CREATING AND MODIFYING RELATIONS USING SQL

Structured Query Language (SQL)

- ✓ SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987
- ✓ Common language for all Databases
- ✓ Fourth generation Language
- ✓ Non procedural Language
- ✓ Commands like an normal English statements
- ✓ SQL is not a case sensitive language
- ✓ All SQL statements should ended with terminator , the default terminator is semi-colon (;)
- ✓ Based on the operation SQL divided into three categories
    - ✓ DDL (Data Definition Language)
    - ✓ DML (Data Manipulation Language)
    - ✓ DCL (Data Control Language)

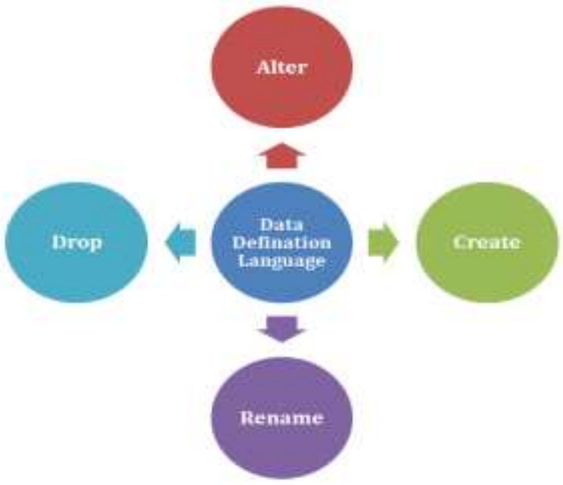
Data types in SQL (ORACLE)

- ✓ CHAR
- ✓ VARCHAR2
- ✓ NUMBER
- ✓ DATE
- ✓ RAW
- ✓ LONG
- ✓ CLOB

- ✓ BLOB , etc.,

Data Definition Language (DDL)

- ✓ DDL is the subset of SQL and part of DBMS
- ✓ DDL relates only with base tables structure and it is no where relates with the information stored in the table.
- ✓ **Note : All the DDL command statements are AUTO COMMIT Statements**



Used to create a new object / schema with a defined structure

CREATE TABLE table **CREATE COMMAND**

_name (column1 datatype, column2 datatype, column3 datatype, ...);

CREATE TABLE EMP(EMPNO NUMBER(4) NOT NULL, ENAME VARCHAR2(10), JOB VARCHAR2(9), MGR NUMBER(4), HIREDATE DATE, SAL NUMBER(7, 2), COMM NUMBER(7, 2), DEPTNO NUMBER(2));

ALTER COMMAND

Alter command used to modify the base table structure

Using this command

a new column can be added with restrictions column data width can be increased / decreased with restrictions

a column can be dropped

Two key words are using in this command

ADD

MODIFY

ALTER TABLE table_name ADD / MODIFY column_name datatype;

EXAMPLE 1: To add a new column in a table

ALTER TABLE emp ADD phone_no number(10);

EXAMPLE 2 : To modify the existing column data width

ALTER TABLE emp MODIFY phone_no number(13);

EXAMPLE: STUDENT RELATION

Table Structure:

StudentID (Primary Key):

Unique identifier for each student.

Data Type: Integer.

Auto-incremented to ensure uniqueness.

Name:

Full name of the student.

Data Type: VARCHAR.

Maximum Length: 50 characters.

Date of Birth:

Birthdate of the student.

Data Type: Date.

Gender:

Gender of the student.

Data Type: CHAR.

Values: 'M' for male, 'F' for female, 'O' for other.

Address:

Residential address of the student.

Data Type: VARCHAR.

Maximum Length: 255 characters.

Email:

Email address of the student.

Data Type: VARCHAR.

Maximum Length: 100 characters.

CREATE TABLE SYNTAX

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

CREATE THE STUDENTS TABLE

To create the students table with the specified columns, the SQL command would look like this:

```
CREATE TABLE students (

    student_id INT PRIMARY KEY,

    fname VARCHAR(50),
    date_of_birth DATE,

    gender CHAR,

    address VARCHAR(255),

   email VARCHAR(100),

);
```

This command creates a table with six columns, with student_id set as the primary key to uniquely identify each student.

