DBMS Concepts for InfyTQ Qualifying Round for 2022 Batch by

TalentBattle

One Stop Platform for Placement Preparation

www.talentbattle.in

For Off-campus updates & Placement Preparation Join our Whatsapp group & Telegram channel here: https://bit.ly/3jyTDkg

InfyTQ Concepts PDF Notes

Complete Placement Preparatory LIVE MasterClass

Complete Placement Preparatory LIVE Mastercla

Preparing for the upcoming placement season? Battle your way through the crowd with the best placement preparation course online.

- This course is a complete end-to-end solution to your hurdles in placement preparation with a perfect blend of **LIVE** & Recorded Classes.
- Aptitude(Quantitative Aptitude, Logical Reasoning, Verbal Ability), Programming concepts, Coding Statement Preparation, Personal and Technical Interview preparation is completely covered in this course
- In this course, our experts are well known for their teaching methods which will help to understand Aptitude, Coding, Programming concepts in the easiest and quickest possible way from basic to advanced level.
- Company Specific Training Modules required for TCS, Wipro, Infosys, Accenture, Cognizant, Capgemini, Mindtree, Deloitte, Tech Mahindra, LTI, IBM, Atos-Syntel, Persistent, Hexaware, and many more companies are covered in this course according to the latest pattern.

More Details and Registration Link: https://bit.ly/39sgQSM

INDEX

INDEX

INDEX

Exam Pattern: 10 Questions

Topics:

Table Properties, Data Types, Key concepts, DCL, DDL & DML Commands, Join Concept, Clauses, Keyword, Operators, Normalization, Constraints, Mongo DB

- Database Concept
- Basic SQL
- Advanced SQL

What is Data?

-**Data** is nothing but facts and statistics stored or free flowing over a network, generally it's raw and unprocessed.

What is a Database?

-A **Database** is a collection of related data organized in a way that data can be easily accessed, managed and updated.

What is DBMS?

- A **DBMS** is a software that allows creation, definition and manipulation of database, allowing users to store, process and analyse data easily.

Characteristics of DBMS:

- Data stored into Tables
- Reduced Redundancy
- Data Consistency
- Support Multiple user and Concurrent Access
- Query Language
- Security
- Transactions

Advantages of DBMS

•Segregation of application program.

•Minimal data duplicity or data redundancy.

•Easy retrieval of data using the Query Language.

•Reduced development time and maintenance need.

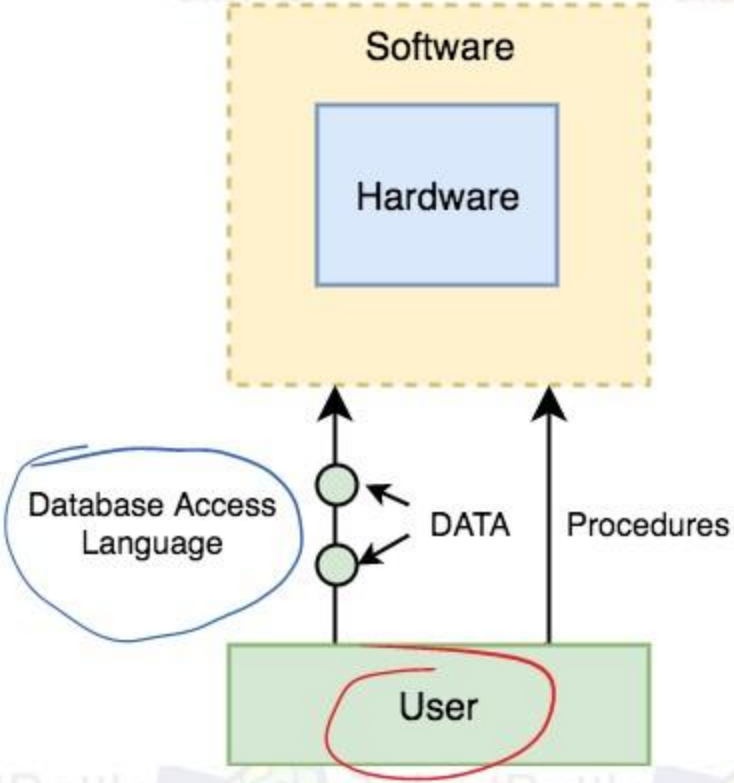
•With Cloud Datacenters, we now have Database Management Systems capable of storing almost infinite data.

•Seamless integration into the application programming languages which makes it very easier to add a database to almost any application or website.

Disadvantages of DBMS

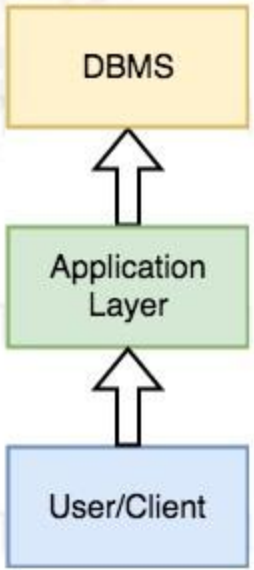
- It's Complexity
- Except MySQL, which is open source, licensed DBMSs are generally costly.
- They are large in size.

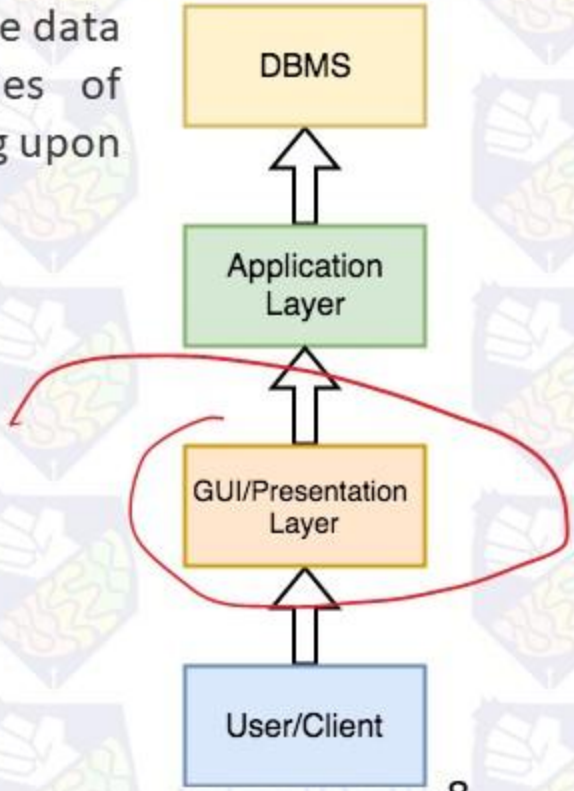
Components of DBMS



Understanding DBMS Architecture

A Database Management system can be **centralised**(all the data stored at one location), **decentralised**(multiple copies of database at different locations) or **hierarchical**, depending upon its architecture.

