

# The Relational Data Model and Relational Database Constraints

Data Intelligence and Learning ([DIAL](#)) Lab

Prof. Jongwuk Lee

# What is a Data Model?

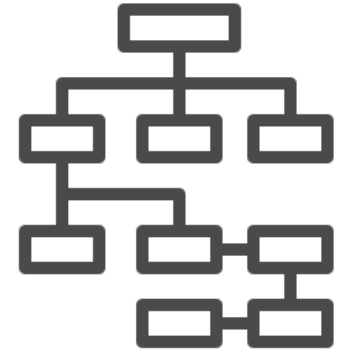


- An abstract model with **structures + operations + constraints**
  - ◆ **Structures** of a database
  - ◆ The **operations** for manipulating the structures
  - ◆ Certain **constraints** that the database should obey



# What is a Data Model?

➤ **Structures** include **elements** (data types) and **groups of elements** (e.g., entity, table) and **relationship** among such groups



➤ **Operations** are used for specifying **database retrievals** and **updates** by referring to the constructs of the data model.



➤ **Constraints** specify some **restrictions** on **valid data**.

- ◆ These constraints must be always enforced.



# Categories of Data Models



## ➤ Conceptual (high-level, semantic) data models

- ◆ Describing the way users perceive data



## ➤ Physical (low-level, internal) data models

- ◆ Describing how data is stored in the computer



## ➤ Self-describing data models

- ◆ Combine the **description of data** with the **data values**.
- ◆ Examples include **XML**, **key-value stores** and some **NOSQL** systems.

# Database Schema vs. State



## ➤ Database **schema**

- ◆ The **description of a database**
- ◆ Includes descriptions of the **database structure, data types, and the constraints** on the database.



## ➤ Database **state**

- ◆ The **actual data stored in a database** at a particular moment in time
- ◆ Also called **database instance** (or occurrence or snapshot).
- ◆ Includes **the collection of all the data in the database.**
- ◆ The term instance is also applied to individual database components, e.g., record instance, table instance, entity instance.



# Database Schema vs. State



- The database schema changes very **infrequently**.
- The database state changes every time **the database is updated**.

**STUDENT**

Name	Student_number	Class	Major
------	----------------	-------	-------

**COURSE**

Course_name	Course_number	Credit_hours	Department
-------------	---------------	--------------	------------

**PREREQUISITE**

Course_number	Prerequisite_number
---------------	---------------------

**SECTION**

Section_identifier	Course_number	Semester	Year	Instructor
--------------------	---------------	----------	------	------------

**GRADE\_REPORT**

Student_number	Section_identifier	Grade
----------------	--------------------	-------

**STUDENT**

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

**COURSE**

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

**SECTION**

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

**GRADE\_REPORT**

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

**PREREQUISITE**

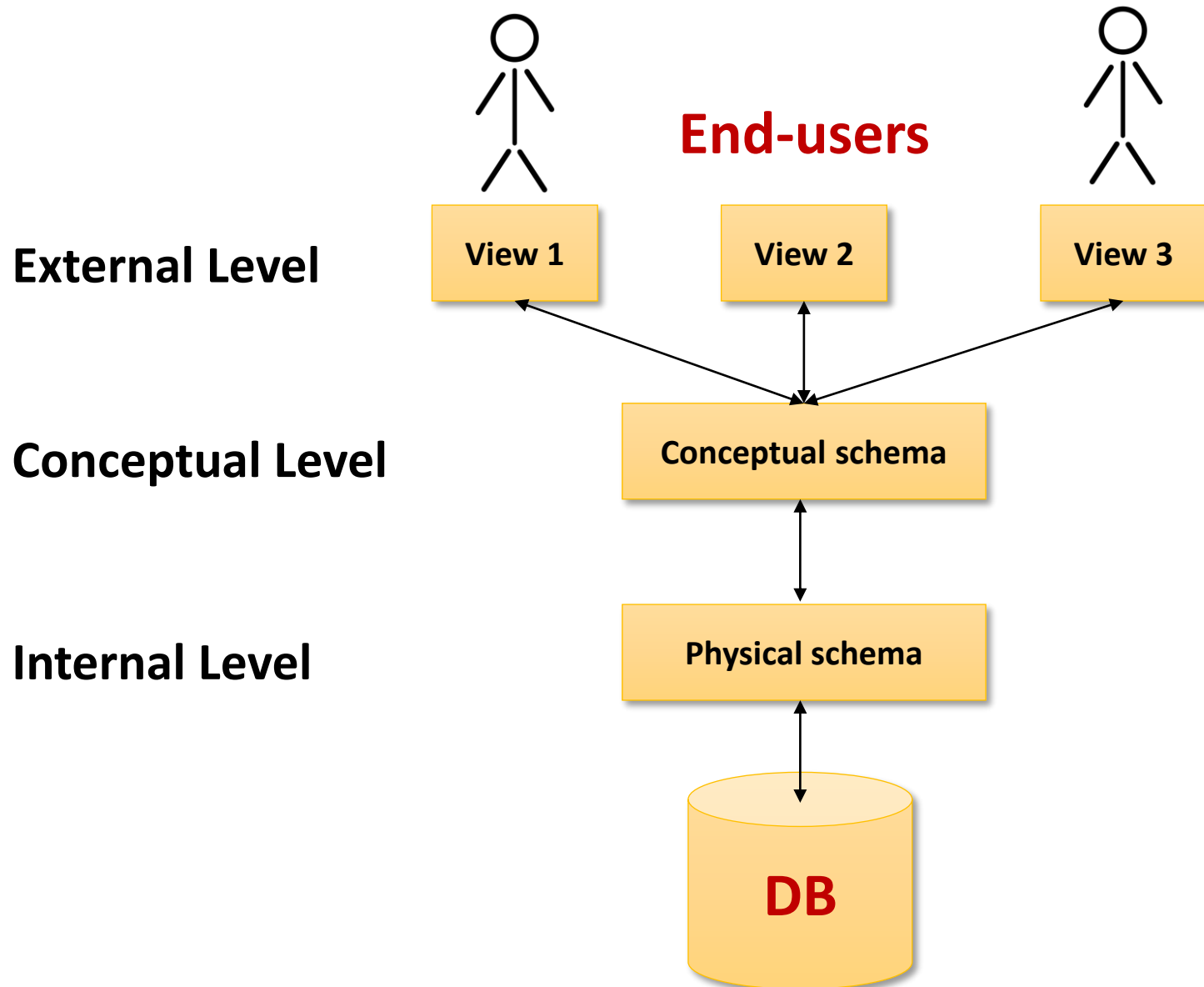
Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310



# Three-Schema Architecture

- **External schemas** to describe the various **user views**
  - ◆ Usually uses the same data model as the conceptual schema.
  
- **Conceptual schema** to describe the **structure and constraints** for the whole database
  - ◆ Uses a conceptual data model.
  
- **Internal schema** to describe **physical storage structures and access paths** (e.g., index)
  - ◆ Typically, uses a physical data model.

# Three-Schema Architecture







# Data Independence

- When a schema at a lower level is changed, only the mappings between this schema and higher-level schemas need to be changed in a DBMS.
  
- **Logical data independence**
  - ◆ The capacity to change the conceptual schema without having to change the external schemas and their related application programs
  
- **Physical data independence**
  - ◆ The capacity to change the internal schema without having to change the conceptual schema
  - ◆ For example, the internal schema may be changed when certain file structures are reorganized.

# Three-Tier Client-Server Architecture

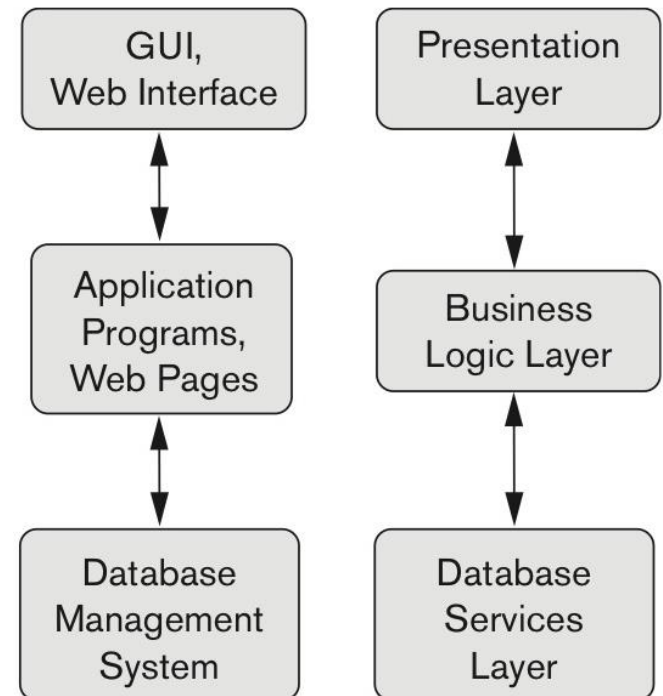


## ➤ Intermediate layer, called application server or Web server

- ◆ Store the web connectivity software and the business logic part of the application used to access the corresponding data.

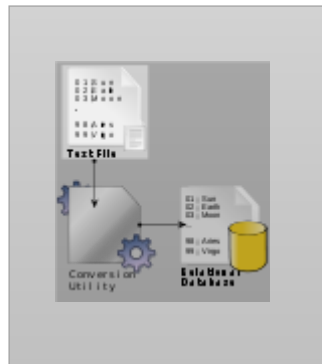
## ➤ Three-tier architecture can enhance **security**.

- ◆ Database server only accessible via middle tier.
- ◆ Clients cannot directly access database server.
- ◆ Clients contain user interfaces and Web browsers.
- ◆ The client is typically a PC or a mobile device connected to the Web.

