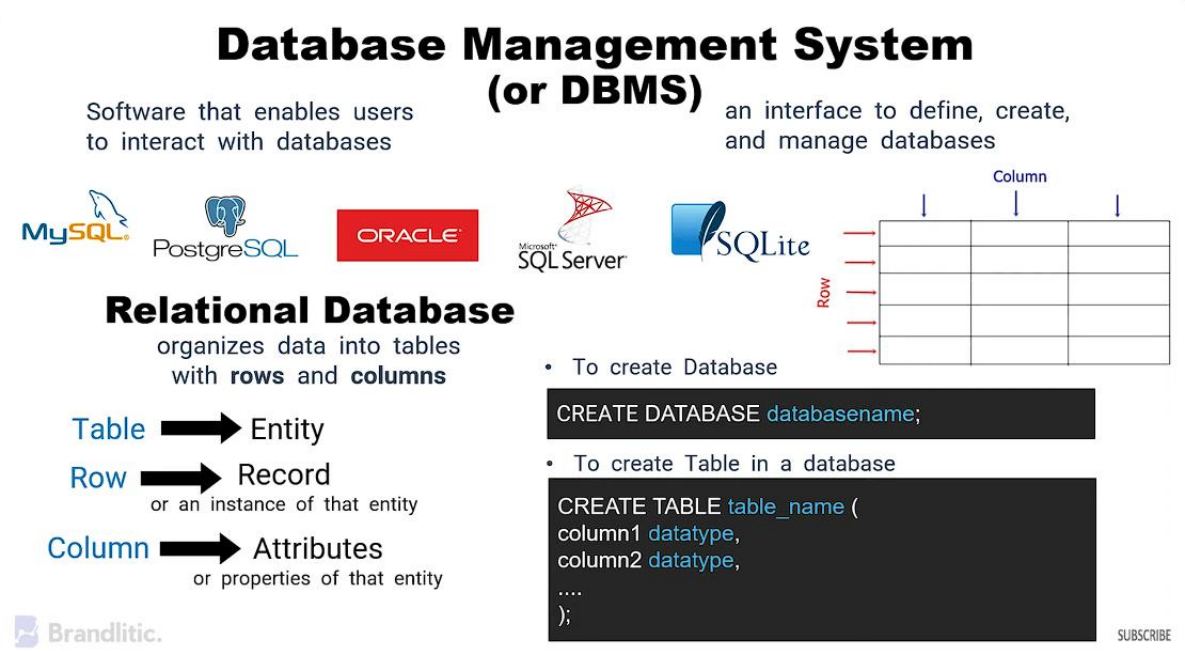
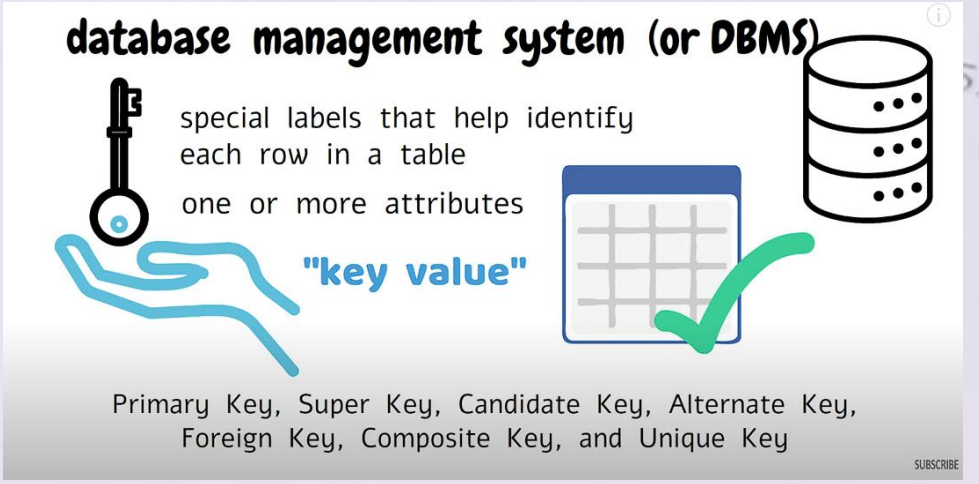
SQL/DBMS

