CS3492 DATABASE MANAGEMENT SYSTEMS

UNIT I RELATIONAL DATABASES **10**

Purpose of Database System – Views of data – Data Models – Database System Architecture – Introduction to relational databases – Relational Model – Keys – Relational Algebra – SQL fundamentals – Advanced SQL features – Embedded SQL– Dynamic SQL

UNIT II DATABASE DESIGN **8**

Entity-Relationship model – E-R Diagrams – Enhanced-ER Model – ER-to-Relational Mapping – Functional Dependencies – Non-loss Decomposition – First, Second, Third Normal Forms, Dependency Preservation – Boyce/Codd Normal Form – Multi-valued Dependencies and Fourth Normal Form – Join Dependencies and Fifth Normal Form

UNIT III TRANSACTIONS **9**

Transaction Concepts – ACID Properties – Schedules – Serializability – Concurrency Control – Need for Concurrency – Locking Protocols – Two Phase Locking – Deadlock – Transaction Recovery - Save Points – Isolation Levels – SQL Facilities for Concurrency and Recovery.

UNIT IV IMPLEMENTATION TECHNIQUES **9**

RAID – File Organization – Organization of Records in Files – Indexing and Hashing –Ordered Indices – B+ tree Index Files – B tree Index Files – Static Hashing – Dynamic Hashing – Query Processing Overview – Algorithms for SELECT and JOIN operations – Query optimization using Heuristics and Cost Estimation.

UNIT V ADVANCED TOPICS **9**

Distributed Databases: Architecture, Data Storage, Transaction Processing – Object-based Databases: Object Database Concepts, Object-Relational features, ODMG Object Model, ODL, OQL - XML Databases: XML Hierarchical Model, DTD, XML Schema, XQuery – Information Retrieval: IR Concepts, Retrieval Models, Queries in IR systems.

TOTAL: 45 PERIODS

TEXT BOOKS:

1. Abraham Silberschatz, Henry F. Korth, S. Sudharshan, —Database System Concepts||,Sixth Edition, Tata McGraw Hill, 2011
2. Ramez Elmasri, Shamkant B. Navathe, —Fundamentals of Database Systems||, Sixth Edition, Pearson, 2011.

REFERENCES:

1. C. J. Date, A.Kannan, S. Swamynathan, —An Introduction to Database Systems||, Eighth Edition, Pearson Education, 2006.
2. Raghu Ramakrishnan, —Database Management Systems||, Fourth Edition, McGraw-Hill College Publications, 2015.
3. G.K.Gupta,"Database Management Systems||, Tata McGraw Hill, 2011.

UNIT I RELATIONAL DATABASES

SYLLABUS

UNIT 1: INTRODUCTION

Purpose of Database System -– Views of data – Data Models – Database Languages — Database System Architecture – Database users and Administrator – Entity–Relationship model (E-R model) – E-R Diagrams -- Introduction to relational databases

1.1 Database Management System (DBMS)

- **Collection of interrelated data**
- **Set of programs to access the data**
- **DBMS contains information about a particular enterprise**
- **DBMS provides an environment that is both convenient and efficient to use.**
- **Database Applications:**
  - Banking: all transactions
  - Airlines: reservations, schedules
  - Universities: registration, grades
  - Sales: customers, products, purchases
  - Manufacturing: production, inventory, orders, supply chain
  - Human resources: employee records, salaries, tax deductions

- **Databases touch all aspects of our lives**

1.2 PURPOSE OF DATABASE SYSTEM

- **In the early days, database applications were built on top of file systems**
- **Drawbacks of using file systems to store data:**
  - **Data redundancy and inconsistency**
    - **Multiple file formats, duplication of information in different files**
  - **Difficulty in accessing data**
    - **Need to write a new program to carry out each new task**
  - **Data isolation — multiple files and formats**
  - **Integrity problems**
    - **Integrity constraints (e.g. account balance > 0) become part of program code**
    - **Hard to add new constraints or change existing ones**
  - **Atomicity of updates**
    - **Failures may leave database in an inconsistent state with partial updates carried out**
    - **E.g. transfer of funds from one account to another should either complete or not happen at all**
  - **Concurrent access by multiple users**
    - **Concurrent accessed needed for performance**
    - **Uncontrolled concurrent accesses can lead to inconsistencies**
      - E.g. two people reading a balance and updating it at the same time
  - **Security problems**

- **Database systems offer solutions to all the above problems**

1.3 VIEWS OF DATA

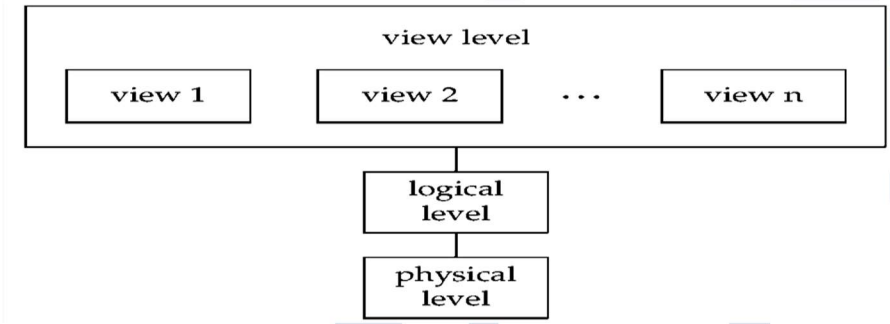
1.3.1 Levels of Abstraction

- **Physical level describes how a record (e.g., customer) is stored.**
- **Logical level: describes data stored in database, and the relationships among the data.**

```
Type customer = record
        name : string;
        street : string;
        city : integer;
    end;
```

- **View level: application programs hide details of data types. Views can also hide information (e.g., salary) for security purposes.**

View of Data



1.3.2 Instances and Schemas

- **Similar to types and variables in programming languages**
- **Schema – the logical structure of the database**
  - e.g., the database consists of information about a set of customers and accounts and the relationship between them)
  - Analogous to type information of a variable in a program
  - Physical schema: database design at the physical level
  - Logical schema: database design at the logical level
- **Instance – the actual content of the database at a particular point in time**
  - **Analogous to the value of a variable**
- **Physical Data Independence – the ability to modify the physical schema without changing the logical schema**
  - **Applications depend on the logical schema**
  - **In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.**

1.4 DATABASE LANGUAGES

1.4.1 Data Manipulation Language (DML)

- Language for accessing and manipulating the data organized by the appropriate data model
  – DML also known as query language
- Two classes of languages
  – **Procedural** – user specifies what data is required and how to get those data
  –**Declarative (nonprocedural)** – user specifies what data is required without specifying how to get those
- SQL is the most widely used query language.