ALTER TABLE - ADD Column

To add a column in a table, use the following syntax:

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

ALTER THE STUDENTS TABLE

The ALTER TABLE command allows to add a new column to the table

Add a Column

Let's consider that if a new column student's phone number is to be added:

```
ALTER TABLE students ADD phone_number VARCHAR(20);
```

ALTER TABLE - MODIFY

To change the data type of a column in a table, use the following syntax:

```
ALTER TABLE table_name MODIFY COLUMN column_name datatype;
```

MODIFY A COLUMN

Perhaps the major column's maximum length of 100 characters is not sufficient, and you want to increase it to 150:

```
ALTER TABLE students MODIFY major VARCHAR(150);
```

ALTER TABLE - DROP COLUMN

To delete a column in a table, use the following syntax :

ALTER TABLE table_name DROP COLUMN column_name;

ALTER TABLE - DROP

If you decide that you no longer need the email column:

ALTER TABLE students DROP COLUMN email;

ALTER TABLE - RENAME COLUMN

To rename a column in a table, use the following syntax:

ALTER TABLE table_name RENAME COLUMN old_name to new_name;

RENAME A COLUMN-EXAMPLE

To rename the phone_number column to contact_number:

ALTER TABLE students RENAME COLUMN phone_number TO contact_number;

INTEGRITY CONSTRAINTS OVER RELATIONS

Domain Constraints:

- These define the permissible values for a given attribute (column) within a relation (table).
- For example, a domain constraint might specify that the age column in a person table must contain integer values between 0 and 150.

Primary Key Constraints:

- A primary key constraint ensures that the values in the primary key column(s) are unique across all rows in the table and that no part of the primary key can be null.
- This is crucial for uniquely identifying each record in a relation.

Foreign Key Constraints:

- A foreign key is an attribute or combination of attributes that is used to establish and enforce a link between the data in two tables.

- The foreign key in one table points to a primary key in another table, ensuring referential integrity.
- For instance, if you have an orders table with a customer_id foreign key, that customer_id must exist in the customers table.

Unique Constraints:

- These constraints ensure that all values in a column or a group of columns are distinct across the rows within the table.
- Unlike primary keys, unique constraints can be nullable.

Check Constraints:

- Check constraints specify a predicate that all the data must satisfy. If data being entered or updated violates this check, the database will not accept the data.
- An example might be ensuring that a percentage column contains only values between 0 and 100.

Not Null Constraints:

- These constraints ensure that a column cannot have a NULL value, which means that you must provide a value for this column whenever you insert or update a row.

DOMAIN CONSTRAINT

- Domain constraints are the most elementary form of integrity constraint.
- They test values inserted in the database, and test queries to ensure that the comparisons make sense.
- CREATE TABLE Employee (ID INT, Age INT CHECK (Age >= 18 AND Age <= 65) -- Domain constraint for 'Age');

PRIMARY KEY CONSTRAINT

- The PRIMARY KEY constraint uniquely identifies each record in a table.

- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns.
- CREATE TABLE Employee (ID INT NOT NULL, Name VARCHAR(100), PRIMARY KEY (ID));

FOREIGN KEY CONSTRAINT

- A foreign key constraint is used to link the data in two tables.
- CREATE TABLE Orders (OrderID INT, EmployeeID INT, FOREIGN KEY (EmployeeID) REFERENCES Employee(ID));

UNIQUE CONSTRAINT

- Unique constraint ensures that all values in a column are different.
- CREATE TABLE Employee (ID INT NOT NULL, Email VARCHAR(255), UNIQUE (Email));

CHECK CONSTRAINT

- Check constraint allows specifying a condition that each row must meet.
- CREATE TABLE Employee (ID INT, Salary DECIMAL CHECK (Salary > 0) -- Check constraint that 'Salary' must be greater than 0);

NOT NULL CONSTRAINT

- NOT NULL constraint ensures that a column cannot have a NULL value.
- CREATE TABLE Employee (ID INT NOT NULL, Name VARCHAR(100) NOT NULL);

INTEGRITY CONSTRAINTS OVER RELATIONS

Constraints

✓ An E-R enterprise schema may define certain constraints to which the contents of a database must conform.

