MODULE 2: Relational Model

PREPARED BY
SHARIKA T R, SNGCE

SYLLABUS

- Structure of Relational Databases - Integrity Constraints, Synthesizing ER diagram to relational schema

- Introduction to Relational Algebra - select, project, cartesian product operations, join - Equi-join, natural join. query examples,

- Introduction to Structured Query Language (SQL), Data Definition Language (DDL), Table definitions and operations – CREATE, DROP, ALTER, INSERT, DELETE, UPDATE.

Relational data model

- Represent database as a collection of relations
- Relation is a table which has values and rows in table is a collection of related data values
- Each row in table is a fact
- Row in relational table is called a tuple, column header is attribute and table is a relation

Attributes

Relation name

EMPLOYEE

EMP_NO	Name	Address	Mobile number	Age	Salary
101	RAM	XYZ	9898989898	20	10000
102	SAM	CVF	9999999999	21	20000
103	SITA	FDFD	888888888	22	30000

Tuples

- In the relational model, all data is logically structured within **relations** (also called **table**)
- Informally a relation may be viewed as a named two-dimensional table representing an entity set.
- A relation has a fixed number of named columns and variable number of rows.

Components of relational database

- The main components of relational database structure are as follows:
1. **Domains**
2. **Tuples (rows)**
3. **Columns**
4. **Keys**
5. **Relations (Tables)**

Domain

- It has three parts
  - Name
  - Data type
  - Format
- A Domain is a set of atomic values.
- Atomic means each value in the domain is indivisible to the relational model.

- A **domain** has a logical definition: e.g."USA_phone_numbers" are the set of 10 digit phone numbers valid in the U.S.

- A domain may have 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. E.g., Dates have various formats such as month name, date, year or yyyy-mm-dd, or dd mm,yyyy etc

Figure 2.2 shows the structure of an instance or extension, of a relation called EMPLOYEE. The EMPLOYEE relation has six attributes (field items), namely EMP-NO, LAST-NAME, FIRST-NAME, DATA-OF-BIRTH, SEX, TEL-NO and SALARY. The extension has seven tuples (records). Each attribute contains values drawn from a particular domain.
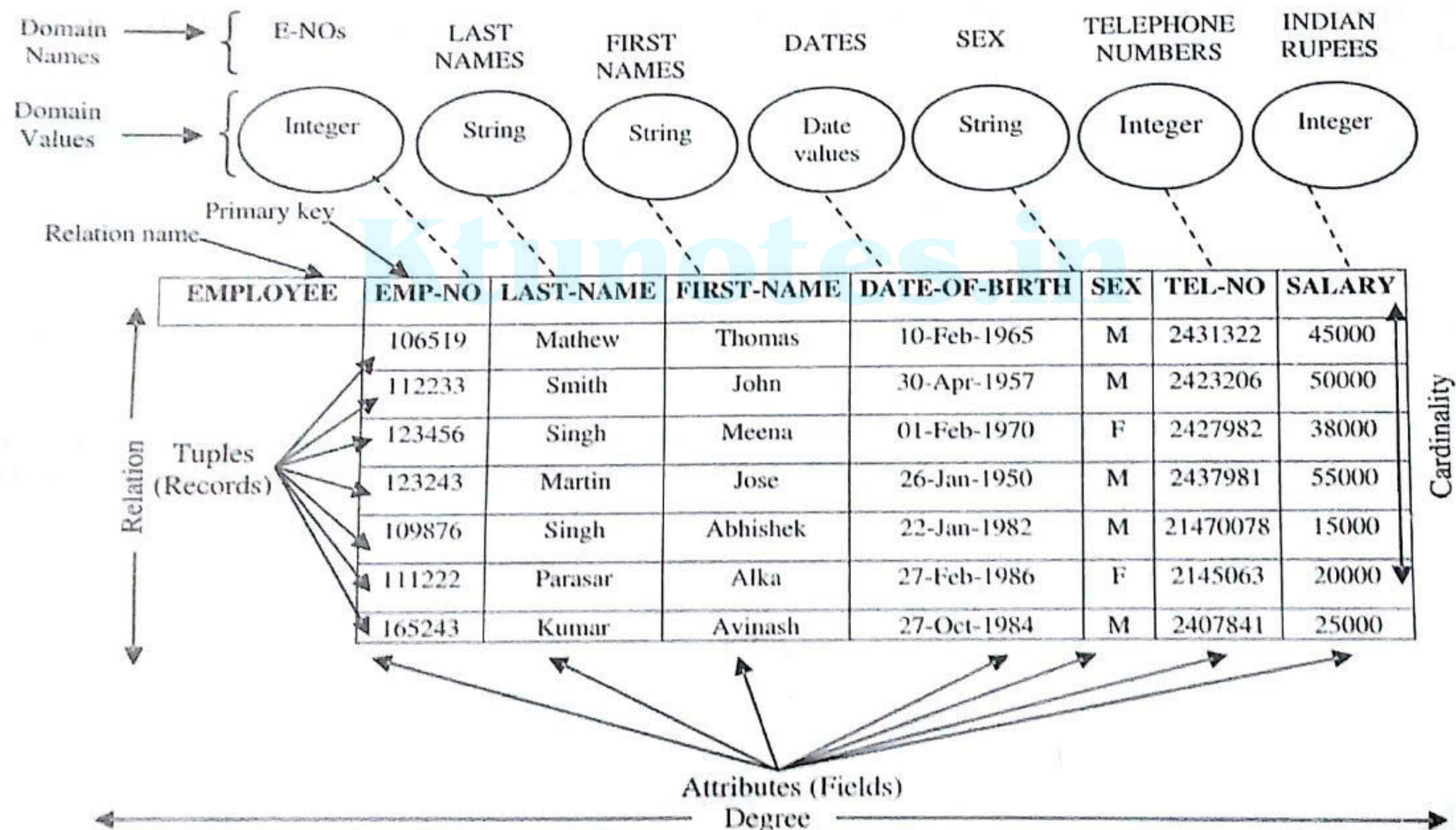

Figure 2.2: EMPLOYEE Relation

Tuples (rows)

- A tuple is an ordered set of values
- Tuple is a portion of a table containing data that described only entity, relationship, or object
- Also known as record
- Each value is derived from an appropriate domain.

Scheme Name

Customer

CFirstName	CLastName	CStreet	CZipCode	CPhone
Kumar	Singh	52/57 store	223001	9889898989

Attribute
Names →

Row (tuple)

Data Cell
(Domain value assigned to attribute Name)

Ktunotes.in

- **<Kumar, Singh, 52/57 store, 223001, 9889898989>** is a tuple belonging to the CUSTOMER relation.

Columns

- Columns in a table are also called **attributes** or **fields** of the relation.
- A single cell in a table called field value, attribute value or data element.
- For example, for the entity person, attributes could include eye colour and height.

Person	Eye Color	Height	Phone No.
Kumar	Gray	6	9889898989

Attribute Name

Person ← Scheme Name

Column (attribute) or → Field

Field Value

Key of a Relation

- Each row has a value of a data item (or set of items) that uniquely identifies that row in the table
    - Called 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*

PREPARED BY
SHARIKA T R, SNGCE

Relations (Tables)

- A table of values
- A relation may be thought of as a **set of rows**.
- A relation may alternately be thought of as a **set of columns**.
- That is a table is perceived as a two-dimensional structure composed of rows and columns.
- Each row represents a fact that corresponds to a real-world **entity** or **relationship**.
- Each row has a value of an item or set of items that uniquely identifies that row in the table

- Sometimes row-ids or sequential numbers are assigned to identify the rows in the table.
- Each column typically is called by its column name or column header or attribute name.

Schema of a Relation

- It is basically an outline of how data is organized
- It is denoted by *R* (A1, A2, …..An)
  - Here R is relation name and
  - it has some attributes A1 to An
- Each attribute have some domain and it is represented by dom(Ai)
- Relation schema is used to describe a relation and R is name of the relation
- Each attribute has a **domain** or a set of valid values.
  - For example, the domain of Cust-id is 6 digit numbers.