1.4.2 Data Definition Language (DDL)

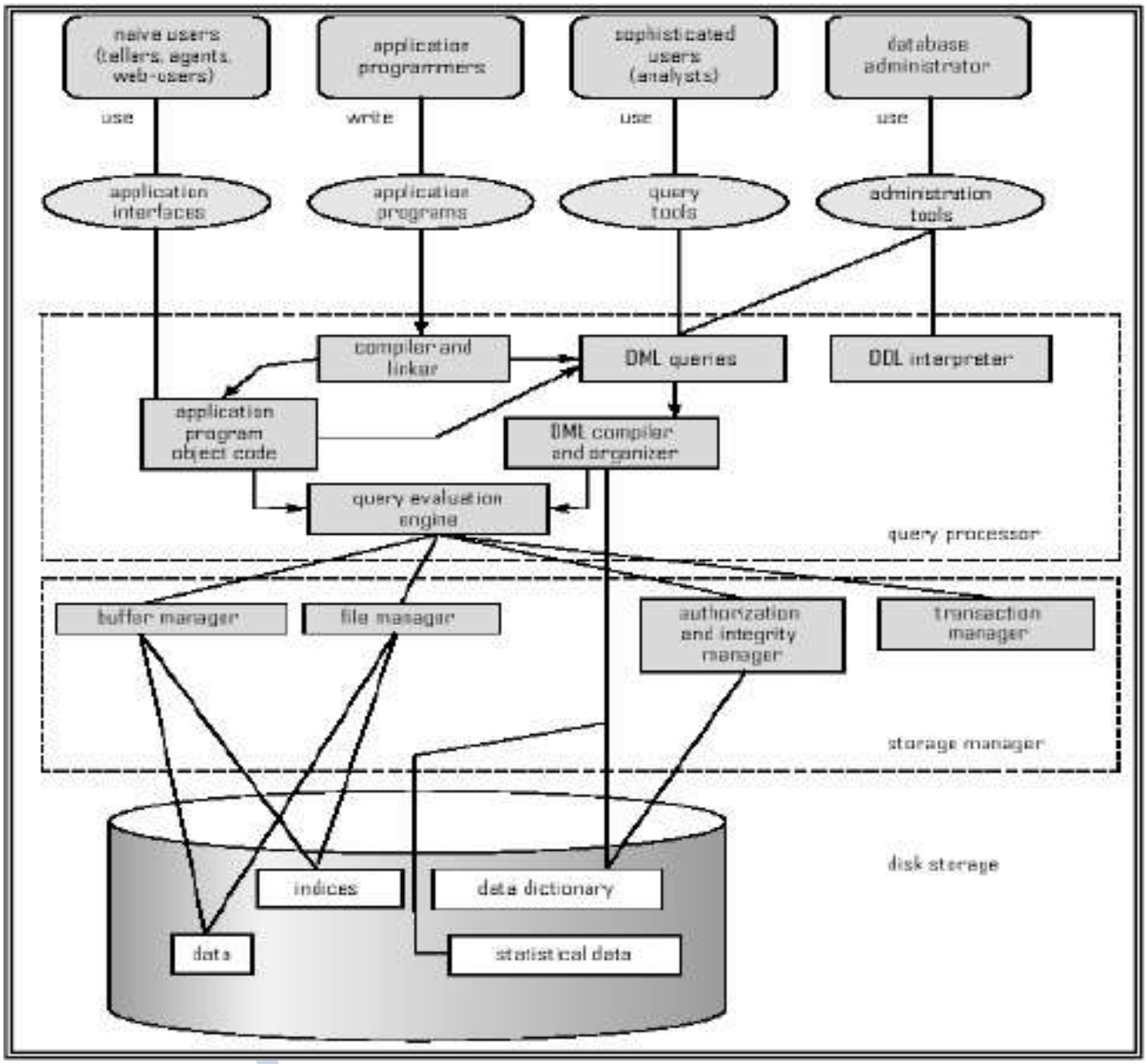
- Specification notation for defining the database schema

Example: **create table** account (
account-number **char**(10),
balance **integer**);

- DDL compiler generates a set of tables stored in a data dictionary
- Data dictionary contains metadata (i.e., data about data)
  – Database schema
  – Data storage and definition language
  --Specifies the storage structure and access methods used
  – Integrity constraints
  --Domain constraints
  --Referential integrity (**references** constraint in SQL)
  --Assertions
  – Authorization

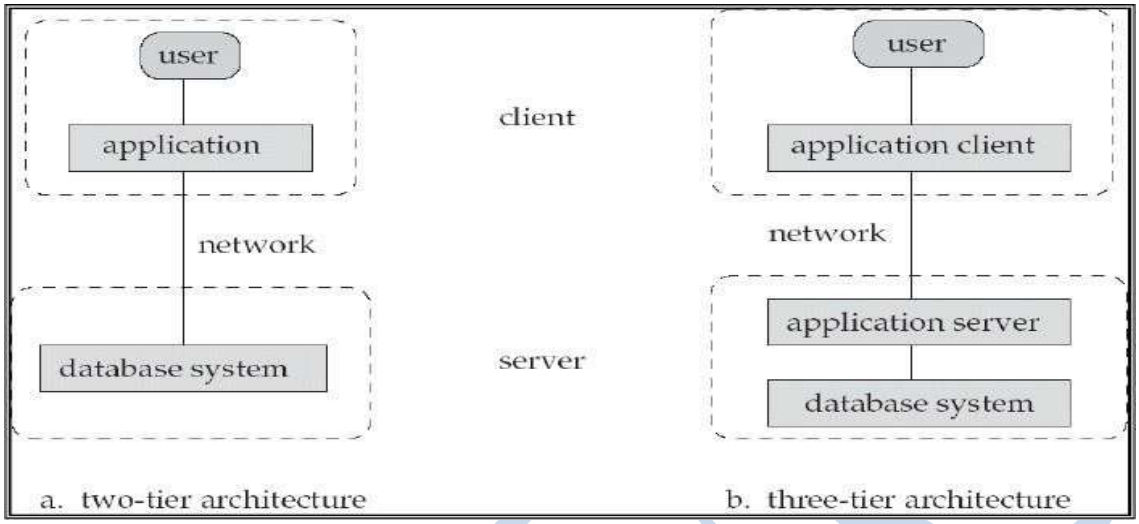
APEC

1.5 DATABASE SYSTEM STRUCTURE



1.5.1 APPLICATION ARCHITECTURES

- Two-tier architecture: E.g. client programs using ODBC/JDBC to communicate with a database
- Three-tier architecture: E.g. web-based applications and applications built using "middleware" .



1.6 DATABASE USERS

Users are differentiated by the way they expect to interact with the system

- **Application programmers** – interact with system through DML calls
- **Sophisticated users** – form requests in a database query language
- **Specialized users** – write specialized database applications that do not fit into the traditional data processing framework
- **Naïve users** – invoke one of the permanent application programs that have been written previously

– Examples, people accessing database over the web, bank tellers, clerical staff.

1.7 DATABASE ADMINISTRATOR

➢ Coordinates all the activities of the database system; the database administrator has a good understanding of the enterprise's information resources and needs.

➢ Database administrator's duties include:

- Schema definition
- Storage structure and access method definition
- Schema and physical organization modification
- Granting user authority to access the database
- Specifying integrity constraints
- Acting as liaison with users
- Monitoring performance and responding to changes in requirements

2. RELATIONAL DATA MODEL

- Relational Model
- Proposed by E.F. Codd in the early seventies.
- Most of the modern DBMS are relational.
- Simple and elegant model with mathematical basis.

- Led to the development of a theory of data dependencies and database design.
- Relational algebra operations – crucial role in query optimization & execution.
- Laid the foundation for the development of Tuple relational calculus and then Database standard SQL.

