Agenda

- Keys
  - Candidate key
  - Primary key
  - Composite key
  - Foreign key
- Introduction to SQL

Candidate Keys

Now let's re-consider **Super Keys**. let's remove the columns that weren't necessary in Super Keys.

Also, let's say we were an offline school, and students don't have email or phone number. In that case, what do you think schools use to uniquely identify a student? Eg: If we remove redundant columns from (name, email, psp), we will be left with (email). Similarly, if we remove redundant columns from (name, email, phone number), we will be left with (phone number) or (email). These are known as `candidate keys`.

A candidate key is a super key from which no column can be removed and still have the property of uniquely identifying a row. If any more column is removed from a candidate key, it will no longer be able to uniquely identify a row.

Example of Candidate Keys using a table

Let's take another example. Consider a table Scaler has for storing student's attendance for every class.

`batches`

student_id	class_id	attendance
1	2	100
1	3	90
2	2	89
2	5	100
2	3	87

| student_id | class_id | attendance |

What do you think are the candidate keys for this table? Do you think (student_id) is a candidate key? Will there be only 1 row with a particular student_id? The student can attend multiple classes at Scaler ex: DBMS1, Keys etc.. in all these cases student_id is same for that particular Student hence it is not unique.

Is (class_id) a candidate key? Will there be only 1 row with a particular class_id? Multiple students can attend multiple classes at Scaler ex: DBMS is having a class_id and multiple students can attend that class hence it is not unique.

Is (student_id, class_id) a candidate key? Will there be only 1 row with a particular combination of student_id and class_id? Yes, a student can attend a class only one time example: Rahul can attend class DBMS once only hence this combination is going to be unique.

Yes! (student_id, class_id) is a candidate key. If we remove any of the columns of this, the remanining part is not a candidate key. Eg: If we remove student_id, we will be left with (class_id). But there can be multiple rows with same class_id. Similarly, if we remove class_id, we will be left with (student_id). But there can be multiple rows with same student_id. Hence, (student_id, class_id) is a candidate key.

> *Activity: Please try to make these pairs from table above to verify this concept.*

Is (student_id, class_id, attendance) a candidate key? Will there be only 1 row with a particular combination of student_id, class_id and attendance?

Yes there is only one row, but can we remove any column from this and still have a candidate key? Eg: If we remove attendance, we will be left with (student_id, class_id). This is a candidate key. Hence, (student_id, class_id, attendance) is not a candidate key.

Now let's have few quizzes:

Quiz 1

Is a candidate key always a super key?

Choices

- [ ] Yes
- [ ] No

Quiz 2

Is a super key always a candidate key?

Choices

- [ ] Yes
- [ ] No

Quiz 3

Which of the following is a Candidate Key for the Employee table?

Choices

- [ ] {EmployeeID, Department}
- [ ] {Email}
- [ ] {FirstName, LastName}
- [ ] {LastName, Department}

Quiz 4

If both EmployeeID and Email are unique for each employee, which of these could be a Candidate Key for the Employee table?

Choices

- [ ] {EmployeeID, Email}
- [ ] {EmployeeID}
- [ ] {Email}
- [ ] Both B and C

Quiz 5

Which of these combinations is NOT a Candidate Key for the Employee table?

Choices

- [ ] {EmployeeID}
- [ ] {Email}
- [ ] {LastName, Department}

Primary Key

Primary Key

We just learnt about super keys and candidate keys. Can 1 table have mulitiple candidate keys? Yes. The Student's table earlier had both `email`, `phone number` as candidate keys. A key in MySQL plays a very important role. Example, MySQL orders the data in disk by the key. Similarly, by default, it returns answers to queries ordered by key. Thus, it is important that there is only 1 key. And that is called primary key. A primary key is a candidate key that is chosen to be the key for the table. In the students table, we can choose `email` or `phone number` as the primary key. Let's choose `email` as the primary key.

