Inheritance

Two types of inheritance available

I **”is a”** inheritance. This shows that the subtype IS a member of the

supertype.

I **”is part of”** inheritance. This shows that the supertype contains, or

is made up of members of the subtypes.

All attributes of the supertype entity are inherited by the subtype

entities. The identifier of the subtypes will be the same as the supertype.

12/25

Entity-Relationship Diagrams, cont. Inheritance

IS A Inheritance

I This type of inheritance happens when you have a **supertype** and

one or more **subtypes** that are members of the supertype.

I Denoted by an upside-down triangle, with the supertype on top,

and the subtypes coming out the bottom.

Person

Student Faculty Staff

13/25

Entity-Relationship Diagrams, cont. Inheritance

Defining IS-A inheritance

There are two things you need to choose when using IS-A inheritance:

I Generalization vs. specialization - can the supertype occur without

being a member of the specified subtypes?

I Overlapped vs. disjoint subtypes - is it possible for a single

occurrence of the supertype to be a member of more than one

subtype?

They are mutually exclusive so you need to pick one of each, ie. GO,

GD, SO, SD

14/25

Entity-Relationship Diagrams, cont. Inheritance

IS-A inheritance - Generalization

Generalization:

I Supertype is the **union** of all of the subtypes.

I This means that an instance of the supertype **CANNOT EXIST**

without belonging to at least one subtype.

15/25

Entity-Relationship Diagrams, cont. Inheritance

IS-A inheritance - Specialization

I The subtype entities specialize the supertype.

I This means that an instance of the supertype **CAN** exist without

being related to any of the subtypes.

16/25

Entity-Relationship Diagrams, cont. Inheritance

IS-A inheritance - Overlapping Subtypes

Overlapping Subtype Entities

I It is possible for an instance of the supertype to be related to more

than one of the subtypes.

I In our example, this would mean that a Person can be a Faculty

member and a Student, or a Student and a Sta\_ member, or all 3.

17/25

Entity-Relationship Diagrams, cont. Inheritance

IS-A inheritance - Disjoint Subtypes

Disjoint Subtype Entities

I the subtype entities are mutually exclusive

I it is **not possible** for an instance of the supertype to be related to

more than one subtype.

I in the above example, this would mean a Person could be Student,

Faculty, or Sta\_, but **not** any combination of two or more of these.

Weak Entity - Example Diagram

To indicate this sort of dependency, we can make the dependent entity a

“weak” entity. This is drawn with a double-edged rectangle, shown

below.

Name City is in State Name

Notice that the City entity is now drawn as a weak entity, with a double

border. The relationship between the weak entity and the strong entity

is also drawn with a double border. The relationship is not weak, per se,

but it is used

Data Models

I A means of describing the structure of data

I A set of operations that manipulate the data (for data models that

are implemented)

I Types of data models

I Conceptual data model

I Logical data models - relational, network, hierarchical, inverted list,

or object-oriented

Other Capabilities of DBMS Systems

Support for at least one data model through which the user can view

the data

▶ There is at least one abstract model of data that allows the user to

see the “information” in the database

▶ Relational, hierarchical, network, inverted list, or object-oriented

14/34

Introduction to Database Concepts Other Capabilities of DBMS Systems

.

Other Capabilities of DBMS Systems

Support for at least one data model through which the user can view

the data

▶ efficient file access which allows us to “find the boss of Susie

Jones”

▶ allows us to “navigate” within the data

▶ allows us to combine values in 2 or more databases to obtain

“information”

15/34

Introduction to Database Concepts Other Capabilities of DBMS Systems

.

Other Capabilities of DBMS Systems

Support for high-level languages that allow the user to define the

structure of the data, access that data, and manipulate it

▶ Data Definition Language (DDL)

▶ Data Manipulation Language (DML)

▶ Data Control Language (DCL)

▶ query language access data

▶ operations such as add, delete, and replace

16/34

Introduction to Database Concepts Other Capabilities of DBMS Systems

.

Transaction Management

Transaction management is a feature that provides correct, concurrent

access to the database, possibly by many users at the same time

▶ ability to simultaneously manage large numbers of *transactions*

▶ procedures operating on the database

▶ often transactions come from around the world

▶ “lock-out” mechanisms

17/34

Introduction to Database Concepts Other Capabilities of DBMS Systems

.

Access Control

Access control is the ability to limit access to data by unauthorized

users along with the capability to check the validity of the data

▶ protect against loss when database crashes

▶ prevent unauthorized access to portions of the data

18/34

Introduction to Database Concepts Other Capabilities of DBMS Systems

.

Resiliency

Resiliency is the ability to recover from system failures without losing

data

▶ Ideally, should be able to recover from **any** type of failure.

▶ sabotage

▶ acts of God

▶ hardware failure

▶ software failure

▶ etc.

▶ Obviously, some of these would require more than just software -

offsite backups, etc.