2.1 RELATIONAL MODEL

- Structure of Relational Databases
- Fundamental Relational- Algebra-Operations
- Additional Relational- Algebra-Operations
- Extended Relational-Algebra- Operations
- Null Values
- Modification of the Database

2.1.1 Example of Relational Model

account_number	*branch_name*	*balance*
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

2.1.2 Basic Structure

- Formally, given sets $D_1, D_2, \ldots D_n$ a **relation** r is a subset of $D_1 \times D_2 \times \ldots \times D_n$

  Thus, a relation is a set of n-tuples $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$
- **Example**: If customer_name = {Jones, Smith, Curry, Lindsay, …}

/* Set of all customer names */

- customer_street = {Main, North, Park, …} **/* set of all street names*/**
- customer_city = {Harrison, Rye, Pittsfield, …} **/* set of all city names */**

Then r = { (Jones, Main, Harrison),

(Smith, North, Rye),

(Curry, North, Rye),

(Lindsay, Park, Pittsfield)

}

is a relation over Customer_name x customer_street x customer_city

2.1.3 Attribute Types

- Each attribute of a relation has a name
- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are (normally) required to be **atomic**; that is, indivisible – **E.g.** the value of an attribute can be an account number, but cannot be a set of account numbers

- Domain is said to be atomic if all its members are atomic
- The special value null is a member of every domain
- The null value causes complications in the definition of many operations – We shall ignore the effect of null values in our main presentation and consider their effect later

2.1.4 Relation Schema

- $A_1, A_2, \ldots, A_n$ are attributes
- $R = (A_1, A_2, \ldots, A_n)$ is a relation schema

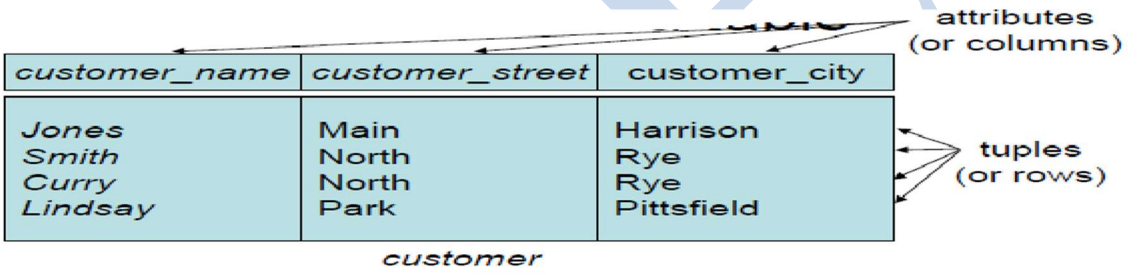
Example:
Customer_schema = (customer_name, customer_street, customer_city)

- r(R) denotes a relation r on the relation schema R

Example:
customer (Customer_schema)

2.1.5 Relation Instance

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



Database

- A database consists of multiple relations
- Information about an enterprise is broken up into parts, with each relation storing one part of the information

account: stores information about accounts
depositor: stores information about which customer owns which account
customer: stores information about customers

- Storing all information as a single relation such as bank(account_number, balance, customer_name, ..) resultsin – repetition of information.
- e.g., if two customers own an account (What gets repeated?) – the need for null values.
- e.g., to represent a customer without an account.
- Normalization deals with how to design relational schemas.

The Customer Relation

customer_name	customer_street	customer_city
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

The Depositor Relation

customer_name	account_number
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

1.2 CATALOG

The catalog is the place where all of the various schemes and all corresponding mappings are kept. In other words catalog contains detailed information, sometimes called Meta data.
Examples of such objects are indexes, Users, Integrity constraints.
For Example, The catalog in an SQl might include two system terms called table and column. The purpose of which is to describe the table in the database and column in those tables.
For the department and employees the database is as follows:

DEPT

DNO	DNAME	BUDGET
D1	A	1000
D2	B	120
D3	C	550

EMPLOYEE

ENO	ENAME	DNO	SALARY
E1	P	D1	40000
E2	Q	D1	20000
E3	R	D2	9500
E4	S	D3	12000

For this database the table and column might look in outline as follows:
Catalog for the departments and employees databases

Table

Table name	Column count	Row count
Dept	3	3
employee	4	4

Column

Table name	Column name
Dept	Dno
Dept	Dname
Dept	Budget
Employee	Eno
Employee	Ename
Employee	Dno
Employee	salary

Some users of the Dept's and employees declares wants to know exactly what columns from Dept contains Eg., (column where table_name = "Dept") { column_name}
Eg., (column where column_name ='eno') { table_name}

2.3 KEYS

- Let K ⊆ R
- K is a **superkey** of R if values for K are sufficient to identify a unique tuple of each possible relation r(R) –by "possible r" we mean a relation r that could exist in the enterprise we are modeling.

–**Example:** {customer_name, customer_street} and {customer_name} are both superkeys of Customer, if no two customers can possibly have the same name

- In real life, an attribute such as customer_id would be used instead of customer_name to uniquely identify customers, but we omit it to keep our examples small, and instead assume customer names are unique.

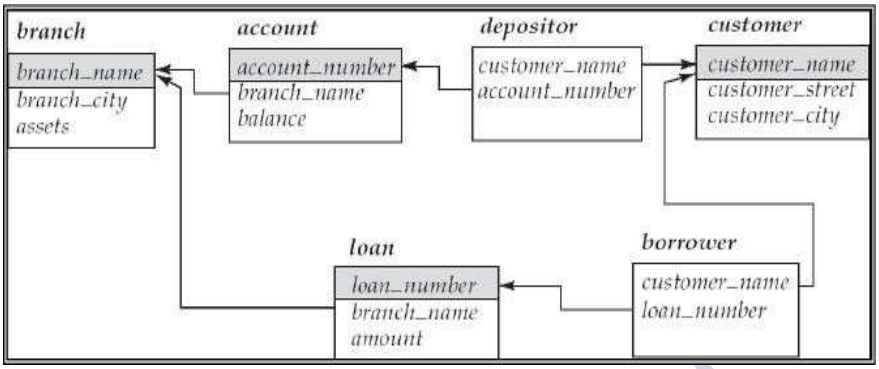
- K is a **candidate key** if K is minimal

Example: {customer_name} is a candidate key for Customer, since it is a superkey and no subset of it is a superkey.

- **Primary key:** a candidate key chosen as the principal means of identifying tuples within a relation – Should choose an attribute whose value never, or very rarely, changes. – E.g. email address is unique, but may change.

- **Foreign Keys:** A relation schema may have an attribute that corresponds to the primary key of another relation. The attribute is called a **foreign key**. – E.g. customer_name and account_number attributes of depositor are foreign keys to customer and account respectively.

– Only values occurring in the primary key attribute of the **referenced relation** may occur in the foreign key attribute of the **referencing relation.**



SCHEMA DIAGRAM

2.4 RELATIONAL ALGEBRA

- ❖ Procedural language
- ❖ Six basic operators
  1. select: σ
  2. project: ∏
  3. union: ∪
  4. set difference: –
  5. Cartesian product: x
  6. rename: ρ
- ❖ The operators take one or two relations as inputs and produce a new relation as a result.

2.4.1 SELACT OPERATION:

- Relation r

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

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

A	B	C	D
α	α	1	7
β	β	23	10

- Notation: $\sigma_p(r)$
- p is called the **selection predicate**
- Defined as:

$$\sigma_p(r) = \{t \mid t \in r \text{ and } p(t)\}$$

Where p is a formula in propositional calculus consisting of **terms** connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
Each **term** is one of:

<attribute> *op* <attribute> or <constant>

where *op* is one of: $=, \neq, >, \geq, <, \leq$

- Example of selection:

$$\sigma_{branch_name="Perryridge"}(account)$$

2.4.2 PROJECT OPERATION:

Relation r

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

$\Pi_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

- Notation:

$$\Pi_{A_1, A_2, \ldots, A_k}(r)$$

where A_1, A_2 are attribute names and r is a relation name.

- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets
- Example: To eliminate the *branch_name* attribute of *account*

$$\Pi_{account_number,\ balance}(account)$$

2.4.3 UNION OPERATION:

Relation r,s:

- Relations *r*, *s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- r ∪ s:

A	B
α	1
α	2
β	1
β	3

❖ Notation: r ∪ s

❖ Defined as: r∪ s = {t | t ∈ r or t ∈ s}

❖ For r ∪ s to be valid.

1. r, s must have the same **arity** (same number of attributes)
2. The attribute domains must be **compatible** (example: 2nd column of r deals with the same type of values as does the 2nd column of s)

❖ Example: to find all customers with either an account or a loan

$\Pi_{customer_name}$ (depositor) ∪ $\Pi_{customer_name}$ (borrower)

2.4.4 SET DIFFERENCE OPERATION:

- Relations *r*, *s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- *r* − *s*:

A	B
α	1
β	1

Notation r – s

- Defined as:

r – s = {t | t ∈ r **and** t ∉ s}

- Set differences must be taken between **compatible** relations.