> *Note: Internally,*
>
> 1. *Database sorts the data by primary key.*
> 2. *Database outputs the result of every query sorted by primary key.*
> 3. *Database creates an index as well on primary key.*

Sometimes, we may have to or want to create a new column to be the primary key. Eg: If we have a students table with columns (name, email, phone number), we may have to create a new column called roll number or studentId to be the primary key. This may be because, let's say, a user can change their email or phone number if they want. Something that is used to uniquely identify a row should ideally never change. Hence, we create a new column called roll number or studentId to be the primary key.

> *A good primary key should:*
>
> 1. *be fast to sort on.*

> 2. *have smaller size (to reduce the space required for behind the scene indexing).*
> 3. *not get changed.*

Therefore, it is preferred to have a primary key with single integer column.

We will see later on how MySQL allows to create primary keys etc. Before we go to foreign keys and composite keys, let's actually get our hands dirty with SQL, post that it will be easy to understand how to create a PK.

Now let's have a quizz:

Quiz 6

Which of the following can be a good PK in students table?

Choices

- [ ] {Email}
- [ ] {Email, Phone_number}
- [ ] {Phone_number}
- [ ] {Student_Id}

Introduction to SQL

First of all, what is SQL? SQL stands for Structured Query Language. It is a language used to interact with relational databases. It allows you to create tables, fetch data from them, update data, manage user permissions etc. Today we will just focus on creation of data. Remaining things will be covered over the coming classes. Why "Structured Query" because it allows to query over data arranged in a structured way. Eg: In Relational databases, data is structured into tables.

Create table in MySQL

A simple query to create a table in MySQL looks like this:

```
CREATE TABLE students (
    id INT AUTO_INCREMENT,
    firstName VARCHAR(50) NOT NULL,
    lastName VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    dateOfBirth DATE NOT NULL,
    enrollmentDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    psp DECIMAL(3, 2) CHECK (psp BETWEEN 0.00 AND 100.00),
    batchId INT,
    isActive BOOLEAN DEFAULT TRUE,
    PRIMARY KEY (id),
);

-- We can add Primary key separately like in this query.
```

Here we are creating a table called students.

- Inside brackets, we mention the different columns that this table has. Along with each columns, we mention the data type of that column. Eg: firstName is of type VARCHAR(50). Please do watch the video on SQL Data Types attached at bottom of notes to understand what VARCHAR, TIMESTAMP etc. means. For this discussion, it suffices to know that these are different data types supported by MySQL.
- After the data type, we mention any constraints on that column. Eg: NOT NULL means that this column cannot be null. In next notes when we will learn how to insert data, if we try to not put a value of this column, we will get an error.
- UNIQUE means that this column cannot have duplicate values. If we insert a new row in a table, or update an existing row that leads to 2 rows having same value of this column, the query will fail and we will get an error.
- DEFAULT specifies that if no value is provided for this column, it will take the given value. Example, for enrollmentDate, it will take the value of current_timestamp, which is the time when you are inserting the row.
- CHECK (psp BETWEEN 0.00 AND 100.00) means that the value of this column should be between 0.00 and 100.00. If some other value is inserted, the query will fail.
- Resources for Data_Type videos is shared at bottom of these notes.
- More on SQL constraints: https://www.scaler.com/topics/sql/constraints-in-sql/

Types of SQL Commands

> *Based on kind of work a SQL query does we have categorised them into following types:*

- DDL(Data Definition Language): To make/perform changes to the physical structure of any table residing inside a database, DDL is used. These commands when executed are auto-commit in nature and all the changes in the table are reflected and saved immediately.
- DML(Data Manipulation Language): Once the tables are created and the database is generated using DDL commands, manipulation inside those tables and databases is done using DML commands. The advantage of using DML commands is, that if in case any wrong changes or values are made, they can be changed and rolled back easily.
- DQL(Data Query Language): Data query language consists of only one command upon which data selection in SQL relies. The SELECT command in combination with other SQL clauses is used to retrieve and fetch data from databases/tables based on certain conditions applied by the user.
- DCL(Data Control Language): DCL commands as the name suggests manage the matters and issues related to the data controller in any database. DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.
- TCL(Transaction Control Language): Transaction Control Language as the name suggests manages the issues and matters related to the transactions in any

database. They are used to roll back or commit the changes in the database.