- SQL (Structured Query Language) is a standard language used to manage and manipulate databases.
- It is a DBMS (Database Management System)
- It is a relational database which organises data into tables with rows and columns
- 

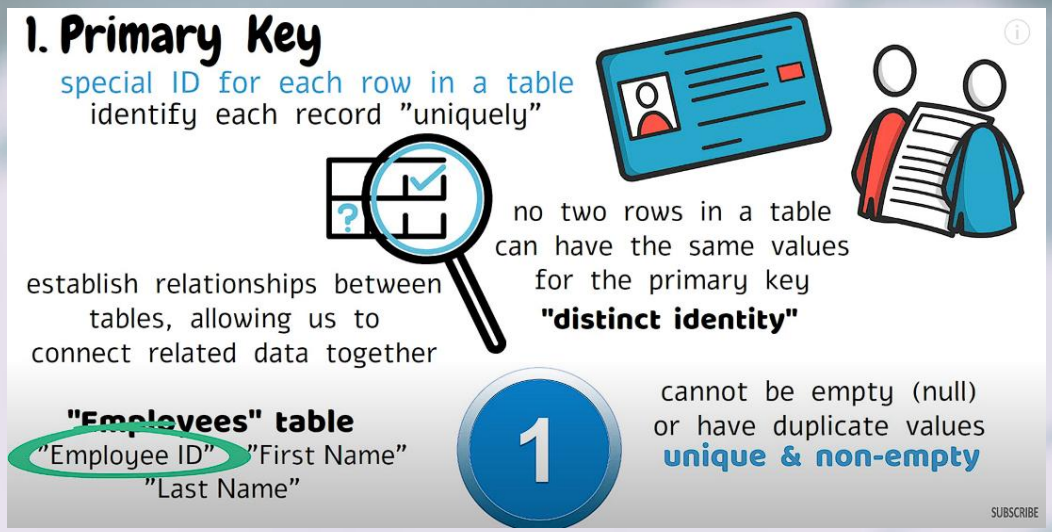
Database Concepts

- Database: A collection of related data organized in a structured format
- Table: A collection of rows and columns in a database.
- Row (Record): A single entry in a table.
- Column (Field): A specific attribute of the data.