Data Independence

Data Independence is a property of an appropriately designed

database system

▶ has to do with the mapping of logical level to physical level, and

logical to external

▶ physical data independence

▶ physical schema can be changed without modifying logical schema

▶ logical data independence

▶ logical schema can be changed without having to modify any of the

external views

Advantages of a Database

▶ Controlled redundancy

▶ Reduced inconsistency in the data

▶ Shared access to data

▶ Standards enforced

▶ Security restrictions maintained

▶ Integrity maintained more easily

▶ Provides capability for backup and recovery

▶ Permitting inferences and actions using rules

32/34

Introduction to Database Concepts Basic Database Terminology

.

Disadvantages of a Database

▶ Increased complexity needed to implement concurrency control

▶ Increased complexity needed for centralized access control

▶ Security needed to allow the sharing of data

▶ Necessary redundancies can cause complexity when updating

33/34

Introduction to Database Concepts Basic Database Terminology

.

Data vs. Information

Discuss data and information: what is the difference?

Data: Measurements you can take from real world

Information: What can be used from that data

Basic Structure

• Keys

– Super Key:

• an attribute or set of attributes that uniquely

identify a tuple

5

• every relation has at least one superkey, the

set of all attributes

• a relation can have more than one superkey

Basic Structure

• Keys

– Candidate Key:

• a minimum set of attributes that uniquely

identify a tuple

6

• a minimal super key

• a relation may have more than one candidate

key

– Primary Key:

• one and only one per relation.

• a chosen candidate key

Foreign Key

– used to reference another relation

– attributes of FK have same domain as the

primary key of the home relation

Terminology

• Domain

– set of atomic valid values of one or more

attribute

may be specified as a data type

11

– • Atomic values

– indivisible data values

• Null value

– designates a MISSING attribute value

– may or may not be allowed for an attribute

Terminology

• Degree

– number of attributes (columns) in a relation

– does not changes dynamically

12

• Cardinality

– number of tuples (rows) in a relation

– changes dynamically with additions and

deletions of tuples using DML

Terminology

• Intention

– a named relation and its attribute names

– also called schema of a relation

th DDL i d t dif th i t ti

13

– the is used to modify the intention

• Extension

– the data (tuples) in a relation

– the state of a relation

– the DML is used to modify the extension

Characteristics of a Relation

• Order Independence

• Two kinds

14

– (1) ordering of tuples within a relation

• do not have any particular order

• considered an unordered set

Characteristics of a Relation

• Order Independence

– (2) ordering of attributes within a relation

d th ti l d l

15

• do not have any particular order as long as

correspondence between the attribute and its

values is maintained

• Example

Student(Stud-ID, Stud-Name, Stud-Address)

Student(Stud-Address, Stud-Name, Stud-ID)

equivalent

relations

Relational Constraints

• Domain / Integrity Constraints

– specify the valid values of each attribute

16

– editing criteria

• salary not > 100k

• height < 8 feet

Relational Constraints

• Entity Integrity Constraint

– states that no attribute of a primary key can

contain a null value

17

• Game ( Date, Location, Time)

– here neither Date nor Location nor both can ever

contain a null value in this relation

Relational Constraints

• Referential Integrity Constraint

– a foreign key can

EITHER t i lid l f th PK i th

18

• contain a valid value of the in the

home relation

• OR contain a NULL value

Section

Sect-ID Sect-

Time

Crse-ID

S1 10:00am C2

S2 2:00pm C2

S3 3:00pm C3

S4 1:00pm null

Course

Crse-ID Crse-Title . . .

. .

C1 Course 1

C2 Course 2

C3 Course 3

05 - Relational Data Model

4

Relational Constraints

• Referential Integrity Constraint

– a foreign key can

EITHER t i lid l f th PK i th

Crse-ID in Section

relation is the

Foreign Key Crse-ID in

Course relation

is the referenced

Primary Key

19

• contain a valid value of the in the

home relation

• OR contain a NULL valueSection

Section

Sect-ID Sect-

Time

Crse-ID

S1 10:00am C2

S2 2:00pm C2

S3 3:00pm C3

S4 1:00pm null

Course

Crse-ID Crse-Title . . .

. .

C1 Course 1

C2 Course 2

C3 Course 3

Relational Operators

• Update operators

– Insert

Delete

20

– – Modify

• Retrieval operators

– Relational Algebra

– Relational Calculus

– SQL

SQL

• Name derived from Structured Query

Language

• Comprehensive database language

21

– DDL

– DML

– view definition

– transaction control

• Can be embedded in a programming

Language

Data Definition

Language

CREATE

DROP

ALTER

Data

Manipulation

(DML)

SELECT

INSERT

UPDATE

DELETE

Data Control

Language

(DCL)

GRANT

REVOKE

*Redundancy*

• When attribute values are repeated

unnecessarily

• Notice that address is repeated for each

item that is supplied

12

*Anomalies*

• Update Anomaly

– caused by redundant information

– must find all copies of information in order

to prevent inconsistencies when updating

*Anomalies*

• Deletion Anomaly

– occurs when data is lost during a deletion

that we do not wish to be lost

– occurs when there are attributes within a

tuple that are logically related to only part

of the primary key

14

*Anomalies*

• Insertion Anomaly

– occurs when we cannot insert some

information into a tuple because of a

violation of a relational constraint

– occurs when a multiple attribute key cannot

be fully completed as necessary for

insertion

15

*Fixing Anomalies*

• Decompose the relation

SP ( Supp-Name, Supp-Addr, Item, Price )