Degree of a relation

- Degree of a relation is number of attributes in a relation
- Eg
- STUDENT(Id, Name, Age, Departmentno)
  - Has degree 4
- Using datatype of each the definition can be written as
- STUDENT(Id:Integer, Name:String,Age:integer, Departmentno:integer)

Relation State

- The **relation state** is a subset of the Cartesian product of the domains of its attributes
  - each domain contains the set of all possible values the attribute can take.
- Example: attribute Cust-name is defined over the domain of character strings of maximum length 25
  - dom(Cust-name) is varchar(25)
- The role these strings play in the CUSTOMER relation is that of the *name of a customer*.
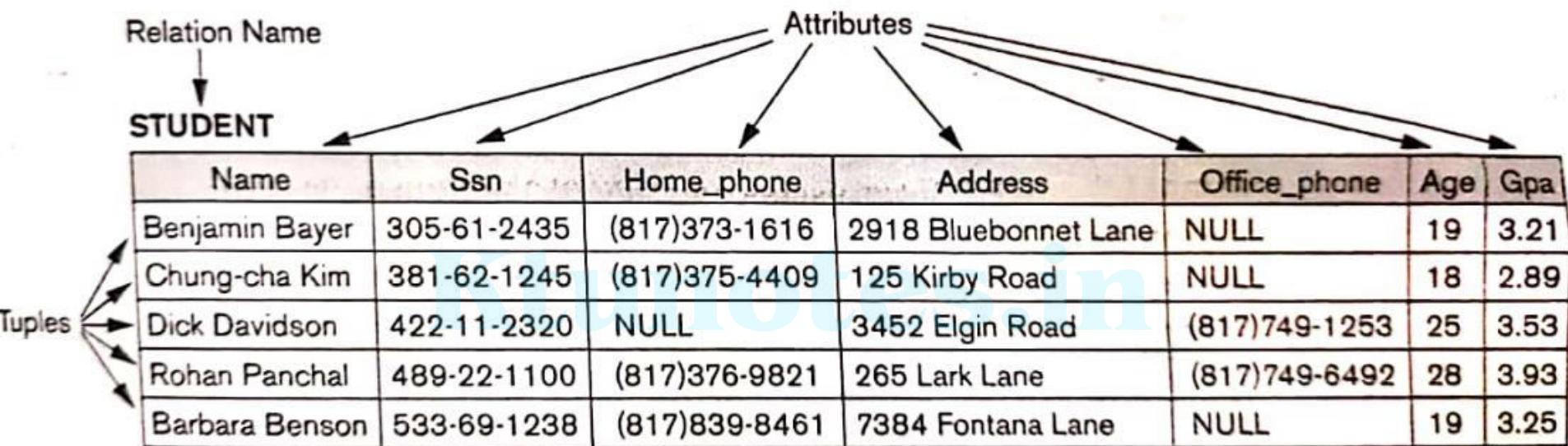
Figure 3.1

The attributes and tuples of a relation STUDENT.

- A relation state r(R) is a mathematical relation of degree n on the domains dom(A1), dom(A2)..., dom(An) which is a subset of Cartesian product(X) of domains that define R

$$r(R) \subseteq (dom(A1) X\, dom(A2)\, X\, ...X\, dom(An))$$

- Cartesian product specifies all possible combination of values from underlying domains

- Cardinality in domain D by |D| then the total number of tuples in cartesiean product is

$$|dom(A1)| \ x \ |dom(A2)| \ x \ldots x \ |dom(An)|$$

Current relation state

- Reflects only the valid tuples that represent a particular state of real world
- As the state of real world changes, so does the relation state, by being transformed into another relation state

Alternative Definition of a Relation

- ordering of values in a tuple *unnecessary*
- a relation schema *R = {A1, A2, ..., An} is a set of attributes, and*
- *a relation state r(R) is a finite set of mappings r = {t1, t2, ..., tm}, where each tuple ti is a **mapping from R to D, and***
- ***D is the union*** (denoted by ∪) of the attribute domains; that is, *D = dom(A1) ∪ dom(A2) ∪ ... ∪* dom(*An*).
- *Here t[Ai] must be in dom(Ai) for 1 ≤ i ≤ n for each mapping t in r.*
- *Each mapping ti is called a tuple.*

- a **tuple can be considered as a set** of (<attribute>, <value>) pairs, where each pair gives the value of the mapping from an attribute *Ai to a value vi from dom(Ai).*

- *The ordering of attributes is not* important, because the *attribute name appears with its value.*

Formal Definitions - Summary

- Formally,
  - Given R(A1, A2, ..., An)
  - r(R) $\subset$ dom (A1) X dom (A2) X ....X dom(An)
- R(A1, A2, ..., An) is the **schema** of the relation
- R is the **name** of the relation
- A1, A2, ..., An are the **attributes** of the relation
- r(R): a specific **state** (or "value" or "population") of relation R – this is a *set of tuples* (rows)
  - r(R) = {t1, t2, ..., tn} where each ti is an n-tuple
  - ti = <v1, v2, ..., vn> where each vj *element-of* dom(Aj)

Formal Definitions - Example

- Let R(A1, A2) be a relation schema:
    - Let dom(A1) = {0,1}
    - Let dom(A2) = {a,b,c}
- Then: dom(A1) X dom(A2) is all possible combinations:
  {<0,a> , <0,b> , <0,c>, <1,a>, <1,b>, <1,c> }

- The relation state r(R) ⊂ dom(A1) X dom(A2)
- For example: r(R) could be {<0,a> , <0,b> , <1,c> }
    - this is one possible state (or "population" or "extension") r
      of the relation R, defined over A1 and A2.
    - It has three 2-tuples: <0,a> , <0,b> , <1,c>

Definition Summary

Informal Terms		Formal Terms
Table		Relation
Column Header		Attribute
All possible Column Values		Domain
Row		Tuple
Table Definition		Schema of a Relation
Populated Table		State of the Relation

Characteristics of Relation

- 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.
- Ordering of attributes in a relation schema R (and of values within each tuple):
  - We will consider the attributes in R(A1, A2, ..., An) and the values in t=<v1, v2, ..., vn> to be ordered .
    - (However, a more general alternative definition of relation does not require this ordering).

Figure 5.2

The relation STUDENT from Figure 5.1 with a different order of tuples.

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21

- Values in a tuple:
  - All values are considered atomic (indivisible).
  - Each value in a tuple must be from the domain of the attribute for that column
    - If tuple t = <v1, v2, ..., vn> is a tuple (row) in the relation state r of R(A1, A2, ..., An)
    - Then each vi must be a value from $dom(Ai)$

  - A special **null** value is used to represent values that are unknown or inapplicable to certain tuples.

- Notation:
  - We refer to **component values** of a tuple t by:
    - t[Ai] or t.Ai
    - This is the value vi of attribute Ai for tuple t
  - Similarly, t[Au, Av, ..., Aw] refers to the subtuple of t containing the values of attributes Au, Av, ..., Aw, respectively in t

Relational Integrity Constraints

- Constraints are **conditions** that must hold on **all** valid relation states.
- There are three *main types* of constraints in the relational model:
  - **Key** constraints
  - **Entity integrity** constraints
  - **Referential integrity** constraints
- Another implicit 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)

Key Constraints

- **Superkey** of R:
  - Is a set of attributes SK of R with the following condition:
    - No two tuples in any valid relation state r(R) will have the same value for SK
    - That is, for any distinct tuples t1 and t2 in r(R), t1[SK] ≠ t2[SK]
    - This condition must hold in *any valid state r(R)*

- **Key** of R:
  - A "minimal" superkey
  - That is, a key is a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey (does not possess the superkey uniqueness property)

- Example: Consider the CAR relation schema:
  - CAR(State, Reg#, SerialNo, Make, Model, Year)
  - CAR has two keys:
    - Key1 = {State, Reg#}
    - Key2 = {SerialNo}
  - Both are also superkeys of CAR
  - {SerialNo, Make} is a superkey but *not* a key.

- In general:
  - Any *key* is a *superkey* (but not vice versa)
  - Any set of attributes that *includes a key* is a *superkey*
  - A *minimal* superkey is also a key

- If a relation has several **candidate keys**, one is chosen arbitrarily to be the **primary key**.
  - The primary key attributes are underlined.
- Example: 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 tuple
  - General rule: Choose as primary key the smallest of the candidate keys (in terms of size)
  - Not always applicable – choice is sometimes subjective

CAR table with two candidate keys – LicenseNumber chosen as Primary Key

CAR

License_number	Engine_serial_number	Make	Model	Year
Texas ABC-739	A69352	Ford	Mustang	02
Florida TVP-347	B43696	Oldsmobile	Cutlass	05
New York MPO-22	X83554	Oldsmobile	Delta	01
California 432-TFY	C43742	Mercedes	190-D	99
California RSK-629	Y82935	Toyota	Camry	04
Texas RSK-629	U028365	Jaguar	XJS	04

Figure 5.4

The CAR relation, with two candidate keys: License_number and Engine_serial_number.

