UNIT-3 STRUCTURED QUERY LANGUAGE

→ HISTORY

(1) → IBM sequel lanag. developed as part of system R project at the IBM San Jose Research Lab.

(2) → Renamed Structured Query Lang. (SQL)

→ ANSI (American National stds. Institute) & ISO (International organizan for standardizan) std. SQL:

(i) SQL-86, SQL-89, SQL-92

(ii) SQ:1999, SQ:2003, SQ:2008

→ Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later stds. & special proprietary features.

(4) → 1970: Dr. Edgar F. "Ted" Codd of IBM is known as the father of Relanal dbses. He described a model for dbses

1974: SQL appeared.

1970: IBM worked to develop Codd's ideas & released a product named System/R.

1986- IBM developed the 1st prototype of relanal. db & standardized by ANSI. The 1st relanal db was released by Relanal s/w which later came to be known as Oracle.

Is a non-procedural langs

→ Why SQL?? OR Advantages of SQL:-

↳ allows users to access data in the relanal db mgmt systems.

→ allows users to describe the data.

→ allows users to define the data in a db & manipulate that data.

→ allows to embed within other lang. using SQL modules, libraries & pre-compilers.

→ allows users to create & drop dbses & tables.

→ allows " " a views, stored procedure, funcs in a db.

→ " " " set permissions on tables, procedures & views.

→ SQL databases : most popular RDBMS's are
- MySQL
- MySQL server
- Oracle
- Ms Access

→ SQL datatypes & Literals:-

(1) char (size) / char(n) : where size is the no. of characters to store. fixed-length strings. ~~space reserved~~ (padded). (augmented)
eg: if the width of a character variable ~~as~~ is 10 & the string stored in it is 'RDBMS', it will be stored as 'RDBMS.....'

(2) varchar2(size): where size is the no. of characters to store. Variable-length strings.
eg: if the width of a character variable is 10 & the string stored in it is 'RDBMS', it will be stored as 'RDBMS'.

(3) number(p,s) : This data type is used to store number fixed or floating point. ~~The p~~
where 'p' is the precision, determines the length of data; while 's' is the scale, determines no. of places after decimal.
eg: number(7,2) is a no. that has 5 digits before the decimal & 2 digits after the decimal.

(4) date : Its format is "DD-MON-YY"
eg: "25-JAN-2005"

(5) long : long data types stores variable length character strings containing upto 2 gigabytes size.
But it has some limits:-
(a) A table can not have more than one LONG type of data field.

(b) it can't
(c) it can't
(d) it can'

→ SQL Co

① Types of

DDL
- create
- alter
- drop
- truncate

select

(b) It can't be indexed.
(c) It can't be used with SQL fun^n's.
(d) It can't appear in WHERE, GROUP BY, ORDER BY clauses.

→ SQL Basic structure ↔ Types of SQL commands
↔ Datatypes & literals.

① Types of SQL commands ↔ DDL
↔ DML
↔ DCL

DDL	DML	DCL	TCL
↳ create	↳ insert	↳ Grant	↳ commit
↳ alter	↳ update	↳ Revoke	↳ rollback
↳ drop	↳ delete		↳ Savepoint
↳ truncate	↳ select		↳ set transaction
	Subqueries; Joins (self, inner, outer, left, right); Queries using exists/Not exists		

Select
① Simple select statement
② Avoiding duplicates (DISTINCT)
③ Row selection/conditions (WHERE clause)
④ FROM clause
⑤ BETWEEN, IN, LIKE, AND, OR, NOT
⑥ column titles using AS
⑦ ~~Sorting~~ IS NULL, IS NOT NULL
⑧ Aggregate func^n's / column func^n's
⑨ Sorting Query Results (order by clause)
⑩ GROUP BY
⑪ HAVING
⑫ Retrieval using UNION
⑬ " " INTERSECT
⑭ " " EXCEPT (difference/minus)

→ DDL: allows the specifica^n of informa^n about relations, including:
- → The schema for each rela^n.
- → The domain of values associated with each attribute.
- → Integrity constraints.