– r and s must have the same arity– attribute domains of r and s must be compatible.

2.4.5 CARTESIAN PRODUCT:

- Relations *r*, *s*:

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

- *r* x *s*:

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

Notation r x s

- Defined as:

r x s = {t q | t ∈ r **and** q ∈ s}

- Assume that attributes of r(R) and s(S) are disjoint. (That is, $R \cap S = \varnothing$).
- If attributes of r(R) and s(S) are not disjoint, then renaming must be used.

2.4.6 RENAME OPEARTION:

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions.
- Allows us to refer to a relation by more than one name.
- Example:

$$\rho_x(E)$$

returns the expression E under the name X

- If a relational-algebra expression E has arity n, then

$$\rho_{x(A_1, A_2, \ldots, A_n)}(E)$$

returns the result of expression E under the name X, and with the attributes renamed to $A_1, A_2, \ldots, A_n$.

2.4.7 BANKING EXAMPLE:

branch (branch_name, branch_city, assets)
customer (customer_name, customer_street, customer_city)
account (account_number, branch_name, balance) loan (loan_number, branch_name, amount) depositor (customer_name, account_number) borrower (customer_name, loan_number)

EXAMPLE QUERIES:

- Find all loans of over $1200

$$\sigma_{amount > 1200}(loan)$$

- Find the loan number for each loan of an amount greater than $1200

$$\prod_{loan_number}(\sigma_{amount > 1200}(loan))$$

- Find the names of all customers who have a loan, an account, or both, from the bank

$$\prod_{customer_name}(borrower) \cup \prod_{customer_name}(depositor)$$

- Find the names of all customers who have a loan at the Perryridge branch.

$$\prod_{customer_name}(\sigma_{branch_name="Perryridge"}(\sigma_{borrower.loan_number = loan.loan_number}(borrower \times loan)))$$

- Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.

$$\prod_{customer_name}(\sigma_{branch_name = "Perryridge"}(\sigma_{borrower.loan_number = loan.loan_number}(borrower \times loan))) - \prod_{customer_name}(depositor)$$

- Find the names of all customers who have a loan at the Perryridge branch.

Query 1

$$\prod_{customer_name}(\sigma_{branch_name = "Perryridge"}(\sigma_{borrower.loan_number = loan.loan_number}(borrower \times loan)))$$

Query 2

$\prod_{customer_name}$ ($\sigma_{loan.loan_number = borrower.loan_number}$ (($\sigma_{branch_name=}$ "Perryridge" (loan)) x borrower))

- Find the largest account balance
  - Strategy:
- Find those balances that are not the largest

– Rename account relation as d so that we can compare each account balance with all others

- Use set difference to find those account balances that were not found in the earlier step.
  - The query is:

$\prod_{balance}$(account) - $\prod_{account.balance}$ ($\sigma_{account.balance < d.balance}$ (account x ρ_d (account)))

2.4.8 FORMAL DEFINITION:

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation
- Let E_1 and E_2 be relational-algebra expressions; the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - E_1 x E_2
  - $\sigma_p (E_1)$, P is a predicate on attributes in E_1
  - $\prod_s(E_1)$, S is a list consisting of some of the attributes in E_1
  - $\rho_x (E_1)$, x is the new name for the result of E_1

2.4.9 ADDITIONAL OPERATIONS:

We define additional operations that do not add any power to the relational algebra, but that simplify common queries.

- Set intersection
- Natural join
- Division
- Assignment

2.4.9.1 SET-INTERSECTION OPERATION

- Notation: $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r$ **and** $t \in s \}$
- Assume:
  – r, s have the same arity
  – attributes of r and s are compatible
- Note: $r \cap s = r - (r - s)$

Relation:

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

A	B
α	2

$r \cap s$

2.4.9.2 NATURAL JOIN OPERATION:

- Notation: r ⋈ s
- Let *r* and *s* be relations on schemas *R* and *S* respectively. Then, ⋈ s is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from *r* and t_s from *s*.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple *t* to the result, where
    - *t* has the same value as t_r on *r*
    - *t* has the same value as t_s on *s*
- Example:
  $R = (A, B, C, D)$
  $S = (E, B, D)$
  - Result schema = (A, B, C, D, E)
  - $r_{\bowtie}$ s is defined as:
    $\Pi_{r.A, r.B, r.C, r.D, s.E} (\sigma_{r.B = s.B \wedge r.D = s.D} (r \times s))$

Relation r, s

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	∈

s

- r ⋈ s

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

2.4.9.3 DIVISION OPERATION:

- Notation: r ÷ s
- Suited to queries that include the phrase "for all".
- Let r and s be relations on schemas R and S respectively where
  - $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$
  - $S = (B_1, \ldots, B_n)$

  The result of r ÷ s is a relation on schema $R - S = (A_1, \ldots, A_m)$

$r \div s = \{ t | t \in \prod_{R-S}(r) \wedge A\ u \in s\ (tu \in r) \}$

Where tu means the concatenation of tuples t and u to produce a single tuple.

Relation r,s:

r

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
δ	4
∈	6
∈	1
β	2

s

B
1
2

r ÷ s:

A
α
β

Another Division Operation:

Relation r,s:

r

A	B	C	D	E
α	a	α	a	1
α	a	γ	a	1
α	a	γ	b	1
β	a	γ	a	1
β	a	γ	b	3
γ	a	γ	a	1
γ	a	γ	b	1
γ	a	β	b	1

s

D	E
a	1
b	1

r ÷ s:

A	B	C
α	a	γ
γ	a	γ

- Property
  - Let q = r ÷ s
  - Then q is the largest relation satisfying q x s ⊆ r
- Definition in terms of the basic algebra operation

Let r(R) and s(S) be relations, and let S ⊆ R

$r \div s = \prod_{R-S}(r) - \prod_{R-S}((\prod_{R-S}(r) \times s) - \prod_{R-S,S}(r))$ To see why

- $\prod_{R-S,S}(r)$ simply reorders attributes of r
- $\prod_{R-S}(\prod_{R-S}(r) \times s) - \prod_{R-S,S}(r))$ gives those tuples t in $\prod_{R-S}(r)$ such that for some tuple u ∈ s, tu ∉ r.

2.4.9.4 ASSIGNMENT OPERATION:

The assignment operation (←) provides a convenient way to express complex queries.

- Write query as a sequential program consisting of a series of assignments followed by an expression whose value is

displayed as a result of the query.
- Assignment must always be made to a temporary relation variable.

- **Example:** Write r ÷ s as

$$temp1 \leftarrow \prod_{R-S} (r)$$
$$temp2 \leftarrow \prod_{R-S} ((temp1 \times s) - \prod_{R-S,S} (r))$$
$$result = temp1 - temp2$$

- The result to the right of the ← is assigned to the relation variable on the left of the ←.
- May use variable in subsequent expressions.

2.4.9.5 BANK EXAMPLE QUERIES:

- Find the names of all customers who have a loan and an account at bank.

$$\Pi_{customer_name} (borrower) \cap \Pi_{customer_name} (depositor)$$

- Find the name of all customers who have a loan at the bank and the loan amount

$$\Pi_{customer_name,\ loan_number,\ amount} (borrower \bowtie loan)$$

- Find all customers who have an account from at least the "Downtown" and the Uptown" branches.
  - Query 1

$$\Pi_{customer_name} (\sigma_{branch_name = \text{"Downtown"}} (depositor \bowtie account)) \cap \Pi_{customer_name} (\sigma_{branch_name = \text{"Uptown"}} (depositor \bowtie account))$$

  - Query 2

$$\Pi_{customer_name,\ branch_name} (depositor \bowtie account) \div \rho_{temp(branch_name)} (\{(\text{"Downtown"}), (\text{"Uptown"})\})$$

Note that Query 2 uses a constant relation.