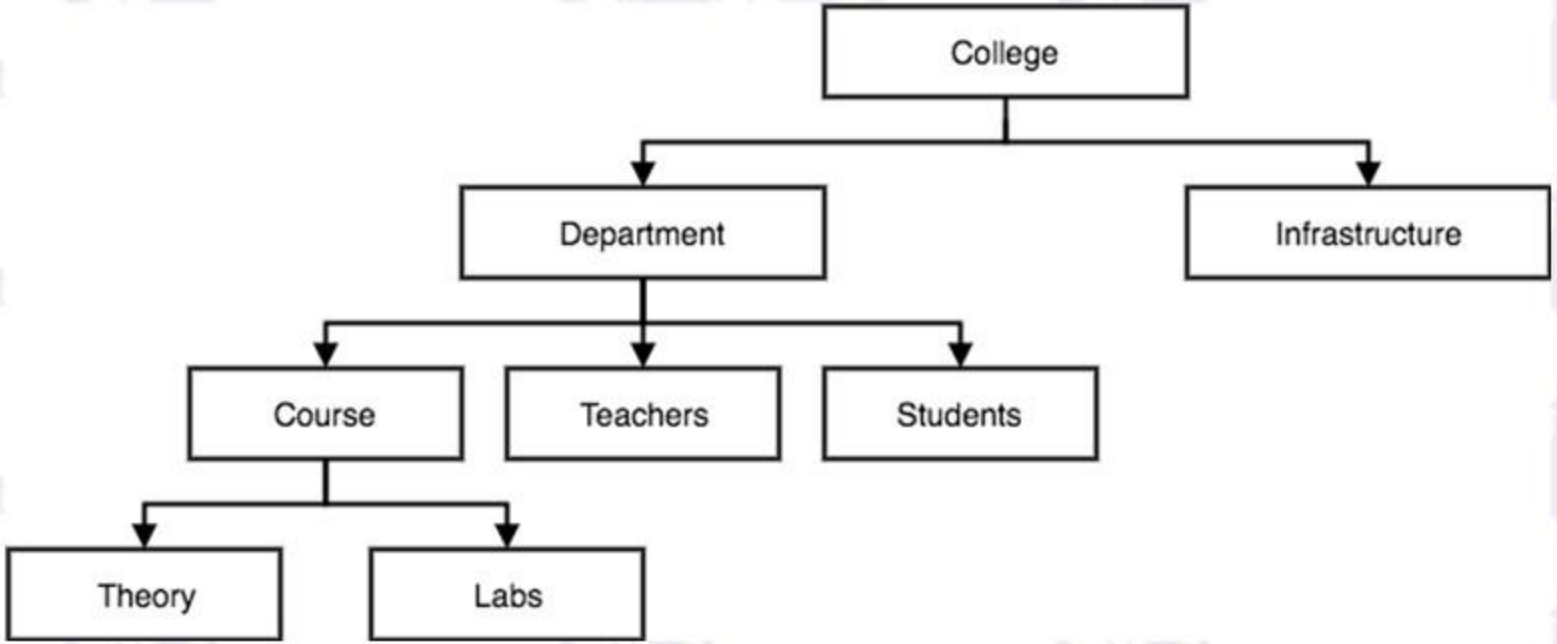




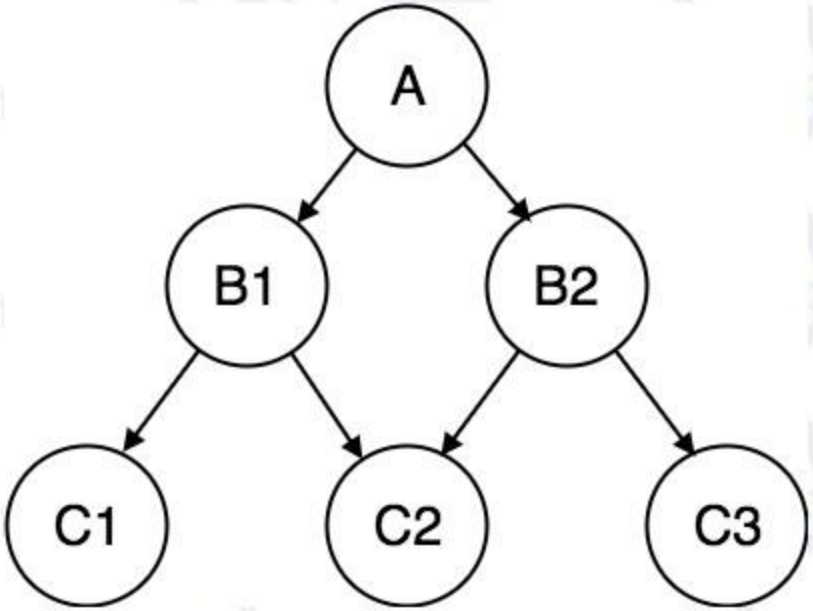
DBMS Database Models

- Hierarchical Model
- Network Model
- Entity-relationship Model
- Relational Model

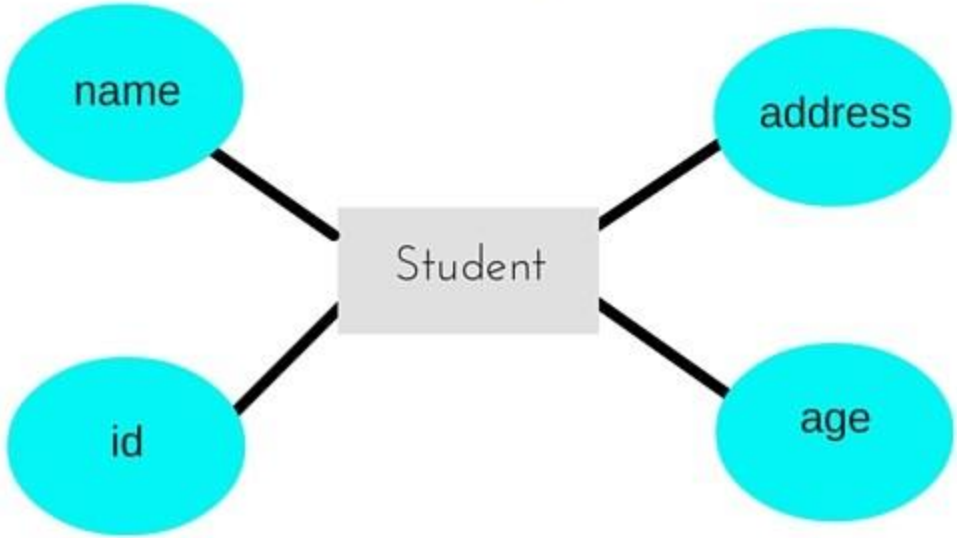
Hierarchical Model



Network Model



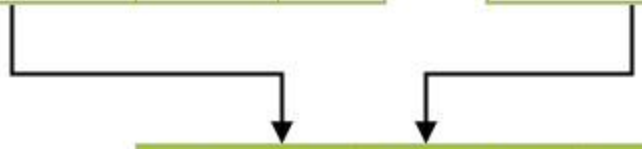
Entity-relationship Model



Relational Model

student_id	name	age
1	Akon	17
2	Bkon	18
3	Ckon	17
4	Dkon	18

subject_id	name	teacher
1	Java	Mr. J
2	C++	Miss C
3	C#	Mr. C Hash
4	Php	Mr. P H P



student_id	subject_id	marks
1	1	98
1	2	78
2	1	76
3	2	88

Basic Concepts of ER Model in DBMS

In short, to understand about the ER Model, we must understand about:

- Entity and Entity Set
- What are Attributes? And Types of Attributes.
- Keys
- Relationships

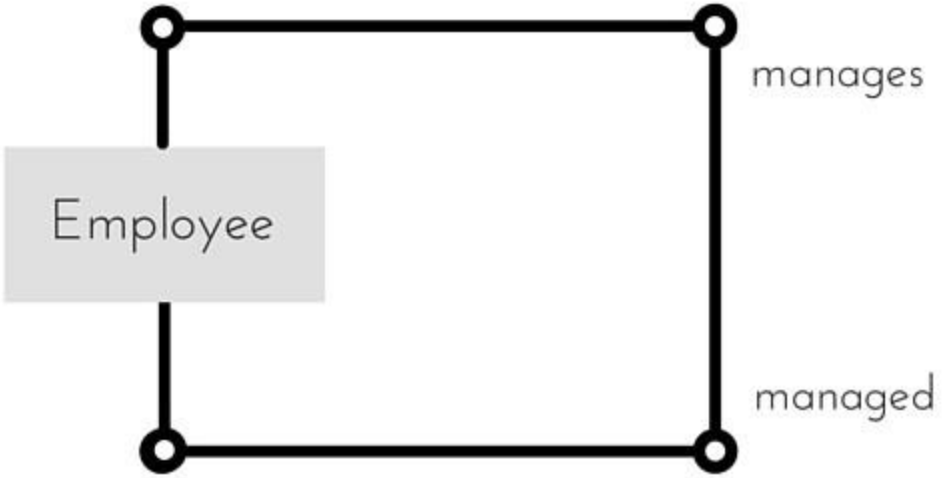
ER Diagram: Relationship

A Relationship describes relation between entities.
Relationship is represented using diamonds or rhombus.

There are three types of relationship that exist between Entities.
1.Binary Relationship
2.Recursive Relationship
3.Ternary Relationship

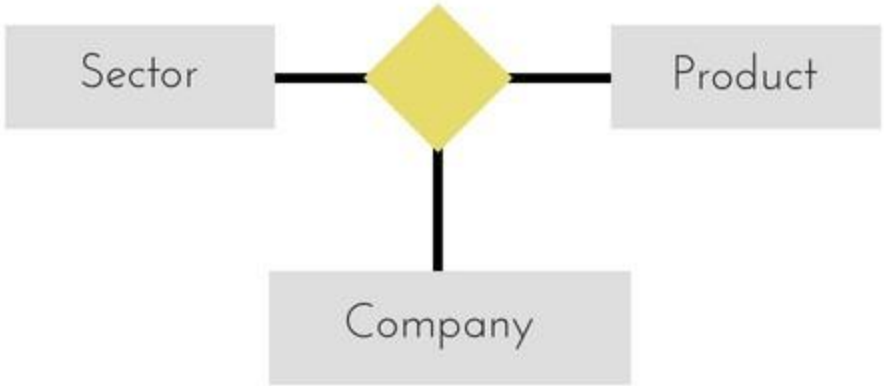
Recursive Relationship

When an Entity is related with itself it is known as **Recursive** Relationship.



Ternary Relationship

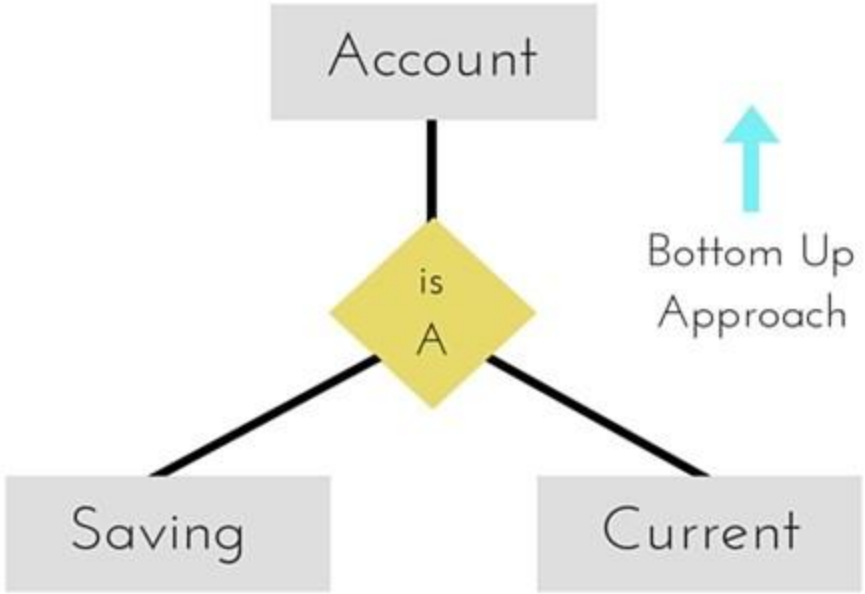
Relationship of degree three is called Ternary relationship.



- The above relationship involves 3 entities.
- Company operates in Sector, producing some Products.

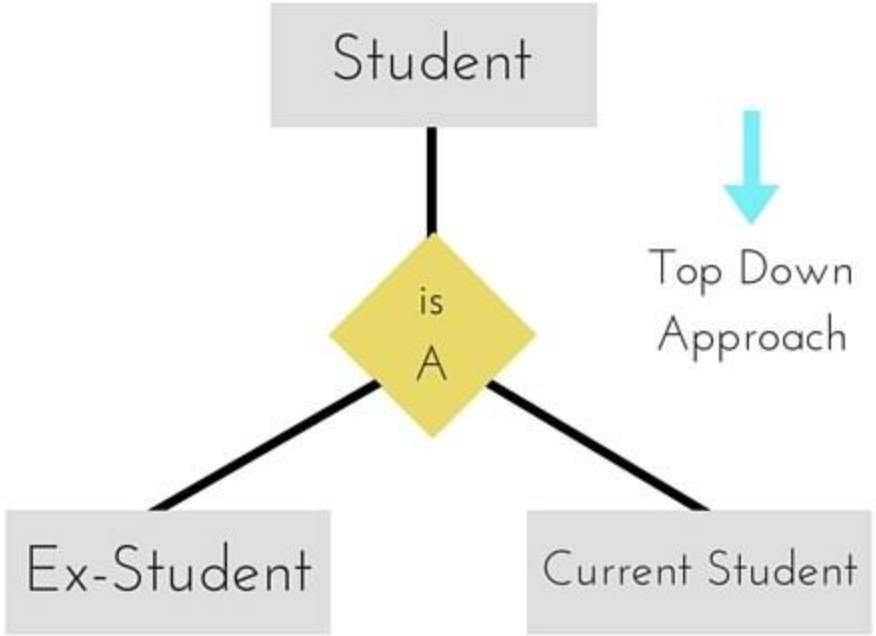
Generalization

Generalization is a bottom-up approach in which two lower level entities combine to form a higher level entity.