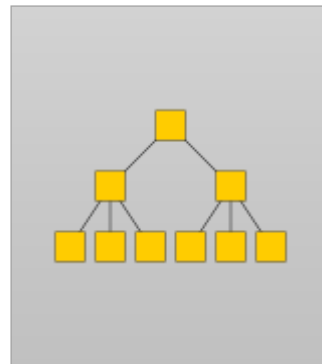


# Evolution of Data Models

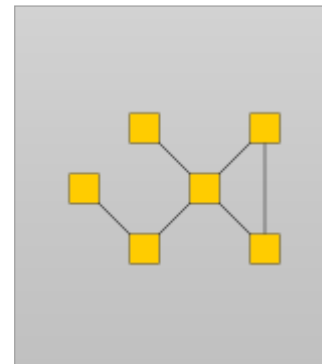
- Flat models
- Network model
- Hierarchical model
- **Relational model**
- Object-relational model



Flat model



Hierarchical model

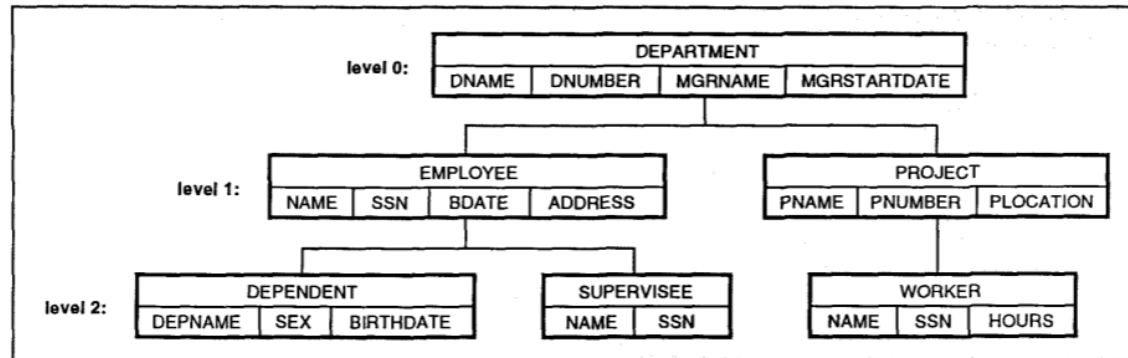


Network model

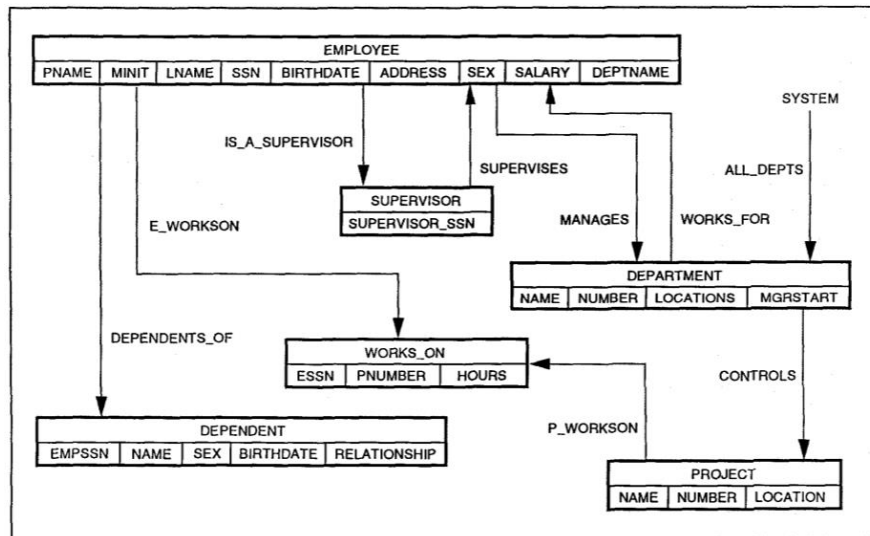


Relational model

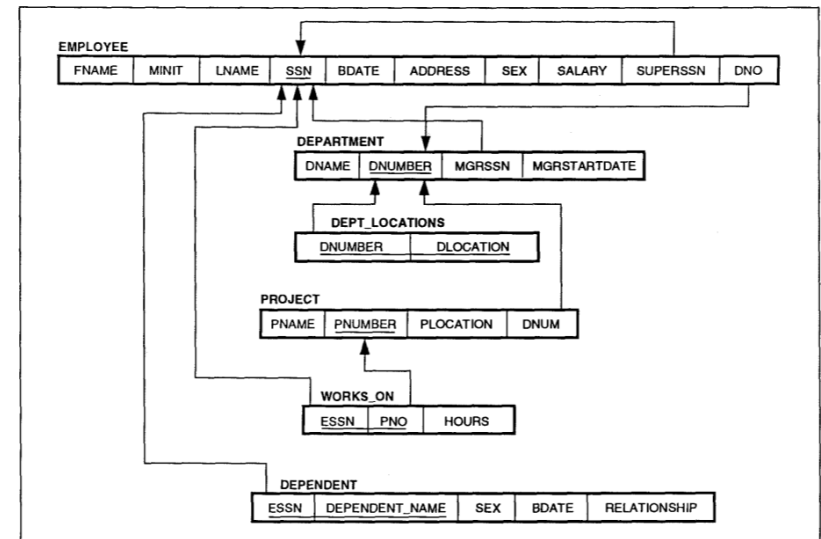
# Evolution of Data Models



**Hierarchical model**



**Network model**



**Relational model**

# Why we Study the Relational Model?



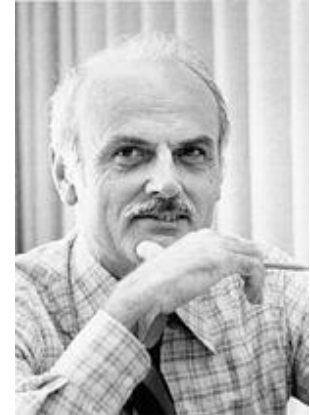
- Proposed in 1970 by E.F. Codd (IBM), first commercial system in 1981-82.

- **Most dominant** for developing database applications

- ◆ Commercial: Oracle, IBM DB2, Microsoft SQL Server
- ◆ Open source: MySQL, Postgres, SQLite, etc.
- ◆ **SQL relational standards**: SQL-89 (SQL1), SQL-92 (SQL2), SQL-99, SQL3

- **Competitors: object-oriented model , XML, NoSQL**

- ◆ A synthesis emerging: **object-relational model**
- ◆ Informix Universal Server, UniSQL, O2, Oracle, DB2
- ◆ Despite its several pros, it is quite **complex** to learn and to use.





# Relational Data Models

# What is the Relational Model?

- **It was first proposed by Dr. E.F. Codd of IBM Research in 1970 in the following paper.**
  - ◆ "A Relational Model for Large Shared Data Banks," Communications of the ACM, June 1970

- **Structures**

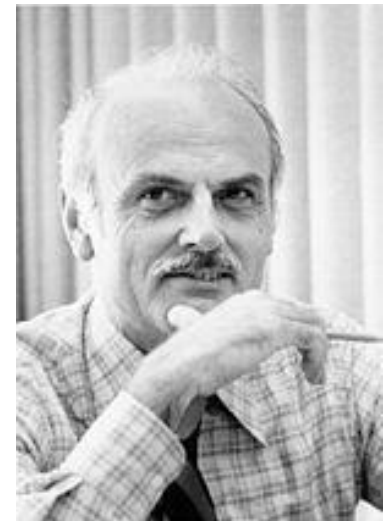
- ◆ Creating/modifying the structures of relations

- **Operations**

- ◆ Inserting/deleting/updating/retrieving records in the database

- **Constraints**

- ◆ Determining which values are permissible and which are not in the database



**Edgar F. Codd**  
(Winner of Turing Award in 1981)

# What is the Relational Model?

- A relational database has **a set of relations**.
- A relation consists of two parts.
  - ◆ **Schema**: relation name + name and type of each column
  - ◆ **Instance**: a table with rows and columns
    - # of rows = **cardinality**, # of columns = **degree, arity**
    - A relation = a set of rows (or **tuples**)

Schema	<b>Courses</b>			
	<i>cid</i>	<i>cname</i>	<i>credit</i>	<i>department</i>
Instance	101	C programming	3	CS
	102	Discrete Math	2	Math
	301	Databases	4	CS
	302	Artificial Intelligence	3	CS
	405	Data Mining	3	CS



# Schema vs. Instance

- The schema is the **logical structure** of the database.
- The instance is a **snapshot** of the data in the database at a given instant in time.

Schema

## *Courses*

<i>cid</i>	<i>cname</i>	<i>credit</i>	<i>department</i>
------------	--------------	---------------	-------------------

Instance

101	C programming	3	CS
102	Discrete Math	2	Math
301	Databases	4	CS
302	Artificial Intelligence	3	CS
405	Data Mining	3	CS

# Definition: Schema

## ➤ The schema of a relation R.

- ◆ Denoted by  $R(A_1, A_2, \dots, A_n)$ 
  - R is the **name** of the relation.
  - The **attributes** of the relation are  $A_1, A_2, \dots, A_n$ .

## ➤ Each attribute has a **domain** or a **set of valid values**.

- ◆ E.g., the domain of *cid* is 6-digit numbers.

## ➤ Example

**Customer**(*cid*, *name*, *address*, *phone*)

- ◆ **Customer** is the relation's name.
- ◆ The relations defined over four attributes: *cid*, *name*, *address*, *phone*.



# Definition: Domain

- The attribute name is used to designate the role by a domain in a relation.
  - ◆ **The meaning of data elements** corresponding to an attribute
  - ◆ E.g., date may be used to define two attributes named “invoice-date” and “payment-date” with different meanings.
  
- A domain has a **data-type** or a **format** defined for it.
  - ◆ The USA\_phone\_numbers may have a format: (ddd)ddd-dddd where each d is a decimal digit.
  - ◆ Dates have various formats such as year, month, date formatted as yyyy-mm-dd, or as dd mm,yyyy etc.