- Find all customers who have an account at all branches located in Brooklyn city.

$$\Pi_{customer_name,\ branch_name} (depositor \bowtie account) \div \Pi_{branch_name} (\sigma_{branch_city = \text{"Brooklyn"}} (branch))$$

2.5 EXTENDED RELATIONAL ALGEBRA OPERATION:

- ❖ Generalized Projection
- ❖ Aggregate Functions
- ❖ Outer Join
- ❖

2.5.1 GENERALIZED PROJECTION

❖ Extends the projection operation by allowing arithmetic functions to be used in the projection list.

$$\Pi_{F_1, F_2, \ldots, F_n}(E)$$

- ❖ E is any relational-algebra expression
- ❖ Each of F_1, F_2, …, F_n are are arithmetic expressions involving constants and attributes in the schema of E.
- ❖ Given relation credit_info(customer_name, limit, credit_balance), find how much more each person can spend:

$\Pi_{\text{customer_name, limit} - \text{credit_balance}}$ (credit_info)

2.5.2 AGGREGATE FUNCTIONS:

- ➢ **Aggregation function** takes a collection of values and returns a single value as a result.
  - **avg**: average value
  - **min**: minimum value
  - **max**: maximum value
  - **sum**: sum of values
  - **count**: number of values

- ➢ **Aggregation operation in Relational algebra**

$$G_1, G_2, \ldots, G_n \; \mathcal{G}_{F_1(A_1), F_2(A_2, \ldots, F_n(A_n)}(E)$$

E is any relational-algebra expression

- G_1, G_2 …, G_n is a list of attributes on which to group (can be empty)
- Each F_i is an aggregate function
- Each A_i is an attribute name

Relation r:

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

$\mathcal{G}_{\text{sum(c)}}(r)$

sum(c)
27

Relation account grouped by **branch-name**:

branch_name	account_number	balance
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$branch_name\ \mathcal{G}\ \mathbf{sum}(balance)\ (account)$

branch_name	sum(balance)
Perryridge	1300
Brighton	1500
Redwood	700

2.5.3 OUTER JOIN:

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses null values:
  - null signifies that the value is unknown or does not exist
  - All comparisons involving null are (roughly speaking) **false** by definition.
- We shall study precise meaning of comparisons with nulls later.

Relation Loan

loan_number	branch_name	amount
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

Relation Borrower

customer_name	loan_number
Jones	L-170
Smith	L-230
Hayes	L-155

Join

$loan \bowtie borrower$

loan_number	branch_name	amount	customer_name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

Left Outer Join

$loan \ ⟕ \ borrower$

loan_number	branch_name	amount	customer_name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*

Right Outer Join

loan ⟕ *borrower*

loan_number	branch_name	amount	customer_name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

Full Outer Join

loan ⟗ *borrower*

loan_number	branch_name	amount	customer_name
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

2.5.4 NULL VALUES:

- It is possible for tuples to have a null value, denoted by null, for some of their attributes
- **null** signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving null is null.
- Aggregate functions simply ignore null values (as in SQL).
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same (as in SQL).
- Comparisons with null values return the special truth value: unknown
  - If false was used instead of unknown, then not (A < 5) would not be equivalent to A >= 5
- Three-valued logic using the truth value unknown:
  - OR: (unknown **or** true) = true, (unknown **or** false)= unknown (unknown **or** unknown) = unknown
  - AND:(true **and** unknown) = unknown, (false **and** unknown)= false, (unknown **and** unknown) = unknown
  - NOT: (**not** unknown) = unknown
  - In SQL "P **is unknown**" evaluates to true if predicate P evaluates to unknown
- Result of select predicate is treated as false if it evaluates to unknown

2.7 MODIFICATION OF DATABASE:

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating
- All these operations are expressed using the assignment operator.

2.7.1 DELETION

- A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.
- Can delete only whole tuples; cannot delete values on only particular attributes
- A deletion is expressed in relational algebra by:
  r ← r – E where r is a relation and E is a relational algebra query.

DELETION EXAMPLE:

- Delete all account records in the Perryridge branch.

$$account \leftarrow account - \sigma_{branch_name = \text{"Perryridge"}}(account)$$

- Delete all loan records with amount in the range of 0 to 50

$$loan \leftarrow loan - \sigma_{amount \geq 0 \text{ and } amount \leq 50}(loan)$$

- Delete all accounts at branches located in Needham.

$$r_1 \leftarrow \sigma_{branch_city = \text{"Needham"}}(account \bowtie branch)$$
$$r_2 \leftarrow \Pi_{account_number,\ branch_name,\ balance}(r_1)$$
$$r_3 \leftarrow \Pi_{customer_name,\ account_number}(r_2 \bowtie depositor)$$
$$account \leftarrow account - r_2$$
$$depositor \leftarrow depositor - r_3$$

2.7.2 INSERTION

- To insert data into a relation, we either:
  - specify a tuple to be inserted
  - write a query whose result is a set of tuples to be inserted.
- In relational algebra, an insertion is expressed by:

  $r \leftarrow r \cup E$ where r is a relation and E is a relational algebra expression.
- The insertion of a single tuple is expressed by letting E be a constant relation containing one tuple.

INSERTION EXAMPLE:

- Insert information in the database specifying that Smith has $1200 in account A-973 at the Perryridge branch.

  account $\leftarrow$ account $\cup$ {("A-973", "Perryridge", 1200)}

  depositor $\leftarrow$ depositor $\cup$ {("Smith", "A-973")}
- Provide as a gift for all loan customers in the Perryridge branch, a $200 savings account. Let the loan number serve as the account number for the new savings account.

$$r_1 \leftarrow (\sigma_{branch_name = \text{"Perryridge"}}(borrower \bowtie loan))$$
$$account \leftarrow account \cup \Pi_{loan_number,\ branch_name,\ 200}(r_1)$$
$$depositor \leftarrow depositor \cup \Pi_{customer_name,\ loan_number}(r_1)$$

2.7.3 UPDATION:

- A mechanism to change a value in a tuple without charging *all* values in the tuple
- Use the generalized projection operator to do this task

$$r \leftarrow \Pi_{F_1, F_2, \ldots, F_n}(r)$$

- Each F_i is either
  - the I^{th} attribute of r, if the I^{th} attribute is not updated, or,
  - if the attribute is to be updated F_i is an expression, involving only constants and the attributes of r, which gives the new value for the attribute

Example:

- Make interest payments by increasing all balances by 5 percent.

$$\text{account} \leftarrow \Pi_{\text{account_number, branch_name, balance} * 1.05}(\text{account})$$

- Pay all accounts with balances over $10,000 6 percent interest and pay all others 5 percent

$$\text{account} \leftarrow \Pi_{\text{account_number, branch_name, balance} * 1.06}(\sigma_{\text{BAL} > 10000}(\text{account})) \cup \Pi_{\text{account_number, branch_name, balance} * 1.05}(\sigma_{\text{BAL} \leq 10000}(\text{account}))$$

2.8 SQL FUNDAMENTALS

INTRODUCTION TO SQL

SQL (structured query language) is a database computer language designed for the retrieval and management of data in relational database systems (RDBMS), database scheme creation and modification, and database object access control management.

FEATURES OF SQL

SQL is both an easy –to-understand language and a comprehensive tool for managing data.