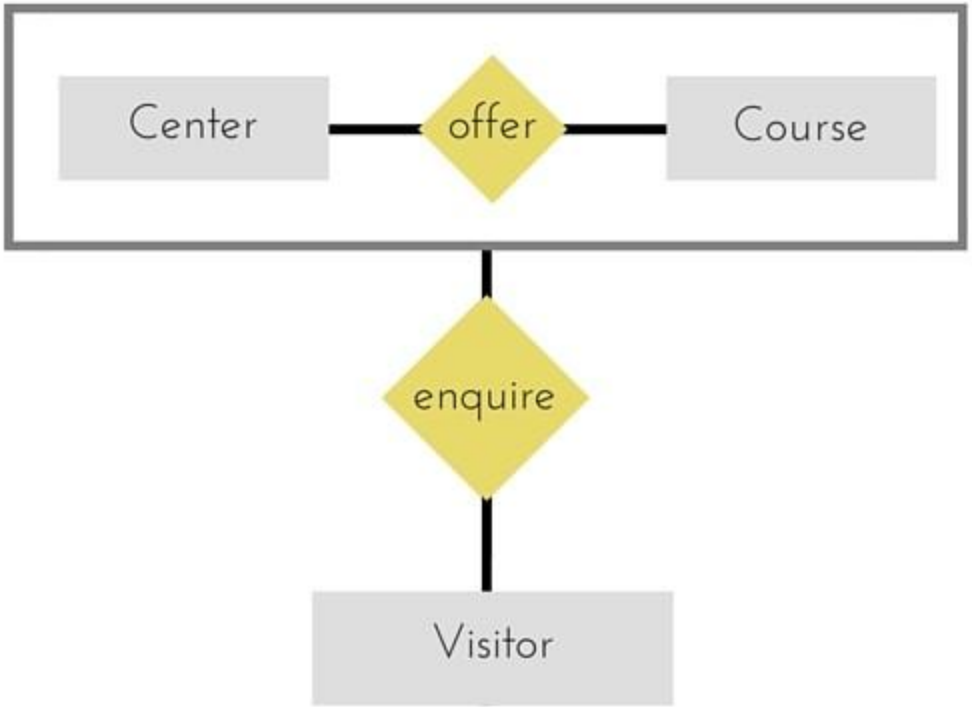
Specialization

Specialization is opposite to Generalization.



Aggregation

Aggregation is a process when relation between two entities is treated as a **single entity**.



Basic Relational DBMS Concepts

RDBMS is used to manage Relational database. **Relational database** is a collection of organized set of tables related to each other, and from which data can be accessed easily. Relational Database is the most commonly used database these days.

RDBMS: What is Table ?

In Relational database model, a **table** is a collection of data elements organized in terms of rows and columns.

ID	Name	Age	Salary
1	Adam	34	13000
2	Alex	28	15000
3	Stuart	20	18000
4	Ross	42	19020

RDBMS: What is a Tuple?

A single entry in a table is called a **Tuple** or **Record** or **Row**.

1	Adam	34	13000

RDBMS: What is an Attribute?

A table consists of several records(row), each record can be broken down into several smaller parts of data known as **Attributes**.

Attribute Domain

When an attribute is defined in a relation(table), it is defined to hold only a certain type of values, which is known as **Attribute Domain**.

Name
Adam
Alex
Stuart - 9/401, OC Street, Amsterdam
Ross

What is a Relation Schema?

A relation schema describes the structure of the relation, with the name of the relation(name of table), its attributes and their names and type.

What is a Relation Key?

A relation key is an attribute which can uniquely identify a particular tuple(row) in a relation(table).

Relational Integrity Constraints

Every relation in a relational database model should abide by or follow a few constraints to be a valid relation, these constraints are called as **Relational Integrity Constraints**.

The three main Integrity Constraints are:

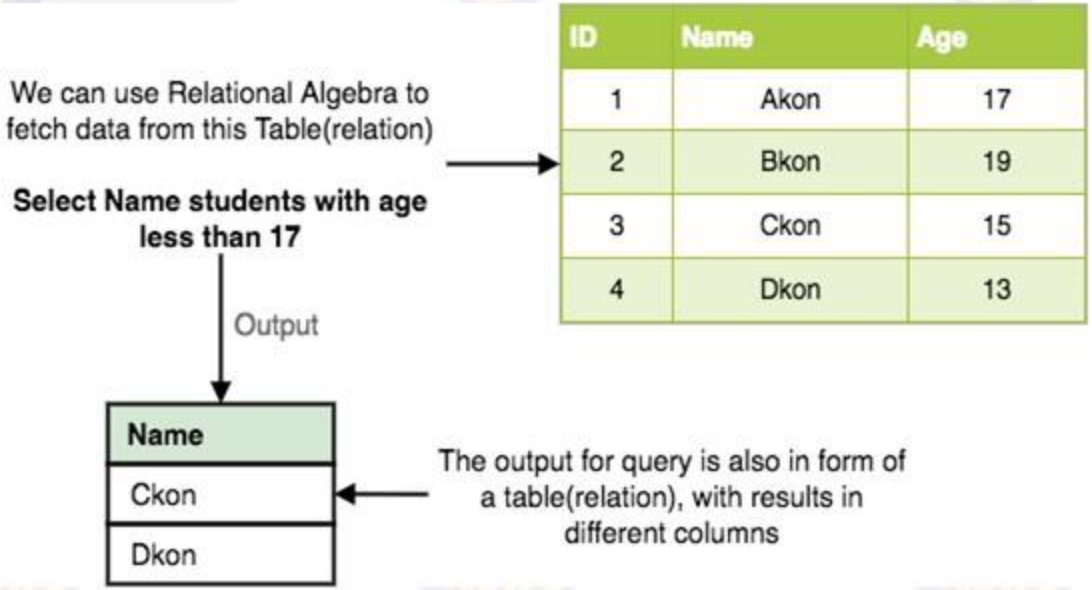
1.Key Constraints

2.Domain Constraints

3.Referential integrity Constraints

What is Relational Algebra?

Relational Algebra is a procedural query language used to query the database tables to access data in different ways.



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

Name
Ckon
Dkon

The primary operations that we can perform using relational algebra are:

1.Select

2.Project

3.Union

4.Set Different

5.Cartesian product

6.Rename

Algebra is also used for **Join** operations like,

- Natural Join
- Outer Join
- Theta join etc.

ER Model to Relational Model

Points to Remember

Similarly we can generate relational database schema using the ER diagram. Following are some key points to keep in mind while doing so:

1.Entity gets converted into Table, with all the attributes becoming fields(columns) in the table.

2.Relationship between entities is also converted into table with primary keys of the related entities also stored in it as foreign keys.

3.Primary Keys should be properly set.

4.For any relationship of Weak Entity, if primary key of any other entity is included in a table, foreign key constraint must be defined.

Introduction to Database Keys

Keys are very important part of Relational database model. They are used to establish and identify relationships between tables and also to uniquely identify any record or row of data inside a table.

A Key can be a single attribute or a group of attributes, where the combination may act as a key.

student_id	name	phone	age
1	Akon	9876723452	17
2	Akon	9991165674	19
3	Bkon	7898756543	18
4	Ckon	8987867898	19
5	Dkon	9990080080	17

Super Key

Super Key is defined as a set of attributes within a table that can uniquely identify each record within a table.

Super Key is a superset of Candidate key.

Candidate Key

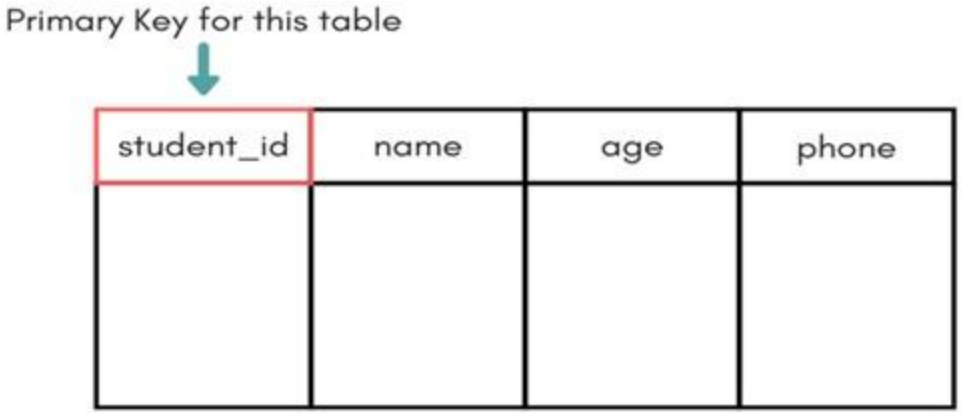
Candidate keys are defined as the minimal set of fields which can uniquely identify each record in a table.

In our example, student_id and phone both are candidate keys for table Student.

- A candidate key can never be NULL or empty. And its value should be unique.
- There can be more than one candidate keys for a table.
- A candidate key can be a combination of more than one columns(attributes).