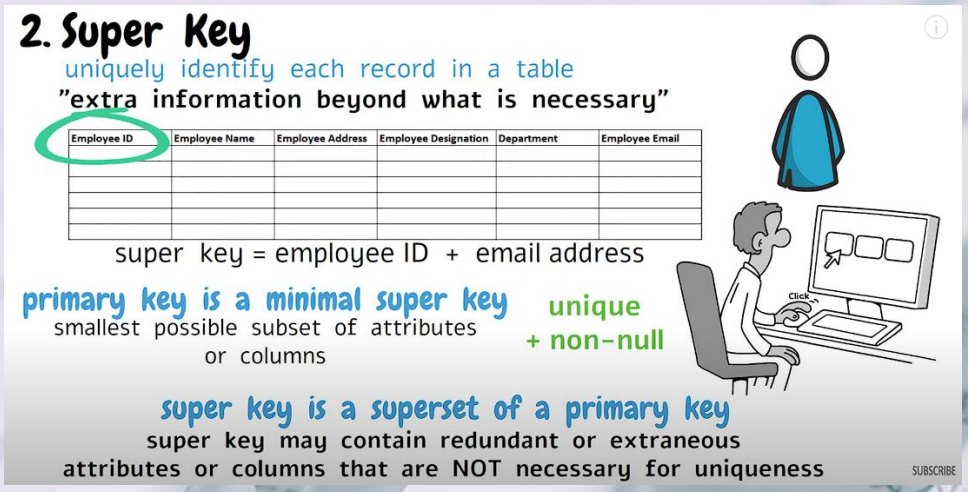
Keys



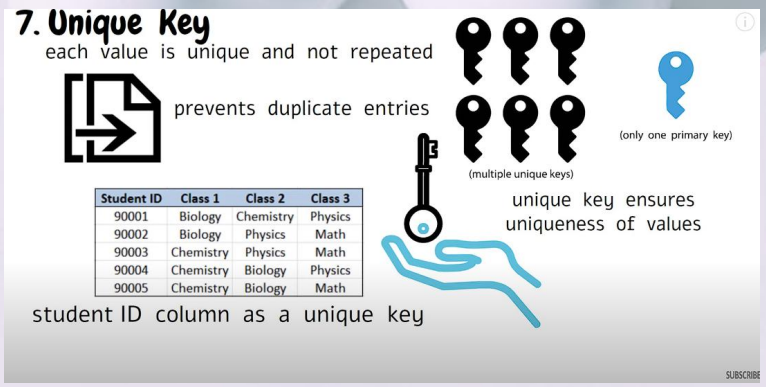
1. Primary Key



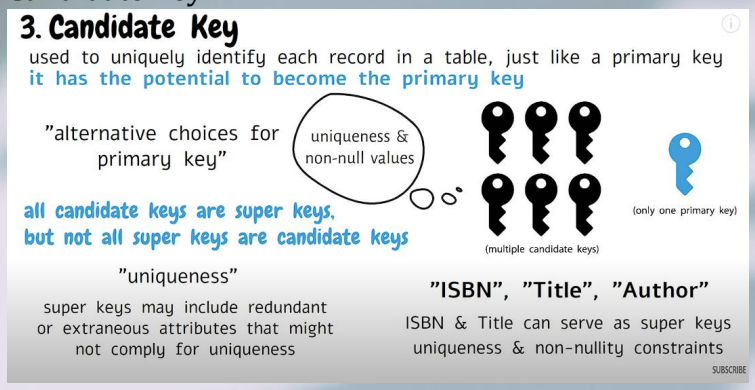
2. Super Key



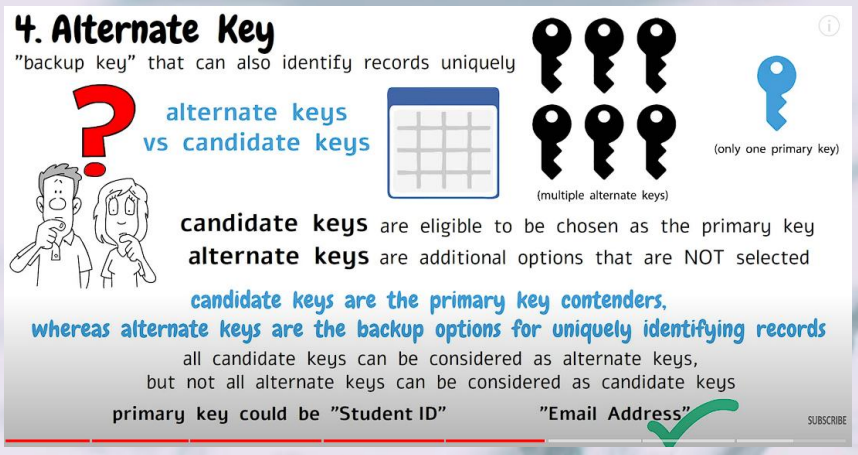
3. Unique Key



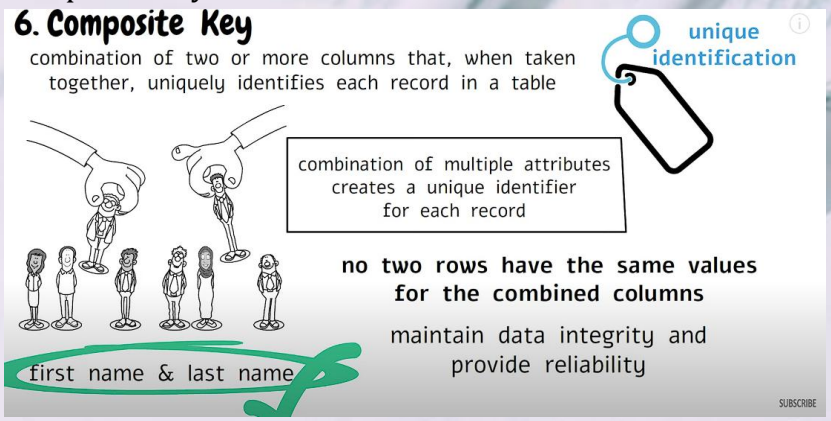
4. Candidate Key



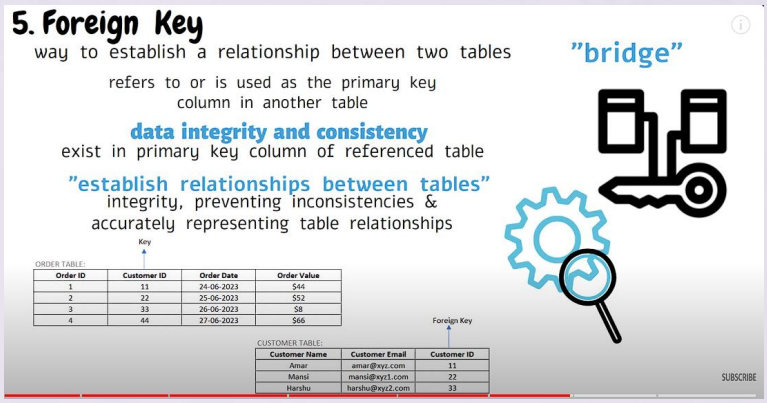
5. Alternate Key



6. Composite Key



7. Foreign Key



Index

- A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional storage and maintenance overhead
- Indexes are crucial for enhancing the performance of queries, especially in large databases

How Indexes Work

Indexes function similarly to an index in a book, allowing the database engine to quickly locate rows without scanning the entire table. Here's how they work in different scenarios:

- B-tree Indexes: Most common type, where indexes are stored in a balanced tree structure, making search operations efficient (logarithmic time complexity).
- Hash Indexes: Use a hash function to map keys to locations, ideal for exact match lookups but not suitable for range queries.
- Bitmap Indexes: Use bitmaps and are efficient for low-cardinality columns in read-heavy environments.

Advantages of Indexes

- Faster Query Performance: Significantly speeds up data retrieval operations.
- Efficient Sorting: Indexes can make ORDER BY operations faster. Create indexes on columns that are frequently used in WHERE clauses, JOIN conditions, and ORDER BY clauses
- Improved Join Operations: Speeds up joins by providing quick access paths to rows.

Disadvantages of Indexes

- Storage Overhead: Indexes consume additional disk space.
- Maintenance Overhead: Inserts, updates, and deletes can be slower because the index must be updated.
- Complexity: Over-indexing can lead to performance degradation and complexity in query optimization.