Relational Database Schema

- **Relational Database Schema:**
  - A set S of relation schemas that belong to the same database.
  - S is the name of the whole **database schema**
  - S = {R1, R2, ..., Rn}
  - R1, R2, ..., Rn are the names of the individual **relation schemas** within the database S
- Following slide shows a COMPANY database schema with 6 relation schemas

PREPARED BY
SHARIKA T R, SNGCE

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Figure 5.5

Schema diagram for the COMPANY relational database schema.

Entity Integrity

- **Entity Integrity:**
  - The *primary key attributes* PK of each relation schema R in S cannot have null values in any tuple of r(R).
    - This is because primary key values are used to *identify* the individual tuples.
    - t[PK] ≠ null for any tuple t in r(R)
    - If PK has several attributes, null is not allowed in any of these attributes
  - Note: Other attributes of R may be constrained to disallow null values, even though they are not members of the primary key.

Referential Integrity

- A constraint involving **two** relations
  - ▫ The previous constraints involve a single relation.
- 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 t1 in R1 is said to **reference** a tuple t2 in R2 if t1[FK] = t2[PK].
- A referential integrity constraint can be displayed in a relational database schema as a directed arc from R1.FK to R2.

- Statement of the constraint
  - The value in the foreign key column (or columns) FK of the the **referencing relation** R1 can be **either**:
    - (1) a value of an existing primary key value of a corresponding primary key PK in the **referenced relation** R2, <u>or</u>
    - (2) a **null**.
- In case (2), the FK in R1 should **not** be a part of its own primary key.

Displaying a relational database schema and its constraints

- Each relation schema can be displayed as a row of attribute names
- The name of the relation is written above the attribute names
- The primary key attribute (or attributes) will be underlined
- A foreign key (referential integrity) constraints is displayed as a directed arc (arrow) from the foreign key attributes to the referenced table
  - Can also point the the primary key of the referenced relation for clarity
- Next slide shows the COMPANY **relational schema diagram**

Figure 5.7

Referential integrity constraints displayed on the COMPANY relational database schema.

Other Types of Constraints

- Semantic Integrity Constraints:
  - based on application semantics and cannot be expressed by the model per se
  - Example: "the max. no. of hours per employee for all projects he or she works on is 56 hrs per week"
- A **constraint specification** language may have to be used to express these
- SQL-99 allows triggers and **ASSERTIONS** to express for some of these

Populated database state

- Each *relation* will have many tuples in its current relation state
- The *relational database state* is a union of all the individual relation states
- Whenever the database is changed, a new state arises
- Basic operations for changing the database:
  - INSERT a new tuple in a relation
  - DELETE an existing tuple from a relation
  - MODIFY an attribute of an existing tuple
- Next slide shows an example state for the COMPANY database

EMPLOYEE

Fname	Minit	Lname	Ssn	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	Dnumber	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

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	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

Pname	Pnumber	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

Essn	Dependent_name	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

PREPARED BY
SHARIKA T R, SNGCE

Update Operations on Relations

- INSERT a tuple.
- DELETE a tuple.
- MODIFY a tuple.
- Integrity constraints should not be violated by the update operations.
- Several update operations may have to be grouped together.
- Updates may **propagate** to cause other updates automatically. This may be necessary to maintain integrity constraints.

Update Operations on Relations

- In case of integrity violation, several actions can be taken:
  - Cancel the operation that causes the violation (RESTRICT or REJECT option)
  - Perform the operation but inform the user of the violation
  - Trigger additional updates so the violation is corrected (CASCADE option, SET NULL option)
  - Execute a user-specified error-correction routine

Possible violations for each operation

- INSERT may violate any of the constraints:
  - Domain constraint:
    - if one of the attribute values provided for the new tuple is not of the specified attribute domain
  - Key constraint:
    - if the value of a key attribute in the new tuple already exists in another tuple in the relation
  - Referential integrity:
    - if a foreign key value in the new tuple references a primary key value that does not exist in the referenced relation
  - Entity integrity:
    - if the primary key value is null in the new tuple

- DELETE may violate only referential integrity:
  - 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 (see Chapter 8 for more details)
      - 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

- 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

homework

- 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.**

Introduction to Relational Algebra

- select,
- project,
- cartesian product operations,
- join - Equi-join,
- natural join
- query examples

PREPARED BY
SHARIKA T R, SNGCE

Relational algebra

- Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output.
- It uses operators to perform queries. An operator can be either **unary** or **binary**.
- They accept relations as their input and yield relations as their output.
- Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

- The basic set of operations for the relational model is the relational algebra.
- These operations enable a user to specify basic retrieval requests as relational algebra expressions.
- The result of a retrieval is a new relation, which may have been formed from one or more relations.
- The algebra operations thus produce new relations, which can be further manipulated using operations of the same algebra.
- A sequence of relational algebra operations forms a relational algebra expression, whose result will also be a relation that represents the result of a database query

Relational Calculus

- Relational calculus is a non-procedural query language, and instead of algebra, it uses mathematical predicate calculus.
- When applied to databases, it is found in two forms.
  - Tuple relational calculus which was originally proposed by Codd in the year 1972 and
  - Domain relational calculus which was proposed by Lacroix and Pirotte in the year 1977
- In first-order logic or predicate calculus, a predicate is a truth-valued function with arguments. When we replace with values for the arguments, the function yields an expression, called a proposition, which will be either true or false.

Unary Relational Operations:

1. **SELECT and**
2. **PROJECT**

The SELECT Operation

- The SELECT operation is used to choose a *subset of the tuples from a relation that* satisfies a **selection condition**

- SELECT operation *restricts the tuples in a relation to only* those tuples that satisfy the condition

- *horizontal partition of the relation into two sets of tuples*

- those tuples that satisfy the condition and are selected, and those tuples that do not satisfy the conditionand are discarded.

- Select the EMPLOYEE tuples whose department is 4, or those whose salary is greater than $30,000

$$\sigma_{Dno=4}(\text{EMPLOYEE})$$
$$\sigma_{Salary>30000}(\text{EMPLOYEE})$$

- In general, the SELECT operation is denoted by

$$\sigma_{<\text{selection condition}>}(R)$$

- *R is generally a relational algebra expression whose result is a* relation—the simplest such expression is just the name of a database relation.
- Therelation resulting from the SELECT operation has the *same attributes as R*.

- The Boolean expression specified in <selection condition> is made up of a number of **clauses of the form**

    <attribute name> <comparison op> <constant value>

or

    <attribute name> <comparison op> <attribute name>

- Clauses can be connected by the standard Boolean operators *and, or, and not to form a general selection condition*

- Select the tuples for all employees who either work in department 4 and make over $25,000 per year, or work in department 5 and make over $30,000,

$$\sigma_{(Dno=4 \text{ AND } Salary>25000) \text{ OR } (Dno=5 \text{ AND } Salary>30000)}(EMPLOYEE)$$

(a)

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
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

- the result of a SELECT operation can be determined as follows:
- The <selection condition> is applied independently to each *individual tuple t in R.*
- *This* is done by substituting each occurrence of an attribute *Ai in the selection condition* with its value in the tuple *t[Ai].*
- *If the condition evaluates to TRUE, then tuple t is* selected.
- All the selected tuples appear in the result of the SELECT operation.

Interpretation of Boolean conditions AND, OR, and NOT

- **(cond1 AND cond2) is TRUE**
  - if both (cond1) and (cond2) are TRUE;
  - otherwise, it is FALSE.
- **(cond1 OR cond2) is TRUE**
  - if either (cond1) or (cond2) or both are TRUE;
  - otherwise, it is FALSE.
- **(NOT cond) is TRUE**
  - if cond is FALSE;
  - otherwise, it is FALSE.

Degree of the relation from SELECT operation

- its number of attributes
- is the same as the degree of R.
- *The* number of tuples in the resulting relation is always *less than or equal to the number* of tuples in R.

$$|\sigma_c (R)| \leq |R|$$

- The fraction of tuples selected by a selection condition is referred to as the **selectivity of the condition.**

Commutative Property of SELECT

- SELECT operation is **commutative**

$$\sigma_{<cond1>}(\sigma_{<cond2>}(R)) = \sigma_{<cond2>}(\sigma_{<cond1>}(R))$$

- a sequence of SELECTs can be applied in any order.