- ✓ This is achieved using

Mapping Cardinalities

Participation Constraints

Mapping Cardinalities

- ✓ Mapping cardinalities, or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set.
- ✓ Mapping cardinalities are most useful in describing binary relationship sets.
- ✓ For a binary relationship set "Assign" between entity sets Programmer and Project the mapping cardinality must be one of the following.

One-to-One (1:1)

One-to-Many (1:M)

Many-to-One (M:1)

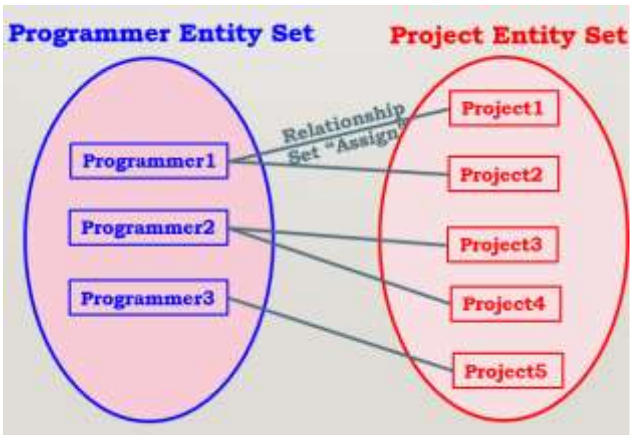
Many-to-Many (M:M)

One-to-One (1:1)

  - ✓ An entity in Programmer is associated with at most one entity in Project, and an entity in Project is associated with at most one entity in Programmer.
  - ✓ The following figure depicts 1:1 mapping cardinality

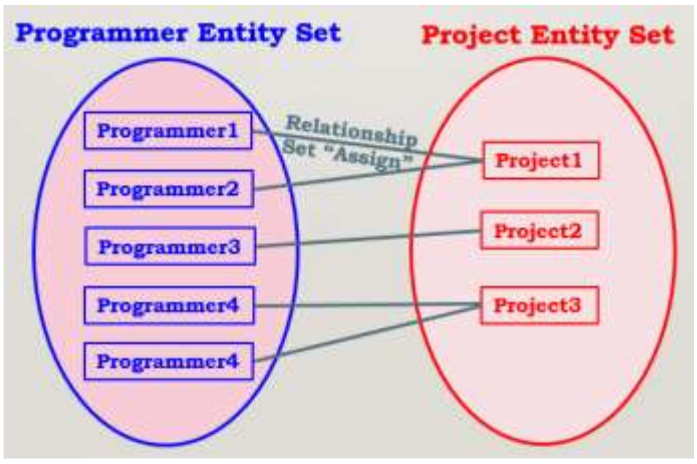
One-to-Many (1:M)

One-to-many. An entity in Programmer is associated with any number (zero or more) of entities in Project. An entity in Project, however, can be associated with at most one entity in Programmer.



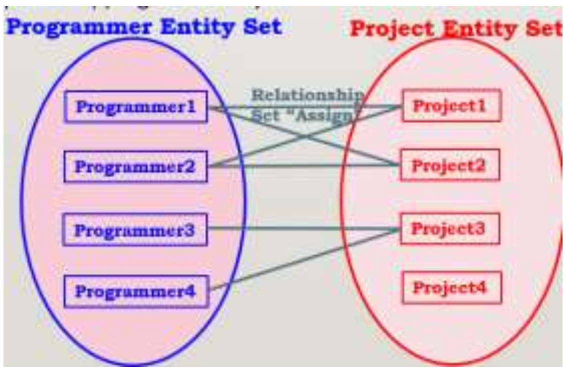
Many-to-One (M:1)

- ✓ An entity in Programmer is associated with at most one entity in Project. An entity in Project, however, can be associated with any number (zero or more) of entities in Programmer.