Supp( Supp-Name, Supp-Addr)

John 10 Main

Jane 20 State

Frank 30 Elm

SP-Item ( Supp-Name, Item, Price )

John Apple $2.00

John Orange 2.50

Jane Grape 1.25

Jane Apple 2.25

Frank Mango 6.00 16

*Fixing Anomalies:*

*Decomposing Relations*

Supp( Supp-Name, Supp-Addr)

John 10 Main

Jane 20 State

Frank 30 Elm

SP-Item ( Supp-Name, Item, Price )

John Apple $2.00

John Orange 2.50

Jane Grape 1.25

Jane Apple 2.25

Frank Mango 6.00

*Decomposition of Relations*

• Disadvantages

– it is more expensive to solve queries

• example: Get the address of suppliers

supplying grapes.

SP (Supp-Name, Supp-Addr, Item, Price )

Only need ONE relation with the original

Supp( Supp-Name, Supp-Addr)

SP-Item ( Supp-Name, Item, Price )

with decomposed schema - need to join two

relations

*Relational Design*

*Methodology*

• How do we tell whether one relation is

better than another?

– Check for anomalies

– Normalization (also called functional

dependency theory)

06 - Normalization to 3NF

4

19

*Functional Dependency*

• Functional dependency

– constraints in the data that depend upon

NOT on the values within a given tuple

BUT on whether or not two tuples agree on

certain components.

20

*Functional Dependency*

• Functional dependency

– Let R be a relation and let X and Y be subsets

of the attributes (one or more) of R we say

• X functionally determines Y

• y is functionally dependent on X

– if for all the tuples of R it is NOT possible that

two tuples agree on X but disagree on Y

X Y

21

*Functional Dependency*

• Functional dependency

– Given a unique value of X, we can ALWAYS

determine a value of Y

22

*Functional Dependency*

• Person (SSN, Age, Gender)

FD = { SSN Age

SSN Gender

Age Gender }

• Since two people of the same age can be

of different genders

23

*Functional Dependency*

• FDs are assertions about the real world

which cannot be proven

• FDs are established by the database

designer by considering the meaning of the

attributes

• FDs ***MUST*** hold for all possible data values

• FDs can be enforced during insertion if

programmed and told to do so by the DBA

24

*Functional Dependency*

Emp-Proj ( Emp-ID, Project, Supv, Dept, Case )

e1 p1 s1 d1 c1

e2 p2 s2 d2 c2

e1 p3 s1 d1 c3

e3 p3 s1 d1 c3

FD = { Emp-ID,Project Project, Supv, Dept, Case

Emp-ID Supv, Dept

Supv Dept }

*Definitions*

• Remember definitions of key

– Super Key:

• an attribute or set of attributes that uniquely

identify a tuple (can be > 1 in a relation)

– Candidate Key:

• a minimum set of attributes that uniquely

identify a tuple (can be > 1 in a relation)

• a minimal super key

– Primary Key:

• one and only one per relation.

• a chosen candidate key

30

*Definitions*

• Remember definitions of key

– A candidate key of a relation functionally

determines ALL attributes of the relation.

*Definitions*

• Fully Dependent

– an attribute set Y is fully dependent on

attribute set X if

**X** **Y**

and Y cannot be determined by any

subset of X

– In Emp-Proj,

– Case is fully dependent on Emp-ID, Project

– Supv and Dept are NOT fully dependent on

Emp-ID, Project

32

*Definitions*

• Prime Attribute

– if an attribute appears in a key of a relation,

then it is a prime attribute.

• In Emp-Proj, Emp-ID is prime

• Non-Prime Attribute

– an attribute not appearing in a key of a

relation.

• In Emp-Proj, Supv is non-prime

33\

*Normal Forms*

• 1NF (First Normal Form)

– all values are atomic

• 2NF (Second Normal Form)

– a relation is in 2NF if it is in 1NF and each

of its *non-prime* attributes are *fully*

*dependent* upon its entire primary key

*Definitions*

• Transitively Dependent

– A non-prime attribute is ***transitively***

***dependent*** upon the primary key of a

relation if there is also a non-prime (non

key) attribute that functionally determines

the attribute.

– In EP1, Dept is transitively dependent upon

Emp-ID since

• Emp-ID Supv

• Supv Dept

40

*Normal Forms*

• 1NF (First Normal Form)

– all values are atomic

• 2NF (Second Normal Form)

– a relation is in 2NF if it is in 1NF and each

of its *non-prime* attributes are *fully*

*dependent* upon its key.

• 3NF (Third Normal Form)

– a relation is in 3NF if it is in 2NF and none

of its non-prime attributes are *transitively*

*dependent* on its key.

Introduction to SQL

• Base table

– a named table that really exists

– each row is something that is stored

physically

• View

– a named table that does not have any real

existence

– derived from one or more underlying base

Tables

SQL General Syntax

• General syntax of SQL retrieval is

SELECT [DISTINCT | ALL] [ \* | [list-of-attributes]

FROM list-of-tables

[WHERE condition]

[GROUP BY column-list HAVING condition]

[ORDER BY column-list] ;

• Operators allowed for relational

comparison:

– Equal to =

– Less than <

– Less than or equal to <=

– Greater than >

– Greater than or equal to >=

– Not equal to <>

Ex6: List the supplier numbers for all pairs

of suppliers such that two suppliers are

located in the same city.

