**A Review of SQL**

(This was my former class notes for SQL, just for your reference)

**Topics**

1. Introduction of SQL
2. DDL (create, drop, alter, truncate)
3. Data type
4. DML (insert, update, delete)
5. Transaction control commands (commit, rollback)
6. Data control language (grant, revoke)
7. Data Retrieval, select statements (simple join, outer join, subquery)
8. Functions, single-row, aggregation (group by)
9. Handle NULLs,
10. Set operations (union, minus, intersect)
11. System catalog

Note: when you try to copy and paste any SQL statement from class notes to the SQL\* Developer, please pay attention that MS word has its own single and double quotes; many times, you may need to re-type them in the worksheet in the SQL\*Developer.

In worksheet/notepad: ' ' (Character string delimiter), " " (Quoted identifier delimiter).

In Microsoft word: ‘ ’, “ ”.

Sample **S, P, SP** tables are from textbook << An Introduction to Database Systems>>

by C.J. Date. (ISBN 0-321-19784-4). The script file crtS\_P\_SP.txt is posted on D2L.

**1. Introduction of SQL**

SQL: stands for Structured Query Language.

Actually, SQL is more than just query.

It is non-procedural, declarative language, no description of how to evaluate a query.

It is based on the theory of Relational Calculus.

A kind of sub language, interactive.

History:

Original developed at IBM, San Jose research Lab (now the Almaden Research

Center) in early 1970s, implemented as part of the system R project.

Standard:

SQL1, SQL-86, ANSI 1986, SQL standard; SQL-89;

SQL2, 1992,

SQL3, 1999, or called SQL-99. In 2003, XML-related features (SQL/XML) are

introduced; in 2016, JSON, row pattern matching, polymorphic table function is

added.

Virtually, all the commercial database products follow the rules.

SQL has several parts:

* Data Retrieval: select
* DDL, Data definition language

Provides commands for defining relation schemes, deleting relation,

creating index, modifying relation schemes.

DDL commands, change the meta-data – system catalog.

System (implicitly auto) COMMITS before and after each DDL statement.

No log record for undo for DDL commands.

Example: create (drop, alter, rename, truncate) table (database, view, index, tablespace, cluster, snapshot, sequence, trigger, user, role, synonym, ...)

* DML, Data manipulation language (interactive)

Commands: insert, delete, update, merge, …

DML does not change database structure.

Needs to commit or rollback.

* DCL, Data Control Language: grant, revoke
* Transaction Control Language: commit, rollback, savepoint.
* View
* Authorization (security, grant, revoke, connect command)
* Integrity (constraint, integrity check)

SQL provides rules for Embedded SQL and PL/SQL

**2. Data Definition Language (DDL)**

create, drop, alter (object\_type, object\_name; such as table, user, index, etc.)

**[CREATE TABLE]**

CREATE TABLE table\_name

(column\_name datatype [default expr] [column\_constraint],

column\_name datatype [default expr] [column\_constraint],

...

primary\_key\_definition,

foreign\_key\_definition, foreign\_key\_definition);

[] Brackets enclose optional items.

Default

specifies a value to be assigned to the column if a subsequent INSERT statement omits a value for the column. The datatype of the expression must match the datatype of the column.

Column\_constraint: defines an integrity constraint as part of the column definition.

Constraint Options: Not Null; Unique; Check; Primary\_key\_definition,

foreign\_key\_definition:

Table\_constraint, defines an integrity constraint as part of the table definition.

Example:

CREATE TABLE *S*

(s# CHAR(5) NOT NULL,

sname VARCHAR2(25) NOT NULL,

status NUMBER(3) CHECK (status => 10),

city VARCHAR2(25),

PRIMARY KEY (*S#*));

**[Example of** Default**]**

DROP TABLE BOOKRETURN;

CREATE TABLE BOOKRETURN (User\_Name char (50),

Return\_Date Date DEFAULT SYSdate+14);

INSERT INTO BOOKRETURN (User\_Name) values('Smith');

SELECT \* FROM BOOKRETURN;

**Create table with Subquery**

# CREATE TABLE Test\_S AS SELECT \* FROM S;

This command will create a new table Test\_S which is a duplication of table S including all the records and the “not null” (only) constraints.

Student can create their tables by simply copying from the instructor’s tables. After the instructor created his/her tables, the instructor should run the “grant” commands, for example:

GRANT SELECT ON employees to public;

This will allow every user in this instance to “read” the table.

From student side, after login to his/her account, type the command:

CREATE TABLE employees AS SELECT \* FROM czhang.employees ;

This command will copy the table definition and table records into the student account.