- High-level .English-like structure
- Interactive, ad hoc queries
- Programmatic database access
- Multiple views of data
- Complete database language
- Dynamic data definition
- Client/server architecture

2.8.1 TYPES OF SQL COMMANDS

SQL is used to control all of the functions that a DBMS provides for its users, including:

- Data definition, SQL lets a user define the structure and organization of the stored data and relationship among the stored data items.
- Data retrivel, SQL allows a user or an application program to retrieve stored data from the database and use it.
- Data manipulation, SQL allows a user or an application program to update the database by adding new data, removing old data, and modifying previously stored data.
- Access control, SQL can be used to restrict a user's ability to retrieve, add, and modify data, protecting stored data against unauthorized access.

- Data sharing, SQL is used to coordinate data sharing by concurrent users, ensuring that they do not interface with one another.
- Data integrity, SQL defines integrity constraints in the database, protecting it from corruption due to inconsistent updates or system failures.

2.8.2 DATA DEFINITION LANGUAGES (DDL)

A Data Definition Language (DDL) statement are used to define the database structure or schema.

DDL commands:

- CREATE
- ALTER
- DROP
- RENAME
- TRUNCATE

SYNTAX'S OF COMMANDS

Create Table:

To make a new database, table, index, or stored query. A create statement in SQL creates an object inside of a relational database management system (RDBMS).

```
CREATE TABLE table_name
(
Column_name1 data_type,
Column_name2 data_type …
);
```

Alter A Table:

To modify an existing database object. Alter the structure of the database.

To add a column in a table

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

To delete a column in a table

```
ALTER TABLE table_nmae DROP column column_nmae;
```

Drop Table:

Delete Objects from the Database

```
DROP TABLE table_name;
```

Truncate Table:

Remove all records from a table, including all spaces allocated for the records are removed.

```
TRUNCATE TABLE table_name;
```

2.8.3 DATA MANIPULATION LANGUAGE (DML)

Data manipulation language allows the users to query and manipulate data in existing schema in object.It allows following data to insert,delete ,update and recovery data in schema object.

DML COMMANDS:

- INSERT
- UPDATE
- DELETE
- SELECT

QUERY:

Query is a statement in the DML that request the retrieval of data from database.

- The portion of the DML used in a Query is called Query language. The SELECT statement is used to query a database

Syntax Of Commands

INSERT:

Values can be inserted into table using insert commands. There are two types of insert commands.They are multiple value insert commands (using '&' symbol)single value insert command(without using '&'symbol)

Syntax:

INSERT INTO table_name VALUES (value1, value2, value3,…..);

(OR)

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

UPDATE:

This allows the user to update the particular column value using the where clause condition.

Syntax:

UPDATE<tablename>SET<col1=value>WHERE<col=value>;

DELETE:

This allows you to delete the particular column values using where clause condition.

Syntax:

DELETE FROM<tablename>WHERE<condition>;

SELECT:

The select statement is used to query a database .This statement is used to retrieve the information from the database. The SELECT statement can be used in many ways. They are:

1. **Selecting some columns** :

   To select specified number of columns from the table the Following command is used.

   Syntax:

   SELECT column name FROM table_name;

2. **Query All Columns**:

   To select all columns from the table * is used instead of column names.

   Syntax:

   SELECT * FROM table_name;

3. **Select using DISTINCT:**

   The DISTINCT keyword is used to return only different values (i.e.) this command does not select the duplicate values from the table .

   Syntax:

   SELECT DISTINCT column name(s) FROM table_name;

4. **Select using IN:**

   If you want to get the rows which contain certain values,the best way to do it is to use the IN conditional expression.

   Syntax:

   SELECT column name(s) FROM table_name WHERE Column name IN (value1,value2,……,value n);

5. **Select using BETWEEN**:

BETWEEN can be used to get those items that fall within a range.

Syntax:

SELECT column name FROM table_name WHERE
Column name BETWEEN value1 AND value2;

6. Renaming:

The select statement can be used to rename either a column or the entire table.

Syntax:

Renaming a column:

SELECT columnname AS new name FROM table_name;

Renaming a table:

SELECT columnname FROM table_name AS newname;

7. **Sorting**:

The select statement with the **order by Clause** is used to sort the contents Table either in ascending or descending order.

Syntax:

SELECT columnname FROM table_name WHERE
Condition ORDER BY columnname ASC/DESC;

8. **To Select by matching some patterns**:

The select statement along with **like clause** I is used to match strings. The **like** condition is used to specify a search pattern in a column.

Syntax:

SELECT column name FROM table_name WHERE Column name LIKE "% or-";

% : Matches any sub string.

- : Matches a single character.

9. SELECT INTO statement:

The SELECT INTO statement is most often used to create backup copies of tables or for archiving records.

Syntax:

SELECT Column_name(s) INTO variable_name(s) FROM table_name WHERE condition.

10. **To Select NULL values**:

We can use the SELECT statement to select the 'null' values also.
For retrieving roes where some of the columns have been defined as NULLs there is a special comparison operator of the form IS[NOT]NULL.

Syntax:

SELECT column name FROM table_name WHERE Column name IS NULL;

11. **Select using AND, OR, NOT**:

We can combine one or more conditions in a SELECT statement using the logical operators AND, OR, NOT.

Syntax:

SELECT column name FROM table_name WHERE Condition1 LOGICAL OPERATOR condition2;

Advantages of DML:

i. Table security

ii. Simple commands

2.9 INTEGRITY CONSTRAINS:

It is a mechanism used to present inva data entry into the table.

Types:

(i).Domain integrity constraints

(ii).Entity integrity constraints

(iii).Referential integrity constraints

2.9.1 Domain Integrity Constraints

These constraints set a range and any violations that take place will present the user from performing the manipulation that can cores the branch.

The two types in domain integrity constraints are

(i) Not Null constraint:

User has to provide a value for the column .

E.g. create table emp (eid number(5),constraint emp nothing ename varchar2(10));

Its used to enforce that the particular column will not accept null values.

(ii) Check constraints

To specify the conditions that each row must satisfy:

e.g.Create table branch(branch_name char(15),branch city char assets integers not null,check (assets>=0));

2.9.2 Entity integrity constraints

***unique constraints**

Unique constraints is used to prevent duplication of values. It allows null value for the column.

e.g.create table employee(ename varchar2(10),eid number (5),constraint eeid unique)

Primary key constraint:

This is used for to prevent duplication of values .It will not allow null values for the column.

e.g.create table employee(ename varchar2(10),eid number(5),constraint eeid primary key, address varchar2(10));

2.9.4 Referential integrity constraint

To establish a parent child relationship between two tables having a common column we can use referential integrity constraints.

To implement this , we should define the column in the parent table as primary key and the same column in the child table as a foreign key referring to the corresponding parent entry.

Create table employee(ename varchar2(10),eno number(5),primary key dno number(5),constraint fdno references department(dno),salary number(5));

Fdno - constraint name

2.12 TRIGGERS

- A **trigger** is a statement that is executed automatically by the system as a side effect of a modification to the database.
- To design a trigger mechanism, we must:
  - Specify the conditions under which the trigger is to be executed.
  - Specify the actions to be taken when the trigger executes.

▪ Triggers introduced to SQL standard in SQL, but supported even earlier using non-standard syntax by most databases.

2.12.1 TRIGGER EXAMPLE

Trigger Example in SQL

create trigger *overdraft-trigger* **after update on**
account
referencing new row as *nrow* **for each row**
when *nrow.balance* < 0
begin atomic
insert into *borrower*
(select *customer-name, account-number*
from *depositor*
where *nrow.account-number* =

▪ Suppose that instead of allowing negative account balances, the bank deals with overdrafts by
- setting the account balance to zero
- creating a loan in the amount of the overdraft
- giving this loan a loan number identical to the account number of the overdrawn account

▪ The condition for executing the trigger is an update to the account relation that results in a negative balance value.

