

Module 06

Partha Pratim
Das

Week Recap

Objectives &
OutlineExample of a
Relation

Attributes

Schema and
Instance

Keys

Relational Query
Languages

Module Summary

- Consider

Students = Roll#, First Name, Last Name, DoB, Passport#, Aadhaar#, Department
relation

- The set of allowed values for each attribute is called the **domain** of the attribute
 - **Roll #:** Alphanumeric string
 - **First Name, Last Name:** Alpha String
 - **DoB:** Date
 - **Passport #:** String (Letter followed by 7 digits) – nullable (optional)
 - **Aadhaar #:** 12-digit number
 - **Department:** Alpha String
- Attribute values are (normally) required to be **atomic**; that is, indivisible
- The special value **null** is a member of every domain. Indicates that the value is *unknown*
- The null value may cause complications in the definition of many operations

Attribute Types

Module 06

Partha Pratim
Das

Week Recap

Objectives &
OutlineExample of a
Relation

Attributes

Schema and
Instance

Keys

Relational Query
Languages

Module Summary

- For
 $Students = Roll\#, First\ Name, Last\ Name, DoB, Passport\#, Aadhaar\#, Department$
- And **domain** of the attributes as:

- **Roll #:** Alphanumeric string
- **First Name, Last Name:** Alpha String
- **DoB:** Date
- **Passport #:** String (Letter followed by 7 digits) – nullable (optional)
- **Aadhaar #:** 12-digit number
- **Department:** Alpha String

Roll #	First Name	Last Name	DoB	Passport #	Aadhaar #	Department
15CS10026	Lalit	Dubey	27-Mar-1997	L4032464	1728-6174-9239	Computer
16EE30029	Jatin	Chopra	17-Nov-1996	null	3917-1836-3816	Electrical

Relation Schema and Instance

Module 06

Partha Pratim
Das

Week Recap

Objectives &
OutlineExample of a
Relation

Attributes

Schema and
Instance

Keys

Relational Query
Languages

Module Summary

- A_1, A_2, \dots, A_n are *attributes*
- $R = (A_1, A_2, \dots, A_n)$ is a *relation schema*
Example: *instructor* = (*ID*, *name*, *dept_name*, *salary*)
- Formally, given sets D_1, D_2, \dots, D_n a **relation** r is a subset of

$$r \subseteq D_1 \times D_2 \times \dots \times D_n$$

$$A_1 \Rightarrow D_1$$

Thus, a relation is a set of n -tuples (a_1, a_2, \dots, a_n) where each $a_i \in D_i$

- The current values (**relation instance**) of a relation are specified by a table
- An element t of r is a tuple, represented by a row in a table
- Example:

instructor \equiv (*String*(5) \times *String* \times *String* \times *Number* +), where $ID \in$ *String*(5), $name \in$ *String*, $dept_name \in$ *String*, and $salary \in$ *Number* +



Relations are Unordered with Unique Tuples

Module 06

Partha Pratim
Das

Week Recap

Objectives &
OutlineExample of a
Relation

Attributes

Schema and
Instance

Keys

Relational Query
Languages

Module Summary

- **Order of tuples / rows is irrelevant** (tuples may be stored in an arbitrary order)
- **No two tuples / rows may be identical**
- Example: *instructor* relation with unordered tuples

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Module 06

Partha Pratim
Das

Week Recap

Objectives &
OutlineExample of a
Relation

Attributes

Schema and
Instance

Keys

Relational Query
Languages

Module Summary

- Let $K \subseteq R$, where R is the set of attributes in the relation
- K is a **superkey** of R if values for K are sufficient to identify a unique tuple of each possible relation $r(R)$ name is not a superkey in itself, but it is when in conjunction with ID
 - Example: $\{ID\}$ and $\{ID, name\}$ are both superkeys of *instructor*
- Superkey K is a **candidate key** if K is minimal candidate keys not chosen as primary key are called secondary/alternate key
 - Example: $\{ID\}$ is a candidate key for *instructor*
- One of the candidate keys is selected to be the **primary key**
 - Which one?
- A **surrogate key** (or synthetic key) in a database is a unique identifier for either an *entity* in the modeled world or an *object* in the database
 - The surrogate key is *not* derived from application data, unlike a *natural* (or *business*) key which is derived from application data



Module 06

Partha Pratim
Das

Week Recap

Objectives &
OutlineExample of a
Relation

Attributes

Schema and
Instance

Keys

Relational Query
Languages

Module Summary

- *Students = Roll#, First Name, Last Name, DoB, Passport#, Aadhaar#, Department*
- **Super Key:** Roll #, {Roll #, DoB}
- **Candidate Keys:** Roll #, {First Name, Last Name}, Aadhaar#
 - **Passport # cannot be a key. Why?**
 - **Null values are allowed for Passport #** (a student may not have a passport) unique key allow null values whereas primary key does not
- **Primary Key:** Roll #
 - Can Aadhaar# be a key?
 - **It may suffice for unique identification. But Roll# may have additional useful information.** For example: 14CS92P01
 - ▷ Read 14CS92P01 as 14-CS-92-P-01
 - ▷ 14: Admission in 2014
 - ▷ CS: Department = CS
 - ▷ 92: Category of Student
 - ▷ P: Type of admission: *Project*
 - ▷ 01: Serial Number

Keys

Module 06

Partha Pratim
Das

Week Recap

Objectives &
OutlineExample of a
Relation

Attributes

Schema and
Instance

Keys

Relational Query
Languages

Module Summary

- **Secondary / Alternate Key:** {First Name, Last Name}, Aadhaar #
- **Simple Key:** Consists of a *single attribute*
- **Composite Key:** {First Name, Last Name}
 - Consists of more than one attribute to uniquely identify an entity occurrence
 - One or more of the attributes, which make up the key, are **not** simple keys in their own right

<u>Roll #</u>	<u>First Name</u>	<u>Last Name</u>	<u>DoB</u>	<u>Passport #</u>	<u>Aadhaar #</u>	<u>Department</u>
15CS10026	Lalit	Dubey	27-Mar-1997	L4032464	1728-6174-9239	Computer
16EE30029	Jatin	Chopra	17-Nov-1996	null	3917-1836-3816	Electrical
15EC10016	Smriti	Mongra	23-Dec-1996	G5432849	2045-9271-0914	Electronics
16CE10038	Dipti	Dutta	02-Feb-1997	null	5719-1948-2918	Civil
15CS30021	Ramdin	Minz	10-Jan-1997	X8811623	4928-4927-5924	Computer



Keys

Module 06

Partha Pratim
Das

Week Recap

Objectives &
OutlineExample of a
Relation

Attributes

Schema and
Instance

Keys

Relational Query
Languages

Module Summary

- **Foreign key constraint:** Value in one relation must appear in another attribute is a key in a different table
 - Referencing relation
 - ▷ Enrolment: Foreign Keys – Roll #, Course #
 - Referenced relation
 - ▷ Students, Courses
- A compound key consists of *more than one attribute* to uniquely identify an entity occurrence
 - Each attribute, which makes up the key, is a simple key in its own right
 - {Roll #, Course #}

A composite key is made up of elements that may or may not be foreign keys.
A compound key is a key whereby any part of the key is a foreign key.

