Date → 'DD-MON-YY' 8/2/23 (Give only last 2 digits → 19__)

STUDENT (ROLL, NAME, DOB, DCODE)

→ insert into student values (1, 'LOL', '02-FEB-2010', 'D1');

(*) (Here, char(5) → works like varchar → use all 5 as '0' for empty)

→ TO_DATE (date_string, format) → formats are → 'DD' (numeric day), 'MM' (num^m month), 'YYYY' (numeric year), 'HH' (Hour), 'MI' (Minutes), 'SS' (Second), 'AM', 'PM', 'MON' (o/p like 'JAN', 'MAR'), 'MONTH' (o/p like 'JANUARY', 'MARCH')

⟹ (Can give like 'MONTH, YYYY')

→ select * — [* → all col^m / all att^r] (acts on individual tuples)

→ select roll, name from student where dcode = 'D1'; (!= for others)

[where dcode is null → absence of value]

[where dcode is not null] ~~not condition~~

[where mydate >= '31-DEC-2022']

[where to_date (mydate, 'MON') = 'JAN']

(#) subject (scode, sname) student (roll) result (roll, scode)

Regex →

[where name like 'RA%']

'%' → matches with 0/many # any char

'_' → matches with any single char

pioneerpaper.co
Page :
Date : / /

SQL FUNCTIONS →

↳ 2 types → ① SINGLE ROW F^N → Acts on individual row & provides an O/P...

② AGGREGATE F^N → Acts on a set of rows & provides 1 O/P for a set...

① SINGLE ROW F^N →

NUMERIC

FLOOR(number), CEIL(number), POW(m,n) [m^n], MOD(m,n) [m%n], TRUNCATE(number, upto_which_place)

*/ truncate(175.7831, 2) → 175.78

*/ ——, 0) → 175 */ ——, -1) → 170

ROUND(number, upto_which_place)

*/ round(175.7831, 2) → 175.78 */ ——, 1) → 175.8 [follows ≥5 rule]

*/ ——, 0) → 176 */ ——, -1) → 180

ABS(number)

→ select round(172.17, 1) from dual;

[student → prints same result for #student times]

[dual → print 1 time...]

DUAL → Single dummy tuple in table dual (given by SQL)

STRING

LENGTH(str), ASCII(char), UPPER(str/char), LOWER(s/c),

INITCAP(s) → For each word, do 1st char UPPER, rest Lower (& return formatted string)

RTRIM(s), LTRIM(s),

SUBSTR(s, pos^n, numb of char) → starting from pos^n (1-indexed), return #chars (eg:- substr(to_char(DOB, 'DD-MM-YYYY'), 4, 2) → '02')

(also interpret as leaving first n chars)

INSTR(string, substring, startpos^n, which_occurance) → by default = 1

→ returns pos^n where it finds (all are 1-indexed)

not found/not present/invalid → return 0

[eg:- 'abcdcbac', 'c', 4, 2 → 8]

MONTHS_BETWEEN(date1, date2),

LAST_DAY(date1) → last day of month in that date,

ADD_MONTHS(date1, n) → return date after adding

*/ select * from lol order by roll; → Ascending

*/ " " " " " " roll desc; → Descending

[For string → Lexicographical order]

Query processor at runtime see what to select intermediate for other conditions & orderings → DOES IT → then selects user selections from it

select name from _____ order by roll, score; → primary ordering attribute → First by roll, then for same roll by score

(In classical compiler, selection must have atleast to work our expression on ...)

_____ order by X desc, Y asc

AGGREGATE FN → Cf(COL-NAME) →

(SUM, AVG) → (Null Ignored), (MAX, MIN) → (Not null),

VARIANCE, STDDEV, COUNT, DISTINCT (unique values)

(null is also 1 distinct val)

(do like count(distinct(dept)))

count(all dept) → Count without null

count(dept) → Count with null

(2) (out of depts enrolled in)

NOTE → select avg(score), roll ____ → theoretically can give many #values but practically doesn't work...

13/2/23

DEPT(DCODE, DNAME)

SUBJECT(SCODE, SNAME, FH, FM)

STUDENT(ROLL, NAME, ____)

RESULT(ROLL, SCORE, ____)

count(attr) → #Tuples with not null

count(all attr) → null is counted

count(distinct dcode) → null is counted 1 time if present

(only aggregate functions & grouping attributes can be together)

#St in 'DI' →

select count(*) from student where dcode = 'DI'.

select count(score), max(score)

→ 'OK' → Can have multiple aggregate functions together

select dcode, count(*) from student group by dcode

(for every dept, their count of students)

(#) select city, dcode, count(*) from students group by city, dcode

MULTIPLE GROUPING

Alias

(ordering has no effect)