(Note, it will not copy the constraints except “Not Null”. So “alter table” commands will be needed if you need to add the constraints to the tables).

Name of object (not case-sensitive, “describe all\_tab\_columns”):

-length: less than or equal to 30 char, minimum 1

-could be letters, numbers, plus $, #, \_

-begins with an alphabetic character

Name qualification rule:

Unqualified names are acceptable if they cause no ambiguity.

**[DROP TABLE]** that delete the table and all its records.

DROP TABLE *TABLE\_NAME;*

Example: DROP TABLE *S*;

Note:

* Pay attention to the order of creating table and dropping table when there is foreign key constraint defined.
* You may drop that parent table with “CASCADE CONSTRAINT” without complying to the order.
* Pay attention to the style of scripts, they used to drop tables before creating.
* If there is space in the path, then you need to use the double quotes as below. @"D:\CH\453\File\_for\_posting\_09Winter\Script for creating tables\crtS\_P\_SP.sql"

**[TRUNCATE TABLE]** itremoves all the records from a table, but still keep the definition of the table. It is different from “DELETE TABLE\_NAME” which is a DML statement.

TRUNCATE TABLE table\_name; -- It is more efficient, but cannot rollback

Example: TRUNCATE TABLE s;

[ALTER TABLE]

ALTER TABLE *table\_name*

[ ADD ( {column\_element |table\_constraint } [, {….}] …) ]

[ MODIFY ( column\_element [, column\_element ] …) ]

[ DROP Column\_name ]

[ DROP constraint *constraint\_name* ]

{ } Braces enclose items only one of which is required.

| A vertical bar separates alternatives within brackets or braces.

... Ellipsis points show that the preceding syntactic element can be repeated.

column\_element : column\_name datatype constraint

Example:

ALTER TABLE S

ADD (firstname VARCHAR2 (20), SNN CHAR (12) );

- firstname is a newly added column.

ALTER TABLE S

MODIFY (status NUMBER(5));

( status is a column that already exists)

ALTER TABLE TEST

ADD ( constraint testname primary key (name)) ;

ALTER TABLE test5

MODIFY ( NAME CHAR(50) constraint test5name primary key ) ;

**[RENAME table\_name to new\_name] (rename view, sequence, synonym)**

Example:

Rename S to SUPPLIER ;

**[CONSTRAINTS**]

Table constraintand column constraint [ CONSTRAINT *constraint\_name* ]

**Column constraint**:

Example: (S# char(3) CONSTRAINT pk\_s PRIMARY KEY)

// do not need to indicate the column list.

**Table constraint**

CREATE TABLE S

( S# varchar2 (5) ,

, ...,

CONSTRAINT pk\_s PRIMARY KEY (S#));

**[Foreign Key constraint]**

Column constraint, example

deptno varchar2(3) CONSTRAINT fk\_emp REFERENCES department(deptno),

Table constraint

CONSTRAINT fk\_emp FOREIGN KEY (deptno) REFERENCES department(deptno)

[on delete cascade]

Using col\_constraint to define a PK or FK, it only works for single attribute key; for combination keys, table constraint is needed.

DROP constraint:

Example

CREATE TABLE junk

(Name varchar2 (25),

constraint jk\_pk primary key (Name));

alter table junk drop constraint jk\_pk;

If you do not give the constraint\_name, the system will assign a name for you

(like SYS\_C00418), you can get the information from table user\_constraints.

Samples for your future use:

DROP VIEW my\_pk ;

CREATE VIEW my\_pk AS

SELECT a.constraint\_name "Constraint Name", a.table\_name "Table Name",

b.column\_name "Col Name", constraint\_type T

FROM user\_constraints a, user\_cons\_columns b

WHERE a.constraint\_name = b.constraint\_name and a.owner=b.owner;

SELECT \* FROM my\_pk ORDER BY 4, 1, 2, 3;

**3. Data Types**

Each value manipulated by a database has a **data type**. The data type of a value associates a fixed set of properties with the value. These properties cause DBMS to treat values of one data type differently from values of another. For example, you can add values of NUMBER data type, but not values of RAW data type.

A few built-in Data types in Oracle:

(1) Character datatypes

CHAR (n): fixed length,

n: max # of bytes of data it can hold, maximum of n is 2,000, default is 1.

comparison: if two char strings both are defined as “char”, using blank-

padded comparison semantics (padding blank to the shorter one first)

VARCHAR2 (n): variable length

n: Max # of bytes it can hold, maximum of n is 4,000; must specify the n.

(non-padded comparison semantics)

VARCHAR(n) (temporarily) = VARCHAR2 (n)

NCHAR(n), NVARCHAR2 (n), Unicode Character string,

Literal, add N before ' '. Example:

INSERT INTO tablename (col) VALUES (N 'any characters you want');

SELECT . . . WHERE colname = N 'characters same as not unicode '.