Many-to-Many (M:M)

- ✓ An entity in Programmer is associated with any number (zero or more) of entities in Project, and an entity in Project is associated with any number (zero or more) of entities in Programmer.
- ✓ The following figure depicts mapping cardinality M:M



Participation Constraints

Total Participation :

The participation of an entity set E in a relationship set R is said to be total if every entity in E participates in at least one relationship in R.

Partial Participation :

If only some entities in E participate in relationships in R, the participation of entity set E in relationship R is said to be partial.

Example :

- ✓ In Figure : A, the participation of Project Entity Set in the relationship set is total while the participation of A in the relationship set is partial.
- ✓ In Figure : B, the participation of both Programmer Entity Set and Project Entity Set in the relationship set are total.

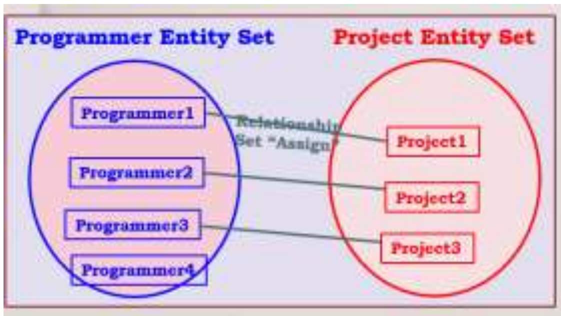


Figure : A

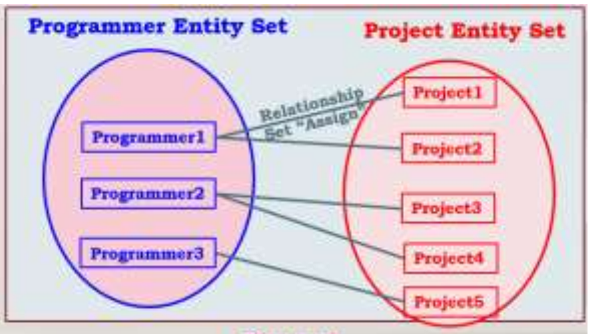


Figure : B

KEY CONSTRAINTS

A key is a set of attributes that can identify each tuple uniquely in the given relation.

Types Of Keys in DBMS

- Super key
- Candidate key
- Primary key
- Foreign key
- Unique key

Super Key

- A super key is a set of attributes that can identify each tuple uniquely in the given relation.
- A super key is not restricted to have any specific number of attributes.
- Thus, a super key may consist of any number of attributes.

Example-

Consider the following Student schema and identify the super key.

Student (roll , name , sex , age , address , class , section)

In the Given table,the following are the examples of super keys since each set can uniquely identify each student in the Student table-

- (roll , name , sex , age , address , class , section)
- (class , section , roll)
- (class , section , roll , sex)
- (name , address)

Candidate Key

- A minimal super key is called as a candidate key.
- A set of minimal attribute(s) that can identify each tuple uniquely in the given relation is called as a candidate key.

Example

Consider the following Student schema-

Student (roll , name , sex , age , address , class , section)

Candidate Keys in the above schema

- (class , section , roll)
- (name , address)

Points to be remembered for Candidate Key

- All the attributes in a candidate key are sufficient as well as necessary to identify each tuple uniquely.
- Removing any attribute from the candidate key fails in identifying each tuple uniquely.
- The value of candidate key must always be unique.
- The value of candidate key can never be NULL.
- It is possible to have multiple candidate keys in a relation.
- Those attributes which appears in some candidate key are called as prime attributes.

Primary Key

- A primary key is a candidate key that the database designer selects while designing the database.
- Candidate key that the database designer implements is called as a primary key.

Example

Consider the following Student schema and Identify the primary key.