- we can always combine a **cascade (or sequence) of SELECT operations into a single** SELECT operation with a conjunctive (AND) condition; that is,

$$\sigma_{<cond1>}(\sigma_{<cond2>}(...(\sigma_{<condn>}(R))...)) = \sigma_{<cond1> \text{ AND} <cond2> \text{ AND...AND } <condn>}(R)$$

PROJECT Operation

- **PROJECT operation** selects certain *columns from the table and discards the other columns.*
- If we are interested in only certain attributes of a relation, we use the PROJECT operation to *project the relation over these attributes only*
- *vertical partition of the relation into two relations*
  - one has the needed columns (attributes) and contains the result of the operation, and
  - the other contains the discarded columns.

The general form of PROJECT

$$\pi_{<\text{attribute list}>}(R)$$

- Example: list each employee's first and last name and salary

$$\pi_{\text{Lname, Fname, Salary}}(\text{EMPLOYEE})$$

- The result of the PROJECT operation has only the attributes specified in <attribute list> *in the same order as they appear in the list.*

- *Hence, its **degree is equal to the number of attributes** in <attribute list>.*

Duplicate Elimination in PROJECT

- The PROJECT operation *removes any duplicate tuples, so the result of the* PROJECT operation is a set of distinct tuples, and hence a valid relation.
- This is known as **duplicate elimination.**

Example

$$\pi_{Sex, Salary}(EMPLOYEE)$$

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

EMPLOYEE

Fname	Minit	Lname	Ssn	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

- The number of tuples in a relation resulting from a PROJECT operation is always less than or equal to the number of tuples in R.

$$\pi_{<list1>} (\pi_{<list2>} (R)) = \pi_{<list1>} (R)$$

- as long as <list2> contains the attributes in <list1>;
- otherwise, the left-hand side is an incorrect expression.
- commutativity *does not hold on* PROJECT.

EMPLOYEE

Fname	Minit	Lname	Ssn	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

$\sigma_{\text{(Dno=4 AND Salary>25000) OR (Dno=5 AND Salary>30000)}}$ (EMPLOYEE).

(a)

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
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

(b) $\pi_{\text{Lname, Fname, Salary}}$(EMPLOYEE).

(c) $\pi_{\text{Sex, Salary}}$(EMPLOYEE).

(b)

Lname	Fname	Salary
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

(c)

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

Sequences of Operations and the RENAME Operation

- for most queries, we need to apply several relational algebra operations one after the other
  - Either we can write the operations as a single relational algebra expression by nesting the operations, or
  - we can apply one operation at a time and create intermediate result relations
    - we must give names to the relations that hold the intermediate results.

Example- In-Line relational algebra expression

- retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a SELECT and a PROJECT operation

$$\pi_{\text{Fname, Lname, Salary}}(\sigma_{\text{Dno}=5}(\text{EMPLOYEE}))$$

(a)

Fname	Lname	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

Example- Intermediate Relation

$$DEP5\_EMPS \leftarrow \sigma_{Dno=5}(EMPLOYEE)$$
$$RESULT \leftarrow \pi_{Fname, Lname, Salary}(DEP5\_EMPS)$$

PREPARED BY
SHARIKA T R, SNGCE

Rename

- To rename the attributes in a relation, we simply list the new attribute names in parentheses,

$$TEMP \leftarrow \sigma_{Dno=5}(EMPLOYEE)$$

$$R(First\_name, Last\_name, Salary) \leftarrow \pi_{Fname, Lname, Salary}(TEMP)$$

(b)

TEMP

Fname	Minit	Lname	Ssn	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
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

R

First_name	Last_name	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

- If no renaming is applied, the names of the attributes in the resulting relation of a SELECT operation are the same as those in the original relation and in the same order.

- a PROJECT operation with no renaming, the resulting relation has the same attribute names as those in the projection list and in the same order in which they appear in the list.

RENAME Operation General form

- RENAME operation when applied to a relation *R of degree n is denoted by any of the* following three forms

$$\rho_{S(B1,\ B2,\ ...,\ Bn)}(R) \quad or \quad \rho_S(R) \quad or \quad \rho_{(B1,\ B2,\ ...,\ Bn)}(R)$$

- symbol ρ (rho) is used to denote the RENAME operator,
- *S is the new relation* name, and
- *B1, B2, ..., Bn are the new attribute names.*
  - *The first expression* renames both the relation and its attributes,
  - the second renames the relation only, and
  - the third renames the attributes only.
- If the attributes of *R are (A1, A2, ..., An) in* that order, then each *Ai is renamed as Bi.*

The UNION, INTERSECTION, and MINUS Operations

- UNION:
  - The result of this operation, denoted by $R \cup S$, *is a relation that* includes all tuples that are either in *R or in S or in both R and S.*
  - *Duplicate* tuples are eliminated.
- INTERSECTION:
  - The result of this operation, denoted by $R \cap S$, *is a relation* that includes all tuples that are in both *R and S.*
- SET DIFFERENCE (or MINUS):
  - The result of this operation, denoted by $R - S$, *is a relation that includes all tuples that are in R but not in S.*

Example

- retrieve the Social Security numbers of all employees who either work in department 5 or directly supervise an employee who works in department 5, we can use the UNION operation as follows

$$DEP5\_EMPS \leftarrow \sigma_{Dno=5}(EMPLOYEE)$$
$$RESULT1 \leftarrow \pi_{Ssn}(DEP5\_EMPS)$$
$$RESULT2(Ssn) \leftarrow \pi_{Super\_ssn}(DEP5\_EMPS)$$
$$RESULT \leftarrow RESULT1 \cup RESULT2$$

- UNION operation produces the tuples that are in either RESULT1 or RESULT2 or both, while eliminating any duplicates.

DEP5_EMPS

Fname	Minit	Lname	Ssn	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
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

$$\text{DEP5\_EMPS} \leftarrow \sigma_{Dno=5}(\text{EMPLOYEE})$$
$$\text{RESULT1} \leftarrow \pi_{Ssn}(\text{DEP5\_EMPS})$$
$$\text{RESULT2(Ssn)} \leftarrow \pi_{Super\_ssn}(\text{DEP5\_EMPS})$$
$$\text{RESULT} \leftarrow \text{RESULT1} \cup \text{RESULT2}$$

RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

Ssn
333445555
888665555

RESULT

Ssn
123456789
333445555
666884444
453453453
888665555

- UNION, INTERSECTION, and SET DIFFERENCE are binary operations;
  - that is, each is applied to two sets

Union compatibility or Type compatibility

- Two relations *R(A1, A2, ..., An) and S(B1, B2, ..., Bn) are said to be union compatible (or*
- type compatible)
  - if they have the same degree *n and if dom(Ai) = dom(Bi) for 1 <=i<=n.*
  - *This means that the two relations have the same number of attributes and each* corresponding pair of attributes has the same domain.

(a) Two union-compatible relations.

(a) STUDENT

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

(b) STUDENT ∪ INSTRUCTOR

(b)

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c) STUDENT ∩ INSTRUCTOR

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

(d) STUDENT – INSTRUCTOR

(d)

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e) INSTRUCTOR – STUDENT

(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

The UNION, INTERSECTION, and MINUS Properties

- UNION and INTERSECTION are *commutative operations*

$$R \cup S = S \cup R \text{ and } R \cap S = S \cap R$$

- Both UNION and INTERSECTION can be treated as *n-ary operations applicable to* any number of relations because both are also *associative operations*

$$R \cup (S \cup T) = (R \cup S) \cup T \text{ and } (R \cap S) \cap T = R \cap (S \cap T)$$

- The MINUS operation is *not commutative; that is, in general,*

$$R - S \neq S - R$$

- INTERSECTION can be expressed in terms of union and set difference as follows

$$R \cap S = ((R \cup S) - (R - S)) - (S - R)$$

CARTESIAN PRODUCT (CROSS PRODUCT) OR CROSS JOIN

- denoted by ×
- This is also a binary set operation, but the relations on which it is applied do *not have to be union compatible*
- the result of $R(A1, A2, …, An) × S(B1, B2, …, Bm)$ *is a relation Q with degree n + m attributes Q(A1, A2, …, An, B1, B2, …, Bm), in that order.*
- The resulting relation Q *has one tuple for each combination of tuples—one from R and one from S.*
- *Hence, if R has n_R tuples (denoted as |R| = n_R), and S has n_S tuples, then $R × S$ will have $n_R * n_S$ tuples.*

n-ary CARTESIAN PRODUCT

- produces new tuples by concatenating all possible combinations of tuples from *n underlying relations*