(2) Number datatypes

NUMBER (P, S) fixed-point number

P: precision: total # of digits (1~38, range from 1 to 38)

S: scale: # of digits to the right of the decimal point (-84 ~ 127)

NUMBER (P) -- scale = 0

NUMBER -- floating point (P = 38)

Example

Actual data Specified as stored as

----------- -------------- -----------

1234567.89 Number 1234567.89 /\* floating point \*/

1234567.89 Number(9) 1234568 /\* p = 9, S = 0 \*/

1234567.89 Number(9, 2) 1234567.89 /\* p = 9, s = 2 \*/

1234567.89 Number(9, 1) 1234567.9 /\* p = 9, s = 1 \*/

1234567.89 Number(6) exceeds precision /\* see **\*1** below \*/

1234567.89 Number(7, 2) exceeds precision /\* see **\*2** below \*/

1234567.89 Number(15, 1) 1234567.9

1234567.89 Number(7, -2) 1234600 /\* s = -2 \*/

0.01234 Number(4, 5) 0.01234 /\* P = 4, do not count 0s \*/

0.00012 Number(4, 5) 0.00012

0.000127 Number(4, 5) 0.00013

0.0000012 Number(2, 7) 0.0000012

0.00000123 Number(2, 7) 0.0000012 /\* s = 7, --> 0000012 \*/

**\*1** p=6, s=0, but there are 7 digits 1234567

**\*2** p=7, s=2, allowed total number of digits is 7, but there are

totally 9 (p) digits 123456789 \*/

(3) Date datatypes

Stores *yyyymmddhhmmss* year, month, day, hour, minute, and second.

default data format: DD-MON-YY (MON are letters)

Example 07-OCT-96, 23-APR-97

You can specify the date format yourself, for example:

Alter session set NLS\_DATE\_FORMAT =’DD-MON-YY’;

Function **Sysdate**  returns the current date and the time.

Example

SELECT sysdate FROM DUAL;

create table ... (hiredate date default SYSDATE, ...

DUAL is a dummy table in Oracle. (Dual has one column and one tuple)

(think, why is there only one tuple in the table DUAL)

Function: To\_Char (date, format)

TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE (omitted).

(4) Other,

Long: Character data of variable length up to 2 gigabytes. 231 - 1

Raw (size): Raw binary data of length *size* bytes. Maximum *size* is 2,000 bytes*.*

Long Raw: Raw binary data of variable length up to 2 gigabytes.

Rowid: Hexadecimal string representing the unique address of a row in its table.

LOB, large Object data types, the built-in BLOB, CLOB, NCLOB, and BFILE can store

large and unstructured data such as text, image, video, and spatial data.

**4. Data Manipulation Language (DML)**

(1) **INSERT:** (single record insert into one table)

To add rows to a table (or to a view's base table).

INSERT INTO *TABLE\_NAME* [ (column [, column] ...) ]

{VALUES ( expr [, expr] ... ) | subquery }

* If you do not specify any column (other words, you omit [ (column [, column] ...) ]), then you have to give values for ALL the attributes of this table (ordered).
* If you specified the column names, then you can selectively load columns in a table, and you can also change the order of columns.

Ex : (single row insert

)

INSERT INTO *S* ( sname, s# ) VALUES ( ‘Noname’, ‘S9’ );

* If you use a subquery, “values” must be omitted.
* Pay attention to the case: SQL command is not case sensitive, but the literal is.

Insert multiple records:

CREATE TABLE NEWTB

( snumber VARCHAR2( 5),

scity VARCHAR2(25));

INSERT INTO newtb

SELECT s#, city FROM S WHERE status > 20 ;

(2) **DELETE**

Delete tuples from a table.

DELETE [FROM] table\_name [WHERE condition];

For the w*here* condition, the default is TRUE, that means it will delete every

tuple in this table.

Example

DELETE FROM SP WHERE qty < 200;

Example with subselect; delete all shipments for suppliers in London.

DELETE FROM SP WHERE 'London' =

(SELECT city from s where sp.s# = s.s#);

result 9 rows deleted.

(3) **UPDATE** (modifying existing record(s))

UPDATE table\_name

SET *field* = [*, field* = , ...... ]

[WHERE condition ]

For the *where* condition, the default is TRUE, that means it will update every tuple in this table.

Example

UPDATE S SET city = ‘Chicago’, status = 25

WHERE city = ‘London’ ;

Set shipment to zero for all suppliers in London.