Students						
Roll #	First Name	Last Name	DoB	Passport #	Address #	Department

Courses				
Course #	Course Name	Credits	L-T-P	Department

Enrolment		
Roll #	Course #	Instructor ID

Module 06

Partha Pratim
Das

Week Recap

Objectives &
Outline

Example of a
Relation

Attributes

Schema and
Instance

Keys

Relational Query
Languages

Module Summary

Procedural viz-a-viz Non-procedural or Declarative Paradigms

- **Procedural programming** requires that the programmer tell the computer what to do
 - That is, *how* to get the output for the range of required inputs
 - The programmer must know an appropriate **algorithm**
- **Declarative programming** requires a more descriptive style
 - The programmer must know *what* relationships hold between various entities



Relational Query Languages (Cont..)

Module 06

Partha Pratim
Das

Week Recap

Objectives &
Outline

Example of a
Relation

Attributes

Schema and
Instance

Keys

Relational Query
Languages

Module Summary

Procedural vs. Non-procedural or Declarative Paradigms

- **Example: Square root of n**

- Procedural

- a) Guess x_0 (close to root of n)
 - b) $i \leftarrow 0$
 - c) $x_{i+1} \leftarrow (x_i + n/x_i)/2$
 - d) Repeat Step 2 if $|x_{i+1} - x_i| > delta$

- Declarative

- ▷ Root of n is m such that $m^2 = n$

Module 06

Partha Pratim
Das

Week Recap

Objectives &
Outline

Example of a
Relation

Attributes

Schema and
Instance

Keys

Relational Query
Languages

Module Summary

- “Pure” languages:
 - Relational algebra
 - Tuple relational calculus
 - Domain relational calculus
- The above 3 pure languages are equivalent in computing power
- We will concentrate on relational algebra
 - Not Turing-machine equivalent
 - ▷ Not all algorithms can be expressed in RA
 - Consists of 6 basic operations

Basic Properties of Relations

Module 07

Partha Pratim
DasObjectives &
OutlineRelational
OperatorsAggregation
Operators

Module Summary

- A relation is set. Hence,
- Ordering of rows / tuples is inconsequential

A	B
a1	b1
a1	b2
a2	b1
a2	b2

A	B
a1	b1
a2	b1
a2	b2
a1	b2

is same as:

- All rows / tuples must be distinct

A	B
a1	b1
a1	b2
a1	b2
a1	b1

is not valid

A	B
a1	b1
a1	b2

is

Select Operation – selection of rows (tuples)

Module 07

Partha Pratim
Das

Objectives &
Outline

Relational
Operators

Aggregation
Operators

Module Summary

if they satisfy a particular condition

- Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

and v is or

- $\sigma_{A=B \wedge D>5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

Project Operation – selection of columns (Attributes)

Module 07

Partha Pratim
DasObjectives &
OutlineRelational
OperatorsAggregation
Operators

Module Summary

- Relation r

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

- $\pi_{A,C}(r)$

$$\begin{array}{c} \begin{array}{c} A \quad C \\ \hline \alpha & 1 \\ \alpha & 1 \\ \beta & 1 \\ \beta & 2 \end{array} & = & \begin{array}{c} A \quad C \\ \hline \alpha & 1 \\ \beta & 1 \\ \beta & 2 \end{array} \end{array}$$

Union of two relations

Module 07

Partha Pratim
DasObjectives &
OutlineRelational
OperatorsAggregation
Operators

Module Summary

- Relation r, s

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cup s$

A	B
α	1
α	2
β	1
β	3

Set difference of two relations

Module 07

Partha Pratim
DasObjectives &
OutlineRelational
OperatorsAggregation
Operators

Module Summary

- Relation r, s

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r - s$

A	B
α	1
β	1

Set intersection of two relations

Module 07

Partha Pratim
DasObjectives &
OutlineRelational
OperatorsAggregation
Operators

Module Summary

- Relation r, s

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cap s$

A	B
α	2

Note: $r \cap s = r - (r - s)$

Joining two relations – Cartesian-product

Module 07

 Partha Pratim
 Das

 Objectives &
 Outline

 Relational
 Operators

 Aggregation
 Operators

Module Summary

- Relation r, s

A	B	C	D	E
α	1			
β	2			
		r		

C	D	E	
α	10	a	
β	10	a	
β	20	b	
γ	10	b	
		s	

$$2^*4=8$$

- $r \times s$

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

you can get the orginal relations by projecting on r and s

$$\begin{aligned} r &= \pi_{r}(r \times s) \\ s &= \pi_{s}(r \times s) \end{aligned}$$

Cartesian-product – naming issue

Module 07

Partha Pratim
Das

Objectives &
Outline

Relational
Operators

Aggregation
Operators

Module Summary

- Relation r, s

A	B			
α	1	α	10	a
β	2	β	10	a
		β	20	b
		γ	10	b

r

B	D	E		
α	10	a	α	10
β	10	a	β	10
β	20	b	β	20
γ	10	b	γ	10

s

- $r \times s$

A	$r.B$	$s.B$	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b



Renaming a Table

Module 07

Partha Pratim
Das

Objectives &
Outline

Relational
Operators

Aggregation
Operators

Module Summary

- Allows us to refer to a relation, (say E) by more than one name.

$$\rho_X(E)$$

returns the expression E under the name X

- Relations r

A	B
α	1
β	2
r	

- $r \times \rho_s(r)$
self cross product

$r.A$	$r.B$	$s.A$	$s.B$
α	1	α	1
α	1	β	2
β	2	α	1
β	2	β	2



Joining two relations – Natural Join

Module 07

Partha Pratim
Das

Objectives &
Outline

Relational
Operators

Aggregation
Operators

Module Summary

- Let r and s be relations on schemas R and S respectively. Then, the “natural join” of relations R and S is a relation on schema $R \cup S$ obtained as follows:
 - Consider each pair of tuples t_r from r and t_s from s .
 - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple t to the result, where
 - ▷ t has the same value as t_r on r
 - ▷ t has the same value as t_s on s