# Definition: Tuple

➤ A **tuple** is an **ordered set of values** enclosed in angled brackets ' $\langle \dots \rangle$ '.

- ◆ Each value is derived from an appropriate domain.

➤ **Example**

**Customer**(*cid, name, address, phone*)

- ◆ A row in the **Customer** relation is a 4-tuple with four values:
  - This is called a 4-tuple as it has 4 values.
  - E.g.,  $\langle 632895, \text{"John Smith"}, \text{"101 Main St. Atlanta, GA 30332"}, \text{"(404) 894-2000"} \rangle$
- ◆ A tuple (row) in the **Customer** relation

➤ A **relation** is formed as a set of tuples (rows).

# Definition: Tuple

## ➤ Values in a tuple

- ◆ All values are considered **atomic**.

## ➤ Each value in a tuple must be from the **domain** of the attribute for that column.

- ◆ If tuple  $t = \langle v_1, v_2, \dots, v_n \rangle$  is a **tuple** (row) in the relation state  $r$  of  $R(A_1, A_2, \dots, A_n)$ , each  $v_i$  must be a value from  $dom(A_i)$

## ➤ A special **null value** is used to represent values that are **unknown, not available or inapplicable** in certain tuples.

# Definition: State

➤ Formally, given  $R(A_1, A_2, \dots, A_n)$ , then

$$r(R) \subseteq \text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n)$$

- ◆ The schema is denoted by  $R(A_1, A_2, \dots, A_n)$ .
- ◆ **R** is the **name** of the relation.
- ◆ The **attributes** of the relation are  $A_1, A_2, \dots, A_n$
- ◆ The **relation state** is a subset of **the Cartesian product** of the domains of its attributes.
  - $r(R)$ : **A specific state (or “population”)** of relation R
    - $r(R) = \{t_1, t_2, \dots, t_m\}$ , where each  $t_i$  is an  $n$ -tuple.
    - $t_i = \langle v_1, v_2, \dots, v_n \rangle$ , where each  $v_j$  is  $v_j \in \text{dom}(A_j)$ .

# Example: Relation and State

➤ Let  $R(A_1, A_2)$  be a relation schema.

- ◆ Let  $dom(A_1) = \{0, 1\}$  and  $dom(A_2) = \{a, b, c\}$ .
- ◆ Then,  $dom(A_1) \times dom(A_2)$  is all possible combinations:
  - $\{ \langle 0, a \rangle, \langle 0, b \rangle, \langle 0, c \rangle, \langle 1, a \rangle, \langle 1, b \rangle, \langle 1, c \rangle \}$
- ◆ A relation state  $r(R)$  holds:  $r(R) \subseteq dom(A_1) \times dom(A_2)$ 
  - E.g.,  $r(R) = \{ \langle 0, a \rangle, \langle 0, b \rangle, \langle 1, c \rangle \}$
  - This is **one possible state** of the relation  $R$ .

A possible relation state

$A_1$	$A_2$
0	a
0	b
1	c

$\subseteq$

A relation with all possible combinations

$A_1$	$A_2$
0	a
0	b
0	c
1	a
1	b
1	c

# Summary of Formal Definitions



Informal Terms	Formal Terms
<b>Table</b>	<b>Relation</b>
<b>Column Header</b>	<b>Attribute (or Field)</b>
<b>All possible column values</b>	<b>Domain</b>
<b>Row</b>	<b>Tuple</b>
<b>Table Definition</b>	<b>Schema of a Relation</b>
<b>Populated Table</b>	<b>State of a Relation</b>



# Characteristics of Relations

- Q: Why not a **tabular** data model?
- A: The “**mathematical relation**” is formally different from the table.
  
- **Ordering of tuples in a relation  $r(R)$** 
  - ◆ The tuples **are not considered to be ordered**, even though they appear to be in the tabular form.
  - ◆ Tuples may be stored in an **arbitrary** order.
- **Ordering of attributes in a relation schema  $R$  (and of values within each tuple)**
  - ◆ We usually consider the attributes in  $R(A_1, A_2, \dots, A_n)$  and the values in  $t = \langle v_1, v_2, \dots, v_n \rangle$  to be ordered.
  - ◆ However, **the relation does not require this ordering**.

# Example of Courses Relation

- Do values in all rows have to be distinct?
- Do values in all columns have to be distinct?

**Courses**(*cid*: string, *name*: string, *credit*: integer, *department*: string)

Relation name

Attributes (or Field, Column)

Attribute name

Tuples (or Records, Rows)

	<i>cid</i>	<i>cname</i>	<i>credit</i>	<i>department</i>
	101	C programming	3	CS
	102	Discrete Math	2	Math
	301	Databases	4	CS
	302	Artificial Intelligence	2	CS
	405	Data Mining	5	CS



# Informal Definition: Key

## ➤ Key of a relation

- ◆ Each row has a value of an attribute (a set of attributes) that **uniquely identifies that tuples** in the relation.

## ➤ Example

- ◆ In the STUDENT table, SSN is the key.
- ◆ In the COURSE table, CID is the key.

## ➤ Sometimes, row-ids or sequential numbers are assigned as keys to identify the rows in a table.

- ◆ Called artificial key or surrogate key

# Example of Relational Models

## ➤ Logical schema

- ◆ **Students**(sid: string, sname: string, gender: string)
- ◆ **Courses**(cid: string, cname: string, credits: int)
- ◆ **Enrolled**(sid: string, cid: string, grade: string)

**Students**

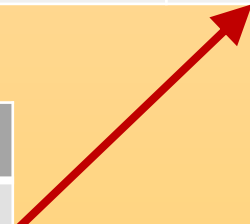
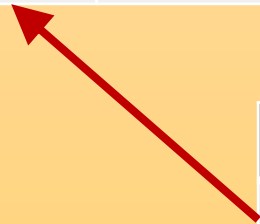
<i>sid</i>	<i>sname</i>	<i>gender</i>
100	Alice	Female
200	Bob	Male

**Courses**

<i>cid</i>	<i>cname</i>	<i>credit</i>
202	DS	3
301	DB	4

**Enrolled**

<i>sid</i>	<i>cid</i>	<i>grade</i>
100	202	A
200	301	B+





# Data Definition Language (DDL)

# What is SQL?



## ➤ Definition

- ◆ The origin of SQL is **relational predicate calculus**
- ◆ SQL comes from the word “SEQUEL”.
  - Popularly known as “**Structured query language.**”
- ◆ SQL is an informal or practical rendering of the relational data model with syntax.



## ➤ SQL tutorial

- ◆ <https://www.w3schools.com/sql/default.asp>
- ◆ Quiz: [https://www.w3schools.com/sql/sql\\_quiz.asp](https://www.w3schools.com/sql/sql_quiz.asp)

# Data Types for Attributes in SQL



## ➤ Numeric data type

- ◆ Integer numbers: INTEGER, INT, and SMALLINT
- ◆ Floating-point (real) numbers: FLOAT or REAL, and DOUBLE PRECISION

## ➤ Character-string data type

- ◆ Fixed length: **CHAR(n)**, CHARACTER(n)
- ◆ Varying length: **VARCHAR(n)**, CHAR VARYING(n),  
CHARACTER VARYING(n)

## ➤ Boolean data type

- ◆ Values of TRUE, FALSE or **NULL**



# Data Types for Attributes in SQL

## ➤ DATE data type

- ◆ Components are YEAR, MONTH, and DAY.
  - Usually, the format is YYYY-MM-DD.
- ◆ Multiple mapping functions available in RDBMS

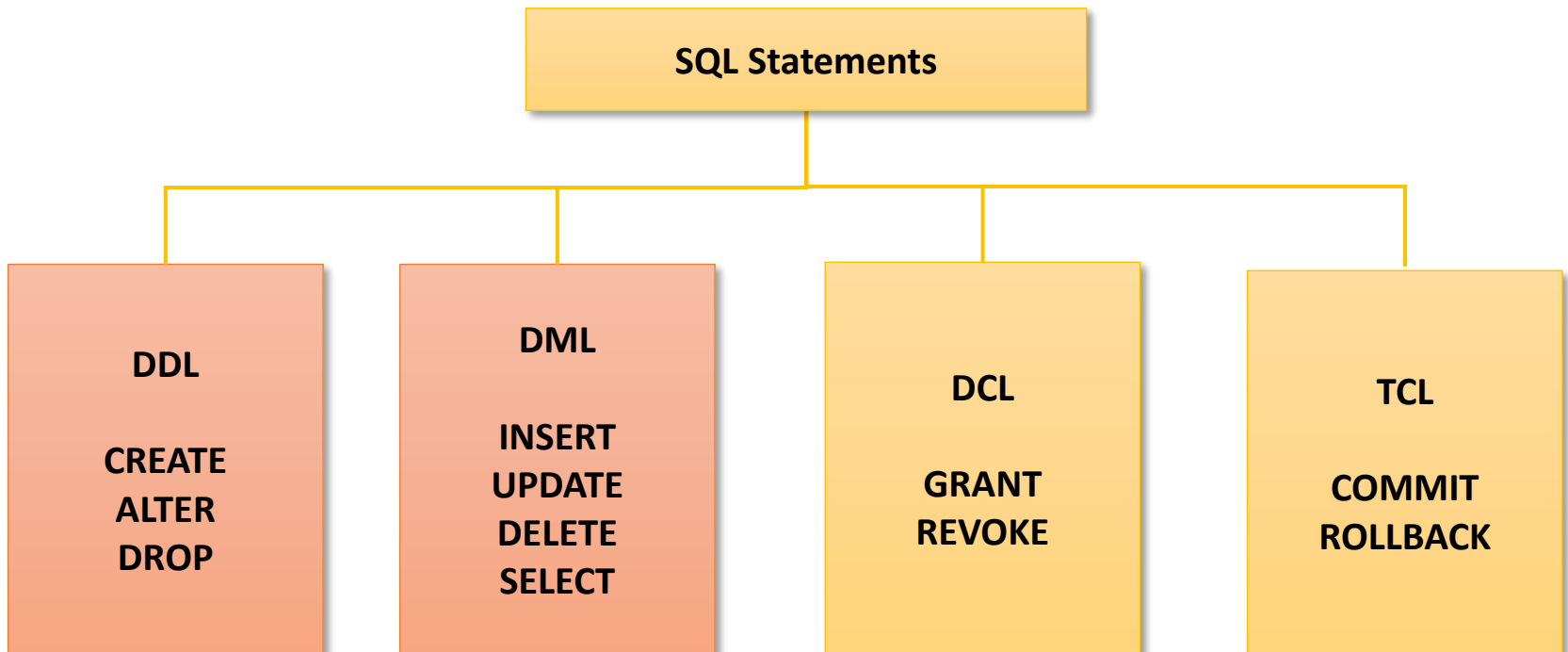
## ➤ Additional data type

- ◆ Timestamp data type Includes the DATE and TIME fields
  - Plus a minimum of six positions for decimal fractions of seconds
  - Optional WITH TIME ZONE qualifier
- ◆ DATE, TIME, and Timestamp data types can be cast or converted to string formats for comparison.