2.12.2 Triggering Events and Actions in SQL

▪ Triggering event can be **insert**, **delete** or **update**
▪ Triggers on update can be restricted to specific attributes
- **E.g. create trigger** overdraft-trigger **after update of** balance **on** account

▪ Values of attributes before and after an update can be referenced
- **referencing old row as** : for deletes and updates
- **referencing new row as :** for inserts and updates

▪ Triggers can be activated before an event, which can serve as extra constraints. E.g. convert blanks to null.

2.12.3 Statement Level Triggers

▪ Instead of executing a separate action for each affected row, a single action can be executed for all rows affected by a transaction

- Use **for each statement** instead of **for each row**
- Use **referencing old table** or **referencing new table** to refer to temporary tables (called **transition tables**) containing the affected rows
- Can be more efficient when dealing with SQL statements that update a large number of rows

External World Actions

▪ We sometimes require external world actions to be triggered on a database update

- E.g. re-ordering an item whose quantity in a warehouse has become small, or turning on an alarm light,

▪ Triggers cannot be used to directly implement external-world actions, BUT

- Triggers can be used to record actions-to-be-taken in a separate table
- Have an external process that repeatedly scans the table, carries out external-world actions and deletes action from table

▪ E.g. Suppose a warehouse has the following tables

- inventory(item, level): How much of each item is in the warehouse
- minlevel(item, level) : What is the minimum desired level of each item
- reorder(item, amount): What quantity should we re-order at a time
- orders(item, amount) : Orders to be placed (read by external process)

```
create trigger reorder-trigger after update of amount on
inventory
referencing old row as orow, new row as nrow
for each row
    when nrow.level <= (select level
                        from minlevel
                        where minlevel.item =
orow.item)
        and orow.level > (select level
                          from minlevel
                          where minlevel.item =
```

2.12.4 When Not To Use Triggers

▪ Triggers were used earlier for tasks such as

- maintaining summary data (e.g. total salary of each department)

- Replicating databases by recording changes to special relations (called **change** or **delta** relations) and having a separate process that applies the changes over to a replica

- There are better ways of doing these now:
  - Databases today provide built in materialized view facilities to maintain summary data
  - Databases provide built-in support for replication
- Encapsulation facilities can be used instead of triggers in many cases
  - Define methods to update fields
  - Carry out actions as part of the update methods instead of through a trigger

2.13 SECURITY

- **Security** - protection from malicious attempts to steal or modify data.
  - **Database system level**
    - Authentication and authorization mechanisms to allow specific users access only to required data
    - We concentrate on authorization in the rest of this chapter
  - **Operating system level**
    - Operating system super-users can do anything they want to the database! Good operating system level security is required.
    - Eavesdropping (unauthorized reading of messages)
    - Network level: must use encryption to prevent
    - Masquerading (pretending to be an authorized user or sending messages supposedly from authorized users)
  - **Physical level**
    - Physical access to computers allows destruction of data by intruders; traditional lock-and-key security is needed
    - Computers must also be protected from floods, fire, etc.
    - –More in Chapter 17 (Recovery)
  - **Human level**
    - Users must be screened to ensure that an authorized users do not give access to intruders
    - Users should be trained on password selection and secrecy
    -

2.12.1 Authorization

Forms of authorization on parts of the database:

- **Read authorization** - allows reading, but not modification of data.
- **Insert authorization** - allows insertion of new data, but not modification of existing data.
- **Update authorization** - allows modification, but not deletion of data.
- **Delete authorization** - allows deletion of data

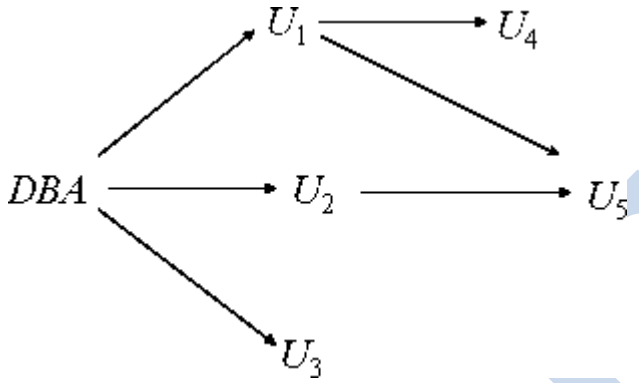
Forms of authorization to modify the database schema:

- **Index authorization** - allows creation and deletion of indices.
- **Resources authorization** - allows creation of new relations.
- **Alteration authorization** - allows addition or deletion of attributes in a relation.
- **Drop authorization** - allows deletion of relations.

Authorization and Views

- Users can be given authorization on views, without being given any authorization on the relations used in the view definition
- Ability of views to hide data serves both to simplify usage of the system and to enhance security by allowing users access only to data they need for their job
- A combination or relational-level security and view-level security can be used to limit a user's access to precisely the data that user needs.

2.12.2 Granting of Privileges



- The passage of authorization from one user to another may be represented by an authorization graph.
- The nodes of this graph are the users.
- The root of the graph is the database administrator.
- Consider graph for update authorization on loan.
- An edge $U_i \rightarrow U_j$ indicates that user U_i has granted update authorization on loan to U_j.

Authorization Grant Graph

- Requirement: All edges in an authorization graph must be part of some path originating with the database administrator
- If DBA revokes grant from U_1:
  - Grant must be revoked from U_4 since U_1 no longer has authorization
  - Grant must not be revoked from U_5 since U_5 has another authorization path from DBA through U_2
- Must prevent cycles of grants with no path from the root:
  - DBA grants authorization to U_7
  - U7 grants authorization to U_8
  - U8 grants authorization to U_7
  - DBA revokes authorization from U_7
- Must revoke grant U_7 to U_8 and from U_8 to U_7 since there is no path from DBA to U_7 or to U_8 anymore.

Security Specification in SQL

- The grant statement is used to confer authorization

  grant <privilege list>
  on <relation name or view name> to <user list>

- <user list> is:
  - a user-id
  - public, which allows all valid users the privilege granted
  - A role (more on this later)

- Granting a privilege on a view does not imply granting any privileges on the underlying relations.
- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).

Privileges in SQL

- **select:** allows read access to relation,or the ability to query using the view
  - Example: grant users U_1, U_2, and U_3 **select** authorization on the branch relation:
    grant select on branch **to** U_1, U_2, U_3
- **insert**: the ability to insert tuples
- **update**: the ability to update using the SQL update statement
- **delete**: the ability to delete tuples.
- **references**: ability to declare foreign keys when creating relations.
- **usage**: In SQL-92; authorizes a user to use a specified domain
- **all privileges**: used as a short form for all the allowable privileges

Privilege To Grant Privileges

- **with grant option**: allows a user who is granted a privilege to pass the privilege on to other users.
  - Example:
    grant select on branch **to** U_1 **with grant option**

gives U_1 the **select** privileges on branch and allows U_1 to grant this privilege to others

Roles

- Roles permit common privileges for a class of users can be specified just once by creating a corresponding "role"
- Privileges can be granted to or revoked from roles, just like user
- Roles can be assigned to users, and even to other roles
- SQL:1999 supports

  roles **create role** teller
  create role manager

  grant select on branch **to** teller
  grant update (balance**) on** account **to** teller
  grant all privileges on account **to** manager
  grant teller **to** manager
  grant teller **to** alice, bob
  grant manager **to** avi

Revoking Authorization in SQL

- The **revoke** statement is used to revoke authorization.
  revoke<privilege list>
  on <relation name or view name> **from** <user list> [**restrict**|**cascade**]
- **Example**:
  revoke select on branch **from** U_1, U_2, U_3 **cascade**
- Revocation of a privilege from a user may cause other users also to lose that privilege; referred to as cascading of the **revoke**.