Types of Index

- Primary - An index on a primary key field which is unique and not null.
- Secondary - An index on non-primary key columns, allowing fast retrieval based on non-primary attributes.
- Composite Index - An index on multiple columns.
- Clustered Index - Determines the physical order of data in a table. There can be only one clustered index per table
- Full Text Index - Special index type optimized for searching text-based columns.
- Unique, Non clustered

Eg:

Example

```sql
-- Creating a primary index (automatically created when defining primary key)
CREATE TABLE users (
    id INT PRIMARY KEY,
    username VARCHAR(50),
    email VARCHAR(100)
);

-- Creating a secondary index
CREATE INDEX idx_username ON users (username);

-- Creating a composite index
CREATE INDEX idx_name ON persons (last_name, first_name);
```

SQL Commands

- **SELECT**: Retrieves data from a database.

```sql
SELECT column1, column2 FROM table_name;
```

- **INSERT**: Adds new rows to a table.

```sql
INSERT INTO table_name (column1, column2) VALUES (value1, value2);
```

- **UPDATE**: Modifies existing rows in a table.

```sql
UPDATE table_name SET column1 = value1, column2 = value2 WHERE condition;
```

- **DELETE**: Removes rows from a table.

```sql
DELETE FROM table_name WHERE condition;
```

- Note: In above CRUD operations, no need to mention the keyword 'TABLE', just mention the table_name
- But in below Create, Alter, Drop, mention the keyword 'TABLE'

Data Definition Language (DDL)

- **CREATE TABLE**: Creates a new table.

```sql
CREATE TABLE table_name (
  column1 datatype,
  column2 datatype,
  ...
);
```

- **ALTER TABLE**: Modifies an existing table.

```sql
ALTER TABLE table_name ADD column_name datatype;
```

- **DROP TABLE**: Deletes a table.

```sql
DROP TABLE table_name;
```

- **JOINS**

- **INNER JOIN**: Returns records that have matching values in both tables.

```sql
SELECT columns FROM table1 INNER JOIN table2 ON table1.column = table2.column;
```

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

```sql
SELECT columns FROM table1 LEFT JOIN table2 ON table1.column = table2.column;
```

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

```sql
SELECT columns FROM table1 RIGHT JOIN table2 ON table1.column = table2.column;
```

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

```sql
SELECT columns FROM table1 FULL JOIN table2 ON table1.column = table2.column;
```

➢

Other Queries

Consider emp table

```sql
INSERT INTO emp (EID, ENAME, EADD, ESAL, EDEP, EDATE) VALUES (40, 'anish', 'mango', 18000, 'marketing','1999-08
INSERT INTO emp (EID, ENAME, EADD, ESAL, EDEP, EDATE) VALUES (20, 'ankush', 'dimna', 17000, 'retail','2019-11-0
INSERT INTO emp (EID, ENAME, EADD, ESAL, EDEP, EDATE) VALUES (50, 'neha', 'noida', 21000, 'sales','2001-03-01')
INSERT INTO emp (EID, ENAME, EADD, ESAL, EDEP, EDATE)VALUES (60, 'kapil', 'mumbai', 16000, 'retail','1995-11-09
INSERT INTO emp (EID, ENAME, EADD, ESAL, EDEP, EDATE) VALUES (30, 'suman', 'dimna', 11000, 'sales','2000-12-09'

ALTER TABLE employees RENAME TO emp;
select * from emp;
```

EID	ENAME	EADD	ESAL	EDEP	EDATE
10	ankit	mango	20000.00	marketing	2021-01-01
20	ankush	dimna	17000.00	sales	2019-11-03
30	rohit	sakchi	13000.00	retail	2020-06-24
40	altaf	goa	23000.00	sales	2018-05-05
40	anish	mango	18000.00	marketing	1999-08-05
20	ankush	dimna	17000.00	retail	2019-11-03
50	neha	noida	21000.00	sales	2001-03-01
60	kapil	mumbai	16000.00	retail	1995-11-09
30	suman	dimna	11000.00	sales	2000-12-09