Natural Join Example

- Relations r, s :

$$R = \begin{array}{|c|c|c|c|} \hline A & B & C & D \\ \hline \alpha & 1 & \alpha & a \\ \beta & 2 & \gamma & a \\ \gamma & 4 & \beta & b \\ \alpha & 1 & \gamma & a \\ \delta & 2 & \beta & b \\ \hline \end{array} \quad r$$

$$S = \begin{array}{|c|c|c|} \hline B & D & E \\ \hline 1 & a & \alpha \\ 3 & a & \beta \\ 1 & a & \gamma \\ 2 & b & \delta \\ 3 & b & \varepsilon \\ \hline \end{array} \quad s$$

$$RUS = (A, B, C, D, E)$$

$$RNS = (B, D)$$

- Natural Join

now $s.B, r.B, s.D, r.D$ is not required

$\circ r \bowtie s$

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

$$\pi_{A,r.B,C,r.D,E}(\sigma_{r.B=s.B \wedge r.D=s.D}(r \times s))$$



Aggregate Operators

Module 07

Partha Pratim
Das

Objectives &
Outline

Relational
Operators

Aggregation
Operators

Module Summary

- Can we compute:

- SUM
- AVG
- MAX
- MIN

Notes about Relational Languages

Module 07

Partha Pratim
Das

Objectives &
Outline

Relational
Operators

Aggregation
Operators

Module Summary

- Each query input is a table (or set of tables)
- Each query output is a table
- All data in the output table appears in one of the input tables
- Relational Algebra is not Turing complete

Summary of Relational Algebra Operators

Module 07

 Partha Pratim
 Das

 Objectives &
 Outline

 Relational
 Operators

 Aggregation
 Operators

Module Summary

Symbol (Name)	Example of Use
σ (Selection)	$\sigma \text{ salary} \geq 85000 \text{ (instructor)}$ Return rows of the input relation that satisfy the predicate.
Π (Projection)	$\Pi ID, \text{salary} \text{ (instructor)}$ Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
\times (Cartesian Product)	$\text{instructor} \times \text{department}$ Output all possible combinations of rows in <i>instructor</i> and <i>department</i> .
\cup (Union)	$\Pi name \text{ (instructor)} \cup \Pi name \text{ (student)}$ Output the union of tuples from the two input relations.
- (Set Difference)	$\Pi name \text{ (instructor)} - \Pi name \text{ (student)}$ Output the set difference of tuples from the two input relations.
\bowtie (Natural Join)	$\text{instructor} \bowtie \text{department}$ Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.

Data Definition Language (DDL)

Module 08

Partha Pratim
Das

Objectives &
Outline

Outline

History of SQL

Data Definition
Language (DDL)

Create Table

Integrity Constraints

Update Table

Data
Manipulation
Language (DML):
Query Structure

Select Clause

Where Clause

From Clause

Module Summary

The **SQL data-definition language (DDL)** allows the specification of information about relations, including:

- The *Schema* for each *Relation*
- The *Domain* of values associated with each *Attribute*
- *Integrity Constraints*
- And, as we will see later, also other information such as
 - The set of *Indices* to be maintained for each relations
 - *Security and Authorization* information for each relation
 - The *Physical Storage Structure* of each relation on disk

Domain Types in SQL

Module 08

Partha Pratim
DasObjectives &
Outline

Outline

History of SQL

Data Definition
Language (DDL)

Create Table

Integrity Constraints

Update Table

Data
Manipulation
Language (DML):
Query Structure

Select Clause

Where Clause

From Clause

Module Summary

- **char(*n*)**. Fixed length character string, with user-specified length *n*
- **varchar(*n*)**. Variable length character strings, with user-specified maximum length *n*
- **int**. Integer (a finite subset of the integers that is machine-dependent)
- **smallint(*n*)**. Small integer (a machine-dependent subset of the integer domain type)
- **numeric(*p, d*)**. Fixed point number, with user-specified precision of *p* digits, with *d* digits to the right of decimal point. (ex., **numeric(3, 1)**, allows 44.5 to be stored exactly, but not 444.5 or 0.32)
- **real, double precision**. Floating point and double-precision floating point numbers, with machine-dependent precision
- **float(*n*)**. Floating point number, with user-specified precision of at least *n* digits
- More are covered in Chapter 4

There is a small difference between **NUMERIC(*p, s*)** and **DECIMAL(*p, s*)** SQL numeric data type. **NUMERIC** determines the exact precision and scale. **DECIMAL** specifies only the exact scale; the precision is equal or greater than what is specified by the coder. **DECIMAL** columns can have a larger-than-specified precision if this is more convenient or efficient for the database system.

Create Table Construct

Module 08

Partha Pratim
Das

Objectives &
Outline

Outline

History of SQL

Data Definition
Language (DDL)

Create Table

Integrity Constraints

Update Table

Data
Manipulation
Language (DML):
Query Structure

Select Clause

Where Clause

From Clause

Module Summary

- An **SQL relation is defined using the `create table` command:**

`create table r (A1D1, A2D2, ..., AnDn),` A-name & D-type of attribute
`(integrity-constraint1),`
`...`
`(integrity-constraintk));`

- r is the name of the relation
- each A_i is an attribute name in the schema of relation r
- D_i is the data type of values in the domain of attribute A_i

Create Table Construct (2)

