id

related_sensor_name

command_id

Equipment_id

Equipment_name

Power_consumption

Start_time

# Check Normal Form

## 2NF

The table is in 2NF

## 3NF

The table is in 3NF

## BCNF

The table is in BCNF

## Normalize to 2NF

Time Humidity_sensor_id ➔ sensor_id Humidity_sensor_data Humidity_sensor_Status Location

Time Sound_sensor_id ➔ sensor_id Sound_sensor_data Sound_sensor_Status Location

Time Temperature_sensor_id ➔ sensor_id Temperature_sensor_data Temperature_sensor_Status Location

Sensor_id ➔ Sensor_Name Sensor_discription sensor_type numberofsensor power_consumption

command_id ➔ Alexa_command related_query

Equipment_id ➔ Equipment_name Equipment_Location Equipment_Work Power_consumption Equipment_status

Equipment_id Start_time ➔ End_time RS.

## Normalize to 3NF

Time Humidity_sensor_id ➜ sensor_id Humidity_sensor_data Humidity_sensor_Status Location

Time Sound_sensor_id ➜ sensor_id Sound_sensor_data Sound_sensor_Status Location

Time Temperature_sensor_id ➜ sensor_id Temperature_sensor_data Temperature_sensor_Status Location

Sensor_id ➜ Sensor_Name Sensor_discription sensor_type numberofsensor power_consumption

command_id ➜ Alexa_command related_query

Equipment_id ➜ Equipment_name Equipment_Location Equipment_Work Power_consumption Equipment_status

Equipment_id Start_time ➜ End_time RS.

## Normalize to BCNF

Time Humidity_sensor_id ➜ sensor_id Humidity_sensor_data Humidity_sensor_Status Location

Time Sound_sensor_id ➜ sensor_id Sound_sensor_data Sound_sensor_Status Location

Time Temperature_sensor_id ➜ sensor_id Temperature_sensor_data Temperature_sensor_Status Location

Sensor_id ➜ Sensor_Name Sensor_discription sensor_type numberofsensor power_consumption

command_id ➜ Alexa_command related_query

Equipment_id ➜ Equipment_name Equipment_Location Equipment_Work Power_consumption Equipment_status

Equipment_id Start_time ➜ End_time RS.

## Uniqueness of project as to existing work:

- It is **truly global,** connecting local resources with platforms on the other side of the globe.
- It covers a **wide variety of industries**, including health, logistics, automotive and energy management to name a few.
- Its **value chain is far more complex** and articulated; involving service providers, module manufacturers, connectivity providers, system integrators and device manufacturers.
- Its most common business model is **Business to Business to Consumer (or B2B2C)**. Most often mobile operators hold a relationship with another business, but no longer directly with the end-user.
- For IoT **connectivity is only an enabler**, whereas in traditional telecommunications it is the core service provided.

Extra topic given to each group

# Triggers

Trigger is invoked by Oracle engine automatically whenever a specified event occurs.Trigger is stored into database and invoked repeatedly, when specific condition match.

Triggers are stored programs, which are automatically executed or fired when some event occurs.

Triggers are written to be executed in response to any of the following events.

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers could be defined on the table, view, schema, or database with which the event is associated.

## Advantages of Triggers

These are the following advantages of Triggers:

- Trigger generates some derived column values automatically
- Enforces referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

# Cursor

When an SQL statement is processed, Oracle creates a memory area known as context area. A cursor is a pointer to this context area. It contains all information needed for processing the statement. In PL/SQL, the context area is controlled by Cursor. A cursor contains information on a select statement and the rows of data accessed by it.

A cursor is used to referred to a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors:

- Implicit Cursors
- Explicit Cursors

# PL/SQL

PL/SQL is a block structured language. The programs of PL/SQL are logical blocks that can contain any number of nested sub-blocks. Pl/SQL stands for "Procedural Language extension of SQL" that is used in Oracle. PL/SQL is integrated with Oracle database (since version 7). The functionalities of PL/SQL usually extended after each release of Oracle database. Although PL/SQL is closely integrated with SQL language, yet it adds some programming constraints that are not available in SQL.

# Procedure

The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.

The procedure contains a header and a body.

- **Header:** The header contains the name of the procedure and the parameters or variables passed to the procedure.
- **Body:** The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

# Conclusion, future work and references

In the future there is a plan for adding more features like full alexa capability with nodemcu and with sql triggers we can get real time query data with alexa ask responses also there is a big plan to develop an alexa skill which helps all IOT project query with speech responses.

We have taken references from,

- Geeksforgeeks
- Javatpoint
- https://www.instructables.com
- Etc