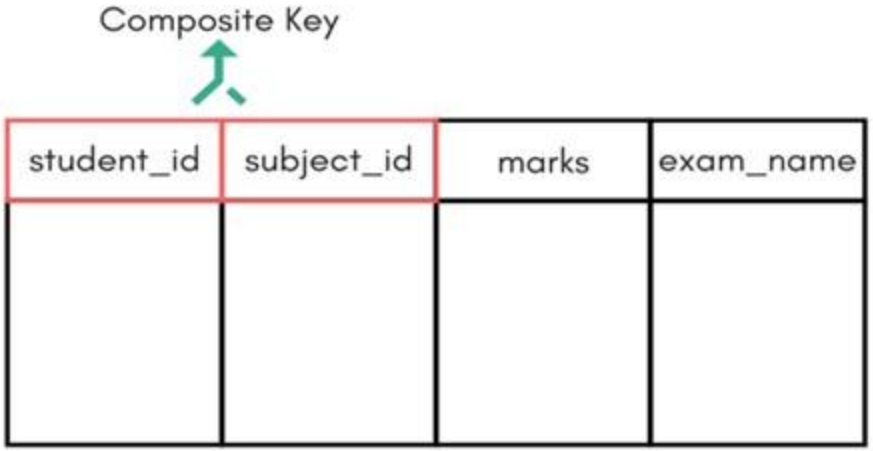
Primary Key

Primary key is a candidate key that is most appropriate to become the main key for any table.
It is a key that can uniquely identify each record in a table.



Composite Key

Key that consists of two or more attributes that uniquely identify any record in a table is called **Composite key**. But the attributes which together form the **Composite key** are not a key independently or individually.



Score Table - To save scores of the student for various subjects.

Secondary or Alternative key

The candidate key which are not selected as primary key are known as secondary keys or alternative keys.

Non-key Attributes

Non-key attributes are the attributes or fields of a table, other than candidate key attributes/fields in a table.

Non-prime Attributes

Non-prime Attributes are attributes other than Primary Key attribute(s)..

Normalization of Database

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.

Normalization is used for mainly two purposes,

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

Normalization Rule

Normalization rules are divided into the following normal forms:

1.First Normal Form

2.Second Normal Form

3.Third Normal Form

4.BCNF

5.Fourth Normal Form

First Normal Form (1NF)

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

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

2.Values stored in a column should be of the same domain

3.All the columns in a table should have unique names.

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

What is First Normal Form (1NF)?

If tables in a database are not even in the 1st Normal Form, it is considered as **bad database design**.

roll_no	name	subject
101	Akon	OS, CN
103	Ckon	Java
102	Bkon	C, C++

roll_no	name	subject
101	**Akon**	**OS**
101	**Akon**	**CN**
103	**Ckon**	**Java**
102	**Bkon**	**C**
102	**Bkon**	**C++**

Second Normal Form (2NF)

For a table to be in the Second Normal Form,

1.It should be in the First Normal form.

2.And, it should not have Partial Dependency.

What is Second Normal Form?

What is Dependency?

student_id	name	reg_no	branch	address
10	Akon	07-WY	CSE	Kerala
11	Akon	08-WY	IT	Gujarat

I can ask from branch name of student with student_id 10, and I can get it. Similarly, if I ask for name of student with student_id 10 or 11, I will get it. So all I need is student_id and every other column depends on it, or can be fetched using it.

This is Dependency and we also call it Functional Dependency.

What is Partial Dependency?

subject_id	subject_name
1	Java
2	C++
3	Php

Together, student_id + subject_id forms a Candidate Key for this table, which can be the Primary key.

score_id	student_id	subject_id	marks	teacher
1	10	1	70	Java Teacher
2	10	2	75	C++ Teacher
3	11	1	80	Java Teacher

Recap(2NF)

1.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.

2.Partial Dependency exists, when for a composite primary key, any attribute in the table depends only on a part of the primary key and not on the complete primary key.

3.To remove Partial dependency, we can divide the table, remove the attribute which is causing partial dependency, and move it to some other table where it fits in well.

Third Normal Form (3NF)

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

1.It is in the Second Normal form.

2.And, it doesn't have Transitive Dependency.

Third Normal Form is an upgrade to Second Normal Form.
When a table is in the Second Normal Form and has no transitive dependency, then it is in the Third Normal Form.

let's use the same example, where we have 3 tables, **Student**, **Subject** and **Score**.

student_id	name	reg_no	branch	address
10	Akon	07-WY	CSE	Kerala
11	Akon	08-WY	IT	Gujarat
12	Bkon	09-WY	IT	Rajasthan

Student Table

subject_id	subject_name	Teacher
1	Java	Java Teacher
2	C++	C++ Teacher
3	Php	Php Teacher

Subject Table

score_id	student_id	subject_id	marks
1	10	1	70
2	10	2	75
3	11	1	80

Score Table

In the Score table, we need to store some more information, which is the exam name and total marks, so let's add 2 more columns to the Score table.

What is Transitive Dependency?

How to remove Transitive Dependency?

The new Exam table

exam_id	exam_name	total_marks
1	Workshop	200
2	Mains	70
3	Practicals	30

Advantage of removing Transitive Dependency

- Amount of data duplication is reduced.
- Data integrity achieved.

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 → Y), X should be a super Key.

Boyce-Codd Normal Form (BCNF)

Boyce-Codd Normal Form or BCNF is an extension to the third normal form, and is also known as 3.5 Normal Form.

Below we have a college enrolment table with columns `student_id`, `subject` and `professor`.

student_id	subject	professor
101	Java	P.Java
101	C++	P.Cpp
102	Java	P.Java2
103	C#	P.Chash
104	Java	P.Java

How to satisfy BCNF?

To make this relation(table) satisfy BCNF, we will decompose this table into two tables, **student** table and **professor** table.

Student Table

student_id	p_id
101	1
101	2
and so on...	

Professor Table

p_id	professor	subject
1	P.Java	Java
2	P.Cpp	C++
and so on...		

In the picture below, we have tried to explain BCNF in terms of relations.

Consider the following relationship : **R (A,B,C,D)**

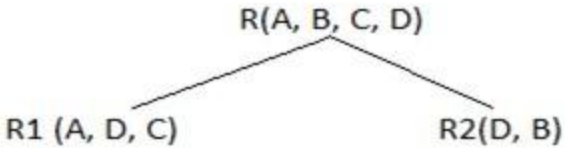
and following dependencies :

A -> BCD
BC -> AD
D -> B

Above relationship is already in 3rd NF. Keys are **A** and **BC**.

Hence, in the functional dependency, **A -> BCD**, A is the super key.
in second relation, **BC -> AD**, BC is also a key.
but in, **D -> B**, D is not a key.

Hence we can break our relationship R into two relationships **R1** and **R2**.



Breaking, table into two tables, one with A, D and C while the other with D and B.

InfyTQ

Basic SQL

TalentBattle

Prerequisite: Lecture 1: Database Concepts

Introduction to SQL

Structure Query Language(SQL) is a database query language used for storing and managing data in Relational DBMS.

SQL is used to perform all types of data operations in RDBMS.

SQL Command

DDL: Data Definition Language

This includes changes to the structure of the table like creation of table, altering table, deleting a table etc. All DDL commands are auto-committed. That means it saves all the changes permanently in the database.

Command	Description
create	to create new table or database
alter	for alteration
truncate	delete data from table
drop	to drop a table
rename	to rename a table

DML: Data Manipulation Language

DML commands are used for manipulating the data stored in the table and not the table itself.

DML commands are not auto-committed. It means changes are not permanent to database, they can be rolled back.

Command	Description
insert	to insert a new row
update	to update existing row
delete	to delete a row
merge	merging two rows or two tables

TCL: Transaction Control Language

These commands are to keep a check on other commands and their affect on the database. These commands can annul changes made by other commands by rolling the data back to its original state. It can also make any temporary change permanent.

Command	Description
commit	to permanently save
rollback	to undo change
savepoint	to save temporarily

DCL: Data Control Language

Data control language are the commands to grant and take back authority from any database user.

Command	Description
grant	grant permission of right
revoke	take back permission.

DQL: Data Query Language

Data query language is used to fetch data from tables based on conditions that we can easily apply.

Command	Description
select	retrieve records from one or more table

SQL: create command

create is a DDL SQL command used to create a table or a database in relational database management system.

Creating a Database
To create a database in RDBMS, create command is used. Following is the syntax,

CREATE DATABASE <DB_NAME>;
Example for creating Database
CREATE DATABASE Test;

Creating a Table

create command can also be used to create tables. Now when we create a table, we have to specify the details of the columns of the tables too. We can specify the names and datatypes of various columns in the create command itself.

Following is the syntax,

```
CREATE TABLE <TABLE_NAME>
(
   column_name1 datatype1,
   column_name2 datatype2,
   column_name3 datatype3,
   column_name4 datatype4
);
```

create table command will tell the database system to create a new table with the given table name and column information.

Example for creating Table

```
CREATE TABLE Student(
   student_id INT,
   name VARCHAR(100),
   age INT);
```

Most commonly used datatypes for Table columns

Here we have listed some of the most commonly used datatypes used for columns in tables.

Datatype	Use
INT	used for columns which will store integer values.
FLOAT	used for columns which will store float values.
DOUBLE	used for columns which will store float values.
VARCHAR	used for columns which will be used to store characters and integers, basically a string.
CHAR	used for columns which will store char values(single character).
DATE	used for columns which will store date values.
TEXT	used for columns which will store text which is generally long in length. For example, if you create a table for storing profile information of a social networking website, then for **about me** section you can have a column of type TEXT.

SQL: ALTER command

alter command is used for altering the table structure, such as,

- to add a column to existing table
- to rename any existing column
- to change datatype of any column or to modify its size.
- to drop a column from the table.