UPDATE SP SET QTY = 0 WHERE ‘London’ =

(SELECT city FROM S WHERE s.s# = sp.s# );

**Transaction Control statements**

COMMIT [WORK]:

To end your current transaction and make permanent all changes performed

in the transaction.

ROLLBACK [WORK]:

To undo work done in the current transaction (up to last commit).

Picture: CPU < -- > RAM < -- > DB in the hard disk.

RAM is volatile memory storage, all the info will be gone after the power down.

Hard disk is the permanent storage for data – where DB resides.

Sample: assume you already have the S – P – SP tables built.

In session #1:

Insert into S values ( ‘S6’, ‘Kevin’, 30, ‘Chicago’ ) ;

SELECT \* FROM S ; - you will see this record in the S table;

In another session #2, log in as the same name,

SELECT \* FROM S ; - you will NOT see this record in the S table;

Then in session #1, you type:

Commit ;

Go to session #2

SELECT \* FROM S ; - Now you see this record in the S table, as it is in the hard disk;

Do similar exercise, but this time with rollback; that means, you tell the system “undo”, or cancel, do not write any changes in the RAM into the DB (hard disk).

For the DML commands (e.g. insert, update and delete) you need to explicitly

type the command “commit” to save the result (into the hard disk).

**Implicit commit**: in the following cases, the system will automatically commit the transaction for you.

* Whenever Oracle executes a DDL (such as “create”, “drop” command)
* “Autocommit” is set on in the environment
* A normal exit from Oracle (such as type exit in the SQL\* Developer command line; an abnormal exit – such as click the “exit” in the window box, will cause rollback ).

**Comments**:

-- Single-Line comments

/\* Multiline Comments, C type comment \*/

REMARK this line is comment

**5.**  **Data Control Language (DCL) (grant** and **revoke** commands):

**GRANT**  privilege [, privilege ...] | **REVOKE** …

on table\_name | on…….

TO user [,user ..] | FROM

[with grant option] | [cascade constraint…]

**Sample**: start Oracle after installation; first login as system, then:

1. You want to create a DBA account for yourself; or someone:

CREATE USER myname IDENTIFIED BY mypassword;

GRANT DBA TO myname WITH ADMIN OPTION;

/\* you can delete the account by: DROP USER myname; \*/

1. You want to create a regular (none DBA) account

CREATE USER someonename IDENTIFIED BY hispassword;

GRANT CONNECT, RESOURCE TO someonename;

(c ) You wan to grant the read for everyone

GRANT SELECT ON employees TO public; -- Or

GRANT SELECT, UPDATE ON employees TO usera, userb;

To cancel it:

REVOKE SELECT ON employees FROM public;

You can put multiple privileges/multiple users in a single statement;

but you can ONLY put a single table (or view) in one statement.

**6. Data Retrieval: SELECT statements**

To retrieve data from one or more tables or views.

**SYNTAX:**

SELECT [DISTINCT | ALL] {attribute\_list...}

FROM table\_name [table\_alias] [,table\_name [table\_alias] ] ...

[WHERE condition]

[GROUP BY attribute\_name [, attribute\_name] ...

[HAVING condition]]

[{UNION | UNION ALL | INTERSECT | MINUS} SELECT command]

[ORDER BY attribute\_name | position} [ASC | DESC]

[, {attribute\_name | position} [ASC | DESC]] ...]

schema.table.attribute:

Example SQL> desc scott.dept

SQL> SELECT scott.dept.deptno , scott.dept.dname

FROM scott.dept;

DISTINCT: returns only one copy of each set of duplicate rows selected.

ALL: returns all rows selected, including all copies of duplicates.

The default is ALL. Distinct is for the row.

**WHERE** clause:

If omitted, condition is true (all tuples selected)

Logical operators: and, or, not

Not != , <>, ^=

With a range search between 1000 and 2000 (inclusive)

salary >=1000 and x <= 2000

Pattern search: [NOT] like ‘%Smith%’,

-- % zero or more match, \_ one and only one

With a search list: S# [NOT] in (‘S1’, ‘S2’);

NULL : is [NOT] NULL

[NOT] exists

nested query (subquery) (subselect)

**Comparison**

Comparison operators:

= ; != , <>, ^= ; >, >=, <, <=

" = ANY ( ) " 🡪 IN;

“!= ALL ( ) " 🡪 NOT IN;

**GROUP BY**

groups the selected rows based on the value of expr for each row and

returns a single row of summary information for each group.

**HAVING**

restricts the groups of rows returned to those groups for which the

specified condition is TRUE. If you omit this clause, Oracle

returns summary rows for all groups.

**ORDER BY**

orders rows returned by the statement. (using alias if exists)

Position: means position of the select list.