Module 08

 Partha Pratim
 Das

 Objectives &
 Outline

Outline

History of SQL

 Data Definition
 Language (DDL)
 Create Table

 Integrity Constraints
 Update Table

 Data
 Manipulation
 Language (DML):
 Query Structure

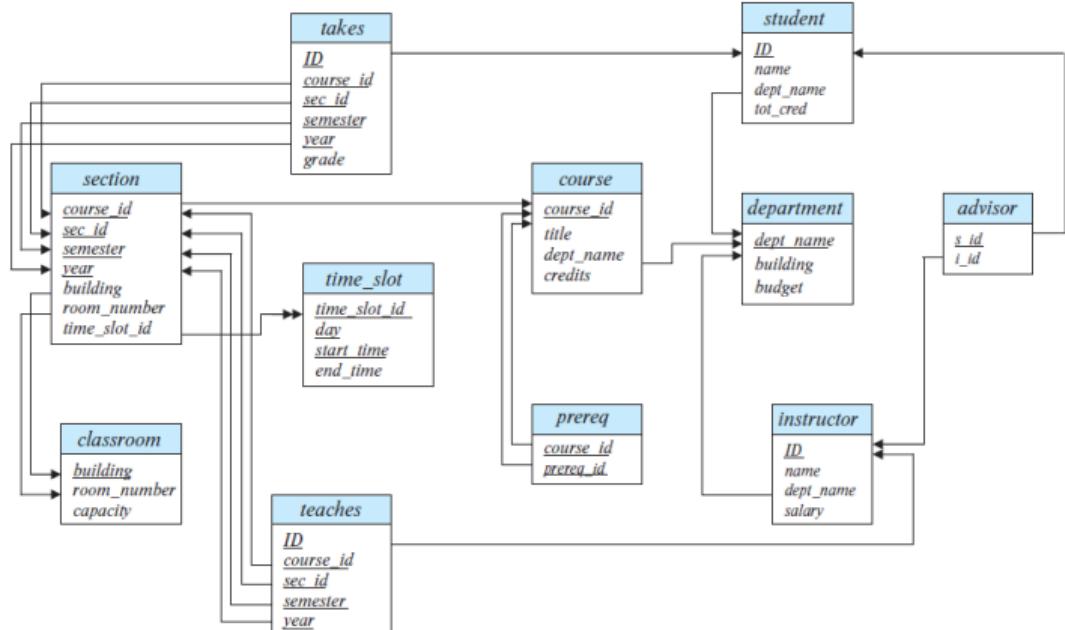
Select Clause

Where Clause

From Clause

Module Summary

```
create table instructor (
  ID char(5),
  name varchar(20)
  dept_name varchar(20)
  salary numeric(8,2));
```



Create Table Construct (3): Integrity Constraints

Module 08

Partha Pratim
DasObjectives &
Outline

Outline

History of SQL

Data Definition
Language (DDL)

Create Table

Integrity Constraints

Update Table

Data
Manipulation
Language (DML):
Query Structure

Select Clause

Where Clause

From Clause

Module Summary

- **not null**
- **primary key (A_1, \dots, A_n)**
- **foreign key (A_m, \dots, A_n) references r**

```
create table instructor (
    ID char(5),
    name varchar(20)
    dept_name varchar(20)
    salary numeric(8, 2));
```

```
create table instructor (
    ID char(5),
    name varchar(20) not null,
    dept_name varchar(20),
    salary numeric(8, 2),
    primary key (ID),
    foreign key (dept_name) references department);
```

✓ primary key declaration on an attribute automatically ensures **not null**



Update Tables

Module 08

Partha Pratim
Das

Objectives &
Outline

Outline

History of SQL

Data Definition
Language (DDL)

Create Table

Integrity Constraints

Update Table

Data
Manipulation
Language (DML):
Query Structure

Select Clause

Where Clause

From Clause

Module Summary

- **Insert (DML command)**

- `insert into instructor values ('10211', 'Smith', 'Biology', 66000);`

- **Delete (DML command)**

- Remove all tuples from the *student* relation

- `delete from student`

- **Drop Table (DDL command)**

- `drop table r`

- **Alter (DDL command)**

- `alter table r add A D`

- ▷ Where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A*
 - ▷ All existing tuples in the relation are assigned *null* as the value for the new attribute

- `alter table r drop A`

- ▷ Where *A* is the name of an attribute of relation *r*
 - ▷ Dropping of attributes not supported by many databases

Basic Query Structure

Module 08

Partha Pratim
Das

Objectives &
Outline

Outline

History of SQL

Data Definition
Language (DDL)

Create Table

Integrity Constraints

Update Table

Data
Manipulation
Language (DML):
Query Structure

Select Clause

Where Clause

From Clause

Module Summary

- A typical SQL query has the form:

select $A_1, A_2, \dots, A_n,$
from r_1, r_2, \dots, r_m
where P

- A_i represents an attribute from r_i 's
 - r_i represents a relation
 - P is a predicate
- The result of an SQL query is a relation

Select Clause

Module 08

Partha Pratim
Das

Objectives &
Outline

Outline

History of SQL

Data Definition
Language (DDL)

Create Table

Integrity Constraints
Update Table

Data
Manipulation
Language (DML):
Query Structure

Select Clause

Where Clause

From Clause

Module Summary

- The **select** clause lists the attributes desired in the result of a query
 - Corresponds to the **projection operation** of the relational algebra

- Example: find the names of all instructors:

```
select name,  
      from instructor
```

- NOTE: **SQL names are case insensitive** (that is, you may use upper-case or lower-case letters)
 - $Name \equiv NAME \equiv name$
 - Some people use upper case wherever we use bold font



Select Clause (2)

Module 08

Partha Pratim
Das

Objectives &
Outline

Outline

History of SQL

Data Definition
Language (DDL)

Create Table

Integrity Constraints

Update Table

Data
Manipulation
Language (DML):
Query Structure

Select Clause

Where Clause

From Clause

Module Summary

✓ **SQL allows duplicates in relations as well as in query results!!!**

- To force the elimination of duplicates, insert the keyword **distinct** after select
- Find the department names of all instructors, and remove duplicates

```
select distinct dept_name  
from instructor
```

- The keyword **all** specifies that **duplicates should not be removed**

```
select all dept_name  
from instructor
```

Select Clause (3)

Module 08

Partha Pratim
DasObjectives &
Outline

Outline

History of SQL

Data Definition
Language (DDL)

Create Table

Integrity Constraints

Update Table

Data
Manipulation
Language (DML):
Query Structure

Select Clause

Where Clause

From Clause

Module Summary

- An asterisk in the select clause denotes *all attributes*

```
select *  
from instructor
```

- An attribute can be a literal with no **from** clause

```
select '437'
```

- Results is a table with one column and a single row with value '437'
 - Can give the column a name using:

```
select '437' as FOO
```

- An attribute can be a literal with **from** clause

```
select 'A'  
from instructor
```

- Result is a table with one column and N rows (number of tuples in the instructors table), each row with value 'A'



Select Clause (4)

Module 08

Partha Pratim
Das

Objectives &
Outline

Outline

History of SQL

Data Definition
Language (DDL)

Create Table

Integrity Constraints

Update Table

Data
Manipulation
Language (DML):
Query Structure

Select Clause

Where Clause

From Clause

Module Summary

The **select** clause can contain arithmetic expressions involving the operation, +, -, *, and /, and operating on constants or attributes of tuples