➢ **ORDER By**

- Sorts the result set in ascending or descending order.

```
#order by clause
select * from emp order by ename;
select * from emp order by eid asc, esal desc;  #first sort by eid
select * from emp order by esal desc, eid asc;  #first sort by esal
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

EID	ENAME	EADD	ESAL	EDEP	EDATE
40	altaf	goa	23000.00	sales	2018-05-05
50	neha	noida	21000.00	sales	2001-03-01
10	ankit	mango	20000.00	marketing	2021-01-01
40	anish	mango	18000.00	marketing	1999-08-05
20	ankush	dimna	17000.00	sales	2019-11-03
20	ankush	dimna	17000.00	retail	2019-11-03
60	kapil	mumbai	16000.00	retail	1995-11-09
30	rohit	sakchi	13000.00	retail	2020-06-24
30	suman	dimna	11000.00	sales	2000-12-09

- **GROUP BY**
  - Groups rows that have the same values in specified columns into summary rows. (Note: if u group by edep, then select edep alone or aggregate func)

```
#group by
select edep from emp group by edep; #only unique
select edep, count(*) from emp group by edep;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

edep	count(*)
marketing	2
sales	4
retail	3

  - You can add condition too wrt to other field/column like print edep who have salary greater than 17000

```
select edep, count(*) from emp where esal>17000 group by edep;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

edep	count(*)
marketing	2
sales	2

  - Department wise get the max/min salary and get the sum/average salaries of the corresponding departments

```
#Department wise get the max/min salary and get the sum/average salaries of the corresponding departments
select edep, max(esal), min(esal), sum(esal), avg(esal) from emp group by edep;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

edep	max(esal)	min(esal)	sum(esal)	avg(esal)
marketing	20000.00	18000.00	38000.00	19000.000000
sales	23000.00	11000.00	72000.00	18000.000000
retail	17000.00	13000.00	46000.00	15333.333333

- **HAVING**

- It is always used with 'GROUP By' clause (need that particular column to 'GROUP by' first, then only you can add 'HAVING' to it)
- It is used to restrict the group (similar to 'where')
- Order of execution is WHERE, GROUP by, HAVING, ORDER by
- Eg: Instead of where, using having by in above above statement

```
#having by
select edep, count(*) from emp  group by edep,esal having esal>17000 ;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

edep	count(*)
marketing	1
sales	1
marketing	1
sales	1

```
select edep, esal, count(*) from emp  group by edep,esal having esal>17000 ;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

edep	esal	count(*)
marketing	20000.00	1
sales	23000.00	1
marketing	18000.00	1
sales	21000.00	1

- **To get simple salary range and also create a duplicate table with same structure**

```
#to find details of employees who have salary as 23000,11000 and 13000
select * from emp where esal in (23000,11000,13000);
#to find details of employees who have salary between 10000 and 15000
select * from emp where esal between 10000 and 15000;

#to create a duplicate table with same structure
create table emp2 as select * from emp;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

EID	ENAME	EADD	ESAL	EDEP	EDATE
30	rohit	sakchi	13000.00	retail	2020-06-24
30	suman	dimna	11000.00	sales	2000-12-09

- **To find patterns**

```
#to find all details of employees whose name starts with 'a'
select * from emp where ename like 'a%';
#to find all details of employees whose name ends with 'a'
select * from emp where ename like '%a';
#to find all details of employees whose name does not have letter 'a'
select * from emp where ename not like '%a%';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

EID	ENAME	EADD	ESAL	EDEP	EDATE
30	rohit	sakchi	13000.00	retail	2020-06-24

- **To find max salary, 2nd max salary or nth max salary**

```
#to find max salary
select * from emp where esal = (select max(esal) from emp);

#to find 2nd max salary
select max(esal) from emp where esal<(select max(esal) from emp);
#or
select max(esal) from emp where esal not in (select max(esal) from emp);

#to find nth max salary (offset = n-1)
select distinct esal from emp order by esal desc limit 1 offset 2;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

esal
20000.00

- **To find repeating/duplicate values of a column like eid or ename**

```
#to find repeating/duplicate values of a column like eid or ename
select eid, count(*) from emp group by eid having count(*)>1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

eid	count(*)
20	2
30	2
40	2

- **To find other patterns**