ALTER Command: Add a new Column

Using ALTER command we can add a column to any existing table. Following is the syntax,

```
ALTER TABLE table_name ADD(
   column_name datatype);
```

Here is an Example for this,

```
ALTER TABLE student ADD(
   address VARCHAR(200)
);
```

ALTER Command: Add multiple new Columns

Using ALTER command we can even add multiple new columns to any existing table. Following is the syntax,

```
ALTER TABLE table_name ADD(
   column_name1 datatype1,
   column-name2 datatype2,
   column-name3 datatype3);
```

Here is an Example for this,

```
ALTER TABLE student ADD(
   father_name VARCHAR(60),
   mother_name VARCHAR(60),
   dob DATE);
```

ALTER Command: Add Column with default value

ALTER command can add a new column to an existing table with a default value too. The default value is used when no value is inserted in the column. Following is the syntax,

```
ALTER TABLE table_name ADD(
   column-name1 datatype1 DEFAULT some_value
);
```

Here is an Example for this,

```
ALTER TABLE student ADD(
   dob DATE DEFAULT '01-Jan-99'
);
```

ALTER Command: Modify an existing Column

ALTER command can also be used to modify data type of any existing column. Following is the syntax,

```
ALTER TABLE table_name modify(
   column_name datatype
);
```

Here is an Example for this,

```
ALTER TABLE student MODIFY(
   address varchar(300));
```

ALTER Command: Rename a Column

Using ALTER command you can rename an existing column. Following is the syntax,

```
ALTER TABLE table_name RENAME
   old_column_name TO new_column_name;
```

Here is an example for this,

```
ALTER TABLE student RENAME
   address TO location;
```

ALTER Command: Drop a Column

ALTER command can also be used to drop or remove columns. Following is the syntax,

```
ALTER TABLE table_name DROP(
   column_name);
```

Here is an example for this,

```
ALTER TABLE student DROP(
   address);
```

SQL Truncate, Drop or Rename a Table

TRUNCATE command
TRUNCATE command removes all the records from a table. But this command will not destroy the table's structure. When we use TRUNCATE command on a table its (auto-increment) primary key is also initialized. Following is its syntax,

```
TRUNCATE TABLE table_name
```

Here is an example explaining it,

```
TRUNCATE TABLE student;
```

The above query will delete all the records from the table student.

DROP command

DROP command completely removes a table from the database. This command will also destroy the table structure and the data stored in it. Following is its syntax,

DROP TABLE table_name

Here is an example explaining it,

DROP TABLE student;

The above query will delete the Student table completely. It can also be used on Databases, to delete the complete database. For example, to drop a database,

DROP DATABASE Test;

The above query will drop the database with name Test from the system.

RENAME query

RENAME command is used to set a new name for any existing table. Following is the syntax,

RENAME TABLE old_table_name to new_table_name

Here is an example explaining it.

RENAME TABLE student to students_info;

The above query will rename the table student to students_info.

INSERT SQL command

Data Manipulation Language (DML) statements are used for managing data in database. DML commands are not auto-committed. It means changes made by DML command are not permanent to database, it can be rolled back.

INSERT command
Insert command is used to insert data into a table. Following is its general syntax,
INSERT INTO table_name VALUES(data1, data2, ...)
Lets see an example,
Consider a table student with the following fields.

s_id	name	age

INSERT INTO student VALUES(101, 'Adam', 15);
The above command will insert a new record into student table.

s_id	name	age
101	Adam	15

Insert value into only specific columns

We can use the INSERT command to insert values for only some specific columns of a row. We can specify the column names along with the values to be inserted like this,

```
INSERT INTO student(id, name) values(102, 'Alex');
```

The above SQL query will only insert id and name values in the newly inserted record.

Insert NULL value to a column

Both the statements below will insert NULL value into age column of the student table.

```
INSERT INTO student(id, name) values(102, 'Alex');
```

Or,

```
INSERT INTO Student VALUES(102,'Alex', null);
```

The above command will insert only two column values and the other column is set to null.

S_id	S_Name	age
101	Adam	15
102	Alex	

Insert Default value to a column

INSERT INTO Student VALUES(103,'Chris', default)

S_id	S_Name	age
101	Adam	15
102	Alex	
103	chris	14

Suppose the column age in our table has a default value of 14.

Also, if you run the below query, it will insert default value into the age column, whatever the default value may be.

INSERT INTO Student VALUES(103,'Chris')

Using UPDATE SQL command

UPDATE command is used to update any record of data in a table. Following is its general syntax,

UPDATE table_name SET column_name = new_value WHERE some_condition;

Lets take a sample table student,

student_id	name	age
101	Adam	15
102	Alex	
103	chris	14

UPDATE student SET age=18 WHERE student_id=102;

S_id	S_Name	age
101	Adam	15
102	Alex	18
103	chris	14

In the above statement, if we do not use the WHERE clause, then our update query will update age for all the columns of the table to 18.

Updating Multiple Columns

We can also update values of multiple columns using a single UPDATE statement.

UPDATE student SET name='Abhi', age=17 where s_id=103;

The above command will update two columns of the record which has s_id 103.

s_id	name	age
101	Adam	15
102	Alex	18
103	Abhi	17

UPDATE Command: Incrementing Integer Value

When we have to update any integer value in a table, then we can fetch and update the value in the table in a single statement.

For example, if we have to update the age column of student table every year for every student, then we can simply run the following UPDATE statement to perform the following operation:

UPDATE student SET age = age+1;

As you can see, we have used age = age + 1 to increment the value of age by 1.

NOTE: This style only works for integer values.

Using DELETE SQL command

DELETE command

DELETE command is used to delete data from a table.

Following is its general syntax,

```
DELETE FROM table_name;
```

Let's take a sample table student:

s_id	name	age
101	Adam	15
102	Alex	18
103	Abhi	17

Delete all Records from a Table

DELETE FROM student;

The above command will delete all the records from the table student.

Delete a particular Record from a Table

In our student table if we want to delete a single record, we can use the WHERE clause to provide a condition in our DELETE statement.

DELETE FROM student WHERE s_id=103;

The above command will delete the record where s_id is 103 from the table student.

S_id	S_Name	age
101	Adam	15
102	Alex	18

Isn't DELETE same as TRUNCATE

TRUNCATE command is different from DELETE command. The delete command will delete all the rows from a table whereas truncate command not only deletes all the records stored in the table, but it also re-initializes the table(like a newly created table).

For eg: If you have a table with 10 rows and an auto_increment primary key, and if you use DELETE command to delete all the rows, it will delete all the rows, but will not re-initialize the primary key, hence if you will insert any row after using the DELETE command, the auto_increment primary key will start from 11. But in case of TRUNCATE command, primary key is re-initialized, and it will again start from 1.

Commit, Rollback and Savepoint SQL commands

COMMIT command

COMMIT command is used to permanently save any transaction into the database.

When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.

To avoid that, we use the COMMIT command to mark the changes as permanent.

Following is commit command's syntax,

```
COMMIT;
```

ROLLBACK command

This command restores the database to last commited state. It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction.

If we have used the UPDATE command to make some changes into the database, and realise that those changes were not required, then we can use the ROLLBACK command to rollback those changes, if they were not commited using the COMMIT command.

Following is rollback command's syntax,

```
ROLLBACK TO savepoint_name;
```

SAVEPOINT command

SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.

Following is savepoint command's syntax,

SAVEPOINT savepoint_name;

In short, using this command we can name the different states of our data in any table and then rollback to that state using the ROLLBACK command whenever required.

Using GRANT and REVOKE

Data Control Language(DCL) is used to control privileges in Database. To perform any operation in the database, such as for creating tables, sequences or views, a user needs privileges. Privileges are of two types,

System: This includes permissions for creating session, table, etc and all types of other system privileges.
Object: This includes permissions for any command or query to perform any operation on the database tables.
In DCL we have two commands,

GRANT: Used to provide any user access privileges or other priviliges for the database.
REVOKE: Used to take back permissions from any user.

Allow a User to create session

When we create a user in SQL, it is not even allowed to login and create a session until and unless proper permissions/privileges are granted to the user.

Following command can be used to grant the session creating privileges.

GRANT CREATE SESSION TO username;

Allow a User to create table

To allow a user to create tables in the database, we can use the below command,

GRANT CREATE TABLE TO username;

Provide user with space on tablespace to store table

Allowing a user to create table is not enough to start storing data in that table. We also must provide the user with privileges to use the available tablespace for their table and data.

ALTER USER username QUOTA UNLIMITED ON SYSTEM;

The above command will alter the user details and will provide it access to unlimited tablespace on system.

NOTE: Generally unlimited quota is provided to Admin users.

Grant all privilege to a User

sysdba is a set of privileges which has all the permissions in it. So if we want to provide all the privileges to any user, we can simply grant them the sysdba permission.

GRANT sysdba TO username

Grant permission to create any table

Sometimes user is restricted from creating come tables with names which are reserved for system tables. But we can grant privileges to a user to create any table using the below command,

GRANT CREATE ANY TABLE TO username

Grant permission to drop any table

As the title suggests, if you want to allow user to drop any table from the database, then grant this privilege to the user,

GRANT DROP ANY TABLE TO username

To take back Permissions

And, if you want to take back the privileges from any user, use the REVOKE command.

REVOKE CREATE TABLE FROM username

Using the WHERE SQL clause

WHERE clause is used to specify/apply any condition while retrieving, updating or deleting data from a table. This clause is used mostly with SELECT, UPDATE and DELETEquery.

When we specify a condition using the WHERE clause then the query executes only for those records for which the condition specified by the WHERE clause is true.

Syntax for WHERE clause
Here is how you can use the WHERE clause with a DELETE statement, or any other statement,

DELETE FROM table_name WHERE [condition];
The WHERE clause is used at the end of any SQL query, to specify a condition for execution.

Example

Consider a table student,

s_id	name	age	address
101	Adam	15	Chennai
102	Alex	18	Delhi
103	Abhi	17	Banglore
104	Ankit	22	Mumbai

```
SELECT s_id, name, age, address FROM student WHERE s_id = 101;
```

Operators for WHERE clause condition

Operator	Description
=	Equal to
!=	Not Equal to
<	Less than
>	Greater than
<=	Less than or Equal to
>=	Greate than or Equal to
BETWEEN	Between a specified range of values
LIKE	This is used to search for a pattern in value.
IN	In a given set of values

SQL LIKE clause

LIKE clause is used in the condition in SQL query with the WHERE clause. LIKE clause compares data with an expression using wildcard operators to match pattern given in the condition.

Wildcard operators

There are two wildcard operators that are used in LIKE clause.

Percent sign % : represents zero, one or more than one character.
Underscore sign _ : represents only a single character.