- The query:

```
select ID, name, salary/12
      from instructor
```

- Would return a relation that is the same as the *instructor* relation, except that the value of the attribute *salary* is divided by 12
- Can rename “*salary/12*” using the **as** clause:

```
select ID, name, salary/12 as monthly_salary
```

Where Clause

Module 08

Partha Pratim
DasObjectives &
Outline

Outline

History of SQL

Data Definition
Language (DDL)

Create Table

Integrity Constraints

Update Table

Data
Manipulation
Language (DML):
Query Structure

Select Clause

Where Clause

From Clause

Module Summary

- The **where** clause specifies conditions that the result must satisfy

- Corresponds to the selection predicate of the relational algebra

- To find all instructors in Comp. Sci. dept

```
select name  
from instructor  
where dept_name = 'Comp. Sci.'
```

- Comparison results can be combined using the logical connectives **and**, **or**, and **not**

- To find all instructors in Comp. Sci. dept with salary > 80000

```
select name  
from instructor  
where dept_name = 'Comp. Sci.' and salary > 80000
```

- Comparisons can be applied to results of arithmetic expressions



From Clause

Module 08

Partha Pratim
Das

Objectives &
Outline

Outline

History of SQL

Data Definition
Language (DDL)

Create Table

Integrity Constraints

Update Table

Data
Manipulation
Language (DML):
Query Structure

Select Clause

Where Clause

From Clause

Module Summary

- The **from** clause lists the relations involved in the query
 - Corresponds to the **Cartesian product operation** of the relational algebra
- Find the Cartesian product *instructor X teaches*

```
select *
  from instructor, teaches
```

 - Generates every possible instructor-teaches pair, with all attributes from both relations
 - For common attributes (for example, *ID*), the attributes in the resulting table are renamed using the relation name (for example, *instructor.ID*)
- Cartesian product not very useful directly, but useful combined with where-clause condition (selection operation in relational algebra)

Examples

Module 09

Partha Pratim
DasObjectives &
OutlineAdditional Basic
Operations

Cartesian Product

Rename AS
Operation

String Values

Order By Clause

Select Top / Fetch
ClauseWhere Clause
Predicates

Duplicates

Module Summary

- Find the names of all instructors who have taught some course and the course_id

```
select name, course_id
```

```
from instructor, teaches
```

```
where instructor.ID = teaches.ID
```

o Equi-Join, Natural Join

instructor				teaches				
ID	name	dept_name	salary	ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2009
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2010
32343	El Said	History	60000	15151	MU-199	1	Spring	2010
33456	Gold	Physics	87000	22222	PHY-101	1	Fall	2009
45565	Katz	Comp. Sci.	75000	32343	HIS-351	1	Spring	2010
58583	Califieri	History	62000	45565	CS-101	1	Spring	2010
76543	Singh	Finance	80000	45565	CS-319	1	Spring	2010
76766	Crick	Biology	72000	76766	BIO-101	1	Summer	2009
83821	inst.ID				teaches.ID			
98345	inst.name				course_id			
	dept_name				sec_id			
	salary				semester			
	teaches.ID				year			

Examples

Module 09

Partha Pratim
Das

Objectives &
Outline

Additional Basic
Operations

Cartesian Product

Rename AS
Operation

String Values

Order By Clause

Select Top / Fetch
Clause

Where Clause
Predicates

Duplicates

Module Summary

- Find the names of all instructors in the Art department who have taught some course and the course_id

```
select name, course_id  
from instructor, teaches  
where instructor.ID = teaches.ID and instructor.dept_name = 'Art'
```

Rename AS Operation

Module 09

Partha Pratim
DasObjectives &
OutlineAdditional Basic
Operations

Cartesian Product

Rename AS
Operation

String Values

Order By Clause

Select Top / Fetch
ClauseWhere Clause
Predicates

Duplicates

Module Summary

- The SQL allows renaming relations and attributes using the **as** clause:
 $old_name \text{ as } new_name$
- Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.

select distinct T.name *instructor X instructor*
from instructor as T, instructor as S, cartesian product with itself
where T.salary > S.salary and S.dept_name = 'Comp. Sci'

- Keyword **as** is optional and may be omitted

instructor as T ≡ instructor T

String Operations

Module 09

Partha Pratim
DasObjectives &
OutlineAdditional Basic
Operations

Cartesian Product

Rename AS
Operation

String Values

Order By Clause

Select Top / Fetch
ClauseWhere Clause
Predicates

Duplicates

Module Summary

- SQL includes a **string-matching operator for comparisons** on character strings. The operator **like** uses patterns that are described using two special characters:
 - percent (%). The % character matches any substring
 - underscore (_). The _ character matches any character
- Find the names of all instructors whose name includes the substring “dar”

```
select name  
from instructor  
where name like '%dar--'
```



- Match the string “100%”
like '100%' escape '\'

- in that above we use backslash (\) as the escape character

String Operations (2)

Module 09

Partha Pratim
DasObjectives &
OutlineAdditional Basic
Operations

Cartesian Product

Rename AS
Operation

String Values

Order By Clause

Select Top / Fetch
ClauseWhere Clause
Predicates

Duplicates

Module Summary

- Patterns are case sensitive
- Pattern matching examples:
 - 'Intro%' matches any string beginning with "Intro"
 - '%Comp%' matches any string containing "Comp" as a substring
 - '___' matches any string of exactly three characters
 - '___ %' matches any string of at least three characters
- SQL supports a variety of string operations such as
 - concatenation (using "||")
 - converting from upper to lower case (and vice versa)
 - finding string length, extracting substrings, etc.

```
-- Extract substring from position 7 to the end
SELECT SUBSTRING('Hello World' FROM 7) AS extracted_substring;
```

```
-- Extract substring starting from position 1 with a length of 5
SELECT SUBSTRING('Hello World' FROM 1 FOR 5) AS extracted_substring;
```

Ordering the Display of Tuples

Module 09

Partha Pratim
Das

Objectives &
Outline

Additional Basic
Operations

Cartesian Product

Rename AS
Operation

String Values

Order By Clause

Select Top / Fetch
Clause

Where Clause
Predicates

Duplicates

Module Summary

- List in alphabetic order the names of all instructors

```
select distinct name  
from instructor  
order by name
```

- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; **ascending order is the default.**
 - Example: **order by name desc**
- Can sort on multiple attributes
 - Example: **order by dept_name, name**

