Relation (table)
Definition: A relation, commonly known as a table in a database, is a collection of data organized in rows and columns. Each table represents a specific entity or concept in the database.
Structure:
- Columns (Attributes): Each column in a table represents a specific attribute of the entity, such as name, age, or ID. Columns have defined data types, such as integer, string, or date.
- Rows (Records or Tuples): Each row in a table represents a single record or instance of the entity.

Tuple (Row) : every *row* need to be unique, every cell don't need to unique
Definition: A tuple is a single entry in a table, representing a set of related data items. It corresponds to a row in the table.
Components:
- A tuple contains values for each attribute defined by the table's columns. For example, a tuple in a student table might include a student's ID, name, age, and GPA.
- Each value in the tuple must correspond to the data type specified for its column.

- **Schema**: A set of attributes specifying the structure/rules of a table.
  - Example: `Schema(sID: uint, Name: string, GPA: real)`

sID (uint)	Name (string)	GPA (real)
1	Harry	3.5
2	Hermione	4.0
3	Ron	4.0
4	Malfoy	3.9

Schema

Primary Key and Candidate Keys

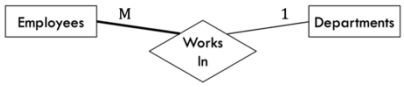
- **Primary Key (PK)**: Uniquely identifies each row in a table.
  - Example: `PRIMARY KEY(sID, cID)`
- **Candidate Keys (UQ)**: Additional unique constraints other than the primary key.
  - Example: `UNIQUE(sID, Grade)`

Foreign key: attribute in one table that uniquely identifies a row in another table, acting like a "pointer"from a child table back to parent table

Handling Foreign Key Constraints
1. Delete Corresponding Records: If the referencing record has no meaning on its own, delete it.
2. Nullify Foreign Key: Keep the data but "unlink" it by setting the foreign key to NULL.
3. Disallow Changes: Prevent changes to the referenced table if they would violate referential integrity, requiring some action to be taken first.

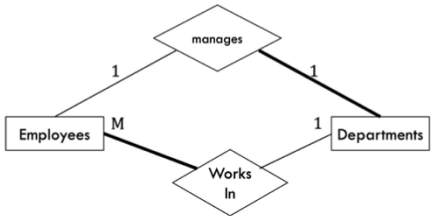
Participation Constraints

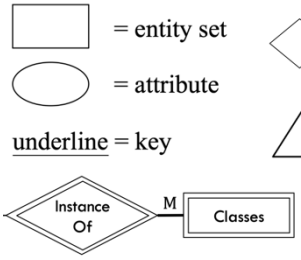
- **Definition**: Constraints indicating whether all entities in an entity set must participate in a relationship.
- **Notation**: Bold lines or double lines indicate mandatory participation.



Example: "An employee must work in one department, but a department does not necessarily have any employees."

•A department can have multiple employees
•An employee works for exactly one department
•A department has exactly one manager
•An employee can manage up to one department





Weak Entity:
- Double rectangle for the entity set
- Double diamond for the supporting relationship

SQL Table Creation Syntax

```
CREATE TABLE <name> (
  <column1Name> <type> <properties>,
  <column2Name> <type> <properties>,
  ...
  <table properties>
);
```

MySQL Data Types

- **Numeric**: `int`, `tinyint`, `smallint`, `mediumint`, `bigint` `unsigned`, `float`, `double`, `decimal`.
- **Dates**: `date`, `datetime`, `timestamp`, `time`, `year`.
- **Strings**: `char(N)`, `varchar(N)`, `blob`, `enum`.

Column Properties

- **NOT NULL**: Ensures a column cannot have NULL values.
- **DEFAULT**: Provides a default value for a column.
- **AUTO_INCREMENT**: Automatically increments the value of a column.
- **PRIMARY KEY**: Uniquely identifies each row.
- **UNIQUE**: Ensures all values in a column are unique.

Example Table Creation:

```
CREATE TABLE Titles (
  ISBN char(14) NOT NULL,
  Title varchar(255) NOT NULL,
  Author varchar(255) NOT NULL,
  PRIMARY KEY (ISBN)
);
```

```
CREATE TABLE AsgCats (
  classID INT,
  Name VARCHAR(50),
  acID INT AUTO_INCREMENT,
  PRIMARY KEY (classID, acID),
  FOREIGN KEY (classID) REFERENCES Classes(classID)
);
```

```
CREATE TABLE Classes (
  corID INT,
  Semester VARCHAR(10),
  classID INT AUTO_INCREMENT,
  PRIMARY KEY (corID, Semester, classID),
  UNIQUE (Semester, corID)
);
```

DML (Data Manipulation Language): Managing data within tables.

- **SELECT**: Querying data.

```
SELECT column_list FROM table_name WHERE condition;
```

- **INSERT**: Adding data.

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

- **DELETE**: Removing data.