(1-)

Create table table-name (column1 datatype, column2 datatype, ...);

(2) INTEGRITY CONSTRAINTS:-

(1.) NOT NULL (2.) UNIQUE (3.) PRIMARY KEY (4.) FOREIGN KEY (5.) CHECK. (6.) DEFAULT.

NOT NULL

eg:→ create table customer (c_id number (10), NOT NULL, c_name varchar (20), add varchar (29));

→ Alter table customer add NOT NULL (c_id);

UNIQUE

eg:→ create table customer (c_id number(9) UNIQUE, c_name varchar (20), add varchar (18));

→ Alter table customer add UNIQUE (c_id);

PRIMARY KEY

eg:→ create table dept (d_id number(9) PRIMARY KEY, d_name varchar(10), location varchar (21));

→ Alter table dept · add PRIMARY KEY (d_id);

FOREIGN KEY

eg:→ create table employee (e_id number(11) PRIMARY KEY, e_name varchar (21), city varchar (18), d_id int FOREIGN KEY references dept (d_id)

CHECK:

eg:→ Create
Cid n
→ Alte

DEFAULT

eg:→ crea
cr
Alt

(2.) Alter

Alter
change
datatype
(3.) C

(3.) drop:

(4.) truncate

→ DML:

(1.) Ins

(1) direct mo
→ insert i

(2) through pa
insert into
Enter value

CHECK:

eg:→ Create table employee (e-id number(10) CHECK (e_id>0), e_name varchar(21));

→ Alter table employee add CHECK (e-id>0).

DEFAULT

eg:→ create table customer (c_id number(10) NOT NULL, c_name varchar(20) DEFAULT "xyz", add varchar(29));

→ Alter table customer ~~add~~ alter column c-name set DEFAULT "xyz";

② Alter:

→ Alter table Employee add e_name char(11);

change datatype → Alter table employee MODIFY ~~alter~~ e_name varchar(18);

③ ~~drop:~~ OR Alter table employee alter column e_name varchar(18);

③ drop: DROP table table_name;

④ truncate: truncate table table_name;

→ DML:

① Insert → direct method
→ through parameter substitn.

② direct method:→ insert into dept values ('101', 'CS', 'Gzb');

the → insert into dept (did, d_name, d_locn) values (101, 'CS', 'Gzb');

③ through parameter substitn:

insert into dept values ('&d_id', '&d_name', '&d_locn');

Enter value for did: 101

" " d_name: CS

" " d_locn: Gzb

② update

UPDATE dept SET d_id='102' WHERE d_name='CS';

③ delete-

- ~~delete table table_name;~~ delete ~~*~~ from tablename;
- delete from table_name WHERE [condition];
  (∵ delete from dept WHERE ~~det~~ d_id=102;)

④ select-

- select * FROM table_name; (gives full table as o/p)

→ SELECT:

① select * FROM table_name;

eg: select e_name from employee;
~~select * from employee;~~

② select DISTINCT e_name from employee;
(∵ It remove redundant data of e_name or repetition or same data of e_name);

③ select e_name from employee WHERE salary=40000;

④ select e_name, e_id, e_city FROM employee;

⑤ Select * from employee WHERE salary BETWEEN 40000 AND 50000;

(∵ here b/w between include 40000 & 50,000 both).

⑥ column titles using AS: SQL-Aliases are used to give a table, or a column in a table, a temporary name.

alias column syntax
→ SELECT column_name AS alias_name FROM table_name;

alias table syntax
→ SELECT column_name ~~AS~~ FROM tablename AS aliasname;

eg: → select ~~column_name~~ customer_name As customer, contact_name As contact_person FROM customertable;

(14) IS NOT NULL:

select column_names FROM table_name WHERE column_name IS NOT NULL;

eg: Select * from employee where salary IS NOT NULL;

IS NULL:

eg: select * from employee where salary IS NULL;

(15) LIKE:

select e_name, emp_no, salary of employee where e_name Like 'a%';
Like '%a';
Like '_r%';
Like 'a%t';
Like '%or%';

OR:

Select * from employee where salary = 40000 OR salary = 50000 OR salary = 6000;