ASC ascending order. ASC is the default.

DESC descending order

Ex: Get supplier #, name and status for suppliers in Paris.

SELECT S#, SNAME , STATUS

FROM S

WHERE CITY = ‘Paris’ ;

S# SNAME STATUS

----- --------------- ----------

S2 Jones 10

S3 Blake 30

Example Retrieval with ordering:

The above question, plus: ordered first by name then by status.

SELECT S#, SNAME , STATUS

FROM S

WHERE CITY = ‘Paris’

ORDER BY SNAME , STATUS ;

-- ORDER BY 1, 2 [ASC] or [DESC] ; ASC is default ;

**Rule of Precedence from highest to lowest (skip)**

1 Arithmetic operators

2 Concatenation operator

3 Comparison conditions

4 IS [NOT] NULL, LIKE, [NOT] IN

5 [NOT] BETWEEN

6 NOT logical condition

7 AND logical condition

8 OR logical condition

**6.1. Temporarily rename a column and**

**temporarily rename a table**

- these only affect the output this statement, nothing is changed in the table definition.

Temporarily rename a table in the SELECT STATEMENT:

SELECT e.Emp\_ID

From employees e;

Here we rename the table employees as e, we have to use the alias e as qualifier everywhere in this statement, be careful with select list.

Temporarily rename a column (heading) in the SELECT STATEMENT

(that will affect the heading display in the output)

SELECT scott.dept.deptno [AS] department

SELECT S# supplier\_number, .....

SELECT S.city " supplier city", ....

SELECT first\_name||' ' ||last\_name "Full Name"

FROM employees;

Note, " " must be used if there is space in the column alias.

**6.2. Extend operation:**

Example For all parts, get the part number and the weight of that part in grams.

SELECT P.P#, P.Weight \* 454 GMWT

FROM P ;

P# GMWT

----- ----------

P1 5448

P2 7718

...

**6.3. Partial matching**:(like, %, \_ )

Example, List all salesperson's names starting with S.

SELECT SNAME

FROM S

WHERE UPPER (SNAME) LIKE 'S%';

/\* COMPARE: WHERE UPPER (SNAME) LIKE '\_HANG' \*/

/\* What will you get if you type where upper (sname) = 'S%' ? \*/

**6.4. Retrieve data from more than one table**

When designing/creating the tables for a business, it is required to follow the normalization rules. We need to keep the data integrity when we need to insert/delete/update (DML) on the contents of the tables - the business information. That means we need to divide the business information into a few relative independent pieces - tables for the business growth, maintenance.

This causes the problem with retrieving information which are separated and located in different tables. Hence joining several tables are needed.

If the information does not require updated for every moment, materialized view will do the work. For those business, where the information does not need the strict maintenance integrity, relation database is not a preferred choice.

When information cuts into tables, they are linked by repeating certain column(s) in two tables, most times, these are primary (unique) key for one table and repeated in another table called/defined as foreign key. When retrieving info from these two tables, we use the condition in the where clause tableA.pk = tableB.fk, plus other condition for the row restrictions.

If joining three tables (or more than three), system will join two first, then join the third, just as we do the arithmetic 2 + 3 + 4 = (2 + 3) + 4.

**Natural join:** most time, the common attribute(s) among the two joining tables is/are the PK-FK (where FK = PK)

Example Natural join of the two tables S and SP. Common attribute: S#

SELECT S.S#, S.SNAME, S.STATUS, S.CITY, SP.P#, QTY

FROM S, SP

WHERE S.S# = SP.S# ; -- explicitly defining the natural join condition, equal on

-- common attributes

S# SNAME STATUS CITY P# QTY

----- ------------------ ----------- ----- ----------

S1 Smith 20 London P1 300

S1 Smith 20 London P2 200

S1 Smith 20 London P3 400

...

Using qualifier to distinguish ambiguous column name

SQL99 Syntax is just for your future reference, not required.

Samples for using SQL 1999 Syntax (no qualifier is used in natural join syntax, assuming that the common attribute(s) has the same name, and no none-join-condition columns have the same name in these two tables)

SELECT S# , S.SNAME , STATUS, CITY, P#, QTY

FROM S NATURAL JOIN SP ; -- same query as above, but using SQL99 syntax

-- The common attribute in S and SP table has the same name of “S#”.

-- Besides “S#”, there is no other attribute in S and SP has the same name.

-- It is not allowed to add qualifier (table name) before the common attribute.

-- If you change “SELECT S# ” into “SELECT S.S# ”, you get error.