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

- **UPDATE**: Modifying data.

```
UPDATE table_name SET column1 = value1 WHERE condition;
```

SQL Joins

- **INNER JOIN**: Combines rows from two or more tables based on a related column.

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

- **NATURAL JOIN**: Automatically joins tables on columns with the same name.

```
SELECT * FROM table1 NATURAL JOIN table2;
```

- **LEFT JOIN, RIGHT JOIN**: Include all rows from one table and matched rows from the other.

```
SELECT * FROM table1 LEFT JOIN table2 ON table1.column = table2.column;
SELECT * FROM table1 RIGHT JOIN table2 ON table1.column = table2.column;
```

Basic Operators in Relational Algebra

- **Projection (π)**: Extracts certain columns from a relation.
  - **Syntax**: πcolumn_list
  - **Example**: πTitle, Author
- **Selection (σ)**: Filters rows based on a condition.
  - $\sigma_{condition}(relation)$
  - Example:
  - $\sigma_{CardNum > 3}(Patrons)$
  - $\pi_{Phone}(\sigma_{CardNum > 3}(Patrons \times Phones))$
- **Cross Product (×)**: Combines two relations to produce all possible pairs of rows.
  - **Syntax**: relation1 × relation2
  - **Example**: R1 × R2
- **Set Difference (–)**: Returns rows that are in one relation but not in another.
  - **Syntax**: relation1 – relation2
  - **Example**: R1 – R2

R1	
Col1	
a	
b	
x	

R2	
Col1	
x	
y	
z	

R1—R2	
Col1	
a	
b	

R2—R1	
Col1	
y	
z	

- **Set Union (∪)**: Combines two relations to include all rows from both.
- **Syntax**: relation1 ∪ relation2
- **Example**: R1 ∪ R2

float	int
...	...

∪

float	int
...	...

✓

∪

float
...

×

1. **Basic Operators**:
   - **Projection (π)**: Selects certain columns from a relation.
   - **Selection (σ)**: Filters rows based on a condition.
   - **Cross Product (×)**: Combines two relations to produce all possible pairs of rows.
   - **Set Difference (–)**: Returns rows that are in one relation but not in another.
   - **Set Union (∪)**: Combines two relations to include all rows from both.
2. **Division Operator**:
   - **Purpose**: Used to find tuples in one relation that are associated with **all tuples** in another relation.
   - **Example**: To find all cars that use both parts p2 and p4.

where `A` is the relation of cars and parts, and `B` is the relation of the parts of interest.

A	
cNo	pNo
c1	p1
c1	p2
c1	p3
c1	p4
c2	p1
c2	p2
c3	p2
c4	p2
c4	p4

B
pNo
p2
p4

C
pNo
p2

D
pNo
p1
p2
p4

A / B
cNo
c1
c4

A / C
cNo
c1
c2
c3
c4

A / D
cNo
c1

3. **Steps for Division**:
   a. Project out the relevant attributes from `A`.
   b. Compute the cross product of the projected relation with `B`.
   c. Subtract `A` from the cross product to remove spurious tuples.
   d. Project out the attributes to get the final result.

4. **Joins**:
   - **Natural Join**: Combines two relations based on common attribute values.

```
R1 ⋈ R2
```

   - **Theta Join**: Combines two relations based on a specified condition.

```
R1 ⋈ condition R2
```

5. **Renaming**:
   - **Purpose**: Useful for disambiguating attribute names and making expressions more readable.

```
ρ(newname, expression)
```

$\rho(\text{HerbertSerials}, \pi_{\text{Serial}}(\sigma_{\text{Author=Herbert}}(\text{Titles} \rhd\lhd \text{Inventory})))$

$\pi_{\text{CardNum}}(\text{HerbertSerials} \rhd\lhd \text{CheckedOut}))$

- **Projection Example**:

```
SELECT Author FROM Titles;
```

  - Relational Algebra: πAuthor

- **Selection Example**:

```
SELECT * FROM Patrons WHERE CardNum > 3;
```

  - Relational Algebra: σCardNum > 3

- **Cross Product Example**:
  - Combines two relations R1 and R2.
  - Result: Each row in R1 is paired with every row in R2.
- **Set Union Example**:
  - Combines all rows from two relations R1 and R2, removing duplicates.
- **Set Difference Example**:
  - Returns rows present in R1 but not in R2.