- We can prevent cascading by specifying **restrict**:
  revoke select on branch **from** U_1, U_2, U_3 **restrict**
  With **restrict**, the **revoke** command fails if cascading revokes are required.
- <privilege-list> may be **all to** revoke all privileges the revokee may hold.
- If <revokee-list> includes **public** all users lose the privilege except those granted it explicitly.
- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation.
- All privileges that depend on the privilege being revoked are also revoked.

Limitations of SQL Authorization

- SQL does not support authorization at a tuple level
  - E.g. we cannot restrict students to see only (the tuples storing) their own grades
- With the growth in Web access to databases, database accesses come primarily from application servers.
  - End users don't have database user ids, they are all mapped to the same database user id
- All end-users of an application (such as a web application) may be mapped to a single database user
- The task of authorization in above cases falls on the application program, with no support from SQL
  - Benefit: fine grained authorizations, such as to individual tuples, can be implemented by the application.
  - Drawback: Authorization must be done in application code, and may be dispersed all over an application
  - Checking for absence of authorization loopholes becomes very difficult since it requires reading large amounts of application code

Audit Trails

- An audit trail is a log of all changes (inserts/deletes/updates) to the database along with information such as which user performed the change, and when the change was performed.
- Used to track erroneous/fraudulent updates.
- Can be implemented using triggers, but many database systems provide direct support.

Encryption

- Data may be encrypted when database authorization provisions do not offer sufficient protection.
- **Properties of good encryption technique**:
  - Relatively simple for authorized users to encrypt and decrypt data.
  - Encryption scheme depends not on the secrecy of the algorithm but on the secrecy of a parameter of the algorithm called the encryption key.
  - Extremely difficult for an intruder to determine the encryption key.
- Data Encryption Standard (DES) substitutes characters and rearranges their order on the basis of an encryption key which is provided to authorized users via a secure mechanism. Scheme is no more secure than the key transmission mechanism since the key has to be shared.
- Advanced Encryption Standard (AES) is a new standard replacing DES, and is based on the Rijndael algorithm, but is also dependent on shared secret keys
- Public-key encryption is based on each user having two keys:
  - public key – publicly published key used to encrypt data, but cannot be used to decrypt data
  - private key -- key known only to individual user, and used to decrypt data. Need not be transmitted to the site doing encryption.
    Encryption scheme is such that it is impossible or extremely hard to decrypt data given only the public key.

- The RSA public-key encryption scheme is based on the hardness of factoring a very large number (100's of digits) into its prime components.

Authentication

- Password based authentication is widely used, but is susceptible to sniffing on a network
- **Challenge-response** systems avoid transmission of passwords
  - DB sends a (randomly generated) challenge string to user
  - User encrypts string and returns result.
  - DB verifies identity by decrypting result
  - Can use public-key encryption system by DB sending a message encrypted using user's public key, and user decrypting and sending the message back
- **Digital signatures** are used to verify authenticity of data
  - E.g. use private key (in reverse) to encrypt data, and anyone can verify authenticity by using public key (in reverse) to decrypt data. Only holder of private key could have created the encrypted data.
  - Digital signatures also help ensure **nonrepudiation:** sender cannot later claim to have not created the data

Digital Certificates

- **Digital certificates** are used to verify authenticity of public keys.
- Problem: when you communicate with a web site, how do you know if you are talking with the genuine web site or an imposter?
  - Solution: use the public key of the web site
  - Problem: how to verify if the public key itself is genuine?
- **Solution**:
  - Every client (e.g. browser) has public keys of a few root-level **certification authorities**
  - A site can get its name/URL and public key signed by a certification authority: signed document is called a **certificate**
  - Client can use public key of certification authority to verify certificate
  - Multiple levels of certification authorities can exist. Each certification authority
    - presents its own public-key certificate signed by a higher level authority, and
    - Uses its private key to sign the certificate of other web sites/authorities

2.13 EMBEDED SQL

- The SQL standard defines embedding of SQl in a variety of programming languages such as C, Java, and cobol.
- A language to which SQL quries are embedded is refered to as a host language.
- EXEC SQL statement is used to identify embedded SQL request to the preprocessor.
  EXEC SQL< embedded SQl statement> END_EXEC.
  Note: this varies by language (for example Java embedding uses #SQL { … };

From within a host language find the names and cities of customers with more than the variable amount dollars in same account.

- Specify the Query in SQL and declare a cursor for.
- Advantages:

1. Efficient in CPU usage.
2. Efficient way of merging the strength of two programming language.
3. Transparent to the application programmer.

```
EXEC SQL
        Declare c cursor for
        Select customer_name, customer_city
        From depositor, customer, account
        Where depositer.customer_name=customer.customer_name
        And depositer.acc_no=account.acc_no
        And account_bal>:amount
END_EXEC
```

The variable C in above example is called a cursor for the query

i. Open cursor statement:
The open statement causes the query to be evaluated
EXEC SQL open c END_EXEC

ii. Fetch statement:
The fetch statement causes the values of one tuple in the query result to be placed to on host language variable
EXEC SQL fetch c into: cn1: cc END EXEC
Variable to hold the name and city value---cn1-customer name--- cc- customer city
A single fetch request returns only one tuple . To obtain all tuples, it must contain a iterative loop.

iii. Close statement:
The close statement causes the database system to delete the temporary relation that holds the result of the query
EXEC SQL close c END_EXEC

2.14 DYNAMIC SQL:

Dynamic SQL allow programs to construct and submit SQl queries at runtime.

Example: char * sqlprog = : "update account set balance=balance * 1.05 where account_number=?"
EXEC SQL prepare dynprog from:sqlprog;
Char account[10]="A-10";
EXEC SQL execute dynprog using: account;
The dynamic SQL program contains a '?', which is a place holder for a value that is provided when the SQL program is executed.

2.16 VIEWS

- **In some cases, it is not desirable for all users to see the entire logical model (i.e., all the actual relations stored in the database.)**
- **Consider a person who needs to know a customer's loan number but has no need to see the loan amount. This person should see a relation described, in the relational algebra, by**

$$\Pi_{\text{Customer-name, loan-number}} \text{ (borrower} \quad \text{loan)}$$

- **Any relation that is not of the conceptual model but is made visible to a user as a "virtual relation" is called a view.**

View Definition

A View is a virtual relation that does not necessarily exist in the database but can be produced upon request by a particular user, at the time of request.

Creating a View:

The format of the CREATE VIEW statement is,

Syntax:

CREATE VIEW viewcolumn [(newcolumn name [,…])] AS subselect [WITH[CASCADE/LOCAL]CHECK OPTION]

The subselect is known as the defining query, if with check option is specified, SQL ensures that if a row fails to satisfy the where clause of the getting query of the view, it is not added to the underlying base table of the view.

Removing A View:

A view is removed from the database with the DROP VIEW statement.

Syntax:

DROP VIEW viewname [RESTRICT/CASCADE]

If CASCADE is specified, DROP VIEW deletes all related dependent objects, all objects that reference the view.

If RESTRICT is specified and there are any other objects that depend for their existence on the continued existence of the view being dropped, the command is rejected. The default setting is RESTRICT.

A horizontal view restricts a user's access to selected rows of one or more tables. A vertical view restricts a user's access to selected columns of one or more tables.

Updatable View:

For a view to be updatable, the DBMS must be able to trace any row or column back to its row or column in the source table.

Examples

Create View

```
create view empview as select empid,empname,dept from emp;
```

Select View

```
select * from empview;
```

Insert Data Into View

```
SQL> insert into empview values(&id,'&name','&dept');
```

Update View Table

```
SQL> update empview set empname='kumaran' where empid=5;
```