SELECT T1.S#, T2.S#

FROM S T1, S T2

WHERE T1.CITY = T2.CITY

AND T1.S# < T2.S#;

Ex7: List the supplier names for suppliers

who supply part P2.

SELECT DISTINCT SNAME

FROM S, SP

WHERE S.S# = SP.S# AND

SP.P# = ‘P2’;

Ex8: List the supplier names for suppliers

who supply at least one red part.

SELECT SNAME

FROM S, SP, P

WHERE S.S# = SP.S# AND

SP.P# = P.P# AND

P.COLOR = ‘RED’;

Multiple-Row Subqueries

•Multiple-row subqueries are nested

queries that can return more than one

row of results to the parent query

– Most commonly used in WHERE and HAVING

clause

•Main rule

– MUST use multiple-row operators

Multiple-Row Subqueries:

IN Operator

• IN is a set operator used to test

membership.

• The condition 'S1' IN ('S2', 'S3', 'S1') is

true, whereas the condition ‘P1' IN (‘P2',

‘P3') is false.

Ex7 revisited: List the supplier names for

suppliers who supply part P2.

SELECT SNAME

FROM S

WHERE S# IN

(SELECT S#

FROM SP

WHERE P# = ‘P2’);

Multiple-Row Subqueries: IN Operator

The system evaluates the nested query by

evaluating the nested subquery first.

In the above query, the subquery

SELECT SNAME

FROM S

WHERE S# IN

(SELECT S#

FROM SP

WHERE P# = ‘P2’);

yields (‘S1', ‘S2', ‘S3’, ‘S4')

Multiple-Row Subqueries: IN Operator

Thus the original query becomes

– SELECT SNAME

FROM S

WHERE S# IN (‘S1', ‘S2', ‘S3’, ‘S4')

Ex8 Revisited: List the supplier names for

suppliers who supply at least one red part.

SELECT SNAME

FROM S

WHERE S# IN

(SELECT S#

FROM SP

WHERE P# IN

(SELECT P#

FROM P

WHERE COLOR = ‘RED’));

Ex8 Revisited: List the supplier names for

suppliers who supply at least one red part.

SELECT SNAME

FROM S

WHERE S# IN

(SELECT S#

FROM SP

WHERE P# IN

(SELECT P#

FROM P

WHERE COLOR = ‘RED’));

Ex8 Revisited: List the supplier names for

suppliers who supply at least one red part.

SELECT SNAME

FROM S

WHERE S# IN

(SELECT S#

FROM SP

WHERE P# IN

(SELECT P#

FROM P

WHERE COLOR = ‘RED’));

Ex9: List the supplier numbers for suppliers who

supply at least one part also supplied by S2.

SELECT DISTINCT S#

FROM SP

WHERE P# IN

(SELECT P#

FROM SP

WHERE S# = ‘S2’);

Ex10: List the part numbers for all parts

supplied by more than one supplier.

SELECT DISTINCT P#

FROM SP, SP SPX

WHERE P# IN

(SELECT P#

FROM SP

WHERE SP.S# ~= SPX.S#);

Multiple-Row Subqueries:

ALL and ANY Operators

• ALL operator is pretty straightforward:

– If the ALL operator is combined with the “greater

than” symbol (>), then the outer query is searching

for all records with a value higher than the highest

valued returned by the subquery (i.e., more than ALL

the values returned)

– If the ALL operator is combined with the “less than”

symbol (<), then the outer query is searching for all

records with a value lower than the lowest values

returned by the subquery (i.e., less than ALL the

values returned)

Multiple-Row Subqueries:

ALL and ANY Operators

• List the supplier number and quantity for

suppliers who supply more quantity than any

other supplier who supplies P4.

SELECT S#, QTY

FROM SP

WHERE QTY >ALL

(SELECT QTY

FROM SP

WHERE P# = ‘P4’);

Multiple-Row Subqueries:

ALL and ANY Operators

• The <ANY operator is used to find records that

have a value less than the highest value

returned by the subquery

• The >ANY operator is used to return records

that have a value greater than the lowest value

returned by the subquery

• The =ANY operator works the same way as the

IN operator does

Ex7 Revisited: List the supplier names for

suppliers who supply part P2.

SELECT SNAME

FROM S

WHERE S# = ANY

(SELECT S#

FROM SP

WHERE P# = ‘P2’);

Ex11: List the supplier numbers for suppliers with

status less than the current maximum status value

in the S table.

SELECT S#

FROM S

WHERE STATUS < ANY

(SELECT STATUS

FROM S);

Ex12: List all the names of suppliers who do

not supply part P2.