Example of LIKE clause

Consider the following Student table.

s_id	s_Name	age
101	Adam	15
102	Alex	18
103	Abhi	17

```
SELECT * FROM Student WHERE s_name LIKE 'A%';
```

The above query will return all records where s_name starts with character 'A'.

Using _ and %

SELECT * FROM Student WHERE s_name LIKE '_d%';

The above query will return all records from Student table where s_name contain 'd' as second character.

Using % only

SELECT * FROM Student WHERE s_name LIKE '%x';

The above query will return all records from Student table where s_name contain 'x' as last character.

SQL ORDER BY Clause

Order by clause is used with SELECT statement for arranging retrieved data in sorted order.
The Order by clause by default sorts the retrieved data in ascending order.
To sort the data in descending order DESC keyword is used with Order by clause.

Syntax of Order By

SELECT column-list|* FROM table-name ORDER BY ASC | DESC;

Using default Order by

Consider the following Emp table,

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

```
SELECT * FROM Emp ORDER BY salary;
```

The above query will return the resultant data in ascending order of the salary.

Using Order by DESC

Consider the Emp table described above,

SELECT * FROM Emp ORDER BY salary DESC;

The above query will return the resultant data in descending order of the salary.

SQL Group By Clause

Group by clause is used to group the results of a SELECT query based on one or more columns. It is also used with SQL functions to group the result from one or more tables.

Syntax for using Group by in a statement.

```
SELECT column_name, function(column_name)
FROM table_name
WHERE condition
GROUP BY column_name
```

Example of Group by in a Statement

Consider the following Emp table.

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	9000
405	Tiger	35	8000

Here we want to find name and age of employees grouped by their salaries or in other words, we will be grouping employees based on their salaries, hence, as a result, we will get a data set, with unique salaries listed, along side the first employee's name and age to have that salary.

group by is used to group different row of data together based on any one column.

SQL query for the above requirement will be,

```
SELECT name, age
FROM Emp GROUP BY salary
```

Example of Group by in a Statement with WHERE clause

Consider the following Emp table

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	9000
405	Tiger	35	8000

SQL query will be,

```
SELECT name, salary
FROM Emp
WHERE age > 25
GROUP BY salary
```

You must remember that Group By clause will always come at the end of the SQL query, just like the Order by clause.

SQL HAVING Clause

Having clause is used with SQL Queries to give more precise condition for a statement. It is used to mention condition in Group by based SQL queries, just like WHERE clause is used with SELECT query.

Syntax for HAVING clause is,

```
SELECT column_name, function(column_name)
FROM table_name
WHERE column_name condition
GROUP BY column_name
HAVING function(column_name) condition
```

Example of SQL Statement using HAVING

Consider the following Sale table.

oid	order_name	previous_balance	customer
11	ord1	2000	Alex
12	ord2	1000	Adam
13	ord3	2000	Abhi
14	ord4	1000	Adam
15	ord5	2000	Alex

Suppose we want to find the customer whose previous_balance sum is more than 3000.

We will use the below SQL query,

```
SELECT *
FROM sale GROUP BY customer
HAVING sum(previous_balance) > 3000
```

DISTINCT keyword

The distinct keyword is used with SELECT statement to retrieve unique values from the table.
Distinct removes all the duplicate records while retrieving records from any table in the database.

Syntax for DISTINCT Keyword

SELECT DISTINCT column-name FROM table-name;

Example using DISTINCT Keyword

Consider the following Emp table. As you can see in the table below, there is employee name, along with employee salary and age.

In the table below, multiple employees have the same salary, so we will be using DISTINCT keyword to list down distinct salary amount, that is currently being paid to the employees.

eid	name	age	salary
401	Anu	22	5000
402	Shane	29	8000
403	Rohan	34	10000
404	Scott	44	10000
405	Tiger	35	8000

SELECT DISTINCT salary FROM Emp;

The above query will return only the unique salary from Emp table.

SQL AND & OR operator

The AND and OR operators are used with the WHERE clause to make more precise conditions for fetching data from database by combining more than one condition together.

AND operator

AND operator is used to set multiple conditions with the WHERE clause, alongside, SELECT, UPDATE or DELETE SQL queries.

Example of AND operator

Consider the following Emp table

eid	name	age	salary
401	Anu	22	5000
402	Shane	29	8000
403	Rohan	34	12000
404	Scott	44	10000
405	Tiger	35	9000

```
SELECT * FROM Emp WHERE salary < 10000 AND age > 25
```

OR operator

OR operator is also used to combine multiple conditions with WHERE clause. The only difference between AND and OR is their behaviour.

When we use AND to combine two or more than two conditions, records satisfying all the specified conditions will be there in the result. But in case of OR operator, at least one condition from the conditions specified must be satisfied by any record to be in the result set.

Consider the following Emp table

eid	name	age	salary
401	Anu	22	5000
402	Shane	29	8000
403	Rohan	34	12000
404	Scott	44	10000
405	Tiger	35	9000

```
SELECT * FROM Emp WHERE salary > 10000 OR age > 25
```

Division Operator in SQL

The division operator is used when we have to evaluate queries which contain the keyword ALL.

Some instances where division operator is used are:

Which person has account in all the banks of a particular city?
Which students have taken all the courses required to graduate?

In above specified problem statements, the description after the keyword 'all' defines a set which contains some elements and the final result contains those units which satisfy these requirements.

Using Division Operator

So now, let's try to find out the correct SQL query for getting results for the first requirement, which is:

Query: Find all the students who can graduate. (i.e. who have taken all the subjects required for one to graduate.)

1. Find all the students

Create a set of all students that have taken courses. This can be done easily using the following command.

```
CREATE TABLE AllStudents AS SELECT DISTINCT Student_Name FROM Course_Taken
```

This command will return the table AllStudents, as the resultset:

2. Find all the students and the courses required to graduate

Next, we will create a set of students and the courses they need to graduate. We can express this in the form of Cartesian Product of AllStudents and Course_Required using the following command.

```
CREATE table StudentsAndRequired AS
SELECT AllStudents.Student_Name, Course_Required.Course
FROM AllStudents, Course_Required
```

3. Find all the students and the required courses they have not taken

Here, we are taking our first step for finding the students who cannot graduate. The idea is to simply find the students who have not taken certain courses that are required for graduation and hence they wont be able to graduate. This is simply all those tuples/rows which are present in StudentsAndRequired and not present in Course_Taken.

```
CREATE  table StudentsAndNotTaken AS
SELECT * FROM StudentsAndRequired WHERE NOT EXISTS
(Select * FROM Course_Taken WHERE StudentsAndRequired.Student_Name =
Course_Taken.Student_Name
AND StudentsAndRequired.Course = Course_Taken.Course)
```

4. Find all students who cannot graduate

All the students who are present in the table StudentsAndNotTaken are the ones who cannot graduate. Therefore, we can find the students who cannot graduate as,

CREATE table CannotGraduate AS SELECT DISTINCT Student_Name FROM StudentsAndNotTaken

5. Find all students who can graduate

The students who can graduate are simply those who are present in AllStudents but not in CannotGraduate.

This can be done by the following query:

```
CREATE Table CanGraduate AS SELECT * FROM AllStudents
WHERE NOT EXISTS
(SELECT * FROM CannotGraduate WHERE
   CannotGraduate.Student_name = AllStudents.Student_name)
```

Now let us see how to write all these 5 steps in one single query so that we do not have to create so many tables.

```
SELECT DISTINCT  x.Student_Name FROM Course_Taken AS x WHERE NOT
EXISTS(SELECT *  FROM Course_Required AS y WHERE NOT
EXISTS(SELECT *  FROM Course_Taken AS z
   WHERE z.Student_name = x.Student_name
   AND z.Course = y.Course ))
```

InfyTQ

Advanced SQL

TalentBattle

Prerequisite: Lecture 1: Database Concepts
Lecture 2: Basic SQL

SQL Constraints

SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table.

Constraints can be divided into the following two types,

Column level constraints: Limits only column data.
Table level constraints: Limits whole table data.
Constraints are used to make sure that the integrity of data is maintained in the database. Following are the most used constraints that can be applied to a table.

NOT NULL
UNIQUE
PRIMARY KEY
FOREIGN KEY
CHECK
DEFAULT

NOT NULL Constraint

By default, a column can hold NULL values. If you do not want a column to have a NULL value, use the NOT NULL constraint.

It restricts a column from having a NULL value.
We use ALTER statement and MODIFY statement to specify this constraint.
One important point to note about this constraint is that it cannot be defined at table level.

Example using NOT NULL constraint:

```
CREATE TABLE Student
(           s_id int NOT NULL,
            name varchar(60),
            age  int
);
```

The above query will declare that the s_id field of Student table will not take NULL value.

UNIQUE Constraint

It ensures that a column will only have unique values. A UNIQUE constraint field cannot have any duplicate data.

It prevents two records from having identical values in a column

We use ALTER statement and MODIFY statement to specify this constraint.

Example of UNIQUE Constraint:

Here we have a simple CREATE query to create a table, which will have a column s_id with unique values.

```
CREATE TABLE Student
(           s_id int  NOT NULL,
            name varchar(60),
            age int  NOT NULL UNIQUE
);
```

The above query will declare that the s_id field of Student table will only have unique values and wont take NULL value.

Primary Key Constraint

Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually Primary Key is used to index the data inside the table.

PRIMARY KEY constraint at Table Level

```
CREATE table Student
(           s_id int PRIMARY KEY,
            Name varchar(60) NOT NULL,
            Age int);
```

The above command will creates a PRIMARY KEY on the s_id.

PRIMARY KEY constraint at Column Level

```
ALTER table Student
ADD PRIMARY KEY (s_id);
```

The above command will creates a PRIMARY KEY on the s_id.

Foreign Key Constraint

Foreign Key is used to relate two tables. The relationship between the two tables matches the Primary Key in one of the tables with a Foreign Key in the second table.

This is also called a referencing key.

We use ALTER statement and ADD statement to specify this constraint.