- Based on above findings following are the examples of SQL commands.

Privilege	Description
SELECT	*select statement on tables*
INSERT	*insert statement on the table*
DELETE	*delete statement on the table*
INDEX	*Create an index on an existing table*
CREATE	*Create table statements*
ALTER	*Ability to perform ALTER TABLE to change the table definition*
DROP	*Drop table statements*
ALL	*Grant all permissions except GRANT OPTION*
UPDATE	*Update statements on the table*
GRANT	*Allow to grant and manage privileges*

- For more detailed analysis please refer to Scaler Topic's aticle: https://www.scaler.com/topics/dbms/sql-commands/

Composite Keys

Now, lets's get into composite key.

A composite key is a key with more than one column. Any key with multiple columns (a collection of columns) is a composite key.

> *Note: Super, Candidate and Primary keys can be of both type - either a single key or a composite key.*

Foreign Keys

Foreign Keys

Now let's get to the last topic of the day. Which is foreign keys. Let's say we have a table called batches which stores information about batches at Scaler. It has columns (id, name). We would want to know for every student, which batch do they belong to. How can we do that?

batch_id	batch_name
1	Batch A
2	Batch B
3	Batch C

student_id	first_name	last_name
1	John	Doe
2	Jane	Doe
3	Jim	Brown
4	Jenny	Smith
5	Jack	Johnson

Correct, We can add batchId column in students table. But how do we know which batch a student belongs to? How do we ensure that the batchId we are storing in the students table is a valid batchId? What if someone puts the value in batchID column as 4 but there is no batch with id 4 in batches table. We can set such kind of constraints using foreign keys. **A foreign key is a column in a table that references a column in another table.** It has nothing to do with primary, candidate, super keys. It can be any column in 1 table that refers to any column in other table. In our case, batchId is a foreign key in the students table that references the id column in the batches table. This ensures that the batchId we are storing in the students table is a valid batchId. If we try to insert any value in the batchID column of students table that isn't present in id column of batches table, it will fail. Another example:

Let's say we have `years` table as: `| id | year | number_of_days |`

and we have a table students as: `| id | name | year |`

Is `year` column in students table a foreign key?

The correct answer is yes. It is a foreign key that references the id column in years table. Again, foreign key has nothing to do with primary key, candidate key etc. It is just any column on one side that references another column on other side. Though often it doesn't make sense to have that and you just keep primary key of the other table as the foreign key. If not a primary key, it should be a column with unique constraint. Else, there will be ambiguities.

Okay, now let's think of what can go wrong with foreign keys?

Correct, let's say we have students and batches tables as follows:

batch_id	batch_name

1	Batch A
2	Batch B
3	Batch C

student_id	first_name	last_name	batch_id
1	John	Doe	1
2	Jane	Doe	1
3	Jim	Brown	2
4	Jenny	Smith	3
5	Jack	Johnson	2

Now let's say we delete the row with batch_id 2 from batches table. What will happen? Yes, the students Jim and Jack will be orphaned. They will be in the students table but there will be no batch with id 2. This is called orphaning. This is one of the problems with foreign keys. Another problem is that if we update the batch_id of a batch in batches table, it will not be updated in students table. Eg: If we update the batch_id of Batch A from 1 to 4, the students John and Jane will still have batch_id as 1. This is called inconsistency.

To fix for these, MySQL allows you to set ON DELETE and ON UPDATE constraints when creating a foreign key. You can specify what should happen in case an update or a delete happens in the other table. What do you think are different possibilities of what we can do if a delete happens?