Example

- We want to retrieve a list of names of each female employee's dependents

EMPLOYEE

Fname	Minit	Lname	Ssn	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

$$\text{FEMALE\_EMPS} \leftarrow \sigma_{Sex='F'}(\text{EMPLOYEE})$$
$$\text{EMPNAMES} \leftarrow \pi_{Fname, Lname, Ssn}(\text{FEMALE\_EMPS})$$
$$\text{EMP\_DEPENDENTS} \leftarrow \text{EMPNAMES} \times \text{DEPENDENT}$$
$$\text{ACTUAL\_DEPENDENTS} \leftarrow \sigma_{Ssn=Essn}(\text{EMP\_DEPENDENTS})$$
$$\text{RESULT} \leftarrow \pi_{Fname, Lname, Dependent\_name}(\text{ACTUAL\_DEPENDENTS})$$

FEMALE_EMPS

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Alicia	J	Zelaya	999887777	1968-07-19	3321Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291Berry, Bellaire, TX	F	43000	888665555	4
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

EMPNAMES

Fname	Lname	Ssn
Alicia	Zelaya	999887777
Jennifer	Wallace	987654321
Joyce	English	453453453

$$FEMALE\_EMPS \leftarrow \sigma_{Sex='F'}(EMPLOYEE)$$
$$EMPNAMES \leftarrow \pi_{Fname, Lname, Ssn}(FEMALE\_EMPS)$$
$$EMP\_DEPENDENTS \leftarrow EMPNAMES \times DEPENDENT$$

EMP_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	...
Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	...
Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	...
Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	...
Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	...
Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	...
Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	...
Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	...
Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	...
Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...
Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	...
Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	...
Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	...
Joyce	English	453453453	333445555	Alice	F	1986-04-05	...
Joyce	English	453453453	333445555	Theodore	M	1983-10-25	...
Joyce	English	453453453	333445555	Joy	F	1958-05-03	...
Joyce	English	453453453	987654321	Abner	M	1942-02-28	...
Joyce	English	453453453	123456789	Michael	M	1988-01-04	...
Joyce	English	453453453	123456789	Alice	F	1988-12-30	...
Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	...

FEMALE_EMPS $\leftarrow \sigma_{Sex='F'}$(EMPLOYEE)

EMPNAMES $\leftarrow \pi_{Fname, Lname, Ssn}$(FEMALE_EMPS)

EMP_DEPENDENTS $\leftarrow$ EMPNAMES $\times$ DEPENDENT

ACTUAL_DEPENDENTS $\leftarrow \sigma_{Ssn=Essn}$(EMP_DEPENDENTS)

RESULT $\leftarrow \pi_{Fname, Lname, Dependent\_name}$(ACTUAL_DEPENDENTS)

ACTUAL_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	. . .
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	. . .

RESULT

Fname	Lname	Dependent_name
Jennifer	Wallace	Abner

- The CARTESIAN PRODUCT creates tuples with the combined attributes of two relations.

- We can SELECT *related tuples only from the two relations by specifying an* appropriate selection condition after the Cartesian product, as we did in the preceding example.

- Because this sequence of CARTESIAN PRODUCT followed by SELECT is quite commonly used to combine *related tuples from two relations, a special operation,* called JOIN, was created to specify this sequence as a single operation

The JOIN Operation

- The JOIN operation, denoted by ⋈, is used to combine *related tuples from two relations* into single "longer" tuples.
- This operation is very important for any relational database with more than a single relation because it allows us to process relationships among relations.

Example

- Retrieve the name of the manager of each department.
  - To get the manager's name, we need to combine each department tuple with the employee tuple whose Ssn value matches the Mgr_ssn value in the department tuple.
  - We do this by using the JOIN operation and then projecting the result over the necessary attributes, as follows

$$\text{DEPT\_MGR} \leftarrow \text{DEPARTMENT} \bowtie_{\text{Mgr\_ssn=Ssn}} \text{EMPLOYEE}$$

$$\text{RESULT} \leftarrow \pi_{\text{Dname, Lname, Fname}}(\text{DEPT\_MGR})$$

DEPT_MGR

Dname	Dnumber	Mgr_ssn	. . .	Fname	Minit	Lname	Ssn	. . .
Research	5	333445555	. . .	Franklin	T	Wong	333445555	. . .
Administration	4	987654321	. . .	Jennifer	S	Wallace	987654321	. . .
Headquarters	1	888665555	. . .	James	E	Borg	888665555	. . .

Figure 6.6

Result of the JOIN operation DEPT_MGR ← DEPARTMENT $\bowtie_{\text{Mgr\_ssn=Ssn}}$ EMPLOYEE.

RESULT

Dname	Lname	Fname
Research	Wong	Franklin
Administration	Wallance	Jennifer
Headquaters	Borg	James

- The JOIN operation can be specified as a CARTESIAN PRODUCT operation followed by a SELECT operation.

EMP_DEPENDENTS ← EMPNAMES × DEPENDENT
ACTUAL_DEPENDENTS ← $\sigma_{Ssn=Essn}$(EMP_DEPENDENTS)

- These two operations can be replaced with a single JOIN operation as follows

ACTUAL_DEPENDENTS ← EMPNAMES ⋈ $_{Ssn=Essn}$ DEPENDENT

General form of a JOIN

- JOIN operation on two relations5 *R(A1, A2, ...,
An) and S(B1, B2, ..., Bm) is*

$$R \bowtie_{<\text{join condition}>} S$$

- The result of the JOIN is a relation *Q with n + m
attributes Q(A1, A2, ..., An, B1, B2, ... , Bm) in
that order;*

- *Q has one tuple for each combination of tuples—
one from R and one from S—whenever the
combination satisfies the join condition*

Difference between JOIN and CARTESIAN PRODUCT

- In JOIN, only combinations of tuples *satisfying the join condition appear in the result,*

- *whereas in the* CARTESIAN PRODUCT *all combinations of tuples are included in the result.*

- *The* join condition is specified on attributes from the two relations *R and S and is evaluated* for each combination of tuples.

- Each tuple combination for which the join condition evaluates to TRUE is included in the resulting relation *Q as a single combined tuple.*

THETA JOIN

<condition> AND <condition> AND...AND <condition>

- where each <condition> is of the form *Ai θ Bj, Ai is an attribute of R, Bj is an attribute* of *S, Ai and Bj have the same domain, and*

- *θ (theta) is one of the comparison* operators {=, <, ≤, >, ≥, ≠}.

- Tuples whose join attributes are NULL or for which the join condition is FALSE *do not appear in the result*

- the JOIN operation does *not necessarily preserve all of the information in the participating relations,*
  - *because* tuples that do not get combined with matching ones in the other relation do not appear in the result.

EQUIJOIN

- a JOIN, where the only comparison operator used is =, is called an **EQUIJOIN**
- in the result of an EQUIJOIN we always have one or more pairs of attributes that have *identical values* in every tuple

NATURAL JOIN

- NATURAL JOIN requires that the two join attributes (or each pair of join attributes) have the same name in both relations.
- If this is not the case, a renaming operation is applied first.

Example

- combine each PROJECT tuple with the DEPARTMENT tuple that controls the project
  - first we rename the Dnumber attribute of DEPARTMENT to Dnum
  - so that it has the same name as the Dnum attribute in PROJECT—and then we apply NATURAL JOIN

$$\text{PROJ\_DEPT} \leftarrow \text{PROJECT} * \rho_{(Dname, Dnum, Mgr\_ssn, Mgr\_start\_date)}(\text{DEPARTMENT})$$

- The same query can be done in two steps by creating an intermediate table DEPT

$$\text{DEPT} \leftarrow \rho_{(Dname, Dnum, Mgr\_ssn, Mgr\_start\_date)}(\text{DEPARTMENT})$$
$$\text{PROJ\_DEPT} \leftarrow \text{PROJECT} * \text{DEPT}$$

(a) PROJ_DEPT ← PROJECT ⋈ DEPT.