To understand FOREIGN KEY, let's see its use, with help of the below tables:

Customer_Detail Table

c_id	Customer_Name	address
101	Adam	Noida
102	Alex	Delhi
103	Stuart	Rohtak

Order_Detail Table

Order_id	Order_Name	c_id
10	Order1	101
11	Order2	103
12	Order3	102

FOREIGN KEY constraint at Table Level

```
CREATE table Order_Detail(
   order_id int PRIMARY KEY,
   order_name varchar(60) NOT NULL,
   c_id int FOREIGN KEY REFERENCES Customer_Detail(c_id)
);
```

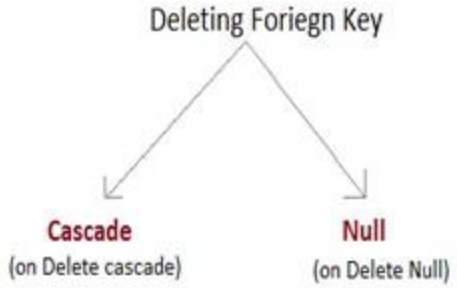
In this query, c_id in table Order_Detail is made as foriegn key, which is a reference of c_id column in Customer_Detail table.

FOREIGN KEY constraint at Column Level

```
ALTER table Order_Detail
ADD FOREIGN KEY (c_id) REFERENCES Customer_Detail(c_id);
```

Behavior of Foreign Key Column on Delete

There are two ways to maintain the integrity of data in Child table, when a particular record is deleted in the main table. When two tables are connected with Foreign key, and certain data in the main table is deleted, for which a record exits in the child table, then we must have some mechanism to save the integrity of data in the child table.



On Delete Cascade : This will remove the record from child table, if that value of foreign key is deleted from the main table.

On Delete Null : This will set all the values in that record of child table as NULL, for which the value of foreign key is deleted from the main table.

If we don't use any of the above, then we cannot delete data from the main table for which data in child table exists. We will get an error if we try to do so.

CHECK Constraint

CHECK constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database. Its like condition checking before saving data into a column.

Using CHECK constraint at Table Level

```
CREATE table Student(
   s_id int NOT NULL CHECK(s_id > 0),
   Name varchar(60) NOT NULL,
   Age int
);
```

The above query will restrict the s_id value to be greater than zero.

Using CHECK constraint at Column Level

```
ALTER table Student ADD CHECK(s_id > 0);
```

What are SQL Functions?

SQL provides many built-in functions to perform operations on data.

These functions are useful while performing mathematical calculations, string concatenations, sub-strings etc.

SQL functions are divided into two categories,

1. **Aggregate Functions**
2. **Scalar Functions**

Aggregate Functions

These functions return a single value after performing calculations on a group of values. Following are some of the frequently used Aggregate functions.

AVG() Function

Average returns average value after calculating it from values in a numeric column.

Its general syntax is,

SELECT AVG(column_name) FROM table_name;

```
SELECT avg(salary) from Emp;
```

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

COUNT() Function

Count returns the number of rows present in the table either based on some condition or without condition.

Its general syntax is,

SELECT COUNT(column_name) FROM table-name

Consider the following Emp table

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to count employees, satisfying specified condition is,

SELECT COUNT(name) FROM Emp WHERE salary = 8000;

Example of COUNT(distinct)

Consider the following Emp table

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query is,

```
SELECT COUNT(DISTINCT salary) FROM emp;
```

Result of the above query will be,

count(distinct salary)
4

FIRST() Function

First function returns first value of a selected column

Syntax for FIRST function is,

SELECT FIRST(column_name) FROM table-name;

Consider the following Emp table

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query will be,

SELECT FIRST(salary) FROM Emp;

and the result will be,

first(salary)

9000

LAST() Function

LAST function returns the return last value of the selected column.

Syntax of LAST function is,

SELECT LAST(column_name) FROM table-name;

Consider the following Emp table

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query will be,

SELECT LAST(salary) FROM emp;

Result of the above query will be,

last(salary)
8000

MAX() Function

MAX function returns maximum value from selected column of the table.

Syntax of MAX function is,

SELECT MAX(column_name) from table-name;

Consider the following Emp table

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to find the Maximum salary will be,

SELECT MAX(salary) FROM emp;

MIN() Function

MIN function returns minimum value from a selected column of the table.

Syntax for MIN function is,

SELECT MIN(column_name) from table-name;

Consider the following Emp table,

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to find minimum salary is,

SELECT MIN(salary) FROM emp;

SUM() Function

SUM function returns total sum of a selected columns numeric values.

Syntax for SUM is,

SELECT SUM(column_name) from table-name;

Consider the following Emp table

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to find sum of salaries will be,

SELECT SUM(salary) FROM emp;

Scalar Functions

Scalar functions return a single value from an input value. Following are some frequently used Scalar Functions in SQL.

UCASE() Function

UCASE function is used to convert value of string column to Uppercase characters.

Syntax of UCASE,

SELECT UCASE(column_name) from table-name;

Consider the following Emp table

eid	name	age	salary
401	anu	22	9000
402	shane	29	8000
403	rohan	34	6000
404	scott	44	10000
405	Tiger	35	8000

SQL query for using UCASE is,

SELECT UCASE(name) FROM emp;

LCASE() Function

LCASE function is used to convert value of string columns to Lowecase characters.

Syntax for LCASE is,

SELECT LCASE(column_name) FROM table-name;

Consider the following Emp table

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	SCOTT	44	10000
405	Tiger	35	8000

SQL query for converting string value to Lower case is,

SELECT LCASE(name) FROM emp;

MID() Function

MID function is used to extract substrings from column values of string type in a table.

Syntax for MID function is,

SELECT MID(column_name, start, length) from table-name;

Consider the following Emp table

eid	name	age	salary
401	anu	22	9000
402	shane	29	8000
403	rohan	34	6000
404	scott	44	10000
405	Tiger	35	8000

SQL query will be,

SELECT MID(name,2,2) FROM emp;

ROUND() Function

ROUND function is used to round a numeric field to number of nearest integer. It is used on Decimal point values.

Syntax of Round function is,

SELECT ROUND(column_name, decimals) from table-name;

Consider the following Emp table

eid	name	age	salary
401	anu	22	9000.67
402	shane	29	8000.98
403	rohan	34	6000.45
404	scott	44	10000
405	Tiger	35	8000.01

SQL query is,

SELECT ROUND(salary) from emp;

SQL JOIN

SQL Join is used to fetch data from two or more tables, which is joined to appear as single set of data. It is used for combining column from two or more tables by using values common to both tables.

JOIN Keyword is used in SQL queries for joining two or more tables.

Minimum required condition for joining table, is (n-1) where n, is number of tables. A table can also join to itself, which is known as, **Self Join.**

Types of JOIN

Following are the types of JOIN that we can use in SQL:

- Inner
- Outer
- Left
- Right

Cross JOIN or Cartesian Product

This type of JOIN returns the cartesian product of rows from the tables in Join. It will return a table which consists of records which combines each row from the first table with each row of the second table.

Cross JOIN Syntax is,

SELECT column-name-list
FROM
table-name1 CROSS JOIN table-name2;

Example of Cross JOIN

class table:

ID	NAME
1	abhi
2	adam
4	alex

class_info table:

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

Cross JOIN query will be,

```
SELECT * FROM
class CROSS JOIN class_info;
```

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	1	DELHI
4	alex	1	DELHI
1	abhi	2	MUMBAI
2	adam	2	MUMBAI
4	alex	2	MUMBAI
1	abhi	3	CHENNAI
2	adam	3	CHENNAI
4	alex	3	CHENNAI

INNER Join or EQUI Join

This is a simple JOIN in which the result is based on matched data as per the equality condition specified in the SQL query.

Inner Join Syntax is,

SELECT column-name-list FROM
table-name1 INNER JOIN table-name2
WHERE table-name1.column-name = table-name2.column-name;

Example of INNER JOIN

class table:

ID	NAME
1	abhi
2	adam
3	alex
4	anu

class_info table:

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

Inner JOIN query will be,

SELECT * from class INNER JOIN class_info where class.id = class_info.id;

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI

Natural JOIN

Natural Join is a type of Inner join which is based on column having same name and same datatype present in both the tables to be joined.

The syntax for Natural Join is,

SELECT * FROM
table-name1 NATURAL JOIN table-name2;

class table:

ID	NAME
1	abhi
2	adam
3	alex
4	anu

class_info table:

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

Natural join query will be,

SELECT * from class NATURAL JOIN class_info;

ID	NAME	Address
1	abhi	DELHI
2	adam	MUMBAI
3	alex	CHENNAI

OUTER JOIN

Outer Join is based on both matched and unmatched data.

Outer Joins subdivide further into,

- Left Outer Join
- Right Outer Join
- Full Outer Join

LEFT Outer Join

The left outer join returns a result set table with the matched data from the two tables and then the remaining rows of the left table and null from the right table's columns.

Syntax for Left Outer Join is,

```
SELECT column-name-list FROM
table-name1 LEFT OUTER JOIN table-name2
ON table-name1.column-name = table-name2.column-name;
```

To specify a condition, we use the ON keyword with Outer Join.

Example of Left Outer Join

Here is the class table,

ID	NAME
1	abhi
2	adam
3	alex
4	anu
5	ashish

and the class_info table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Left Outer Join query will be,

SELECT * FROM class LEFT OUTER JOIN class_info ON (class.id = class_info.id);

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI
4	anu	null	null
5	ashish	null	null

RIGHT Outer Join

The right outer join returns a result set table with the matched data from the two tables being joined, then the remaining rows of the right table and null for the remaining left table's columns.

Syntax for Right Outer Join is,

```
SELECT column-name-list FROM
table-name1 RIGHT OUTER JOIN table-name2
ON table-name1.column-name = table-name2.column-name;
```

Example of Right Outer Join

Once again the class table,

ID	NAME
1	abhi
2	adam
3	alex
4	anu
5	ashish

and the class_info table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Right Outer Join query will be,

SELECT * FROM class RIGHT OUTER JOIN class_info ON (class.id = class_info.id);

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI
null	null	7	NOIDA
null	null	8	PANIPAT

Full Outer Join

The full outer join returns a result set table with the matched data of two table then remaining rows of both left table and then the right table.