# Data Definition Language (DDL)

- It is used to **create and modify the structures of relations and other objects in the database.**
- ◆ CREATE, ALTER, DROP, TRUNCATE
  - ◆ Also, it is used to create VIEW and INDEX.



# Creating and Deleting Relations in SQL



- Deleting a table will result in **the loss of complete information stored in the table!**
- The TRUNCATE TABLE statement is used to **delete the data inside a table**, but not the table itself.

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

```
DROP TABLE table_name;
```

```
TRUNCATE TABLE table_name;
```

# Creating and Deleting Relations in SQL



- Students and Courses relations (for entity)
- Enrolled relation (for relationship)

**Students**(*sid: string, sname: string, gender: string*)  
**Courses**(*cid: string, cname: string, credits: int*)  
**Enrolled**(*sid: string, cid: string, grade: string*)



**CREATE TABLE** *Students*  
(*sid* CHAR(20),  
*sname* CHAR(20),  
*gender* CHAR(10));

**CREATE TABLE** *Courses*  
(*cid* CHAR(20),  
*cname* CHAR(20),  
*credits* INTEGER);

**CREATE TABLE** *Enrolled*  
(*sid* CHAR(20),  
*cid* CHAR(20),  
*grade* CHAR(2));



# Updating Relations in SQL

- Used to add, delete, or modify columns in an existing table
- Used to add and drop various constraints on existing tables.

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

```
ALTER TABLE table_name  
DROP COLUMN column_name;
```

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



# Data Manipulation Language (DML)

# Data Manipulation Language (DML)



## ➤ DML statements affect records in a table.

- ◆ **Inserting** new records
- ◆ **Deleting** unnecessary records
- ◆ **Updating**/modifying existing records
- ◆ **Selecting** a few records from a table

```
INSERT INTO table_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...);
```

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

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

# Manipulating Tuples in SQL

## ➤ Inserting a new tuple into a relation

```
INSERT INTO Students (sid, sname, age, gpa)  
VALUES (101, 'Alice', 21, 3.2);
```

## ➤ Deleting all tuples satisfying a specific condition

```
DELETE FROM Students  
WHERE sname = 'Alice';
```

## ➤ Updating all tuples satisfying a specific condition

```
UPDATE Students  
SET gpa = gpa - 0.1  
WHERE gpa >= 3.3;
```

# Retrieving Tuples in SQL

## ➤ Basic form of the SELECT statement

```
SELECT  [DISTINCT] <attribute list>  
FROM    <table list>  
WHERE   <condition>;
```

- ◆ <attribute list> is a list of attribute names whose values are to be retrieved by the query.
- ◆ <table list> is a list of the relation names required to process the query.
- ◆ <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.
  - Comparisons (attr *op* const or attr1 *op* attr2, where *op* is one of <, <=, >, >=, =, <>) combined with **AND**, **OR**, and **NOT**.



# Retrieving Tuples in SQL

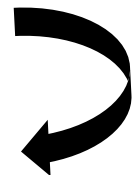
## ➤ Finding all courses whose credit is 3

```
SELECT *  
FROM Courses  
WHERE credit = 3;
```

**Courses**

<i>cid</i>	<i>cname</i>	<i>credit</i>	<i>dept</i>
101	C programming	3	CS
102	Discrete Math	2	Math
301	Databases	4	CS
302	Artificial Intelligence	3	CS
405	Data Mining	3	CS

<i>cid</i>	<i>cname</i>	<i>credit</i>	<i>dept</i>
101	C programming	3	CS
302	Artificial Intelligence	3	CS
405	Data Mining	3	CS



# Retrieving Tuples in SQL

- Finding the names of courses that are enrolled in CS

```
SELECT cname  
FROM Courses  
WHERE dept = 'CS';
```

***Courses***

<i>cid</i>	<i>cname</i>	<i>credit</i>	<i>dept</i>
101	C programming	3	CS
102	Discrete Math	2	Math
301	Databases	4	CS
302	Artificial Intelligence	3	CS
405	Data Mining	3	CS



<i>cname</i>
C programming
Databases
Artificial Intelligence
Data Mining

# Retrieving Tuples in SQL

- Finding the id and names of courses that are enrolled in CS and their credits are more than 3

```
SELECT cid, cname  
FROM Courses  
WHERE credit > 3 and dept = 'CS';
```

**Courses**

<i>cid</i>	<i>cname</i>	<i>credit</i>	<i>dept</i>
101	C programming	3	CS
102	Discrete Math	2	Math
301	Databases	4	CS
302	Artificial Intelligence	3	CS
405	Data Mining	3	CS



<i>cid</i>	<i>cname</i>
301	Databases

# Retrieving Tuples in SQL

- Finding distinct credits of courses that are enrolled in CS

```
SELECT DISTINCT credit  
FROM Courses  
WHERE dept = 'CS';
```

**Courses**

<i>cid</i>	<i>cname</i>	<i>credit</i>	<i>dept</i>
101	C programming	3	CS
102	Discrete Math	2	Math
301	Databases	4	CS
302	Artificial Intelligence	3	CS
405	Data Mining	3	CS



<i>credit</i>
3
4



# Constraints in Relational Models

# What are Constraints?

- Constraints determine which values are **permissible** and which **are not in the database**.
- **Schema-based or explicit constraints**
  - ◆ Expressed in the schema by using the facilities provided by the model.
- **Inherent or Implicit constraints**
  - ◆ These are based on the data model itself.
- **Application based or semantic constraints**
  - ◆ Beyond the expressive power of the model and must be specified and enforced by the application programs.

# Relational Constraints

- **Conditions that must hold on all tuples in a relation instance**
  - ◆ **Should be specified** when schemas are defined.
  - ◆ **They are checked** whenever relations are modified.
    - E.g. insert/delete/update operations
- **A legal relation instance satisfies all specified constraints.**
  - ◆ DBMS **should not** allow illegal instances.

***Students***

<i>sid</i>	<i>sname</i>	<i>gender</i>
100	Alice	Female
200	Bob	Male

*SID* should be unique.  
*GPA* should be non-negative.

***Courses***

<i>cid</i>	<i>cname</i>	<i>credit</i>
202	DS	3
301	DB	4

*CID* should be unique.  
*Credits* should be non-negative.

***Enrolled***

<i>sid</i>	<i>cid</i>	<i>grade</i>
100	202	A
200	301	B+

*SID* and *CID* should exist in  
*Students* and *Courses*.

# Relational Constraints



- Three main types of explicit schema-based constraints that can be expressed in the relational model as
  - ◆ **Key constraints**
  - ◆ **Entity integrity constraints**
  - ◆ **Referential integrity constraints**
  
- Another schema-based constraint is the domain constraint.
  - ◆ Every value in a tuple must be from the domain of its attribute (or **it could be null, if allowed for that attribute**).



# Common Constraints in SQL



## ➤ Constraints are used to specify rules for the data.

- ◆ Limiting the **type of data** that can go into a table
- ◆ Ensuring the **accuracy and reliability** of the data in the table
- ◆ Constraints can be a **column level** or a **table level**.

## ➤ Common constraints used in SQL

- ◆ **NOT NULL**: Ensures that a column cannot have a NULL value.
- ◆ **UNIQUE**: Ensures that all values in a column are different.
- ◆ **PRIMARY KEY**: A combination of a NOT NULL and UNIQUE.
  - **Uniquely identifies each row in a table.**
- ◆ **FOREIGN KEY**: Uniquely identifies a row/record in another table.
- ◆ **CHECK**: Ensures that all values in a column satisfies a specific condition.

# Specifying Constraints in SQL



Permissibility of NULL values    Value constraints for each column

```
CREATE TABLE table_name (  
    column1 datatype [NULL | NOT NULL] [CHECK(condition1)],  
    column2 datatype [NULL | NOT NULL] [CHECK(condition2)],  
    column3 datatype [NULL | NOT NULL] [CHECK(condition3)],  
  
    ....  
  
    PRIMARY KEY (col1, col2, ... coln)  
  
    CONSTRAINT fk_name  
    FOREIGN KEY (child_col1, child_col2, ... child_col_n)  
    REFERENCES parent_table (parent_col1, parent_col2, ... parent_col_n)  
    ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]  
);
```

Key integrity for the primary key

Referential integrity for the foreign key

# Key Constraints

- The superkey SK of relation R is a set of attributes SK of R with the following condition:
  - ◆ No two tuples in R will have the same value for SK.
  - ◆ For any distinct tuples  $t_1, t_2 \in r(R)$ ,  $t_1[SK] \neq t_2[SK]$ .
- The key K of R is a "minimal" superkey.
  - ◆ A key K is a superkey such that removal of any attribute from K results in a set of attributes that is not a superkey.
- Note: A key is a superkey but not vice versa.