(a)

PROJ_DEPT

Pname	Pnumber	Plocation	Dnum	Dname	Mgr_ssn	Mgr_start_date
ProductX	1	Bellaire	5	Research	333445555	1988-05-22
ProductY	2	Sugarland	5	Research	333445555	1988-05-22
ProductZ	3	Houston	5	Research	333445555	1988-05-22
Computerization	10	Stafford	4	Administration	987654321	1995-01-01
Reorganization	20	Houston	1	Headquarters	888665555	1981-06-19
Newbenefits	30	Stafford	4	Administration	987654321	1995-01-01

- In the PROJ_DEPT relation, each tuple combines a PROJECT tuple with the DEPARTMENT tuple for the department that controls the project, but *only one join attribute value is kept.*

- If the attributes on which the natural join is specified already *have the same names in both relations, renaming is unnecessary*

- to apply a natural join on the Dnumber attributes of DEPARTMENT and DEPT LOCATIONS

DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS

(b)

DEPT_LOCS

Dname	Dnumber	Mgr_ssn	Mgr_start_date	Location
Headquarters	1	888665555	1981-06-19	Houston
Administration	4	987654321	1995-01-01	Stafford
Research	5	333445555	1988-05-22	Bellaire
Research	5	333445555	1988-05-22	Sugarland
Research	5	333445555	1988-05-22	Houston

- the join condition for NATURAL JOIN is constructed by equating *each pair of join attributes that have* the same name in the two relations and combining these conditions with **AND.**
- There can be a list of join attributes from each relation, and each corresponding pair must have the same name.

- more general, *but nonstandard definition for NATURAL JOIN is*

$$Q \leftarrow R *_{(<\text{list1}>),(<\text{list2}>)} S$$

- <list1> specifies a list of *i attributes from R, and*

- *<list2> specifies a list* of *i attributes from S.*

- *The lists are used to form equality comparison conditions* between pairs of corresponding attributes, and

- the conditions are then ANDed together.

- Only the list corresponding to attributes of the first relation *R—<list1>—* is kept in the result *Q.*

- if no combination of tuples satisfies the join condition, the result of a JOIN is an empty relation with zero tuples.

Table 6.1   Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{<\text{selection condition}>}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{<\text{attribute list}>}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{<\text{join condition}>} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{<\text{join condition}>} R_2$, OR $R_1 \bowtie_{(<\text{join attributes 1}>),\ (<\text{join attributes 2}>)} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{<\text{join condition}>} R_2$, OR $R_1 *_{(<\text{join attributes 1}>),\ (<\text{join attributes 2}>)} R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Different Types of SQL JOINs

- INNER JOIN
  - Returns records that have matching values in both tables
    1. Theta join
    2. EQUI join
    3. Natural join
- OUTER JOIN
  - In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.
    1. Left Outer JOIN
    2. Right Outer Join
    3. Full Outer Join

Left Outer Join $(A \rhd\!\!\bowtie B)$

- In the left outer join, operation allows keeping all tuple in the left relation.
- if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.
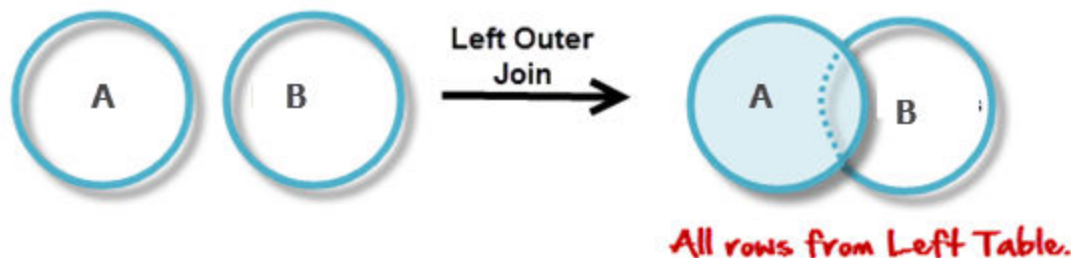
Left Outer Join

All rows from Left Table.

Consider the following 2 Tables

A	
Num	Square
2	4
3	9
4	16

B	
Num	Cube
2	8
3	18
5	75

A ⋈ B		
Num	Square	Cube
2	4	4
3	9	9
4	16	-

Right Outer Join (A ⋈ B)

- In the right outer join, operation allows keeping all tuple in the right relation.
- However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.

Right Outer Join

All rows from Right Table.

Consider the following 2 Tables

A

Num	Square
2	4
3	9
4	16

B

Num	Cube
2	8
3	18
5	75

(A ⋈ B)

Num	Cube	Square
2	8	4
3	18	9
5	75	-

Full Outer Join $(A \bowtie B)$

- In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.

Consider the following 2 Tables

A

Num	Square
2	4
3	9
4	16

B

Num	Cube
2	8
3	18
5	75

(A ⋈ B)

Num	Cube	Square
2	4	8
3	9	18
4	16	-
5	-	75

Left Outer Join(⟕)	In the left outer join, operation allows keeping all tuple in the left relation.
Right Outer join(⟖)	In the right outer join, operation allows keeping all tuple in the right relation.
Full Outer Join(⟗)	In a full outer join, all tuples from both relations are included in the result irrespective of the matching condition.

DIVISION Operation

- Retrieve the names of employees who work on all the projects that 'John Smith' works on.
  - query using the DIVISION operation, proceed as follows.
  - First, retrieve the list of project numbers that 'John Smith' works on in the intermediate relation SMITH   PNOS:

  $$\text{SMITH} \leftarrow \sigma_{\text{Fname='John' AND Lname='Smith'}}(\text{EMPLOYEE})$$
  $$\text{SMITH\_PNOS} \leftarrow \pi_{\text{Pno}}(\text{WORKS\_ON} \bowtie_{\text{Essn=Ssn}} \text{SMITH})$$

  - Next, create a relation that includes a tuple <Pno, Essn> whenever the employee whose Ssn is Essn works on the project whose number is Pno in the intermediate relation SSN_PNOS:

  $$\text{SSN\_PNOS} \leftarrow \pi_{\text{Essn, Pno}}(\text{WORKS\_ON})$$

- Finally, apply the DIVISION operation to the two relations, which gives the desired employees' Social Security numbers:

$$SSNS(Ssn) \leftarrow SSN\_PNOS \div SMITH\_PNOS$$
$$RESULT \leftarrow \pi_{Fname, Lname}(SSNS * EMPLOYEE)$$

Figure 6.8
The DIVISION operation. (a) Dividing SSN_PNOS by SMITH_PNOS. (b) $T \leftarrow R \div S$.

(a)

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30

SMITH_PNOS

Pno
1
2

SSNS

Ssn
123456789
453453453

(b)

R

A	B
a1	b1
a2	b1
a3	b1
a4	b1
a1	b2
a3	b2
a2	b3
a3	b3
a4	b3
a1	b4
a2	b4
a3	b4

S

A
a1
a2
a3

T

B
b1
b4

- **Query 1. Retrieve the name and address of all employees who work for the** 'Research' department.

$$\text{RESEARCH\_DEPT} \leftarrow \sigma_{\text{Dname='Research'}}(\text{DEPARTMENT})$$
$$\text{RESEARCH\_EMPS} \leftarrow (\text{RESEARCH\_DEPT} \bowtie_{\text{Dnumber=Dno}} \text{EMPLOYEE})$$
$$\text{RESULT} \leftarrow \pi_{\text{Fname, Lname, Address}}(\text{RESEARCH\_EMPS})$$

As a single in-line expression, this query becomes:

$$\pi_{\text{Fname, Lname, Address}} (\sigma_{\text{Dname='Research'}}(\text{DEPARTMENT} \bowtie_{\text{Dnumber=Dno}} (\text{EMPLOYEE})))$$

- This query could be specified in other ways; for example, the order of the JOIN and
- SELECT operations could be reversed, or the JOIN could be replaced by a NATURAL JOIN after renaming one of the join attributes to match the other join attribute name.

- **Query 2. For every project located in 'Stafford', list the project number, the** controlling department number, and the department manager's last name, address, and birth date.