SELECT S# , S.SNAME , STATUS, CITY, P#, QTY

FROM S JOIN SP USING (S#) ;

SELECT S.S# , S.SNAME , STATUS, CITY, P#, QTY

FROM S JOIN SP ON (S.S# = SP.S#) ;

SELECT *table1.column, table2.column*

FROM *table1*

[CROSS JOIN *table2*] | -- Cartesian product

[NATURAL JOIN *table2*] | -- names are same on the common attributes

[JOIN *table2* USING (*column\_name*)] |

[JOIN *table2*

ON(*table1.column\_name* = *table2.column\_name*)] |

[LEFT|RIGHT|FULL OUTER JOIN *table2*

ON (*table1.column\_name* = *table2.column\_name*)];

SELECT sname FROM S CROSS JOIN SP ; -- 60 rows selected

SELECT sname FROM S NATURAL JOIN sp; -- 12 rows selected

SELECT sname FROM S JOIN sp on (S.S# =SP.S#); -- 12 rows selected

SELECT sname FROM S JOIN sp USING (S#); -- 12 rows selected

Example (Three tables)

SELECT s.s#, sname, S.CITY SCITY, p.p#, pname, P.CITY PCITY, QTY

FROM S, P, SP

WHERE S.S# = SP.S# and P.P#= SP.P#;

Get all pairs of city names such that a supplier located in the

first city supplies a part stored in the second city.

SELECT DISTINCT S.CITY SCITY, P.CITY PCITY

FROM S, P, SP

WHERE S.S# = SP.S# and P.P#= SP.P#;

SELECT DISTINCT S.CITY SCITY, P.CITY PCITY

FROM S JOIN SP ON (S.S# = SP.S#) JOIN P ON ( P.P#= SP.P# );

SELECT DISTINCT S.CITY SCITY, P.CITY PCITY

FROM S **NATURAL JOIN** SP JOIN P ON ( P.P#= SP.P#);

SCITY PCITY

--------------- ---------------

London London

London Paris

London Rome

Paris London

Paris Paris

Please pay attention to the “DISTINCT”.

Notes about “**Natural Join**” syntax. -- (skip it, if you do not use SQL99).

Pay attention, in “S NATURAL JOIN SP”, system will assume/find the common attribute with same name. But sometimes, the columns with same name in two tables may not have relation of PK-FK. S table and P table both have a column called “CITY”, but these are not related as PK-FK, the natural join syntax will not produce the right output for this kind case.

There will be no error to run the statement below, as there is one common attribute “CITY” between S and P, but the output is not the usual output for natural join. The system assumes the attribute with same name with the PK-FK relationship.

SELECT S# , S.SNAME , STATUS, CITY, P#

FROM S Natural JOIN P;

The statement below will not produce the real natural join result, as the result table of

(S natural join SP) has two columns with same names in P, CITY and P#.

SELECT S#, sname, status, city, P#, QTY, Pname, color, weight

FROM (S NATURAL JOIN SP) natural JOIN P ;

The statement below avoids this problem.

SELECT S#, sname, status, city, P#, QTY, Pname, color, weight

FROM SP NATURAL JOIN S

natural JOIN (Select P#, pname, color, weight, city Pcity from P)

order by 1, 5;

Unless you are sure that the common attributes for joining condition have the same column name, and NO other none-join-condition attribute has the same name, do not use this syntax.

In Oracle HR. there are two tables: employees, departments.

These two tables have two columns with exactly same name: department\_id, manager\_ID.

For joining these two tables, the correct column for joining should be only one column: the department\_id.

According the rule, Natural join will equal on all common columns with same name,

department\_id and manager\_ID.

-- IF using inner join on cola = colb

SELECT d.department\_id, d.department\_name, sum(salary) AS tot\_salary

FROM departments d inner JOIN employees e

on d.department\_id = e.department\_id

where e.department\_id is not null

GROUP BY d.department\_id, department\_name

order by 1;

It is same as:

SELECT d.department\_id, d.department\_name, sum(salary) AS tot\_salary

FROM departments d , employees e

where d.department\_id = e.department\_id and e.department\_id is not null

GROUP BY d.department\_id, department\_name

order by 1;

That gives the correct result:

DEPARTMENT\_ID DEPARTMENT\_NAME TOT\_SALARY

------------- ------------------------------ ----------

10 Administration 4400

20 Marketing 19000

30 Purchasing 24900

40 Human Resources 6500

50 Shipping 156400

60 IT 28800

70 Public Relations 10000

80 Sales 304500

90 Executive 58000

100 Finance 51600

110 Accounting 20300

11 rows selected.