SELECT DISTINCT SNAME

FROM S

WHERE ‘P2’ ~= ALL

(SELECT P#

FROM SP

WHERE SP.S# = S.S#);

Multiple-Row Subqueries:

EXISTS Operator

• The EXISTS operator is used to determine

whether a condition is present in a

subquery

• The results are boolean

– TRUE if the condition exists

– FALSE if it does not

Multiple-Row Subqueries:

EXISTS Operator

• "EXISTS (SELECT ... FROM ...)" evaluates

to true

– if and only if the result of evaluating the

"SELECT ... FROM ..." is not empty.

Ex7 Revisited: List the supplier names for

suppliers who supply part P2

SELECT SNAME

FROM S

WHERE EXISTS

(SELECT \*

FROM SP

WHERE SP.S# = S.S# AND

P# = ‘P2’);

Multiple-Row Subqueries:

NOT EXISTS Operator

• EXISTS, used in conjunction with NOT,

which allows people to express two types

of queries:

– Query that involves the SET DIFFERENCE

– Query that involves the concept of "EVERY".

Ex12 Revisited: List all the names of

suppliers who do not supply part P2.

SELECT SNAME

FROM S

WHERE NOT EXISTS

(SELECT \*

FROM SP

WHERE SP.S# = S.S# AND

P# = ‘P2’);

Ex13: List the supplier names for suppliers

who supply all the parts.

SELECT SNAME

FROM S

WHERE NOT EXISTS

(SELECT \*

FROM P

WHERE NOT EXISTS

(SELECT \*

FROM SP

WHERE SP.S# = S.S#

AND SP.P# = P.P#));

Ex15: Get supplier numbers for all

suppliers who supply at least all those

parts supplied by S2.

SELECT S#

FROM S, SP

WHERE EXISTS

(SELECT \*

FROM SP

WHERE SP.S#=‘S2’);

Notice that S2 is in the answer.

44

SQL Examples

• Ex16: Get the part numbers for all parts

that either weigh more than 18 pounds or

are currently supplied by supplier S2.

SELECT P#

FROM P

WHERE WEIGHT > 18

UNION

SELECT P#

FROM SP

WHERE S# = ‘S2’;

45

Group Functions

• Group functions are sometimes called multiplerow

functions

• Have discussed some of these

– SUM ( [DISTINCT | ALL ] n)

– AVG ( [DISTINCT | ALL ] n)

– COUNT( \* | [DISTINCT | ALL ] c)

– MAX ( [DISTINCT | ALL ] c)

– MIN ( [DISTINCT | ALL ] c)

– STDDEV ( [DISTINCT | ALL ] n)

– VARIANCE ( [DISTINCT | ALL] n)

46

Group Functions: Group By

• In many cases, we want to apply the

aggregate functions to subgroups of

tuples in a relation

• Each subgroup of tuples consists of the

set of tuples that have the same value for

the grouping attribute(s).

47

Group Functions: Group By

• The function is applied to each subgroup

independently

• SQL has a GROUP BY clause for

specifying the grouping attributes, which

must also appear in the SELECT clause

48

Group Functions: Group By

• When using the GROUP BY clause remember

the following

– If a group function is used in the SELECT clause,

then any individual column listed in the SELECT

clause must also be listed in the GROUP BY clause

– Columns used to group data in the GROUP BY

clause do not have to be listed in the SELECT clause.

They are only included in the SELECT clause to have

the groups identified in the output

07 - Data Retrieval in SQL

9

49

Group Functions: Group By

•When using the GROUP BY clause

remember the following

– Column aliases cannot be used in the

GROUP BY clause

– Results returned from a SELECT statement

that include a GROUP BY clause will present

the results in ascending order of the

column(s) listed in the GROUP BY clause. To

present the results in a different order, use

the ORDER BY clause.

50

Group Functions:

Group By - Examples

• Ex17a: For each Part, get the P# and the

total number of suppliers supplying the

part.

SELECT P#, COUNT(\*)

FROM SP

GROUP BY P#

Ex17: For each part being supplied, get

the part number and the total quantity.

SELECT P#, SUM(QTY)

FROM SP

GROUP BY P#;

SUM(QTY)

600

100

400

500

500

100

P#

P1

P2

P3

P4

P5

P6

52

Group Functions:

Use of Having

• Sometimes we want to retrieve the

values of these functions for only those

groups that satisfy certain conditions.

• The HAVING clause is used for specifying

a selection condition on groups (rather

than on individual tuples)

53

SQL Examples

• Ex18: List the part numbers for all parts

supplied by more than one supplier.

(same as Ex11)

SELECT P#

FROM SP

GROUP BY P#

HAVING COUNT(\*) > 1;

P#

P1

P2

P4

P6

54

SQL Examples

• Ex19: Get the total number of suppliers.

SELECT COUNT(\*)

FROM S;

COUNT

5

07 - Data Retrieval in SQL

10

55

SQL Examples

• Ex20: Get the total number of suppliers

currently supplying parts.

SELECT DISTINCT COUNT(S#)

FROM SP;

COUNT

4

56

SQL Examples

• Ex21: Get the number of shipments for

part P2.

SELECT COUNT(\*)

FROM SP

WHERE P# = ‘P2’;

COUNT

4

57

SQL Examples

• Ex22: Get the total quantity of part P2

being supplied.