# Example: Key Constraints

➤ Q: How many keys are possible?

- ◆ For any distinct tuples  $t_1, t_2 \in r(R)$ ,  $t_1[SK] \neq t_2[SK]$ .

**Courses**

<i>cid</i>	<i>cname</i>	<i>credit</i>	<i>department</i>
101	C programming	3	CS
102	Discrete Math	2	Math
301	Databases	4	CS
302	Artificial Intelligence	2	CS
405	Data Mining	5	CS

# Example: Key Constraints

➤ Q:  $\{cid, cname\}$  is a superkey?

***Courses***

<i>cid</i>	<i>cname</i>	<i>credit</i>	<i>department</i>
101	C programming	3	CS
102	Discrete Math	2	Math
301	Databases	4	CS
302	Artificial Intelligence	2	CS
405	Data Mining	5	CS



# What is the Primary Key?

- If a relation has several **candidate keys**, one is chosen arbitrarily to be the **primary key**.
  - ◆ The primary key attributes are underlined.
  
- Consider the Car relation schema.
  - ◆ **CAR**(State, Reg#, SerialNo, Make, Model, Year)
  - ◆ We chose SerialNo as the primary key.
  
- The primary key value is used to **uniquely identify** each tuple in a relation.
  - ◆ Provides the tuple identity.
  
- Also used to **reference** the tuple from another relation.

# Example: COMPANY Database Schema



## EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
-------	-------	-------	------------	-------	---------	-----	--------	-----------	-----

## DEPARTMENT

Dname	<u>Dnumber</u>	Mgr_ssn	Mgr_start_date
-------	----------------	---------	----------------

## DEPT\_LOCATIONS

<u>Dnumber</u>	<u>Dlocation</u>
----------------	------------------

## PROJECT

Pname	<u>Pnumber</u>	Plocation	Dnum
-------	----------------	-----------	------

## WORKS\_ON

<u>Essn</u>	<u>Pno</u>	Hours
-------------	------------	-------

## DEPENDENT

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
-------------	-----------------------	-----	-------	--------------

# Specifying the Primary Key in SQL



- Several candidate keys are specified using **UNIQUE** and one of which is chosen as the **primary key**.

- **Example**

- ◆ For a given student and course, there is a single grade.
- ◆ Students can take only one course, and receive a single grade for that course.
  - That is, no two students in a course receive the same grade.

```
CREATE TABLE Enrolled  
(sid CHAR(20),  
cid CHAR(20),  
grade CHAR(2),  
PRIMARY KEY(sid, cid);
```

```
CREATE TABLE Enrolled  
(sid CHAR(20),  
cid CHAR(20),  
grade CHAR(2),  
PRIMARY KEY(sid),  
UNIQUE (cid, grade));
```



# NULL Constraints

➤ **The primary key attributes PK cannot have NULL value.**

- ◆ This is because primary key values are used to identify the individual tuples.
- ◆  $t[PK] \neq NULL$  for any tuple  $t \in r(R)$ .
- ◆ If PK has several attributes, NULL is not allowed in any of these attributes.

➤ **Other attributes of  $R$  may be constrained to disallow NULL values, even though they are not in the primary key.**

```
CREATE TABLE Emp (  
    idx INT,  
    no INT NOT NULL,  
    name VARCHAR(32),  
    PRIMARY KEY (idx)  
);
```

# UNIQUE Constraints

➤ **UNIQUE**: Allow NULL but disallow duplicate values.

➤ **Example**

```
CREATE TABLE Emp (  
  idx INT,  
  no INT NULL,  
  name VARCHAR(32),  
  PRIMARY KEY (idx),  
  UNIQUE (no));
```

<i>Emp</i>		
<i>Sid</i>	<i>no</i>	<i>name</i>
1	100	Kee
2	NULL	Lee
3	NULL	Yoo

```
INSERT INTO Emp (idx, no, name) VALUES (1, 100, 'Kee');  
INSERT INTO Emp (idx, no, name) VALUES (2, NULL, 'Lee');  
INSERT INTO Emp (idx, no, name) VALUES (3, NULL, 'Yoo');
```

} **Insertions are allowed.**

# What is the Foreign Key?

- A constraint involving two relations
- Used to specify a relationship among tuples in two relations:
  - ◆ The **referencing relation** and the **referenced relation**
- Tuples in the referencing relation  $R1$  have attributes FK (called foreign key attributes) that reference the primary key attributes PK of the referenced relation  $R2$ .
  - ◆ A tuple  $t$  in  $R1$  is said to **reference** a tuple  $t'$  in  $R2$  if  $t[FK] = t'[PK]$ .
  - ◆ Displayed in a **directed arc** from  $R1.FK$  to  $R2.PK$



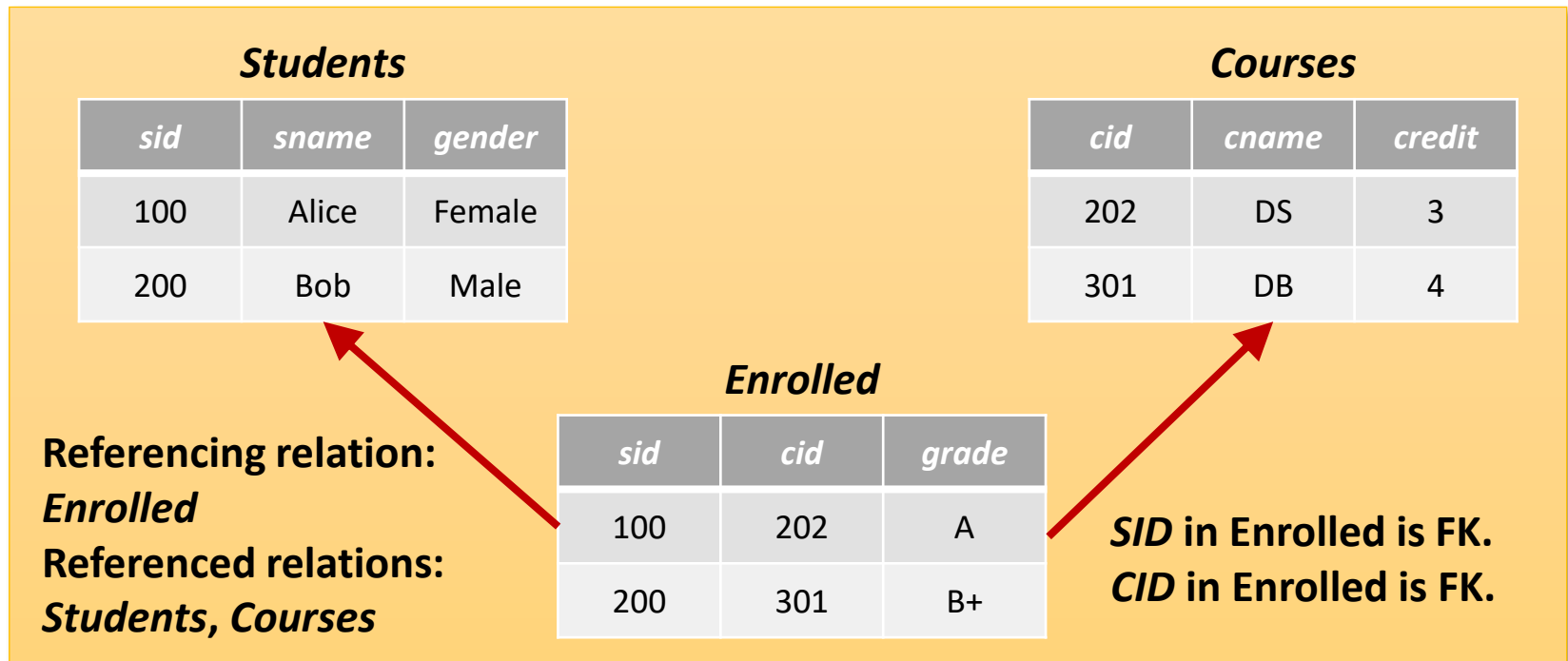
# What is the Foreign Key?

- The value in the foreign key column (or columns) FK of the referencing relation  $R1$  can be either:
  - ◆ (1) A value of **the primary key** in the **referenced relation  $R2$** ,
  - or
  - ◆ (2) A **null value**
  
- Note: In case (2), the FK in  $R1$  should not be a part of its own primary key.

# What is the Foreign Key?

## ➤ Used to specify a relationship among tuples in two relations:

- ◆ **Foreign key**: Tuples in  $R1$  have attributes that **reference** the primary key attributes PK of the relation  $R2$ .
- ◆ A tuple  $t \in r(R1)$  is said to **reference** a tuple  $t' \in r(R2)$  if  $t[FK] = t'[PK]$ .



# Example: COMPANY Database Schema



## EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
-------	-------	-------	------------	-------	---------	-----	--------	-----------	-----

## DEPARTMENT

Dname	<u>Dnumber</u>	Mgr_ssn	Mgr_start_date
-------	----------------	---------	----------------

## DEPT\_LOCATIONS

<u>Dnumber</u>	<u>Dlocation</u>
----------------	------------------

## PROJECT

Pname	<u>Pnumber</u>	Plocation	Dnum
-------	----------------	-----------	------

## WORKS\_ON

<u>Essn</u>	<u>Pno</u>	Hours
-------------	------------	-------

## DEPENDENT

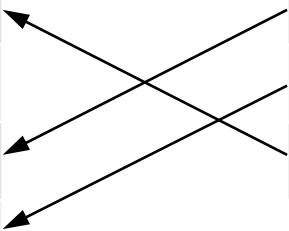
<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
-------------	-----------------------	-----	-------	--------------

# Specifying Foreign Keys in SQL

- **Constraint:** Only students listed in the Students relation should be allowed to enroll for courses.

```
CREATE TABLE Enrolled
(sid CHAR(20), cid CHAR(20), gender CHAR(10),
PRIMARY KEY (sid,cid),
FOREIGN KEY (sid) REFERENCES Students(sid),
FOREIGN KEY (cid) REFERENCES Courses(cid));
```

<i><b>Students</b></i>			<i><b>Enrolled</b></i>		
<i>sid</i>	<i>sname</i>	<i>gender</i>	<i>sid</i>	<i>cid</i>	<i>grade</i>
100	Alice	Female	300	202	A
200	Bob	Male	400	301	B+
300	Carol	Male	100	301	A+
400	David	Male			



# Enforcing Referential Integrity



- Integrity constraints should **not be violated** by the update operations.
  - ◆ Inserting/deleting/modifying tuples
  
- In case of integrity violation, several actions can be taken:
  - ◆ **Cancel the operation** that causes the violation.
  - ◆ **Trigger additional updates** so the violation is corrected (CASCADE option, SET NULL option).