Syntax of Full Outer Join is,

```
SELECT column-name-list FROM
table-name1 FULL OUTER JOIN table-name2
ON table-name1.column-name = table-name2.column-name;
```

Example of Full outer join is,

The class table,

ID	NAME
1	abhi
2	adam
3	alex
4	anu
5	ashish

and the class_info table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Full Outer Join query will be like,

SELECT * FROM class FULL OUTER JOIN class_info ON (class.id = class_info.id);

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI
4	anu	null	null
5	ashish	null	null
null	null	7	NOIDA
null	null	8	PANIPAT

SQL Alias - AS Keyword

Alias is used to give an alias name to a table or a column, which can be a result set table too. This is quite useful in case of large or complex queries. Alias is mainly used for giving a short alias name for a column or a table with complex names.

Syntax of Alias for table names,
SELECT column-name FROM table-name AS alias-name
Example:
SELECT * FROM Employee_detail AS ed;

Syntax for defining alias for columns will be like,
SELECT column-name AS alias-name FROM table-name;
Example using alias for columns,
SELECT customer_id AS cid FROM Emp;

Example of Alias in SQL Query

Consider the following two tables,

SQL Alias seems to be quite a simple feature of SQL, but it is highly useful when you are working with more than 3 tables and have to use JOIN on them.

The class table,

ID	Name
1	abhi
2	adam
3	alex
4	anu
5	ashish

and the class_info table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Below is the Query to fetch data from both the tables using SQL Alias,

SELECT C.id, C.Name, Ci.Address from Class AS C, Class_info AS Ci where C.id = Ci.id;

ID	Name	Address
1	abhi	DELHI
2	adam	MUMBAI
3	alex	CHENNAI

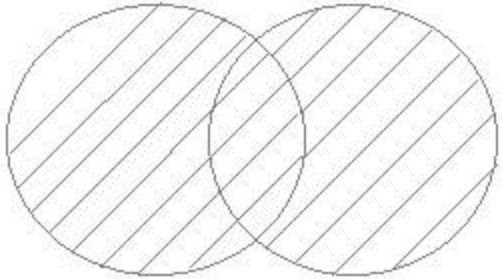
SET Operations in SQL

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.

1. UNION
2. UNION ALL
3. INTERSECT
4. MINUS

UNION Operation

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



Example of UNION

The First table,

ID	Name
1	abhi
2	adam

The Second table,

ID	Name
2	adam
3	Chester

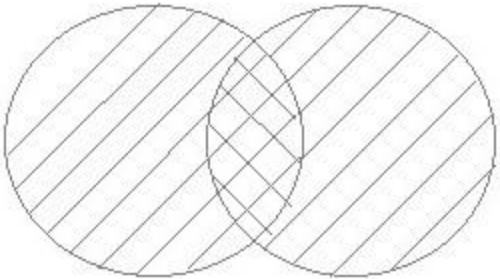
Union SQL query will be,

```
SELECT * FROM First
UNION
SELECT * FROM Second;
```

ID	NAME
1	abhi
2	adam
3	Chester

UNION ALL

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



Example of Union All

The First table,

ID	NAME
1	abhi
2	adam

The Second table,

ID	NAME
2	adam
3	Chester

Union All query will be like,

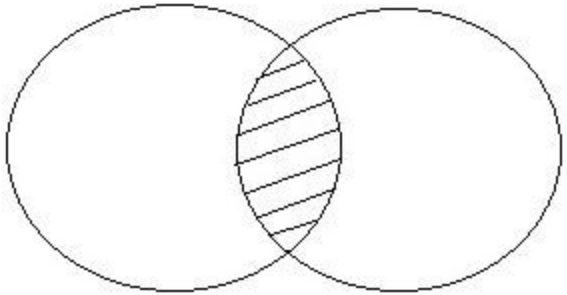
```
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
```

ID	NAME
1	abhi
2	adam
2	adam
3	Chester

INTERSECT

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

NOTE: MySQL does not support INTERSECT operator.



Example of Intersect

The First table,

ID	NAME
1	abhi
2	adam

The Second table,

ID	NAME
2	adam
3	Chester

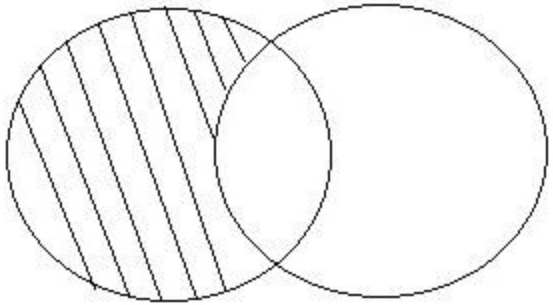
Intersect query will be,

```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

ID	NAME
2	adam

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.



Example of Minus

The First table,

ID	NAME
1	abhi
2	adam

The Second table,

ID	NAME
2	adam
3	Chester

Minus query will be,

```
SELECT * FROM First
MINUS
SELECT * FROM Second;
```

ID	NAME
1	abhi

What is an SQL Sequence?

Sequence is a feature supported by some database systems to produce unique values on demand. Some DBMS like MySQL supports AUTO_INCREMENT in place of Sequence.

AUTO_INCREMENT is applied on columns, it automatically increments the column value by 1 each time a new record is inserted into the table.

Sequence is also some what similar to AUTO_INCREMENT but it has some additional features too.

Creating a Sequence

Syntax to create a sequence is,

```
CREATE SEQUENCE sequence-name
   START WITH initial-value
   INCREMENT BY increment-value
   MAXVALUE maximum-value
   CYCLE | NOCYCLE;
```

- The initial-value specifies the starting value for the Sequence.
- The increment-value is the value by which sequence will be incremented.
- The maximum-value specifies the upper limit or the maximum value up to which sequence will increment itself.
- The keyword CYCLE specifies that if the maximum value exceeds the set limit, sequence will restart its cycle from the beginning.
- And, NO CYCLE specifies that if sequence exceeds MAXVALUE value, an error will be thrown.

Using Sequence in SQL Query

Let's start by creating a sequence, which will start from 1, increment by 1 with a maximum value of 999.

```
CREATE SEQUENCE seq_1
START WITH 1
INCREMENT BY 1
MAXVALUE 999
CYCLE;
```

Below we have a class table,

ID	NAME
1	abhi
2	adam
4	alex

The SQL query will be,

```
INSERT INTO class VALUE(seq_1.nextval, 'anu');
```

ID	NAME
1	abhi
2	adam
4	alex
1	anu

SQL VIEW

A VIEW in SQL is a logical subset of data from one or more tables. View is used to restrict data access.

Syntax for creating a View,

```
CREATE or REPLACE VIEW view_name
   AS
   SELECT column_name(s)
   FROM table_name
   WHERE condition
```

As you may have understood by seeing the above SQL query, a view is created using data fetched from some other table(s).
It's more like a temporary table created with data.

Creating a VIEW

Consider following Sale table,

oid	order_name	previous_balance	customer
11	ord1	2000	Alex
12	ord2	1000	Adam
13	ord3	2000	Abhi
14	ord4	1000	Adam
15	ord5	2000	Alex

SQL Query to Create a View from the above table will be,

```
CREATE or REPLACE VIEW sale_view
AS
SELECT * FROM Sale WHERE customer = 'Alex';
```

The data fetched from SELECT statement will be stored in another object called sale_view. We can use CREATE and REPLACE separately too, but using both together works better, as if any view with the specified name exists, this query will replace it with fresh data.

Displaying a VIEW

The syntax for displaying the data in a view is similar to fetching data from a table using a SELECT statement.

```
SELECT * FROM sale_view;
```

Force VIEW Creation

FORCE keyword is used while creating a view, forcefully. This keyword is used to create a View even if the table does not exist. After creating a force View if we create the base table and enter values in it, the view will be automatically updated.

```
CREATE or REPLACE FORCE VIEW view_name AS
   SELECT column_name(s)
   FROM table_name
   WHERE condition;
```

Update a VIEW

UPDATE command for view is same as for tables.

UPDATE view-name SET VALUE
WHERE condition;

NOTE: If we update a view it also updates base table data automatically.

Read-Only VIEW

We can create a view with read-only option to restrict access to the view.

```
CREATE or REPLACE FORCE VIEW view_name AS
   SELECT column_name(s)
   FROM table_name
   WHERE condition  WITH read-only;
```

The above syntax will create view for read-only purpose, we cannot Update or Insert data into read-only view. It will throw an error.

Types of View

There are two types of view,

- Simple View
- Complex View

Simple View	Complex View
Created from one table	Created from one or more table
Does not contain functions	Contain functions
Does not contain groups of data	Contains groups of data

MongoDB - Overview

MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

Database

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

Relationship of RDBMS terminology with MongoDB

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)
Database Server and Client	
Mysqld/Oracle	mongod
mysql/sqlplus	mongo

Sample Document

```
{
  _id: ObjectId(9df87ad1234c)
  title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',
  by: ‘talent battle',
  url: 'http://www.talentbattle.in',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100,
  comments:  [
    {
      user:'user1',
      message: 'My first comment',
      dateCreated: new Date(2011,1,20,2,15),
      like: 0
    },
    {
      user:'user2',
      message: 'My second comments',
      dateCreated: new Date(2011,1,25,7,45),
      like: 5
    }
  ]
}
```

_id is a 12 bytes hexadecimal number which assures the uniqueness of every document.

You can provide _id while inserting the document. If you don’t provide then MongoDB provides a unique id for every document.

These 12 bytes first 4 bytes for the current timestamp, next 3 bytes for machine id, next 2 bytes for process id of MongoDB server and remaining 3 bytes are simple incremental VALUE.

Advantages of MongoDB over RDBMS

- **Schema less** – MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.
- Structure of a single object is clear.
- No complex joins.
- Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
- Tuning.
- Ease of scale-out – MongoDB is easy to scale.
- Conversion/mapping of application objects to database objects not needed.
- Uses internal memory for storing the (windowed) working set, enabling faster access of data.

Why Use MongoDB?

- Document Oriented Storage – Data is stored in the form of JSON style documents.
- Index on any attribute
- Replication and high availability
- Auto-sharding
- Rich queries
- Fast in-place updates
- Professional support by MongoDB

Where to Use MongoDB?

- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure
- User Data Management
- Data Hub

THANK YOU!!!