SELECT SUM(QTY)

FROM SP

WHERE P# = ‘P2’;

COUNT

1000

58

Single-Row Subqueries

• A single-row subquery is used when the

results of the outer query are based on a

single, unknown value

• A single-row subquery can return to the

outer query only ONE row of results that

consists of only ONE column

59

Single-Row Subqueries

• Single-row subquery in a WHERE clause

SELECT Title, Cost

FROM Books

WHERE Cost >

(SELECT Cost

FROM Books

WHERE Title = ‘DATABASES’)

AND Category = ‘COMPUTER’;

60

Single-Row Subqueries:

Use with HAVING clause

SELECT Category, AVG (Retail-Cost)

“Average Profit”

FROM Books

GROUP BY Category

HAVING AVG (Retail-Cost) >

(SELECT AVG(Retail-Cost)

FROM Books

WHERE Category = ‘LIT’);

07 - Data Retrieval in SQL

11

61

SQL Examples

• Ex23: List the supplier numbers for suppliers who

are located in the same city as supplier S1.

SELECT S#

FROM S

WHERE CITY =

(SELECT CITY

FROM S

WHERE S# = ‘S1’);

62

SQL Examples

• Ex24: Get supplier numbers for suppliers

whose status is less than the current

maximum status

SELECT S#

FROM S

WHERE STATUS <

(SELECT MAX(STATUS)

FROM S);

S#

S1

S2

S4

63

Single-Row Subqueries:

In a SELECT Clause

SELECT Title, Retail,

(SELECT AVG(Retail)

FROM Books) “Overall Average”

FROM Books;

64

Retrieval using LIKE

– string matching

• List suppliers whose name starts with

letter S.

– SELECT \*

FROM S

WHERE SNAME LIKE 'S%'

65

Retrieval using LIKE

– string matching

• In general, a "LIKE condition" takes the form

– column LIKE string-literal

•Where "column" must designate a column of

string type. For a given record, the condition

evaluates to true if the value within the

designated column conforms to the pattern

specified by "literal"

66

Retrieval using LIKE

– string matching

• Characters within "literal" are interpreted

as follows:

– The "-" character stands for any single

character.

– The "%" character stands for any sequence

of n characters

•(where n may be zero).

• All other characters simply stand for themselves.

07 - Data Retrieval in SQL

12

67

Retrieval using LIKE

– string matching

• CITY LIKE "%BERKELEY%'

– will evaluate to true if ADDRESS contains the

string "BERKELEY" anywhere inside it.

• SNAME LIKE 'S\_\_'

– will evaluate to true if SNAME is exactly

three character long and the first is an "S".

68

Retrieval using LIKE

– string matching

• PNAME LIKE '%c\_\_\_'

– will evaluate to true if PNAME is four

character long or more and the last but three

is a "c"

• CITY NOT LIKE "%E%’

– will evaluate to true if CITY does not contain

an "E"

69

Retrieval using LIKE

– string matching

• Using escape character

– sname like ‘%\%%’ will match ?

•Abc%def

•Abcdef

•%

•%%

•%%%

70

Some Single-Row Functions

• Case Conversion Functions

– Temporarily alters the case of data stored in

a field or character string

– Does not affect how data are stored only

how data are viewed by Oracle9i during

execution of a specific query

– LOWER, UPPER and INITCAP

71

Some Single-Row Functions

• Case Conversion Functions

SELECT Firstname, Lastname

FROM Customers

WHERE LOWER(Lastname) = ‘nelson’;

SELECT LOWER(Firstname),

LOWER(Lastname)

FROM Customers

WHERE LOWER(Lastname) = ‘nelson’;

72

Some Single-Row Functions

• Case Conversion Functions

SELECT Firstname, Lastname

FROM Customers

WHERE Lastname = UPPER (‘nelson’);

SELECT INITCAP(Firstname), INITCAP(Lastname)

FROM Customers

WHERE Lastname = ‘NELSON’;

– Converts to mixed case

07 - Data Retrieval in SQL

13

73

Single-Row Functions: Character

Manipulation Functions

• Determine length, extract portions of a

string, or reposition a string

– SUBSTR (c, p, l)

– LENGTH (c)

– LPAD (c, l, s) and RPAD (c, l, s)

– LTRIM (c, s) and RTRIM (c, s)

– REPLACE (c, s, r)

– CONCAT (c1, c2)

74

Single-Row Functions:

Number Functions

•Manipulates numeric data

– Most related to trigonometry like COS, SIN,

etc.