Advanced Concepts

- **Set Intersection (∩)**: Can be formulated using basic operators.
  - Example: R1 ∩ R2 = R1 – (R1 – R2)

Translating Relational Algebra to SQL

1. **Basic Translations**:
   - **Projection**:

```
SELECT col1, col2 FROM table;
```

   - **Selection**:

```
SELECT * FROM table WHERE condition;
```

2. **Joins**:
   - **Natural Join**:

```
SELECT * FROM R1 NATURAL JOIN R2;
```

   - **Theta Join**:

```
SELECT * FROM R1 JOIN R2 ON condition;
```

3. **Set Operations**:
   - **Union**:

```
SELECT * FROM R1 UNION SELECT * FROM R2;
```

   - **Difference**:

```
SELECT * FROM R1 EXCEPT SELECT * FROM R2;
```

Exercises

Students

sID	Name	DOB
1	Hermione	1980
2	Harry	1979
3	Ron	1980
4	Malfoy	1982

Enrolled

sID	cID	Grd
1	3500	A
1	3810	A-
1	5530	A
2	3810	A
2	5530	B

Courses

cID	Name
3500	SW Practice
3810	Architecture
5530	Databases

- **Operators**:
  - `x IN A`: True if x is in A.
  - `EXISTS A`: True if A is not empty.
  - `x OP y ANY A`: True if there exists a y in A such that x op y is True.
  - `x OP y ALL A`: True if for ALL y in A, x op y is True.

1. **Names of Students who earned an A or B**

$$Result \leftarrow \pi_{Name}((\pi_{sID}(\sigma_{Grd='A' \vee Grd='B'}(Enrolled))) \rhd\lhd_{sID=Students.sID} Students)$$

2. **Names of Students who earned an A *and* B**

π_{Name}

$$((\pi_{sID}(\sigma_{Grd='A'}(Enrolled)) \cap \pi_{sID}(\sigma_{Grd='B'}(Enrolled))) \rhd\lhd_{sID=Students.sID} Students)$$

1. **Names of Courses with a student born in 1979 or a student born in 1982:**

```
SELECT DISTINCT c.Name
FROM Courses c
JOIN Enrolled e ON c.cID = e.cID
JOIN Students s ON e.sID = s.sID
WHERE s.DOB = 1979 OR s.DOB = 1982;
```

2. **Names of Students taking all Courses:**

```
SELECT s.Name
FROM Students s
WHERE NOT EXISTS (
  SELECT c.cID
  FROM Courses c
  WHERE NOT EXISTS (
    SELECT e.sID
    FROM Enrolled e
    WHERE e.cID = c.cID AND e.sID = s.sID
  )
);
```

- **Common Functions**:
  - `COUNT()`
  - `MAX()`
  - `MIN()`
  - `SUM()`
  - `AVG()`

Translating Relational Algebra (RA) to SQL

1. **Basic Translations from RA to SQL**:
   - **Projection (π) to SELECT**:
     - RA: πcol1,col2(relation)
     - SQL: `SELECT col1, col2 FROM relation;`
     - RA (no projection): `SELECT * FROM relation;`
   - **Selection (σ) to WHERE**:
     - RA: σcondition(relation)

       σcondition(relation)
     - SQL: `SELECT * FROM relation WHERE condition;`

- Renaming a table:

```
SELECT p.CardNum
FROM Patrons p
WHERE p.Name = 'Joe';
```

- Renaming a column:

```
SQL
SELECT p.CardNum AS CN
FROM Patrons p
WHERE p.Name = 'Joe';
```

5. **Set Operations**:
   - **Union (∪)**:
     - RA: R1∪R2

       R1∪R2
     - SQL: `SELECT * FROM R1 UNION SELECT * FROM R2;`
     - Example:

```
SELECT Addr FROM CorporateLocs
UNION
SELECT Addr FROM RetailLocs;
```

- **Union All**: Includes duplicates.
  - SQL: `SELECT * FROM R1 UNION ALL SELECT * FROM R2;`
- **Intersection (∩)**:
  - RA: R1∩R2
  - SQL: `SELECT * FROM R1 INTERSECT SELECT * FROM R2;`
  - MySQL does not support `INTERSECT`, so it can be formulated using `NATURAL JOIN` or `IN`:

```
sqlCopy code
SELECT * FROM
(SELECT Addr FROM CorporateLocs) AS corp
NATURAL JOIN
(SELECT Addr FROM RetailLocs) AS retail;
```

- **Set Difference (−)**:
  - RA: R1−R2
  - SQL: `SELECT * FROM R1 WHERE column NOT IN (SELECT column FROM R2);`
  - Example:

```
SELECT Addr FROM CorporateLocs
WHERE Addr NOT IN (SELECT Addr FROM RetailLocs);
```

2. **Finding All Patrons Who Have Checked Out Both 'The Lorax' and 'Harry Potter':**

```
SELECT Name
FROM (SELECT CardNum
      FROM CheckedOut
      NATURAL JOIN Inventory
      NATURAL JOIN Titles
      WHERE Titles.Title = 'The Lorax') AS lorax
NATURAL JOIN
     (SELECT CardNum
      FROM CheckedOut
      NATURAL JOIN Inventory
      NATURAL JOIN Titles
      WHERE Titles.Title = 'Harry Potter') AS hp
NATURAL JOIN Patrons;
```

Outer Join

- **Types**:
  - **LEFT JOIN**: Includes all rows from the left table and
  - matched rows from the right table. Unmatched rows
  - from the right table will contain NULL.
  - **RIGHT JOIN**: Includes all rows from the right table and
  - matched rows from the left table. Unmatched rows from
  - the left table will contain NULL.

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