STAFFORD_PROJS ← $\sigma_{Plocation='Stafford'}$(PROJECT)
CONTR_DEPTS ← (STAFFORD_PROJS ⋈ $_{Dnum=Dnumber}$ DEPARTMENT)
PROJ_DEPT_MGRS ← (CONTR_DEPTS ⋈ $_{Mgr\_ssn=Ssn}$ EMPLOYEE)
RESULT ← $\pi_{Pnumber, Dnum, Lname, Address, Bdate}$(PROJ_DEPT_MGRS)

- we first select the projects located in Stafford, then join them with their controlling departments, and then join the result with the department managers.
- Finally, we apply a project operation on the desired attributes

- **Query 3. Make a list of project numbers for projects that involve an employee** whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

$\text{SMITHS(Essn)} \leftarrow \pi_{Ssn} \, (\sigma_{Lname='Smith'}(\text{EMPLOYEE}))$

$\text{SMITH\_WORKER\_PROJS} \leftarrow \pi_{Pno}(\text{WORKS\_ON} * \text{SMITHS})$

$\text{MGRS} \leftarrow \pi_{Lname, \, Dnumber}(\text{EMPLOYEE} \bowtie_{Ssn=Mgr\_ssn}\text{DEPARTMENT})$

$\text{SMITH\_MANAGED\_DEPTS(Dnum)} \leftarrow \pi_{Dnumber} \, (\sigma_{Lname='Smith'}(\text{MGRS}))$

$\text{SMITH\_MGR\_PROJS(Pno)} \leftarrow \pi_{Pnumber}(\text{SMITH\_MANAGED\_DEPTS} * \text{PROJECT})$

$\text{RESULT} \leftarrow (\text{SMITH\_WORKER\_PROJS} \cup \text{SMITH\_MGR\_PROJS})$

SYLLABUS

- Structure of Relational Databases - Integrity Constraints, Synthesizing ER diagram to relational schema
- Introduction to Relational Algebra - select, project, cartesian product operations, join - Equi-join, natural join. query examples,
- Introduction to Structured Query Language (SQL), Data Definition Language (DDL), Table definitions and operations – CREATE, DROP, ALTER, INSERT, DELETE, UPDATE.

SQL

- SQL provides
  - A data definition language (DDL)
  - A data manipulation language (DML)
  - A data control language (DCL)
- In addition SQL
  - Can be used from other languages
  - Is often extended to provide common programming constructs (such as if-then tests, loops, variables, etc.)

Cont..

- SQL is a declarative (non-procedural) language
  - Procedural - say exactly what the computer has to do
  - Non-procedural – describe the required result (not the way to compute it)
- SQL is based on the relational model
  - It has many of the same ideas
  - Databases that support SQL are often described as relational databases
  - It is not always true to the model

Cont..

- E/R designs can be implemented in SQL
  - Entities, attributes, and relationships can all be expressed in terms of SQL
  - Many-to-many relationships are a problem, so should be removed

Relations, Entities, Tables

Relational model	E/R Diagram	SQL
Relation	Entity	Table
Tuple	Instance	Row
Attribute	Attribute	Column or Field
Foreign Key	M:1 Relationship	Foreign Key
Primary Key		Primary Key

Implementing E/R Designs

- Given an E/R design
  - The entities become SQL tables
  - Attributes of an entity become columns in the corresponding table
  - Relationships may be represented by foreign keys

- Each entity becomes a table in the database
  - The name of the table is often the name of the entity
  - The attributes become columns of the table with the same name

- A table called Student
- With columns for ID, Name, Address, and Year

Schema and Catalog Concepts in SQL

- An SQL schema is identified by a schema name, and includes an authorization identifier to indicate the user or account who owns the schema, as well as descriptors for each element in the schema.

- Schema elements include tables, constraints, views, domains, and other constructs that describe the schema

- A schema is created via the CREATE SCHEMA statement, which can include all the schema elements definitions.

- Creates a schema called COMPANY, owned by the user with authorization identifier 'Jsmith'.

CREATE SCHEMA COMPANY AUTHORIZATION 'Jsmith';

- not all users are authorized to create schemas and schema elements.
- The privilege to create schemas, tables, and other constructs must be explicitly granted to the relevant user accounts by the system administrator or DBA.

SQL environment

- SQL environment is basically an installation of an SQL-compliant RDBMS on a computer system
- SQL uses the concept of a catalog a named collection of schemas in an SQL environment.
- A catalog always contains a special schema called INFORMATION_SCHEMA,
  - which provides information on all the schemas in the catalog and all the element descriptors in these schemas

Cont..

- Integrity constraints such as referential integrity can be defined between relations only if they exist in schemas within the same catalog.
- Schemas within the same catalog can also share certain elements, such as domain definitions.

DATABASE LANGUAGES

1. Data Definition Language (DDL):
   - It is used to specify a database conceptual schema using set of definitions.
   - It supports the definition or declaration of database objects.
   - The more common DDL commands are
     - a.CREATE TABLE:
     - ALTER TABLE
     - DROP TABLE
     - TRUNCATE
     - COMMENT
     - RENAME

2.   Data Manipulation Language (DML)

- It provides a set of operations to support the basic data manipulation operations on the data held in the database.

- It is used to query, update or retrieve data stored in a database.

- Some of the tasks that come under DML are

  - SELECT
    - Used to query and display data from a database.
  - INSERT
    - Adds new rows to a table.
  - UPDATE
    - Changes an existing value in a column or group of columns in a table.
  - DELETE:
    - Removes a specified row or set of rows from a table.
  - MERGE.

SQL Data Definition

- The set of relations in a database are specified using a data-definition language (DDL)
- The SQL DDL allows specification of not only a set of relations, but also information about each relation, including:
  - The schema for each relation.
  - The types of values associated with each attribute.
  - The integrity constraints.
  - The set of indices to be maintained for each relation.
  - The security and authorization information for each relation.
  - The physical storage structure of each relation on disk.

Basic Schema Definition: CREATE TABLE Command in SQL

- CREATE TABLE command is used to specify a new relation by giving it a name and specifying its attributes and initial constraints.
- The attributes are specified first, and each attribute is given a name, a data type to specify its domain of values, and any attribute constraints, such as NOT NULL.
- The key, entity integrity, and referential integrity constraints can be specified within the CREATE TABLE statement

CREATE TABLE

- Specifies a new base relation by giving it a name, and specifying each of its attributes and their data types (INTEGER, FLOAT, DECIMAL(i,j), CHAR(n), VARCHAR(n))
- A constraint NOT NULL may be specified on an attribute

```
CREATE TABLE DEPARTMENT (
    DNAME                VARCHAR(10)    NOT
NULL,
    DNUMBER         INTEGER            NOT
NULL,
    MGRSSN          CHAR(9),
    MGRSTARTDATE    CHAR(9)   );
```

The general form of the **create table** command is:

create table r
$(A_1 \quad D_1,$
$A_2 \quad D_2,$
$\dots,$
$A_n \quad D_n,$
$\langle\text{integrity-constraint}_1\rangle,$
$\dots,$
$\langle\text{integrity-constraint}_k\rangle);$

- The relations declared through CREATE TABLE statements are called base tables
- this means that the relation and its tuples are actually created and stored as a file by the DBMS

- In SQL2, can use the CREATE TABLE command for specifying the primary key attributes, secondary keys, and referential integrity constraints (foreign keys).
- Key attributes can be specified via the PRIMARY KEY and UNIQUE phrases

```
CREATE TABLE DEPT (
  DNAME            VARCHAR(10)    NOT
NULL,
  DNUMBER          INTEGER        NOT
NULL,
  MGRSSN      CHAR(9),
  MGRSTARTDATE     CHAR(9),
  PRIMARY KEY (DNUMBER),
  UNIQUE (DNAME),
  FOREIGN KEY (MGRSSN) REFERENCES EMP
);
```

Attribute Data Types and Domains in SQL

- The basic data types available for attributes include
  - numeric,
  - character string,
  - bit string,
  - Boolean,
  - date, and time
  - timestamp

Specifying Constraints in SQL

- Specifying Attribute Constraints and Attribute Defaults
- Specifying Key and Referential Integrity Constraints
- Giving Names to Constraints
- Specifying Constraints on Tuples Using CHECK

Specifying Attribute Constraints and Attribute Defaults

- a constraint NOT NULL may be specified if NULL is not permitted for a particular attribute.
- This is always implicitly specified for the attributes that are part of the primary key of each relation,
  - but it can be specified for any other attributes whose values are required not to be NULL
- It is also possible to define a default value for an attribute by appending the clause DEFAULT <value> to an attribute definition.

- If no default clause is specified, the default default value is NULL for attributes that do not have the NOT NULL constraint.

- Another type of constraint can restrict attribute or domain values using the CHECK clause following an attribute or domain definition.