– ROUND (n, p)

– TRUNC (n, p)

75

Single-Row Functions:

Date Functions

• Date function displays date values in a

dd-mon-yy format

– (i.e., 02-FEB-04)

•MONTHS\_BETWEEN (d1, d2)

• ADD\_MONTHS (d, m)

• NEXT\_DAY (d, day)

• TO\_DATE (d, f)

76

Single-Row Functions:

Miscellaneous Functions

• NVL (x, y)

–Where y represents the value to be

substituted for if x is NULL

SELECT Order#, OrderDate,

NVL(Shipdate, ’07-APR-03’),

NVL (Shipdate, ’07-APR-03’) – OrderDate “Delay”

FROM Orders

WHERE Order# = 1018;

77

Single-Row Functions:

Miscellaneous Functions

• TO\_CHAR (n, ‘f’) where n is the date or number

to be formatted and f is the format model to be

used

SELECT Title,

TO\_CHAR(PubDate, ‘MONTH DD YYYY”)

“Publication Date”,

TO\_CHAR(retail, ‘$999.99’) “Retail Price”

FROM books

WHERE ISBN = 0401140733;

78

Single-Row Functions:

Miscellaneous Functions

• DECODE (V, L1, R1, L2, R2,…., D)

– Where V is the value being searched for

L1 represents the first value in the list

R1 represents the results being returned if L1 and V are

equivalent, etc., and

D is the default result to return if no match is found

– Similar to CASE or IF….Then ….ELSE in many

languages

SELECT Customer#, State,

DECODE(State, ‘CA’, .08, ‘FL’, .07, 0) “Sales Tax Rate”

FROM Customers;

07 - Data Retrieval in SQL

14

79

Single-Row Functions:

Miscellaneous Functions

• SOUNDEX (c)

–Where c is the character string being

referenced for phonetic representation

shown as a letter and number sequence

SELECT Lastname, SOUNDEX (Lastname)

FROM Customers

WHERE Lastname LIKE ‘M%’

ORDER BY SOUNDEX(Lastname);

80

Single-Row Functions:

Nesting Functions

• Any of the Single-Row functions can be nested

inside other Single-Row functions as long as

the rules as followed

– All arguments required for each function must be

provided

– For every open parenthesis, there must be a

corresponding closed parenthesis

– The nested, or inner, function is solved first. The

result of the inner function is passed to the outer

function, and the outer function is executed

Self-Joins

• Traditional Method

– List table name twice using an alias for each

SELECT r.Firstname, r.Lastname,

c.Lastname, c.Referred

FROM Customers c, Customers r,

WHERE c.Referred = r.Customer#;

Self-Joins

• JOIN Method

– List table name twice using an alias for each

SELECT r.Firstname, r.Lastname,

c.Lastname, c.Referred

FROM Customers c JOIN Customers r,

ON c.Referred = r.Customer#;

More Joins

• Inner Join

– This is what we have been doing with our

joins

•A join that compares the tables in the FROM

clause and lists as output only those rows that

satisfy the condition in the WHERE clause

• Outer Join

• Product

Outer Joins

• Keyword OUTER JOIN in SQL includes

records of a table in the output even if

there is no matching record in the other

table

• In a sense SQL will join the “dangling”

record to a NULL record in the other table

07 - Data Retrieval in SQL

15

Outer Joins

• Traditional Method

– Use the outer join operator which is a plus sign in

parenthesis (+) placed in the WHERE clause

immediately after the column name of the NULL

tuple

SELECT Lastname, Firstname, Order#

FROM Customers c, Orders o

WHERE c.Customer# = o.Customer#(+)

ORDER BY c.Customer#;

Outer Joins

• JOIN Method

– Include the keyword LEFT, RIGHT, or FULL

with the JOIN

Outer Joins

• Display the customer number, name, order

number, and order date for all orders. Include

all customers in the results.

SELECT c.Customer\_num, c.Customer\_Name,

Order\_num, Order\_date

FROM Customer c

LEFT OUTER JOIN Orders o

on c.Customer\_num = o.Customer\_num

ORDER BY c.Customer\_num;

• Called Cartesian Product

– Of two tables is the combination of all the

rows of the first table and all the rows of the

second table.

07 - Data Retrieval in SQL

16

Product

• Form the product of the CUSTOMER and

ORDERS tables.

SELECT c.Customer\_Num, Customer\_Name,

Order\_Num, Order\_Date

FROM Customer c, Orders;

Product

• Form the product of the CUSTOMER and

ORDERS tables.

SELECT c.Customer\_Num, Customer\_Name,

Order\_Num, Order\_Date

FROM Customer c, Orders;

Create Table - Syntax

• CREATE TABLE <table\_name> (

<attribute> <type>

[not null][unique][primary key])

{,<attribute> <type>

[not null][unique][primary key] }

[,<primary key constraint>]

[ {,<foreign key constraint> } ] );

3

Create Table - Syntax

• where

– <attribute> is the name of attribute in the

table to be defined

– <type> can be

•INTEGER

•FLOAT/REAL

•DECIMAL(I,J)

•CHAR(N)

•VARCHAR(N)

4

Create Tables - Examples

• Create table person (

ssn char(9) primary key,

fname char(10) not null,

lname char(10) not null,

phone# char(10) );

5

Create Tables - Examples

• Create table student (

ssn char(9),

classification char(6),

gpa decimal(4,3),

total\_hours integer,

primary key (ssn),

foreign key (ssn) references

person(ssn) ); 6

Alter Table - Syntax

• alter table <table\_name> add (

<attribute> <type>

{, <attribute> <type>} );

• alter table <table\_name> modify (

<attribute> <new\_length>

{, <attribute> <new\_length>} );

08 - SQL Data Definition Language and

Update Operations

2

7

Alter Table - Examples

• alter table person add (

birth\_date char(8) );

• alter table section modify (

title char (25),

description char (50) );