Selecting Number of Tuples in Output

Module 09

Partha Pratim
Das

Objectives &
Outline

Additional Basic
Operations

Cartesian Product

Rename AS
Operation

String Values

Order By Clause

Select Top / Fetch
Clause

Where Clause
Predicates

Duplicates

Module Summary

- The **Select Top** clause is used to specify the number of records to return
- The **Select Top** clause is useful on large tables with thousands of records. Returning a large number of records can impact performance

```
select top 10 distinct name  
from instructor
```

- Not all database systems support the SELECT TOP clause.
 - SQL Server & MS Access support **select top**
 - MySQL supports the **limit** clause
 - Oracle uses **fetch first n rows only** and **rownum**

```
select distinct name  
from instructor  
order by name  
fetch first 10 rows only
```

Where Clause Predicates

Module 09

Partha Pratim
Das

Objectives &
Outline

Additional Basic
Operations

Cartesian Product

Rename AS
Operation

String Values

Order By Clause

Select Top / Fetch
Clause

Where Clause
Predicates

Duplicates

Module Summary

- SQL includes a **between** comparison operator
- Example: Find the names of all instructors with salary between \$90,000 and \$100,000 (that is, $\geq \$90,000$ and $\leq \$100,000$)

```
select name  
from instructor  
where salary between 90000 and 100000
```

- Tuple comparison

```
select name, course_id  
from instructor, teaches  
where (instructor.ID, dept_name) = (teaches.ID, 'Biology');
```

In Operator

Module 09

Partha Pratim
Das

Objectives &
Outline

Additional Basic
Operations

Cartesian Product

Rename AS
Operation

String Values

Order By Clause

Select Top / Fetch
Clause

Where Clause
Predicates

Duplicates

Module Summary

- The **in** operator allows you to specify multiple values in a **where** clause
- The **in** operator is a shorthand for multiple **or** conditions

```
select name  
from instructor  
where dept_name in ('Comp. Sci.', 'Biology')
```

Duplicates

Module 09

Partha Pratim
DasObjectives &
OutlineAdditional Basic
Operations

Cartesian Product

Rename AS
Operation

String Values

Order By Clause

Select Top / Fetch
ClauseWhere Clause
Predicates

Duplicates

Module Summary

- In relations with duplicates, SQL can define how many copies of tuples appear in the result
- Multiset versions of some of the relational algebra operators – given multiset relations r_1 and r_2 :
 - a) $\sigma_\theta(r_1)$: If there are c_1 copies of tuple t_1 in r_1 , and t_1 satisfies selections σ_θ , then there are c_1 copies of t_1 in $\sigma_\theta(r_1)$
 - b) $\Pi_A(r)$: For each copy of tuple t_1 in r_1 , there is a copy of tuple $\Pi_A(t_1)$ in $\Pi_A(r_1)$ where $\Pi_A(t_1)$ denotes the projection of the single tuple t_1
 - c) $r_1 \times r_2$: If there are c_1 copies of tuple t_1 in r_1 and c_2 copies of tuple t_2 in r_2 , there are $c_1 \times c_2$ copies of the tuple $t_1.t_2$ in $r_1 \times r_2$

Duplicates (2)

Module 09

Partha Pratim
DasObjectives &
OutlineAdditional Basic
Operations

Cartesian Product

Rename AS
Operation

String Values

Order By Clause

Select Top / Fetch
ClauseWhere Clause
Predicates

Duplicates

Module Summary

- Example: Suppose multiset relations $r_1(A, B)$ and $r_2(C)$ are as follows:
 $r_1 = \{(1, a)(2, a)\}$ $r_2 = \{(2), (3), (3)\}$
- Then $\Pi_B(r_1)$ would be $\{(a), (a)\}$, while $\Pi_B(r_1) \times r_2$ would be
 $\{(a, 2), (a, 2), (a, 3), (a, 3), (a, 3), (a, 3)\}$
- SQL duplicate semantics:

```
select A1, A2, ..., An
from r1, r2, ..., rm
where P
```

is equivalent to the *multiset* version of the expression:

$$\Pi_{A_1, A_2, \dots, A_n}(\sigma_P(r_1 \times r_2 \times \dots \times r_m))$$

Set Operations

Module 10

Partha Pratim
DasObjectives &
Outline

Set Operations

Null Values

Three Valued Logic

Aggregate
FunctionsGroup By
Having

Null Values

Module Summary

- Find courses that ran in Fall 2009 or in Spring 2010
`(select course_id from section where sem = 'Fall' and year = 2009)`
union
`(select course_id from section where sem = 'Spring' and year = 2010)`
- Find courses that ran in Fall 2009 and in Spring 2010
`(select course_id from section where sem = 'Fall' and year = 2009)`
intersect
`(select course_id from section where sem = 'Spring' and year = 2010)`
- Find courses that ran in Fall 2009 but not in Spring 2010
`(select course_id from section where sem = 'Fall' and year = 2009)`
except
`(select course_id from section where sem = 'Spring' and year = 2010)`

Set Operations (2)

Module 10

Partha Pratim
DasObjectives &
Outline

Set Operations

Null Values

Three Valued Logic

Aggregate
Functions

Group By

Having

Null Values

Module Summary

- Find the salaries of all instructors that are less than the largest salary

```
select distinct T.salary
from instructor as T, instructor as S
where T.salary < S.salary
```

- Find all the salaries of all instructors

```
select distinct salary
from instructor
```

- Find the largest salary of all instructors

```
(select "second query" )
except
(select "first query" )
```



Set Operations (3)

Module 10

Partha Pratim
Das

Objectives &
Outline

Set Operations

Null Values

Three Valued Logic

Aggregate
Functions

Group By

Having

Null Values

Module Summary

- Set operations **union**, **intersect**, and **except**
 - Each of the above operations automatically eliminates duplicates
- To retain all duplicates use the corresponding multiset versions **union all**, **intersect all**, and **except all**.
- Suppose a tuple occurs m times in r and n times in s , then, it occurs:
 - $m + n$ times in r **union all** s
 - $\min(m, n)$ times in r **intersect all** s
 - $\max(0, m - n)$ times in r **except all** s





Null Values

Module 10

Partha Pratim
Das

Objectives &
Outline

Set Operations

Null Values

Three Valued Logic

Aggregate
Functions

Group By

Having

Null Values

Module Summary

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist
- The result of any arithmetic expression involving *null* is *null*
 - Example: $5 + \text{null}$ returns *null*
- The predicate **is null** can be used to check for null values
 - Example: Find all instructors whose salary is null

```
select name  
from instructor  
where salary is null
```

- It is not possible to test for **null** values with comparison operators, such as $=$, $<$, or \neq .
We need to use the **is null** and **is not null** operators instead