# Enforcing Referential Integrity

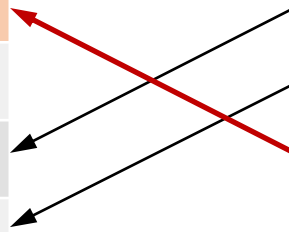


- What should be done if an Enrolled tuple with a non-existent student id is inserted? **Reject it!**
  
- What should be **done if a Students tuple is deleted?**
  - ◆ Possible options
    - **No action:** Disallow deletion of a Students tuple that is referred to.
    - **Cascade:** Delete all Enrolled tuples that refer to it.
    - **Set NULL:** Set sid in Enrolled tuples that refer to it to NULL.
  
- **Similar if primary key of Students tuple is updated.**

# Enforcing Referential Integrity

➤ What if we try to delete Alice from the Student table?

<i>Students</i>			<i>Enrolled</i>		
<i>sid</i>	<i>sname</i>	<i>gender</i>	<i>sid</i>	<i>cid</i>	<i>grade</i>
100	Alice	Female	300	202	A
200	Bob	Male	400	301	B+
300	Carol	Male	100	301	A+
400	David	Male			



➤ What happens in the Enrolled table?

- ◆ **No action:** Deleting the Alice row is rejected.
- ◆ **Cascade:** The referenced rows in Enrolled are also deleted together.
- ◆ **Set NULL:** The referenced rows in Enrolled are set as NULL.

# Example: No Action

➤ Deleting the Alice row is rejected.

*Students*

<i>sid</i>	<i>sname</i>	<i>gender</i>
100	Alice	Female
200	Bob	Male
300	Carol	Male
400	David	Male

*Enrolled*

<i>sid</i>	<i>cid</i>	<i>grade</i>
300	202	A
400	301	B+
100	301	A+



*Students*

<i>sid</i>	<i>sname</i>	<i>gender</i>
100	Alice	Female
200	Bob	Male
300	Carol	Male
400	David	Male

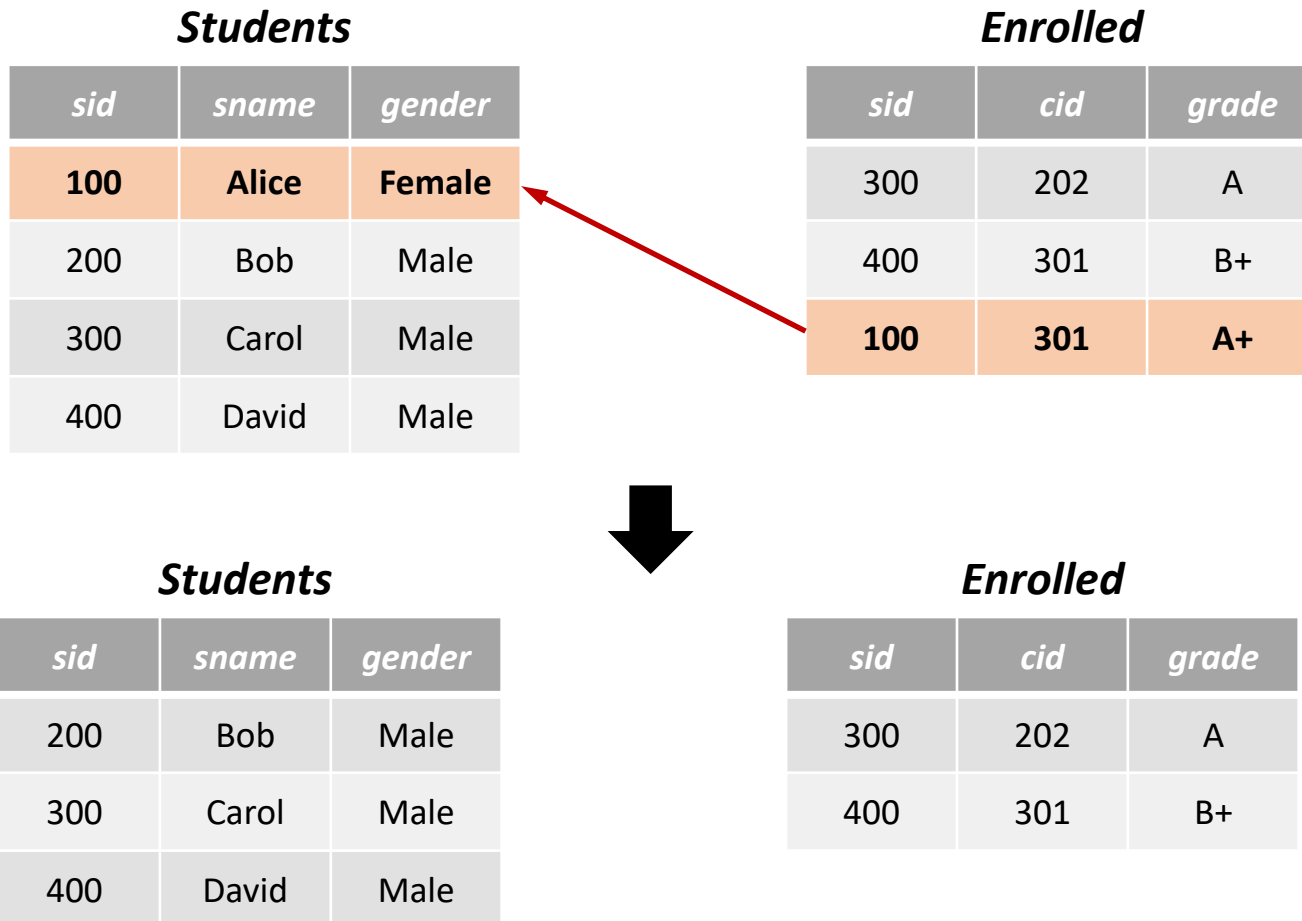
*Enrolled*

<i>sid</i>	<i>cid</i>	<i>grade</i>
300	202	A
400	301	B+
100	301	A+

# Example: Cascade

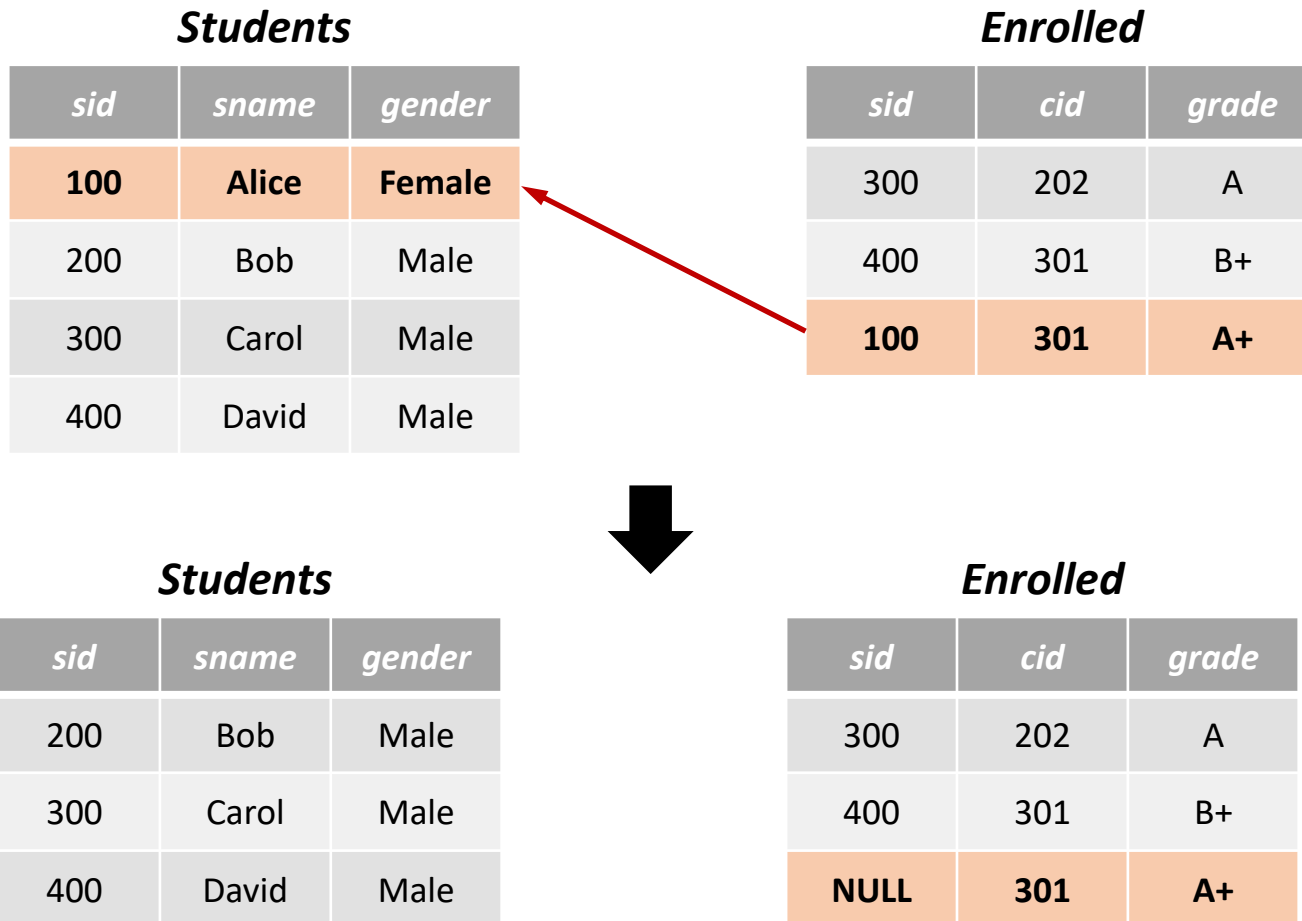


- The referenced rows in Enrolled are also deleted together.