(except this subject)

where scode != 'EVS'

(of grouping)

as total

select roll, sum(score) from result group by roll

Total score for each student

order by total desc

(OR) order by 2 desc

(By 2nd selected Attribute)

To filter out total > 400

(Can't do where

Applied on every tuple

SOLN → having total > 400 → Works on result of groups by

(There is no having without group by)

select → from student S, dept D → select student s, roll r → another way of writing alias

Ans = select roll, sum(score) as total from result where scode != 'EVS' group by roll having ~~total~~ total > 400 order by total desc

Another Way → (where) →

select roll, total from
(
select roll, sum(score) as total from result where scode != 'EVS'
group by roll
) where total > 400 order by total desc;

(Cartesian Product) (#) **SUBJECT NAME & AVG. SCORE** →

JOINING {
select sname, avg(score) from subject S, result R
where S.scode = R.scode group by S.scode, sname

SIMPLE SUBQUERY

• select name from student where dcode = (select dcode from dept where upper(dname) = 'CSE')

(*) Inner query (executed once) is executed first, outcome of inner query is used in evaluating the outer query.
Type of result returned by inner query must be compatible with the type expected by outer query.
Inner query must return a single value else special measure has to be taken.

• select max(score) from result where score is not null
/OR/ • select max(nvl(score, 0)) from result
→ if null, replace with value
/OR/ • select ~~max~~ score from result where score is not null and score >= all (select distinct score from result where score is not null)

(#) ~~where~~ [in (v1, v2, v3)] ≡ [x = v1 or x = v2 or x = v3]

• select * from student where dcode in (select dcode from dept where upper(campus) = 'MAIN')

pioneerpaper.co
Page :
Date : / /

Another way of selecting min score →

- select * from result where score is not null and score > any (select distinct score from result where score is not null)

CORRELATED SUBQUERY → Find name of dept in which atleast 10 students are enrolled

select dname from dept D where 10 <= (select count(*) from student S where S.dcode = D.dcode)

CORRELATED ← outer query record is referred here in inner query (table)

- inner query executed multiple times
- → tuple of outer query is referred in inner query (say candidate tuple)
- → takes a tuple from relation of outer query
- → For each & every cand^t key inner query is executed
- → Result is used to decide the action on cand^t tuple
- → Repeat for every candidate tuple

Alt → JOIN →

CORR^L SUB^Q → Less Memory but more CPU used

JOIN → Memory Heavy but less CPU used

(same dname found out many times for same dcode)

- Delete dept where not student is studying →

delete from dept D where 0 = (select count(*) from student S where D.dcode = S.dcode)

→ Here, cannot do JOIN → (corr^L SQ cannot be avoided)

- Show dept where atleast 1 student studies → (Not efficient)
  - → Subquery (like above) with 1 <= (→)
  - → Don't need to count all 1000 if any 1 is found...
  - → Use exists → EFFICIENT

select dname from dept D where exists (select * from student S where D.dcode = S.dcode)

find atleast 1 tuple in result → Exist returns true immediately

NOTE → Abt prev q. → nobody studies → ~~Same time (another)~~ ~~Needed~~ → (if any exist → immediately exclude) Better with not exists

- insert into backlog values (select roll, score from result where score is null or score < 40) ???
- create backlog* as (Same schema) (Changing diff^t names) (ROLL NO, SUB CODE)

(#) Changing Schema → alter schema tablename (add (column1 desc1, column 2 desc2, ——)
(modify (col1 desc1, col2 desc2, ——)
→ type can be changed only if no tuples
→ size can be ↑ (↓ if no tuples)
→ add constraint (not null if no null values are already in the tuples)
→ drop
(drop constraint (const name)
table
drop tablename → whole table removed (DDL)
rename oldname newname

15/2/23

DB DESIGN →

E-R Model → Formal way of designing DB → NORMALISATION
Mapping to relation → Based on the concept of key & functional constraints

(*) In a relation, we have a set of semantically related (attr) & we have some interpretation associated with them → INTER-RELATED
Subject Code here won't be meaningful
(STUDENT (ROLL, NAME, ——)

■ GUIDELINES →
(1) In a schema, we will have 1 type of entity or relation type. -slip
The schema should be easy to interpret.
⇒ Update Anomalies → STUDENT (ROLL, NAME, DOB, ADDR, COURSE ID, CNAME, DURATION, FEES, ——)
⇒ Insert Anomalies → New Course, No Students → For Student Attr, allow NULL → Key NULL not allowed (when storing course info but no student info → roll) → Also, course info (repeated multiple times → Redundancy
⇒ Deletion Anomaly → Only 1 student in course → Student deleted → Course info lost
⇒ Modification Anomaly → Course info has to be modified in many places → else inconsistency ...
⇒ Redundancy & Update Anomalies → ...