Null Values (2): Three Valued Logic

Module 10

Partha Pratim
DasObjectives &
Outline

Set Operations

Null Values

Three Valued Logic

Aggregate
Functions

Group By

Having

Null Values

Module Summary

- Three values – **true, false, unknown**
- Any comparison with *null* returns *unknown*
 - Example: $5 < \text{null}$ or $\text{null} < > \text{null}$ or $\text{null} = \text{null}$
- Three-valued logic using the value *unknown*:
 - OR: $(\text{unknown or true}) = \text{true}$,
 $(\text{unknown or false}) = \text{unknown}$
 $(\text{unknown or unknown}) = \text{unknown}$
 - AND: $(\text{true and unknown}) = \text{unknown}$,
 $(\text{false and unknown}) = \text{false}$,
 $(\text{unknown and unknown}) = \text{unknown}$
 - NOT: $(\text{not unknown}) = \text{unknown}$
 - “*P is unknown*” evaluates to *true* if predicate *P* evaluates to *unknown*
- Result of **where clause** predicate is treated as *false* if it evaluates to *unknown*



Aggregate Functions

Module 10

Partha Pratim
Das

Objectives &
Outline

Set Operations

Null Values

Three Valued Logic

Aggregate
Functions

Group By

Having

Null Values

Module Summary

- These functions operate on the multiset of values of **a column** of a relation, and **return a value**

avg: average value

min: minimum value

max: maximum value

sum: sum of values

count: number of values

Aggregate Functions (2)

Module 10

Partha Pratim
DasObjectives &
Outline

Set Operations

Null Values

Three Valued Logic

Aggregate
Functions

Group By

Having

Null Values

Module Summary

- Find the average salary of instructors in the Computer Science department

```
select avg (salary)
from instructor
where dept_name = 'Comp. Sci';
```
- Find the total number of instructors who teach a course in the Spring 2010 semester

```
select count (distinct ID)
from teaches
where semester = 'Spring' and year = 2010;
```
- Find the **number of tuples** in the *course* relation

```
select count (*)
from courses;
```

 count will be same no matter what attribute you choose ???

Aggregate Functions (3): Group By

Module 10

 Partha Pratim
 Das

 Objectives &
 Outline

Set Operations

Null Values

Three Valued Logic

 Aggregate
 Functions

Group By

Having

Null Values

Module Summary

- Find the average salary of instructors in each department

```
select dept_name, avg(salary) as avg_salary
```

```
from instructor
```

```
group by dept_name;
```

applies on sub-group

ID	name	dept_name	salary
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	avg_salary
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Aggregate Functions (4): Group By

Module 10

Partha Pratim
Das

Objectives &
Outline

Set Operations

Null Values

Three Valued Logic

Aggregate
Functions

Group By

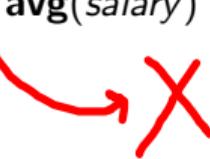
Having

Null Values

Module Summary

- Attributes in **select** clause outside of aggregate functions must appear in **group by** list

```
/* erroneous query */  
select dept_name, ID, avg(salary)  
from instructor  
group by dept_name;
```



Aggregate Functions (5): Having Clause

Module 10

Partha Pratim
Das

Objectives &
Outline

Set Operations

Null Values

Three Valued Logic

Aggregate
Functions

Group By
Having

Null Values

Module Summary

- Find the names and average salaries of all departments whose average salary is greater than 42000

```
select dept_name, avg(salary)
  from instructor
 group by dept_name
 having avg(salary) > 42000;
```

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups



Null Values and Aggregates

Module 10

Partha Pratim
Das

Objectives &
Outline

Set Operations

Null Values

Three Valued Logic

Aggregate
Functions

Group By

Having

Null Values

Module Summary

- Total all salaries

```
select sum (salary)  
from instructor;
```

- Above statement ignores null amounts
- Result is *null* if there is no non-null amount
- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes
- What if collection has only null values?
 - count returns 0
 - all other aggregates return null

- Why do we need a new number system for computers and other computing machines?
 - ✓ They represent an ON state 1 and OFF state 0, which is why they are easy to represent in electronics.
 - ✓ Lesser circuitry required to design devices.
 - ✓ Computers work by taking logical decisions at every step of their work.
A decision making process is itself binary in nature, either YES (1) or NO (0).

- The smallest unit of binary data is a bit. A string of 8 bits make 1 Byte of data.
- The Base of binary number system is 2 and so there are two digits in this number system – 0 and 1.
- How many values can a given number of bits represent?
 - ✓ 'n' bits can represent 2^n possible values in binary number system. For example with 2 bits we can represent $2^2 = 4$ values (00, 01, 10, 11).

representing from 0 to $2^n - 1$

✓ Steps for Converting Decimal number to Binary (Integer part)

- 1) Divide the decimal number by 2, note down the remainder & divide the quotient obtained.
- 2) Repeat step 1 till the quotient becomes 0
- 3) Place the remainder values from bottom to top of the division process with the bottom-most remainder as the most significant bit and the top most remainder as the least significant bit.
- 4) Obtained number is the binary equivalent of the given decimal number.

✓ Steps for Converting Decimal number to Binary (Fractional part)

- 1) Multiply the decimal fractional number by 2 and note down the integer part of the result.
- 2) Separate out the fractional part of the result and multiply 2 with it.
- 3) Repeat step 2 until fractional part is 0 or a desired level of precision is obtained.
In each repetition note down the integer part
- 4) Write the integer parts from top to bottom to get the binary number.