IN: (when we there is need of multiple OR)

Select * from employee where salary IN(40000, 50000, 60000);
NOT IN (40000, 50000, 60000);

AND:

Select * from employee where salary = 40000 AND salary = 50000;

NOT:

Select * from employee where salary != 40000;
where NOT salary = 40000;

(16) Aggregate funcns:
- count()
- sum()
- avg()
- min()
- max()

⊙ count(): this func^{n} returns the no. of rows that matches a specified criteria.

eg: SELECT count(column_name) FROM table_name WHERE condⁿ;

eg: SELECT count(product_id) FROM products;

⊙ Sum(): this func^{n} returns the total sum of a numeric column.

SELECT SUM(column_name) FROM table_name
WHERE condⁿ;

eg: SELECT SUM(Quantity) FROM orders;

⊙ avg(): func^{n} returns the avg. value of a numeric column.

SELECT avg(column_name) FROM table_name where condⁿ;

eg: SELECT avg(Price) FROM products;

⊙ min(): this func^{n} returns the smallest value of the selected column.

SELECT MIN(column_name) FROM tablename WHERE condⁿ;

eg: SELECT min(price) AS smallestPrice FROM products.

o/p →

smallestPrice
2.5

⊙ max(): this func^{n} returns the largest value of the selected column.

SELECT MAX(column_name) FROM tablename WHERE condⁿ;

eg: select max(price) AS largestprice FROM products;

o/p →

Largestprice
263.5

(9.) ORDER BY: ~~is~~ keyword is sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

eg: SELECT column1, column2 FROM tablename ORDER BY column1, DESC;

eg: → select * from customers ORDER BY country DESC;
→ select * from customers ORDER BY country Asc, Cust_name DESC;

(10.) GROUP BY: this is often used with agg. func^n's (count, max, min, sum, avg) to group the result-set by one or more columns.

SELECT column_name(s) FROM tablename WHER condition
GROUP BY column_name(s) ORDER BY column_name(s);

eg: → SELECT COUNT (CustomerID), country FROM customers
GROUP BY Country;
→ SELECT COUNT(customerID), country FROM customers GROUP BY country
ORDER BY count(cust_id) DESC;

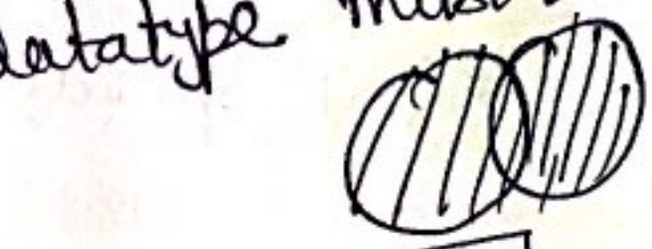
(11.) HAVING: ~~cla~~ this clause was added to SQL bcoz the WHERE keyword could not be used with agg. func^n's.

SELECT column_name(s) FROM tablename WHERE condition GROUP BY column_name(s) HAVING condition ORDER BY column_name(s);

eg: select count(cust_id), country FROM customers GROUP BY country
HAVING COUNT (custID > 5);

(12.) Retrieval using SET OPERATIONS (UNION, INTERSECT & EXCEPT).

UNION is used to combine the results of 2 or more select statements. However, it will eliminate duplicate rows from its result set. In case of UNION, the no. of columns & datatype must be same in both the tables.



First

ID	Name
1	Abhi
2	Avi

Second

ID	Name
2	Avi
3	Amit

```
select * from First
 UNION
select * from second;
```

Result:→

ID	Name
1	Abhi
2	Avi
3	Amit

UNION ALL This operation is similar to UNION. But it also shows the duplicate rows.