P.T.O.

(3) Design must be to remove/minimize redundancy & update anomalies.

1 ⇒ Null Value : If we have a fat relation (large # attr maybe of diff entity or relationship type)
(attr not applicable → NULL) (subset of attr may be NULL for a tuple)
→ Wastage of space → Problem in JOINS & AGGREGATE FUNCTNS

(2) Design must be such to avoid frequent occurance of NULL values.

(4) Design st joining does not give rise to spurious tuples.
Joining will be based on Primary Key Foreign Key
STUDENT (—, DURATION) COURSE (—, DURATION) → We can have multiple courses of same duration
⇒ Equijoin with arbitrary attr → gives redundant / irrelevant tuples (SPURIOUS TUPLES)
Diff from derived attr (DOB & age)
DOB → age changes with time func not same always

• FUNCTIONAL DEPENDANCY →

R
X ⊆ R
Y ⊆ R

X and Y are subsets of schema R.
X → Y is a func^t dependancy that holds on R provided for any 2 tuples $t_1, t_2 \in r(R)$, if $t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$
→ If I know X, I can uniquely determine Y
→ X functionally determines Y
eg:— RESULT (SCORE, GRADE, —) [SCORE (X) → GRADE (Y)] given the FD

*) Constraint on the schema R, semantic relation b/t 2 sets of attr
(NOTE: Looking into the instance, we can't tell if FD is valid/not
eg:— NAME → DOB for 1 state but if same name, diff DOB
→ Y in is not valid → Y in FD is not valid)

*) Set of FD 'F' on 'R' each $t \in r(R)$ satisfied each FD $f \in F$
→ Valid / Legal State

[X → R]

*) If X is a CANDIDATE KEY → unique identifier → Y is any subset of R
*) If X → Y, it doesn't guarantee Y → X
(Not Commutative (eg:— Score → Grade ≠ Grade → Score)
*) Given a set of F.D. 'F', specified on a relation 'R', we can infer other F.D. → CLOSURE of 'F' → F^+ → Set of all all possible F.D. that can be inferred from F & holds on r(R)

A → B
B → C
⇒ A → C (Transitive)

$(XY = YX)$ $(XX = X)$

pioneerpaper.co
Page :
Date : / /

#) Inference Rules → (I) ARMSTRONG'S Axioms →

1. REFLEXIVITY → $Y \subseteq X \Rightarrow X \to Y$ (aka TRIVIAL RULE)
2. AUGMENTATION → $X \to Y \Rightarrow WX \to WY$
3. TRANSITIVE → $X \to Y, Y \to Z \Rightarrow X \to Z$

(II) EXTENSION RULES →

4. DECOMPOSITION → $X \to YZ \Rightarrow X \to Y$, $X \to Z$
5. UNION → $X \to Y, X \to Z \Rightarrow X \to YZ$
6. PSEUDO-TRANSITIVE → $X \to Y, WY \to Z \Rightarrow WX \to Z$

Proofs →

IR1 → $X \to Y \Rightarrow t_1[X] = t_2[X]$ As $Y \subseteq X$ (As Y is subset), $t_1[Y] = t_2[Y]$
Hence, $X \to Y$

[t[X] means extracting X part from tuple instance 't']

IR2 → By Contradiction, $X \to Y$ is true $\Rightarrow t_1[X] = t_2[X]$
$t_1[W] = t_2[W]$
$t_1[Y] = t_2[Y]$
$t_1[WX] = t_2[WX]$
$t_1[WY] = t_2[WY]$
$t_1[WY] \neq t_2[WY]$

IR3 → $X \to Y$, $Y \to Z$
$t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$
$t_1[Y] = t_2[Y] \Rightarrow t_1[Z] = t_2[Z]$
Result: $X \to Z$

IR4 → $X \to YZ$, $YZ \to Y$ (IR1) $\Rightarrow X \to Y$ (IR3)
$\Rightarrow X \to Z$ (Similarly)

IR5 → $X \to Y \Rightarrow XX \to YX$ (IR2) $\Rightarrow X \to XY$
$X \to Z \Rightarrow XY \to XZ$ (IR2)
$\Rightarrow X \to YZ$ (IR3)

IR6 → $X \to Y \Rightarrow WX \to WY$ (IR2)
$WY \to Z$
$\Rightarrow WX \to Z$ (IR3)

Armstrong's Axioms are SOUND & COMPLETE

Given a F.D. set F specified on R, any FD f inferred from F using A.Ax, will be valid & holds on R

By successive application of Armstrong's Axioms on F, till no inference can be made, ALL F.D. on R can be obtained (F^+)