# Example: Set NULL

- The referenced rows in Enrolled are set as NULL.



# Enforcing Referential Integrity in SQL



## ➤ Several options on deletes and updates

- ◆ Default is **NO ACTION**  
(**delete/update is rejected**)
- ◆ **CASCADE** (also delete all tuples that refer to deleted tuple)
- ◆ **SET NULL**  
(sets foreign key value of referencing tuple)

```
CREATE TABLE Enrolled  
(sid CHAR(20),  
cid CHAR(20),  
grade CHAR(2),  
PRIMARY KEY (sid,cid),  
FOREIGN KEY (sid)  
REFERENCES Students(sid)  
ON DELETE CASCADE  
ON UPDATE CASCADE);
```

# Enforcing Referential Integrity in SQL



```
CREATE TABLE products
( product_id INT PRIMARY KEY,
  product_name VARCHAR(50) NOT NULL,
  category VARCHAR(25)
);
```

```
CREATE TABLE inventory
( inventory_id INT PRIMARY KEY,
  product_id INT,
  quantity INT,
  min_level INT,
  max_level INT,
  CONSTRAINT fk_inv_product_id
  FOREIGN KEY (product_id)
  REFERENCES products (product_id)
  ON DELETE SET NULL
);
```

# CHECK Constraints



- **Used to limit the value range that can be placed in a column.**
  - ◆ **Column-level constraints:** It allows only certain values for this column.
  - ◆ **Table-level constraints:** It can limit the values in certain columns based on values in other columns in the rows.

## ➤ Example

```
CREATE TABLE Students  
(sid CHAR(20),  
sname CHAR(20),  
age INT,  
gpa REAL);
```



```
CREATE TABLE Students  
(sid CHAR(20) NOT NULL,  
sname CHAR(20) NOT NULL,  
age INT CHECK(age >= 18),  
gpa REAL CHECK(gpa >= 0.0) ,  
PRIMARY KEY (sid));
```



# Summary: Specifying Constraints in SQL



Permissibility of NULL values    Value constraints for each column

```
CREATE TABLE table_name (  
  column1 datatype [NULL | NOT NULL] [CHECK(condition1)],  
  column2 datatype [NULL | NOT NULL] [CHECK(condition2)],  
  column3 datatype [NULL | NOT NULL] [CHECK(condition3)],  
  ....  
  PRIMARY KEY (col1, col2, ... coln)  
  CONSTRAINT fk_name  
  FOREIGN KEY (child_col1, child_col2, ... child_col_n)  
  REFERENCES parent_table (parent_col1, parent_col2, ... parent_col_n)  
  ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]  
);
```

PRIMARY KEY (col1, col2, ... coln)

CONSTRAINT *fk\_name*

FOREIGN KEY (child\_col1, child\_col2, ... child\_col\_n)

REFERENCES parent\_table (parent\_col1, parent\_col2, ... parent\_col\_n)

ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]

Key integrity for the primary key

Referential integrity for the foreign key

# Possible Violations for INSERT

## ➤ Key constraint

- ◆ Check if the value of a key attribute in the new tuple already exists in another tuple in the relation.

## ➤ Domain constraint

- ◆ Check if one of the attribute values in the new tuple is not of the specified attribute domain.

## ➤ Entity integrity

- ◆ Check if the primary key value is null in the new tuple.

## ➤ Referential integrity

- ◆ Check if a foreign key value in the new tuple references a primary key value that does not exist in the referenced relation.

# Possible Violations for DELETE

- If the primary key value of the tuple being deleted is referenced from other tuples in the database,
  - ◆ Can be remedied by several actions: **RESTRICT**, **CASCADE**, **SET NULL**.
  - ◆ **RESTRICT option**: reject the deletion
  - ◆ **CASCADE option**: propagate the new primary key value into the foreign keys of the referencing tuples
  - ◆ **SET NULL option**: set the foreign keys of the referencing tuples to NULL
- One of the above options must be specified during database design for each foreign key constraint.



# Possible Violations for UPDATE

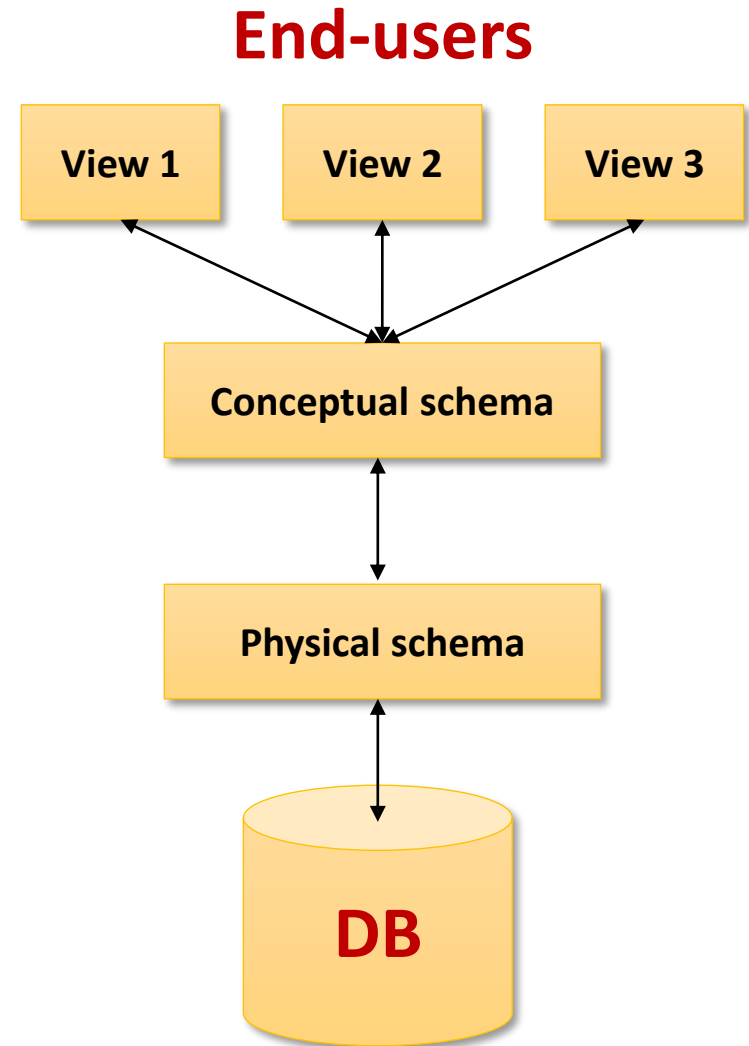
- **UPDATE may violate domain constraint and NOT NULL constraint on an attribute being modified.**
  
- **Any of the other constraints may also be violated, depending on the attribute being updated:**
  - ◆ **Updating the primary key (PK)**
    - Similar to a DELETE followed by an INSERT
    - Need to specify similar options to DELETE
  - ◆ **Updating a foreign key (FK)**
    - May violate referential integrity
  - ◆ **Updating an ordinary attribute (neither PK nor FK)**
    - Can only violate domain constraints



# Views: Virtual Relations

# Levels of Schemata

- **External schemas (or views)**
  - ◆ Describe how users see the data.
  - ◆ Provide **logical data independence**.
- **Conceptual (or logical) schema**
  - ◆ Defines logical structures.
- **Physical (or internal) schema**
  - ◆ Describes the **files** and **indexes**.
    - Relations as **unordered files**
    - Some data in sorted order (index)



# What is a View?

➤ A view is a **virtual relation**.

- ◆ Only store a **definition** rather than a set of tuples.

```
CREATE VIEW view_name AS  
SELECT column1, column2, ...  
FROM table_name  
WHERE condition;
```

➤ Views can be dropped using the **DROP VIEW** statement.

➤ View and security

- ◆ Views can be used to **present necessary information** (or a summary), while **hiding details in underlying relation(s)**.

# Views and Base Tables

- Creating a HighStudents view such that age > 21

```
CREATE VIEW HighStudents (hid, name)  
AS SELECT  S.sid, S.sname  
FROM Students S  
WHERE S.age > 21;
```

**Students**

<i>sid</i>	<i>sname</i>	<i>age</i>
100	Alice	22
200	Bob	23
300	David	15
400	Eva	12



**HighStudents**

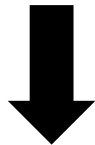
<i>hid</i>	<i>name</i>
100	Alice
200	Bob



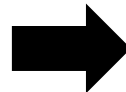
# Queries on Views

- Queries on the view is automatically rewritten by DBMS.

```
SELECT S.sid  
FROM HighStudents S;
```



```
SELECT S.sid  
FROM Students S  
WHERE S.age > 21;
```



<i>sid</i>
100
200



**Students**

<i>sid</i>	<i>sname</i>	<i>age</i>
100	Alice	22
200	Bob	23
300	David	15
400	Eva	12

# Updates on Views

## ➤ View is also another relation.

- ◆ Ideally, every updates (insert/delete/update) should be applicable.
- ◆ But, certain views are **not updatable**.

## ➤ Updatable view creation

```
CREATE OR REPLACE VIEW view_name AS  
SELECT column1, column2, ...  
FROM table_name  
WHERE condition;
```

## ➤ Deleting a view with the DROP VIEW command

```
DROP VIEW view_name;
```

# Summary



- **A tabular representation of data.**
  - ◆ Simple and intuitive, currently the most widely used.
  