8

Drop Tables -

Syntax and Example

• drop table <table\_name>

[cascade constraints];

• drop table person

cascade constraints;

9

Insert - Syntax

• insert into <table\_name>

values (<value-list>);

• insert into <table\_name>

(<attribute-list>)

values (<value-list>);

• insert into <table\_name>

select \* from <another\_table\_name>;

10

Insert - Examples

• insert into student

values(‘293844323’,’senior',3.294,110);

• insert into person

(ssn, fname, lname)

values (‘384729479',‘Susie’,’Jones’);

• insert into section

select \* from temp\_section;

11

Delete - Syntax and Examples

• delete from <table\_name>

[where <condition>];

• delete from person

where ssn = ’394837497‘;

• delete from section;

12

To Change Data Type

of an Attribute

• If a table contains data, the data type of

an attribute cannot be directly changed

• To do so

– a temporary table is created and populated

with the tuples of the original table

– the tuples of the original table are deleted

and the table is modified

– the data is copied from the temporary table

back into the modified original table

08 - SQL Data Definition Language and

Update Operations

3

13

To Change Data Type

of an Attribute

• Example to change the data type of a

character attribute to integer

– create table temp\_table as

select \* from section;

– delete section;

– alter table section (

modify max\_enroll integer);

– insert into section

select \* from temp\_table;

– drop temp\_table cascade constraints; 14

Update tables

• Used to modify attribute values of one or

more selected tuples.

• A WHERE-clause selects the tuples to be

modified.

• An additional SET-clause specifies the

attributes to be modified and their new

values.

• Each command modifies tuples in the

same relation.

15

Update - Syntax and Examples

• update <table\_name>

set <attribute>=<value>

{,<attribute>=<value>}

[ where <condition>];

• update student

set classification = ‘senior’

where total\_hours > 90;

• update section

set max\_enroll = 0; 16

Views

• A view is a single virtual table that is

derived from other tables

• The other tables could be base tables or

previously defined views.

• A view does not exist in physical form,

which limits the possible update

operations that can be applied to views.

• There are no limitations on querying a

view

17

Views

Table 1

View 2

Table 2 Table n-1 Table n

View 3 View m

View 1

18

Create View - Syntax

• create view <view-name>

[(<view-col-name> [, <view-col-name>]...)]

as select <attr-name> [,<attr-name>] ...

from <table-or-view-name>

[,table-or-view-name] ...

where <condition>;

08 - SQL Data Definition Language and

Update Operations

4

19

Create View - Examples

• create view dekalb\_people

(ssn, first\_name, last\_name)

as select ssn, fname, lname

from person

where zip = ‘60115’;

20

Create View - Examples

• create view grade\_count

(grade, number\_grade)

as select grade, count(\*)

from transcript

group by grade;

21

Create View - Examples

• create view offered\_sections

as select \*

from section, course

where section.course# =

course.course#;

22

Query on Views

• Views can be queried by using the same

select operation as with tables just

replace the table name with the view

name

• There are some restrictions on insertion,

deletion, and update function when used

on views

23

Drop Views -

Syntax and Example

• drop view <view\_name> ;

• drop view dekalb\_people;

24

Advantages of Views

• Views allow different users to see the

data in different forms.

• Views can free users from complicated

DML operations, especially in the case

where the views involve joins.

• Views can enhance security

08 - SQL Data Definition Language and

Update Operations

5

25

System Catalog Views

• To find out what tables you have created,

type

– select table\_name from user\_tables;

• To find out information about a particular

table (person, for example) type:

– describe person;

26

Some Built-in Functions

• lower(string)

– converts string to lower case

• upper(string)

– converts string to upper case

• Examples:

select \*

from c

where lower(dept) = ‘music';

27

Some Built-in Functions

• lpad(x, y[,z])

– returns the column padded on the left side

of the data in the column passed as x to a

width passed as y.

– The optional passed value z indicates the

character(s) that lpad() will insert into the

column.

– If no character is specified, a space will be

used.

• rpad(x,y[,z]) similar to lpad.

28

Some Built-in Functions

• Example

– select ssn, course#, lpad(grade, 6), “GRADE”

from transcript;

29

Some Built-in Functions

• The default input format for date type is

DD-MON-YY.

– insert '22-Mar-99' into a date attribute

30

Some Built-in Functions

• To insert a date value of different format into a date

field, you need to use to\_date function

– insert into emp(empno, hiredate)

values(8989, to\_date('99-12-31', 'yy-mm-dd'));

– insert into emp(empno, hiredate)

values(8988,

to\_date('99-12-31 14:35:00', 'yy-mm-dd hh24:mi:ss'));

*ER to Relational Conversion*

1 Consider all strong entities not subtypes

(do not consider “date” entities here)

2 Consider sub-type entities

- two methods

3 Consider weak entities

4 Consider One-to-many binary

relationships

09 - ER to Relational Conversion

2

7

*ER to Relational Conversion*

5 Consider many-to-many binary

relationships

6 Consider relationships greater than

binary (other than those involving “date”

entities)

7 Consider relationships greater than

binary involving a “date” entity

8 Consider recursive relationships