11101011 - (1 byte.) 8 bits

i) $1729 = \text{Taxi Cab No.}$

2	1729
2	864 — 1
2	432 — 0
2	216 — 0
2	108 — 0
2	54 — 0
2	27 — 0
2	13 — 1
2	6 — 1
2	3 — 0
2	1 — 1
	0 — 1

$$1000001_2 = (1729)_{10}$$

ii) $2.718 = ?$

Integer part: $\frac{2}{2} \begin{matrix} 1 \\ 2 \end{matrix} \frac{1}{2} \begin{matrix} 1 \\ 0 \end{matrix} \frac{0}{2} \begin{matrix} 1 \\ 1 \end{matrix} (2)_{10} = (10)_2$

$$(10 \cdot 1011011)_2 = (2.718)_{10}$$

Fractional part: 1

$$0.718 \times 2 = 1.436 \quad \begin{matrix} 1 \\ 0 \end{matrix}$$

$$0.436 \times 2 = 0.872 \quad \begin{matrix} 1 \\ 0 \end{matrix}$$

$$0.872 \times 2 = 1.744 \quad \begin{matrix} 1 \\ 1 \end{matrix}$$

$$0.744 \times 2 = 1.488 \quad \begin{matrix} 1 \\ 1 \end{matrix}$$

$$0.488 \times 2 = 0.976 \quad \begin{matrix} 0 \\ 0 \end{matrix}$$

$$0.976 \times 2 = 1.952 \quad \begin{matrix} 1 \\ 1 \end{matrix}$$

$$0.952 \times 2 = 1.904 \quad \begin{matrix} 1 \\ 1 \end{matrix}$$

$$(0.718)_{10} = (011011)_2$$

Binary to Decimal conversion rules

✓ Steps for Converting Binary number to Decimal (Integer part)

- 1) Write down the bits of binary numbers with their corresponding power of 2.
- 2) Multiply the bits with their corresponding power of 2.
- 3) Add each term obtained as a result of the product in step 2 to get the final answer.

✓ Steps for Converting Binary number to Decimal (Fractional part)

Same as conversion of Integer part just the powers of 2 are negative.
Starting from -1.

Q) Convert the given binary numbers into equivalent decimal representation

i) 110110000001

10 9 8 7 6 5 4 3 2 1 0
1 | 0 1 1 0 0 0 0 0 |
 $1 \times 2^9 + 1 \times 2^8 + 0 \times 2^7 + 1 \times 2^6 + \dots + 1 \times 2^0$

$$= 1024 + 512 + 0 + 128 + 64 + 0 + 0 + 0 + 0 + 1.
= (1729)_{10}.$$

ii) 10 10110111

$$1 \times 2^1 + 0 \times 2^0 = (2)_{10}.$$

$$\begin{array}{cccccccc} -1 & -2 & -3 & -4 & -5 & -6 & -7 & -8 \\ . & 1 & 0 & 1 & 1 & 0 & 1 & 1 \end{array}$$

$$\begin{aligned} &= 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4} + 0 \times 2^{-5} \\ &\quad + 1 \times 2^{-6} + 1 \times 2^{-7} + 1 \times 2^{-8}. \end{aligned}$$

$$\begin{aligned} &= 0.5 + 0 + 0.125 + 0.0625 + \\ &\quad 0.0156 + 0.0078 + 0.0039. \\ &= (0.7148)_{10}. \end{aligned}$$

$$(2 \cdot 7148)_{10}$$

- **Character Encoding**, it is a system in which characters are mapped to a number and is stored as a stream of bits (binary equivalent of that number) in the memory of computer.

● ASCII – American Standard Code for Information Interchange

Its the basic character encoding scheme used for representing text (symbols and characters) in computers. It is originally a 7 bit code and therefore represents 128 characters.

Present day we use "Extended ASCII" which is 8 bits and encodes 256 characters.
Some common ranges of ASCII encoding

- ✓ Numbers (0 to 9) : ASCII value range – 48 to 57
- ✓ Upper Case Alphabets (A,B,...Z) : ASCII value range – 65 to 90
- ✓ Lower Case Alphabets (a,b,...z) : ASCII value range – 97 to 122.

Example : How is the letter W stored in memory?

- ✓ ASCII code maps 'W' to the number 87, binary equivalent of 87 = 1010111
So W is stored in memory as 1010111. Please note in actual machines it is easier to store 8 bits together as they form a byte, hence the characters may be stored as 1 Byte long data with a 0 added as the 8th bit.
That's why many languages have 1 byte long data type for Char

1010111

- **UNICODE** is a multilingual character encoding system. It has up to 16 bits for encoding characters (UTF-16).



Till now we have seen the binary number system and how numbers and characters are represented in binary. Now we will see what kind of operations can be done on these binary numbers and their result.

Fun fact: Aristotle worked on a system of formal logic and reasoning. For centuries mathematicians tried to solve these logic problems with conventional mathematics but couldn't. **George Boole** devised his own mathematical system of logic and was able to solve the problems of Aristotle using this system. Boole's paper "**An Investigation of The Laws of the Thought**" got published in 1854 that led to the development of a new system of algebra – "Boolean Algebra".

A Boolean variable stores truth value for some logic ie it can either be 1 or 0.

Basic operations performed in Boolean variables are :

- ✓ Conjunction or AND denoted by (.)
- ✓ Disjunction or OR denoted by (+)
- ✓ Negation NOT denoted by (1, overbar)

There are other operations also like XOR, NAND, NOR, XNOR but these operations are derivable using the three basic operations.

We are learning these operations because these operations are applied strategically on binary strings to solve many problems and implement concepts and techniques in computers. Eg- Recovery of lost data, Memory management, Error correction etc.

AND operation

It is called logical multiplication in boolean algebra. The rules of AND operation are

- ✓ $0 \cdot 0 = 0$
- ✓ $1 \cdot 0 = 0$
- ✓ $0 \cdot 1 = 0$
- ✓ $1 \cdot 1 = 1$

Logical operations are implemented using electronic devices called "gates".

Symbol for AND gate :

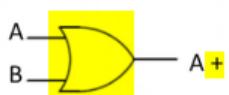


OR operation

It is called logical addition in boolean algebra. The rules of OR operation are

- ✓ $0 + 0 = 0$
- ✓ $1 + 0 = 1$
- ✓ $0 + 1 = 1$
- ✓ $1 + 1 = 1$

Symbol of OR gate :

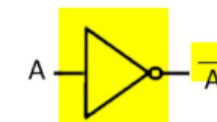


NOT operation

It is called logical complementation in boolean algebra. The rules of NOT operation are

- ✓ $\overline{0} = 1$
- ✓ $\overline{1} = 0$

Symbol for NOT gate :



XOR operation

It is denoted by the symbol (^) and is a compound operation equivalent to

$$A \wedge B = A\bar{B} + \bar{A}B$$

The rules of XOR operation

- ✓ $0 \wedge 0 = 0$
- ✓ $1 \wedge 0 = 1$
- ✓ $0 \wedge 1 = 1$
- ✓ $1 \wedge 1 = 0$



Symbol of XOR gate :



1 when they are different, else 0

There can be many other electronic gates and corresponding boolean operation like NAND, NOR, XNOR. All of them are just a specific cascade of multiple basic gates.

Also there are various laws for solving boolean expression like Associative law, Complementary law, De Morgan's Laws etc. If you want to learn about them in detail refer standard books. For the purpose of the course we only need the basic boolean operations.