- **Powerful and natural query languages exist.**
  - ◆ **Structured Query Language (SQL)**
  
- **Integrity constraints can be **specified** by the DBA, based on application semantics.**
  - ◆ DBMS **checks** for violations.
  - ◆ Two important ICs: **primary** and **foreign keys**
  - ◆ In addition, we *always* have domain constraints.

- **Chapter 5. *The Relational Data Model and Relational Database Constraints*, Fundamentals of Database systems**



# Example: COMPANY Schema



## EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
-------	-------	-------	------------	-------	---------	-----	--------	-----------	-----

## DEPARTMENT

Dname	<u>Dnumber</u>	Mgr_ssn	Mgr_start_date
-------	----------------	---------	----------------

## DEPT\_LOCATIONS

<u>Dnumber</u>	<u>Dlocation</u>
----------------	------------------

## PROJECT

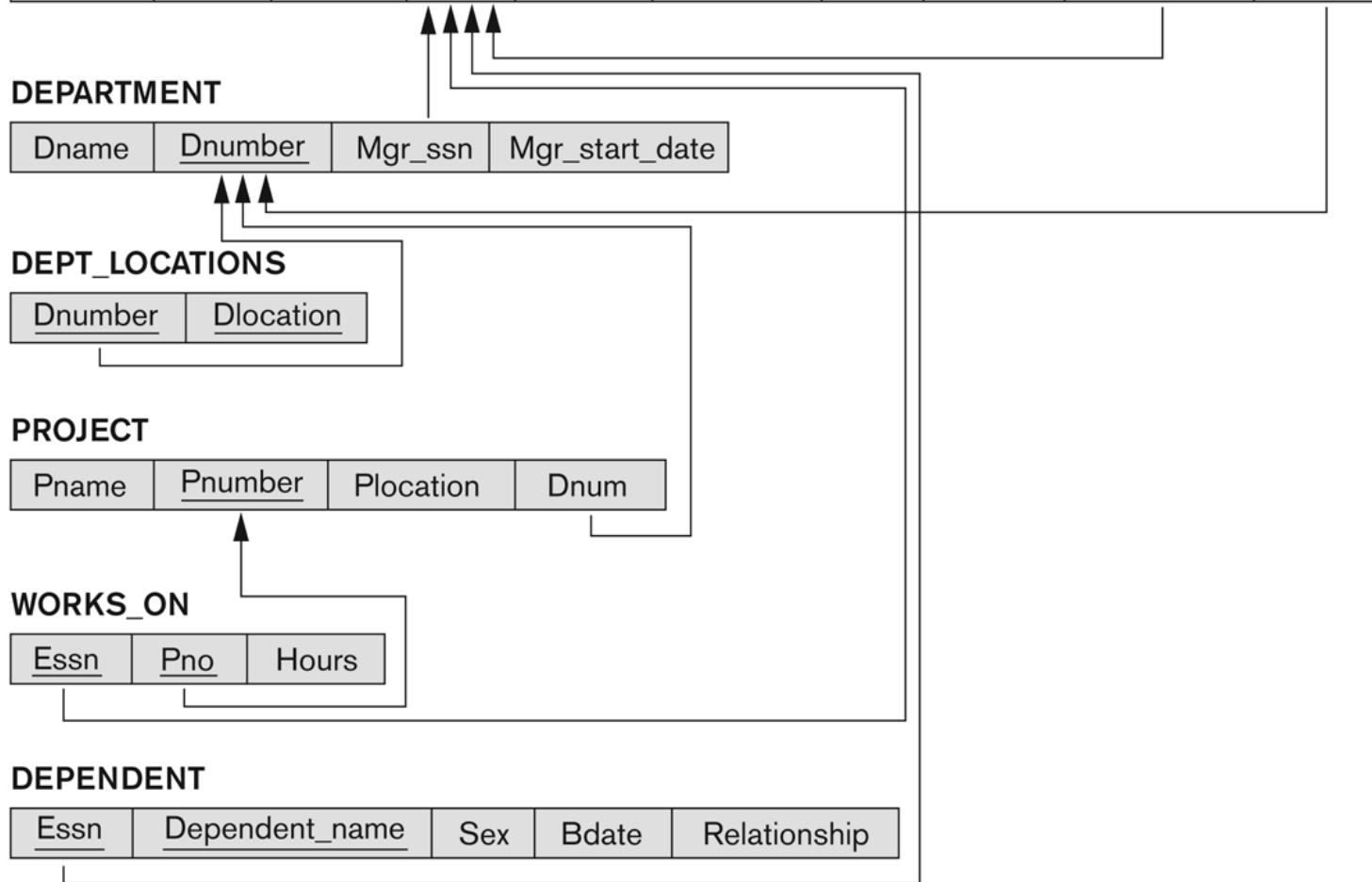
Pname	<u>Pnumber</u>	Plocation	Dnum
-------	----------------	-----------	------

## WORKS\_ON

<u>Essn</u>	<u>Pno</u>	Hours
-------------	------------	-------

## DEPENDENT

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
-------------	-----------------------	-----	-------	--------------



# Example: COMPANY Relation



**CREATE TABLE EMPLOYEE**

( Fname	VARCHAR(15)	NOT NULL,
Minit	CHAR,	
Lname	VARCHAR(15)	NOT NULL,
Ssn	CHAR(9)	NOT NULL,
Bdate	DATE,	
Address	VARCHAR(30),	
Sex	CHAR,	
Salary	DECIMAL(10,2),	
Super_ssn	CHAR(9),	
Dno	INT	NOT NULL,

**PRIMARY KEY (Ssn),**

**CREATE TABLE DEPARTMENT**

( Dname	VARCHAR(15)	NOT NULL,
Dnumber	INT	NOT NULL,
Mgr_ssn	CHAR(9)	NOT NULL,
Mgr_start_date	DATE,	

**PRIMARY KEY (Dnumber),**

**UNIQUE (Dname),**

**FOREIGN KEY (Mgr\_ssn) REFERENCES EMPLOYEE(Ssn) );**

**CREATE TABLE DEPT\_LOCATIONS**

( Dnumber	INT	NOT NULL,
Dlocation	VARCHAR(15)	NOT NULL,

**PRIMARY KEY (Dnumber, Dlocation),**

**FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );**

# Example: COMPANY Relation



**CREATE TABLE PROJECT**

( Pname	VARCHAR(15)	NOT NULL,
Pnumber	INT	NOT NULL,
Plocation	VARCHAR(15),	
Dnum	INT	NOT NULL,

**PRIMARY KEY** (Pnumber),

**UNIQUE** (Pname),

**FOREIGN KEY** (Dnum) **REFERENCES** DEPARTMENT(Dnumber) );

**CREATE TABLE WORKS\_ON**

( Essn	CHAR(9)	NOT NULL,
Pno	INT	NOT NULL,
Hours	DECIMAL(3,1)	NOT NULL,

**PRIMARY KEY** (Essn, Pno),

**FOREIGN KEY** (Essn) **REFERENCES** EMPLOYEE(Ssn),

**FOREIGN KEY** (Pno) **REFERENCES** PROJECT(Pnumber) );

**CREATE TABLE DEPENDENT**

( Essn	CHAR(9)	NOT NULL,
Dependent_name	VARCHAR(15)	NOT NULL,
Sex	CHAR,	
Bdate	DATE,	
Relationship	VARCHAR(8),	

**PRIMARY KEY** (Essn, Dependent\_name),

**FOREIGN KEY** (Essn) **REFERENCES** EMPLOYEE(Ssn) );

# Example: COMPANY Relation



## ➤ Adding referential integrity

```
CREATE TABLE EMPLOYEE
(
    ... ,
    Dno          INT          NOT NULL          DEFAULT 1,
    CONSTRAINT EMPPK
        PRIMARY KEY (Ssn),
    CONSTRAINT EMPSUPERFK
        FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn)
            ON DELETE SET NULL          ON UPDATE CASCADE,
    CONSTRAINT EMPDEPTFK
        FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber)
            ON DELETE SET DEFAULT       ON UPDATE CASCADE);

CREATE TABLE DEPARTMENT
(
    ... ,
    Mgr_ssn CHAR(9)          NOT NULL          DEFAULT '888665555',
    ... ,
    CONSTRAINT DEPTPK
        PRIMARY KEY (Dnumber),
    CONSTRAINT DEPTSK
        UNIQUE (Dname),
    CONSTRAINT DEPTMGRFK
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn)
            ON DELETE SET DEFAULT       ON UPDATE CASCADE);

CREATE TABLE DEPT_LOCATIONS
(
    ... ,
    PRIMARY KEY (Dnumber, Dlocation),
    FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber)
        ON DELETE CASCADE              ON UPDATE CASCADE);
```



# Example: COMPANY Relation



## EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

## DEPARTMENT

Dname	<u>Dnumber</u>	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

## DEPT\_LOCATIONS

<u>Dnumber</u>	<u>Dlocation</u>
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

# Example: COMPANY Relation



**WORKS\_ON**

<u>Essn</u>	<u>Pno</u>	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

**PROJECT**

<u>Pname</u>	<u>Pnumber</u>	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

**DEPENDENT**

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

# In-Class Exercise

- Consider the following relations for a database that keeps track of student enrollment in courses and the books adopted for each course:
  - ◆ STUDENT(SSN, Name, Major, Bdate)
  - ◆ COURSE(Course#, Cname, Dept)
  - ◆ ENROLL(SSN, Course#, Quarter, Grade)
  - ◆ BOOK\_ADOPTION(Course#, Quarter, Book\_ISBN)
  - ◆ TEXT(Book\_ISBN, Book\_Title, Publisher, Author)
  
- Draw a **relational schema diagram specifying the foreign keys** for this schema.