Student (roll , name , sex , age , address , class , section)

Points to be remembered for Primary Key

- The value of primary key can never be NULL.
- The value of primary key must always be unique.
- The values of primary key can never be changed i.e. no updation is possible.
- The value of primary key must be assigned when inserting a record.
- A relation is allowed to have only one primary key.

Foreign Key

- An attribute 'X' is called as a foreign key to some other attribute 'Y' when its values are dependent on the values of attribute 'Y'.
- The attribute 'X' can assume only those values which are assumed by the attribute 'Y'.
- Here, the relation in which attribute 'Y' is present is called as the **referenced relation**.
- The relation in which attribute 'X' is present is called as the **referencing relation**.
- The attribute 'Y' might be present in the same table or in some other table.

Unique Key

Unique key is a key with the following properties-

- It is unique for all the records of the table.
- Once assigned, its value can not be changed i.e. it is non-updatable.
- It may have a NULL value.

Example

The best example of unique key is Adhaar Card Number.

- The Adhaar Card Number is unique for all the citizens (tuples) of India (table).
- If it gets lost and another duplicate copy is issued, then the duplicate copy always has the same number as before.

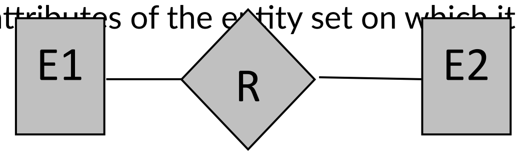
- Thus, it is non-updatable.
- Few citizens may not have got their Adhaar cards, so for them its value is NULL.

FOREIGN KEY CONSTRAINTS

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.
  - Example: If "Perryridge" is a branch name appearing in one of the tuples in the *account* relation, then there exists a tuple in the *branch* relation for branch "Perryridge".

Referential Integrity in the E-R Model

- Consider relationship set *R* between entity sets E_1 and E_2. The relational schema for *R* includes the primary keys K_1 of E_1 and K_2 of E_2.
  Then K_1 and K_2 form foreign keys on the relational schemas for E_1 and E_2 respectively.
- Weak entity sets are also a source of referential integrity constraints.
  - For the relation schema for a weak entity set must include the primary key attributes of the entity set on which it depends



Checking Referential Integrity on Database Modification

When a database is modified, ensuring referential integrity is essential to maintain the consistency of the data model. This involves checking that any changes made to the database do not violate the relationships established by foreign keys. Here's how referential integrity is checked for different types of database modifications:

1. Insert Operations

- Child Table: When a new record is inserted into a table that contains a foreign key, the database system checks that the foreign key value exists in the corresponding parent table. If the value does not exist in the parent table, the insert operation fails.

Parent Table: Inserting into a parent table does not usually affect referential integrity unless the table contains a self-referencing foreign key

Update Operations

- Child Table (Foreign Key Update): If a foreign key value in a child table is updated, the database system checks that the new value exists in the parent table. If it doesn't, the update operation is rejected.
- Parent Table (Primary Key Update): Updating a primary key value that is referenced by a foreign key in another table can lead to referential integrity issues. Most database systems either do not allow this operation or require that a cascade update rule be defined.

Delete Operations

- Child Table: Deleting a record from a child table generally does not affect referential integrity, as it does not affect the parent table.
- Parent Table: Deleting a record from a parent table that has corresponding foreign key references in a child table is more complex. The database system will check for dependent records and handle them according to the specified referential actions.
- CASCADE: Automatically deletes the dependent records in the child table.
- SET NULL: Sets the foreign key in the dependent records to NULL.
- SET DEFAULT: Sets the foreign key in the dependent records to its default value.
- RESTRICT or NO ACTION: Prevents the delete if there are dependent records.
- Custom Trigger: Custom logic can be defined to handle dependent records.

Truncate Operations

- Truncating a table quickly removes all records from that table. If the table is a parent table with foreign keys referenced in other tables, the truncate operation will usually be disallowed unless all referencing tables are also truncated simultaneously or the constraints are temporarily disabled.