You can set 4 values for ON DELETE and ON UPDATE. They are:

1. CASCADE: If the referenced data is deleted or updated, all rows containing that foreign key are also deleted or updated.
2. SET NULL: If the referenced data is deleted or updated, the foreign key in all rows containing that foreign key is set to NULL. This assumes that the foreign key column is not set to NOT NULL.
3. NO ACTION: If the referenced data is deleted or updated, MySQL will not execute the delete or update operation for the parent table. This is the default action.
4. SET DEFAULT: If the referenced data is deleted or updated, the foreign key in all the referencing rows is set to its default values. This is only functional with tables that use the InnoDB engine and where the foreign key column(s) have not been defined to have a NOT NULL attribute.

Practical example - Foreign Keys

Now let's see how to create a table with a foreign key. Let's say we want to create a table called students with columns (id, name, batch_id). We want batch_id to be a foreign key that references the id column in batches table. We want that if a batch is deleted, all students in that batch should also be deleted. We can do that as follows:

```
-- Creating 'batches' table
CREATE TABLE batches (
```

```
    batch_id INT PRIMARY KEY,
    batch_name VARCHAR(50) NOT NULL
);

-- Inserting dummy data into 'batches' table
INSERT INTO batches(batch_id, batch_name) VALUES
(1, 'Batch A'),
(2, 'Batch B'),
(3, 'Batch C');

-- Creating 'students' table with ON DELETE and ON UPDATE constraints
CREATE TABLE students (
    student_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    batch_id INT,
    FOREIGN KEY (batch_id) REFERENCES batches(batch_id) ON DELETE CASCADE ON UPDATE
CASCADE
);

-- Inserting dummy data into 'students' table
INSERT INTO students(first_name, last_name, batch_id) VALUES
('John', 'Doe', 1),
('Jane', 'Doe', 1),
('Jim', 'Brown', 2),
('Jenny', 'Smith', 3),
('Jack', 'Johnson', 2);
```

Now, let's try to delete a batch and see what happens:

```
DELETE FROM batches WHERE batch_id = 1;
```

Answer: It will delete the row from the `batches` table where the `batch_id` is 1. Since the `batch_id` column in the `batches` table is defined as the primary key, which uniquely identifies each row, this query will delete the specific batch with the ID of 1.

Additionally, due to the foreign key constraint defined in the `students` table with the `ON DELETE CASCADE` option, all associated rows in the `students` table with the matching `batch_id` will also be deleted. In this case, the students John Doe and Jane Doe (who belong to Batch A) will also be deleted.

Now, let's see what happens if we update a batch:

```
UPDATE batches SET batch_id = 4 WHERE batch_id = 2;
```

Answer: It will update the `batch_id` to 4 in the `batches` table where the value was 2. Since `batch_id` is the primary key of the `batches` table, this update will modify the specific row with the ID of 2.

Since the `batch_id` column is referenced as a foreign key in the `students` table, updating the `batch_id` in the `batches` table will also update the corresponding value in the `students` table due to the `ON UPDATE CASCADE` option specified in the foreign

key constraint. Therefore, any students associated with Batch B (which had the original batch_id of 2) will now be associated with Batch 4.

We can also add foreign keys to a table after the table has been created by using the ALTER command. Let's look at the syntax:

```
ALTER TABLE table_name
ADD FOREIGN KEY (column_name)
REFERENCES other_table(column_in_other_table);
```

Data Types in SQL

What are **Data Types in SQL**?

A data type is a property that describes the sort of data that an object can store, such as **integer data**, **character data**, **monetary data**, **date and time data**, **binary strings**, and so on.