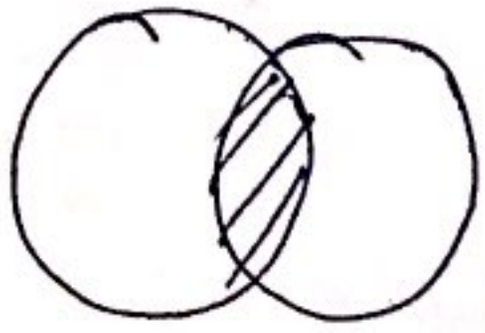
select * from first UNION ALL select * from second;

Result:→

ID	Name
1	Abhi
2	Avi
2	Avi
3	Amit

INTERSECT operation is used to combine 2 select statements, but it only returns the records which are common from both select stmts. In case of INTERSECT the no. of columns & datatype must be same.

*MySQL doesn't support INTERSECT operator.



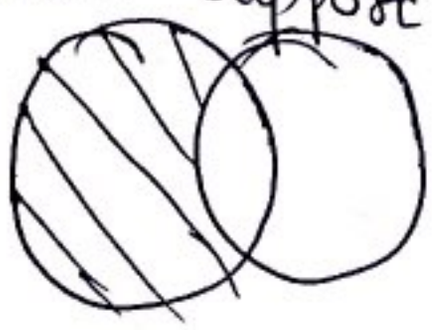
select * from first INTERSECT select * from second;

Result!→

ID	Name
2	Avi

EXCEPT / MINUS / DIFFERENCE operan combines result of 2 select stmts & return only those result which belongs to first set of result.

* MySQL doesn't support INTERSECT operator.



select * from First MINUS select * from Second;

Result!→

ID	Name
1	Abhi

→ **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.
  Ex: 5 + null returns null
- The predicate is null can be used to check for null values.
  Ex: select name from instructor where salary is NULL;
  " " " " " is NOT NULL;
- Any comparison with null returns unknown
  Ex: 5 < null, or null <, <> null, null = null.
- Three-valued logic using the truth value unknown:

DEPA

S. NO.

1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.
13.

bject Fa
kanksha

⊙ How to test for NULL values?

It is not possible to test for NULL values with comparison operators, such as =, <, <, >,

→ IS NULL Syntax:

Select column_names FROM table_name WHERE column_name IS NULL;

→ IS NOT NULL syntax:

Select column_names FROM table_name WHERE column_name IS NOT NULL;

→ SUBQUERIES / NESTED QUERY / INNER QUERY

⊙ A subquery or inner query or a Nested query is a query within another SQL query & embedded within the WHERE clause.

⊙ Subqueries that return more than one row can only be used with multiple values operators such as IN operator.

⊙ Sub-queries → Independent Sub-Queries.
→ Co-related " "

⊙ Independent sub-query:

→ Inner Query is independent of Outer Query.
→ " " executed first & the result are stored
→ Outer Query then runs on the stored results.

Eg: To list the cust_id & Loan_No for all customers who have taken a loan of amount greater than the loan amount of customer (Custid = 104).

comparison

columnname

tablename

: QUERY

query
the

only
s

y.
t are
ts-

es who
n amount

```
select cust_id , loan_No. FROM Customer_Loan ~~WHERE~~
WHERE Amount_in_dollars >
   (SELECT Amount_in_Dollars
        FROM customer_loan
            WHERE Cust_Id = 104);
```

Sub-Query

```
SELECT Amount_in_dollars
 FROM customer_loan
   WHERE cust_id=104;
```

data

cust_id	Loan-No	Amount_in_Dollars
101	1011	0755.00
103	2010	2555.00
104	2056	3050.00
103	2015	2000.00

step 1

3050.00

step-2 3050.00 compared with values in Amount-in-Dollars

cust_id	Loan_NO	Amount_in_Dollars
101	1011	0755.00
103	2010	2555.00
104	2056	3050.00
103	2015	2000.00

Query Result

101	1011

- ~~The~~ The inner query needs to be executed only once, since it returns one constant value irrespective of the outer query.