-- If using SQL99 syntax “Natural join”, the result will miss three rows; and the sum are not correct,

SELECT department\_id, department\_name, sum(salary) AS tot\_salary

FROM departments d NATURAL JOIN employees e

where department\_id is not null

GROUP BY department\_id, department\_name

order by department\_id;

It is the same as query below:

SELECT d.department\_id, d.department\_name, sum(salary) AS tot\_salary

FROM departments d, employees e

where d.department\_id = e.department\_id and

d.manager\_id = e.manager\_id and e.department\_id is not null

GROUP BY d.department\_id, department\_name

order by department\_id;

DEPARTMENT\_ID DEPARTMENT\_NAME TOT\_SALARY

------------- ------------------------------ ----------

20 Marketing 6000

30 Purchasing 13900

50 Shipping 25400

60 IT 19800

80 Sales 51000

90 Executive 34000

100 Finance 39600

110 Accounting 8300

8 rows selected.

**6.5. Outer Join Operator: (+)**

The outer join extends the result of a simple join. An outer join returns all the rows returned by the simple join as well as those rows from one table that do not match any rows from the other table. Append (+) to the table and column, and you will get NULLs for columns of this table whenever this table has no rows to join to a row in the other table.

In the following sample, some supplier, such as S5, does not appear in the SP table. In the simple join, S5 will not show in the result table.

Example

List the s#, sname, city of the suppliers, and the part name, quantity they supply. The result should include the suppliers who do not supply anything now. - this requires to use outer join.

set null NULL

SELECT S.S#, SNAME, CITY, SP.P#, QTY

FROM S, SP

WHERE S.S# = SP.S#(+); -- left outer join

-- or using right outer join: WHERE SP.S# (+) = S.S#;

SELECT S.S#, SNAME, CITY, SP.P#, SP.QTY

FROM S LEFT OUTER JOIN SP ON (S.S# = SP.S#);

S# SNAME CITY P# QTY

----- ---------- --------------- ----- ----------

S1 Smith London P1 300

S1 Smith London P2 200

S1 Smith London P3 400

S1 Smith London P4 200

S1 Smith London P5 100

S1 Smith London P6 100

S2 Jones Paris P1 300

S2 Jones Paris P2 400

S3 Blake Paris P2 200

S4 Clark London P2 200

S4 Clark London P4 300

S4 Clark London P5 400

S5 Adams Athens NULL NULL

The following code will do the same work as Outer join.

SELECT S.S#, SNAME, CITY, SP.P#, QTY

FROM S, SP

WHERE S.S# = SP.S# -- Natural join

UNION -- Plus

SELECT S.S#, SNAME, CITY, null, null -- Add nulls to match the number of columns

FROM S

WHERE S.S# in

(SELECT S.S# FROM S -- select the dangling records which are/is not

MINUS -- matched in Natural join

SELECT SP.S# FROM SP);

**6.6. Join a table to itself**

Example Get all pairs of supplier numbers such that the two suppliers concerned

are collocated.

SELECT A.S#, B.S#

FROM S A, S B

WHERE A.CITY = B.CITY AND A.S# < B.S#;

S# S#

----- -----

S2 S3

S1 s4

-- REMARK “Distinct” will not give the right answer for this question.

select distinct a.s#, b.s# from s a, s b where a.city = b.city ;

S# S#

----- -----

S1 S1

S1 S4

S2 S2

S2 S3

S3 S2

S3 S3

S4 S1

S4 S4

S5 S5

Example Get all the supplier names that ALWAYS have the same status as Smith’s

status.

SELECT A.SNAME

FROM S A , S B

WHERE A.STATUS = B.STATUS AND UPPER ( B.SNAME ) = ‘SMITH’ and

UPPER (A.SNAME ) != ‘SMITH’;

SNAME

---------------

Clark

-- method #2 **(subquery**) (return a single value from subquery here)

SELECT sname

FROM S

WHERE sname != ‘Smith’ and STATUS =

(SELECT status

from S

where upper (sname) = ‘SMITH’);

**Notes:** Some jargons related to joins. (**skip**)

Equijoin equivalent on special columns,

Self join From employees a, employees b

Outer joins discussed

Cartesian products: no join condition, number of rows in output

= row number of table A \* row number of table B

Inner join: (also called simple join) returns only those rows that satisfy the join condition.

From table\_a inner join table\_b on table\_a.col\_1 = table\_b.col\_2 (and ...)

From table\_a natural join table\_b (no qualifier allowed on common attribute in the SQL statement)

The difference:

Natural join: equal on all common columns with same name.

inner join: equal on only the column(s) in the "on cola = colb" list only

**Band** joins nonequivalent (special range, “band”)

**Anti**joins, returns rows from the left side of the predicate for which there are no corresponding rows on the right side of the predicate. It returns rows that fail to match (NOT IN) the subquery on the right side.