Drop Operations

- Dropping a table (or a column that is used as a foreign key) will affect referential integrity. The database system will either:
- Prevent the drop operation if there are existing foreign key references.
- Drop the foreign key constraints if they are defined with the option to cascade on drop

Referential Integrity in SQL

- Primary and candidate keys and foreign keys can be specified as part of the SQL **create table** statement:
    - The **primary key** clause lists attributes that comprise the primary key.
    - The **unique key** clause lists attributes that comprise a candidate key.
    - The **foreign key** clause lists the attributes that comprise the foreign key and the name of the relation referenced by the foreign key.
- By default, a foreign key references the primary key attributes of the referenced table

foreign key (*account-number*) **references** *account*

Referential Integrity in SQL – Example

create table *customer*
(customer-name   char(20),
customer-street   char(30),
customer-city   char(30),
primary key (*customer-name))*

create table *branch*
(branch-name char(15),
branch-city char(30),
assets integer,
primary key *(branch-name))*

SPECIFYING FOREIGN KEY CONSTRAINTS

Purpose of Foreign Key Constraints

Maintaining Referential Integrity:

- Referential integrity ensures that relationships between tables remain consistent. With foreign key constraints, you can enforce rules that prevent orphaned rows or references to non-existent records.

Data Integrity:

- Foreign key constraints help maintain the integrity of your data by ensuring that only valid data can be inserted into the foreign key columns.

Automatic Cascading Actions:

- Foreign key constraints can define actions to take when referenced rows in the parent table are updated or deleted. These actions include cascading updates and deletes, restricting updates and deletes, or setting null values.

CASCADE:

- If a referenced row is updated or deleted, the corresponding rows in the child table are automatically updated or deleted as well.

SET NULL:

- If a referenced row is updated or deleted, the foreign key columns in the child table are set to NULL.

RESTRICT:

- Restricts updates or deletes on the parent table if there are corresponding rows in the child table.

NO ACTION:

- Similar to RESTRICT, it prevents updates or deletes on the parent table if there are corresponding rows in the child table.

CASCADE

The rows in the child table that are directly related to the referenced row are also immediately updated or removed when changes are made to the referenced row.

```
CREATE TABLE Orders (
  OrderID int PRIMARY KEY,
  ProductID int,
  Quantity int,
  FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
    ON DELETE CASCADE
);
```

When a ProductID is deleted from the Products table, all corresponding rows in the Orders table are deleted (ON DELETE CASCADE).

SET NULL

The child table's foreign key columns are set to NULL whenever a referenced record is modified or deleted.

```
CREATE TABLE Orders (
  OrderID int PRIMARY KEY,
  ProductID int,
  Quantity int,
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
        ON DELETE SET NULL
);
```

RESTRICT

Deletes or modifications to the parent table are not allowed if the child table has matching rows.

```
CREATE TABLE Orders (
    OrderID int PRIMARY KEY,
    ProductID int,
    Quantity int,
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
    ON UPDATE RESTRICT
);
```

If you try to update a ProductID in the Products table that is referenced by rows in the Orders table, the update will be restricted (ON UPDATE RESTRICT).

NO ACTION

Just like RESTRICT, it stops the parent table from being edited or deleted if the child table has matching rows.

```
CREATE TABLE Orders (
    OrderID int PRIMARY KEY,
    ProductID int,
    Quantity int,
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)
        ON DELETE NO ACTION
);
```

Enforcing Constraints Across Databases

Few database management systems provide mechanisms or extensions to enforce referential integrity across databases

Challenges:

Distributed Database Environment:

- Tables distributed across multiple databases.
- Data fragmentation for scalability and performance.

Referential Integrity:

- Maintaining relationships across databases.
- Ensuring data consistency and accuracy.

Solutions and Mechanisms

Database Management System Extensions:

- Some DBMSs offer extensions for cross-database referential integrity.
- These extensions vary based on the DBMS used.

Triggers and Procedures:

- Implement custom logic using triggers and stored procedures.
- Perform checks and enforce constraints manually.

Middleware Solutions:

- Integrate middleware for distributed transactions and constraint enforcement.
- Facilitates communication and coordination between databases.