In above example, the innermost query is executed first, the result is stored & then the outer query is executed for each row of the customer-loan table

The inner query is executed only once, while the outer query is executed as many times as the no. of rows in the customer-loan table

Q1. List customers names of all customers who have taken a loan > $3000.00.

```
SELECT cust_name FROM customer WHERE
  Cust_Id IN (SELECT Cust_id from Customer_loan
                WHERE Amount_in_dollars > 3000.00);
```

Q2. List customers names of all customers who have the same Account_type as customer 'Jones Simon'.

```
SELECT cust_name FROM customer WHERE
  Account_type = (SELECT Account_type FROM customer
                    WHERE cust_name = 'Jones Simon');
```

Q3. List customers names of all customers who donot have a fixed deposit.

```
SELECT cust_name FROM customer WHERE
  cust_id NOT IN (SELECT cust_id FROM customer_fixed
                    -deposit);
```

Q4. List customers names of all customers who have either a fixed deposit or a loan but not both at any of the Bank Branches. It will include names that have no fixed deposit & loan as well.

```
SELECT cust_name FROM customer WHERE
  cust_id NOT IN (SELECT cust_id FROM customer_loan
                    WHERE cust_id IN
                      (SELECT cust_id FROM
                        customer_fixed_deposit);
```

(2) Co-related Sub-Queries:

In this, SQL performs a sub-query, once for each row of the main query. The column (s) of ~~from~~ outer-query is always referred in inner-query.

eg:- To list all customers who have a fixed deposit of amount less than the amount in dollars of all ~~their~~ loans.

```
SELECT cust_id, cust_name FROM customer_fixed_deposit
WHERE Amount_in_dollars < (
    SELECT ~~SUM~~ Amount_in_dollars
    FROM customer_loan
    WHERE customer_loan.cust_id = customer_fixed_deposit.
                                  cust_id);
```

customer_fixed_deposit

cust_id	cust_name	cust_email	fixed_deposit_NO	Amount in dollars	Rate_of_interest_in_percent
101	Smith	Smith@gmail	2011	8055.00	6.5
103	Langer	Langer@gmail	2015	2060.00	6.5
104	Quarts	Quarts@gmail	3010	3050.00	6.5

Fig Table-1 (outer query uses this table-1)

customer_loan

cust_id	Loan_No	Amount_in_dollars
101	1011	8755.00
103	2010	2555.00
104	2056	3050.00
103	2015	2000.00

Table-2 (Inner query uses this table-2).

~~The customer~~ Table-1 has 3 records. The inner query will be repeated three times. This is similar to the nested for loop ~~the~~ that has been covered in programming fundamentals course.

⊙ for the 1st record in ~~customer_fixed~~ Table-1.

→ (1) The record with the value of 101 in the cust_id column of the table-1 is read.

→ (2) All records with a value of 101 in cust_id column of table-2 are retrieved & their amount_in_dollars values are summed up. In this, there is only one record with a value of 101 in the cust_id column and the amount_in_dollars value is $0755.00.

table-2

101		
103		
104		
103		

customer_loan records from customer_loan table.

→ (3) This value is compared with ~~$0755.00~~ $0055.00. The target value of $0055.00 is less than the amount_in_dollars value of $0755.00, the record with cust_id value of 01 is part of the query result.

step-3

0055.00 < 0755.00 (True)

The record with cust_id = 101 from ~~custom~~ table-1 will occur in the query results.

~~Same all above 3 points~~

Similarly, ~~all~~ query will be repeated 2 times more for 2nd, 3rd record/row in table-1.

o/p of co-related query

cust_id	cust_name
101	Smith
103	Langer

* Note: For each row of table-1 to be tested by WHERE clause of main query, the cust_id column (which appears in the sub-query as an outer reference) has a different value. Thus SQL carries out this sub-query - once for each row/record in the table-1.

* A sub-query containing an outer reference is called a correlated subquery bcz its results are correlated with each individual row of the main query.

Q1.

SELECT
WHE
AND

Q1. List customer IDs of all customers who have both a fixed deposit & a loan at any of Bank Branches.

```
SELECT cust_id FROM customer.
WHERE cust_id IN
    (SELECT custid FROM customer_loan WHERE
        customer_loan.cust_id = customer.cust_id)
AND cust_id IN
    (SELECT custid FROM customer_fixed_depo WHERE
     customer_fixed_depo.cust_id = customer_~~loan~~.cust_id);
```

Scanned with CamScanner