- suppose that department numbers are restricted to integer numbers between 1 and 20;
- then, we can change the attribute declaration of Dnumber in the DEPARTMENT table

Dnumber INT NOT NULL CHECK (Dnumber > 0 AND Dnumber < 21);

Specifiying Key and Referiential

- ## PRIMARY KEY

  - specifies one or more attributes that make up the primary key of a relation.
  - If a primary key has a single attribute, the clause can follow the attribute directly

  Dnumber INT **PRIMARY KEY;**

- UNIQUE
  - specifies alternate (secondary) keys
  - can also be specified directly for a secondary key if the secondary key is a single attribute

Dname VARCHAR(15) **UNIQUE;**

- FOREIGN KEY
  - a referential integrity constraint can be violated when tuples are inserted or deleted, or when a foreign key or primary key attribute value is modified
  - the schema designer
    - can specify an alternative action to be taken by attaching a referential triggered action clause to any foreign key constraint. The options include:
      - ✓SET NULL,
      - ✓CASCADE, and
      - ✓SET DEFAULT.
    - An option must be qualified with either
      - ✓ON DELETE or
      - ✓ ON UPDATE

```
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);
```

- ON DELETE SET NULL and ON UPDATE CASCADE for the foreign key Super_ssn of EMPLOYEE means
    - if the tuple for a supervising employee is deleted,
    - the value of Super_ssn is automatically set to NULL for all employee tuples that were referencing the deleted employee tuple
    - if the Ssn value for a supervising employee is updated the new value is cascaded to Super_ssn for all employee tuples referencing the updated employee tuple

Giving Names to Constraints

- a constraint may be given a constraint name, following the keyword CONSTRAINT

- The names of all constraints within a particular schema must be unique.

- A constraint name is used to identify a particular constraint

  - in case the constraint must be dropped later and replaced with another constraint

```
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);
```

Specifying Constraints on Tuples Using CHECK

- CHECK clauses is specified at the end of a CREATE TABLE statement

- These can be called tuple-based constraints because they apply to each tuple individually and are checked whenever a tuple is inserted or modified

- The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

DROP TABLE

- The SQL DROP TABLE statement is used to remove a table definition and all the data, indexes, triggers, constraints and permission specifications for that table.

- You should be very careful while using this command because once a table is deleted then all the information available in that table will also be lost forever.

```
DROP TABLE table_name;
```

ALTER TABLE

- The SQL ALTER TABLE command is used to add, delete or modify columns in an existing table.

- You should also use the ALTER TABLE command to add and drop various constraints on an existing table.

- The basic syntax of an ALTER TABLE command to add a New Column in an existing table is as follows.

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

- The basic syntax of an ALTER TABLE command to DROP COLUMN in an existing table is as follows.

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

- The basic syntax of an ALTER TABLE command to change the DATA TYPE of a column in a table is as follows.

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

- The basic syntax of an ALTER TABLE command to add a NOT NULL constraint to a column in a table is as follows.

```
ALTER TABLE table_name MODIFY column_name datatype NOT NULL;
```

- The basic syntax of ALTER TABLE to ADD UNIQUE CONSTRAINT to a table is as follows

```
ALTER TABLE table_name
ADD CONSTRAINT MyUniqueConstraint UNIQUE(column1, column2...);
```

- The basic syntax of an ALTER TABLE command to ADD CHECK CONSTRAINT to a table is as follows.

```
ALTER TABLE table_name
ADD CONSTRAINT MyUniqueConstraint CHECK (CONDITION);
```

- The basic syntax of an ALTER TABLE command to ADD PRIMARY KEY constraint to a table is as follows.

```
ALTER TABLE table_name
ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);
```

- The basic syntax of an ALTER TABLE command to DROP CONSTRAINT from a table is as follows.

```
ALTER TABLE table_name
DROP CONSTRAINT MyUniqueConstraint;
```

INSERT Command

- INSERT is used to add a single tuple to a relation.

- We must specify the relation name and a list of values for the tuple.

- The values should be listed in the same order in which the corresponding attributes were specified in the CREATE TABLE command.

U1:   INSERT INTO   EMPLOYEE
        VALUES   ('Richard', 'K', 'Marini', '653298653', '1962-12-30', '98
                        Oak Forest, Katy, TX', 'M', 37000, '653298653', 4);

- A second form of the INSERT statement allows the user to specify explicit attribute names that correspond to the values provided in the INSERT command.
- This is useful if a relation has many attributes but only a few of those attributes are assigned values in the new tuple.
- However, the values must include all attributes with NOT NULL specification and no default value.
- Attributes with NULL allowed or DEFAULT values are the ones that can be left out.
- For example, to enter a tuple for a new EMPLOYEE for whom we know only the Fname, Lname, Dno, and Ssn attributes

```
U1A:    INSERT INTO    EMPLOYEE (Fname, Lname, Dno, Ssn)
        VALUES         ('Richard', 'Marini', 4, '653298653');
```

- Attributes not specified in U1A are set to their DEFAULT or to NULL, and the values are listed in the same order as the attributes are listed in the INSERT command itself.
- It is also possible to insert into a relation multiple tuples separated by commas in a single INSERT command.
- The attribute values forming each tuple are enclosed in parentheses.

- A DBMS that fully implements SQL should support and enforce all the integrity constraints that can be specified in the DDL

no department no 2 is available

U3:   INSERT INTO   EMPLOYEE (Fname, Lname, Ssn, Dno)
        VALUES   ('Robert', 'Hatcher', '980760540', 2);
        (U2 is rejected if referential integrity checking is provided by DBMS.)

U2A: INSERT INTO   EMPLOYEE (Fname, Lname, Dno)
        VALUES   ('Robert', 'Hatcher', 5);
        (U2A is rejected if NOT NULL checking is provided by DBMS.)

- A variation of the INSERT command inserts multiple tuples into a relation in conjunction with creating the relation and loading it with the result of a query.
- For example,
  - ▫ to create a temporary table that has the employee last name, project name, and hours per week for each employee working on a project

```
U3A:    CREATE TABLE        WORKS_ON_INFO
        ( Emp_name           VARCHAR(15),
          Proj_name          VARCHAR(15),
          Hours_per_week     DECIMAL(3,1) );
```

```
U3B:    INSERT INTO    WORKS_ON_INFO ( Emp_name, Proj_name,
                       Hours_per_week )
        SELECT         E.Lname, P.Pname, W.Hours
        FROM           PROJECT P, WORKS_ON W, EMPLOYEE E
        WHERE          P.Pnumber=W.Pno AND W.Essn=E.Ssn;
```

The DELETE Command

- The DELETE command removes tuples from a relation.
- It includes a WHERE clause, similar to that used in an SQL query, to select the tuples to be deleted.
- Tuples are explicitly deleted from only one table at a time.
- However, the deletion may propagate to tuples in other relations if referential triggered actions are specified in the referential integrity constraints of the DDL

U4A:	DELETE FROM	EMPLOYEE
	WHERE	Lname='Brown';
U4B:	DELETE FROM	EMPLOYEE
	WHERE	Ssn='123456789';
U4C:	DELETE FROM	EMPLOYEE
	WHERE	Dno=5;
U4D:	DELETE FROM	EMPLOYEE;

- Depending on the number of tuples selected by the condition in the WHERE clause, zero, one, or sev_x0002_eral tuples can be deleted by a single DELETE command.
- A missing WHERE clause specifies that all tuples in the relation are to be deleted; however, the table remains in the database as an empty table.
- We must use the DROP TABLE command to remove the table definition

TRUNCATE TABLE COMMAND

- The SQL TRUNCATE TABLE command is used to delete complete data from an existing table.

- You can also use DROP TABLE command to delete complete table but it would remove complete table structure form the database and you would need to re-create this table once again if you wish you store some data.

```
TRUNCATE TABLE  table_name;
```

The UPDATE Command

- The UPDATE command is used to modify attribute values of one or more selected tuples.
- As in the DELETE command, a WHERE clause in the UPDATE command selects the tuples to be modified from a single relation

The basic syntax of the UPDATE query with a WHERE clause is as follows –

```
UPDATE table_name
SET column1 = value1, column2 = value2....., columnN = valueN
WHERE [condition];
```
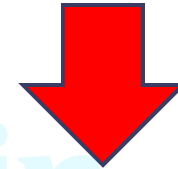
Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

```
SQL> UPDATE CUSTOMERS
SET ADDRESS = 'Pune'
WHERE ID = 6;
```

Now, the CUSTOMERS table would have the following records –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | Pune      |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

- updating a primary key value may propagate to the foreign key values of tuples in other relations if such a referential triggered action is specified in the referential integrity constraints of the DDL
- An additional SET clause in the UPDATE command specifies the attributes to be modified and their new values.
- It is also possible to specify NULL or DEFAULT as the new attribute value
- each UPDATE command explicitly refers to a single relation only.
- To modify multiple relations, we must issue several UPDATE commands.

U5:   UPDATE   PROJECT
        SET   Plocation = 'Bellaire', Dnum = 5
        WHERE   Pnumber=10;

U6:   UPDATE   EMPLOYEE
        SET   Salary = Salary * 1.1
        WHERE   Dno = 5;

MODULE 2 ENDS