> *MySQL String Data Types*

Data Type	Description	Size
CHAR(size)	A FIXED length string containing letters, numbers, and special characters. The size parameter determines the length of the column in characters. The default value is 1.	0 - 255 bytes
VARCHAR(size)	A VARIABLE length string containing letters, numbers, and special characters. The size option sets the maximum character length for a column.	0 - 65535 bytes
BINARY(size)	It is a type of function that stores binary byte strings. The size argument indicates the length of the column in bytes. The default value is 1.	0 - 255 bytes
VARBINARY(size)	It is a type of function that stores binary byte strings. The size argument sets the maximum length of a column in bytes.	0 - 65535 bytes
TINYBLOB	For use with BLOBs (Binary Large Objects).	255 bytes
BLOG(size)	For use with BLOBs (Binary Large Objects).	65,535 bytes
MEDIUMBLOB	For use with BLOBs (Binary Large Objects).	16,777,215 bytes
LONGBLOG	For use with BLOBs (Binary Large Objects).	4,294,967,295 bytes

TINYTEXT(size)	It stores a string.	255 characters
TEXT(size)	It stores a string.	65,535 characters
MEDIUMTEXT(size)	It stores a string.	16,777,215 characters
LONGTEXT(size)	It stores a string.	4,294,967,295 characters
ENUM(val1, val2, val3, ...)	It is used when a string object has only one value, which is selected from a list of possible values. If you enter a value that does not exist in the list, a blank value will be entered.	65535 values
SET(val1, val2, val3, ...)	It is used to describe a string with 0 or more values from a list of available values.	64 values

MySQL Numeric Data Types

Data Type	Description	Size
BOOL/ BOOLEAN	Synonym for TINYINT(1). Nonzero values are deemed true, while zero values are considered false.	
BIT(size)	A type of bit-value. Size specifies the number of bits per value. The size parameter accepts values ranging from 1 to 64. The default value is 1.	
TINYINT(size)	An extremely tiny integer. The signed range is -128 to 127. The unsigned range extends from 0 to 255. The size parameter determines the maximum width of the display (which is 255).	1 byte
SMALLINT(size)	A small integer. The signed range is -32768 to 32767. The unsigned range is 0 to 65535. The size parameter determines the maximum width of the display (which is 255).	2 bytes
MEDIUMINT(size)	A medium integer. The signed range is -8388608 to 8388607. The unsigned range is 0 to 16777215. The size parameter sets the maximum display width (which is 255).	3 bytes
INT(size)	An integer value. The signed range is -2147483648 to -2147483647. The unsigned range is 0 to 4294967295. The size parameter determines the maximum width of the display (which is 255).	4 bytes
INTEGER(size)	As same as to INT(size).	4 bytes

INTEGER(size)	As same as to INT(size).	4 bytes
BIGINT(size)	A large integer. The signed range is -9223372036854775808 to 9223372036854775807. Unsigned numbers range from 0 to 18446744073709551615. The size parameter determines the maximum width of the display (which is 255).	8 bytes
FLOAT(size, d)	A number with a decimal value. Size specifies the total number of digits. The d parameter specifies the number of digits after the decimal point. It has been deprecated in MySQL 8.0.17 and will be removed in future releases.	4 bytes
FLOAT(p)	A number with a decimal value. MySQL utilizes the p value to determine if the final data type is FLOAT or DOUBLE. If p is between 0 and 24, the data type is changed to FLOAT (). If p is between 25 and 53, the data type is changed to DOUBLE ().	4 bytes if 0 <= p <= 24, 8 bytes if 25 <= p <= 53
DOUBLE(size, d)	A standard-size floating-point number. Size specifies the total number of digits. The d option specifies the number of digits after the decimal point.	8 bytes
DOUBLE PRECISION(size, d)	As same as DOUBLE(size, d)	8 bytes

Detailed article on data types in SQL available at Scaler topics: https://www.scaler.com/topics/sql/sql-data-types/

Video link MySQL Data Types: https://drive.google.com/file/d/1GHeBM4nEB-CCZ3SMbRJxhjIwrZ_2TAZx/view

---

Solution to Quizzes:

```
-- Quiz1: Option A (Yes) Quiz2: Option B (No) Quiz3: Option B {Email} Quiz4: Option D (Both B and C) Quiz5: Option C {LastName, Department} Quiz5: Option D {Student_Id} --
```