**Semi**joins, returns rows that match an EXIST subquery without duplicating rows from the left side of the predicate when multiple rows on the right side satisfy the criteria of the subquery.

Example of using **Antijoin**: The following example selects a list of employees who are not in a particular set of departments:

SELECT \* FROM employees

WHERE department\_id **NOT IN**

(SELECT department\_id FROM departments

WHERE location\_id = 1700)

ORDER BY last\_name;

Example of using **Semijoins:**

In the following example, only one row needs to be returned from the departments table, even though many rows in the employees table might match the subquery.

SELECT \* FROM departments

WHERE **EXISTS**

(SELECT \* FROM employees

WHERE departments.department\_id = employees.department\_id

AND employees.salary > 2500)

ORDER BY department\_name;

Example of using **Band** Join**:** (there will be 553 rows selected)

SELECT e1.last\_name || ' has salary between 100 less and 100 more than '

|| e2.last\_name AS "SALARY COMPARISON"

FROM employees e1, employees e2

WHERE e1.salary BETWEEN (e2.salary - 100) AND (e2.salary + 100);

**7. Functions**

Functions: accept input and return some values.

Performing calculations on data; Modifying individual data items; Manipulating output for groups of rows; Formatting dates and numbers for display; Converting column datatypes; etc.

**7.1 Single-Row Functions**

(a) Numeric Functions: some samples (Floor, Power, round, trunk, mod)

FLOOR(n) returns largest integer equal to or less than n

power(m, n) returns m raised to the nth power.

Example

SELECT power (4, 3) "four to three" from DUAL;

four to three

-------------

64

Example

ROUND (98.6548678, 3) 98.655

TRUNC (98.65486, 2) 98.65

MOD (16, 3) 1 -- the remainder of 16 ÷ 1

(b) Character case Conversion Functions

LOWER - Converts character string to lowercase

UPPER - Converts character string to uppercase

INITCAP - Converts the first letter of each word to upper case and the rest of the string to lowercase.

Example

SELECT \* FROM S

WHERE upper (sname)= 'SMITH';

(c) Date Functions (operate on date, timestamp and interval values), some samples:

Sysdate returns the current date and time. requires no arguments.

Months\_between (d1, d2) returns number of months between date d1 and d2.

Example: Select sysdate "TODAY" from Dual;

(d) Character-manipulation Functions, some samples.

Function Result

CONCAT('Hello', 'World') HelloWorld -- joins literals together

SUBSTR('HelloWorld',1,5) Hello -- returns a string of a determined length

LENGTH('HelloWorld') 10 -- returns the length of a string

LPAD('Page 1',15,'\*.') \*.\*.\*.\*.\*Page 1 -- left-pads \*. to length 15.

RPAD (ename,10, ' ') KING -- right-pads space to length 10

(e) Conversion Functions:

TO\_CHAR - converts a number or date value to VARCHAR2.

TO\_DATE - convert a character string representing a date to a date value.

TO\_NUMBER - convert a character string to a number.

SELECT TO\_CHAR(1234.89, '$9,999.00') FROM dual; -- RESULT: $1,234.89

TO\_CHAR(sysdate, 'Month DD, YYYY' -- RESULT: August 16, 2019

TO\_DATE('20190916', 'yyyymmdd') -- RESULT: 16-AUG-19

TO\_DATE('081619', 'MMDDYY') -- RESULT: 16-AUG-19

TO\_NUMBER('1234.56', '999999.99') -- RESULT: 1234.56

TO\_NUMBER('1234.56', '9999.99') -- RESULT: 1234.56

TO\_NUMBER('1234.56') -- RESULT: 1234.56

**7.2. Aggregate Functions (Group functions**)

return a single result row based on groups of rows, rather than on single rows.

(a) **COUNT** (**{** \* |[ Distinct | All ] expr **}**)

Returns the number of rows in the query.

Example, Get total number of suppliers. (count)

SELECT count (\*) number\_of\_suppliers

FROM S;

NUMBER\_OF\_SUPPLIERS

-------------------

5

SELECT count (distinct P#) FROM SP;

COUNT(distinct P#)

--------------------

6

(b) **MAX (**[Distinct | All] expr), **MIN**

Returns the maximum value of expr.

Example Get the maximum and minimum quantity for part P2. (MAX, MIN)

SELECT MAX (QTY), MIN (QTY)

from SP

WHERE p# = 'P2';

MAX(QTY) MIN(QTY)

---------- ----------

400 200

1. **SUM (**[Distinct | All] **n)**

Returns the sum of value of n. Default is All.

Example Get the total shipping quantity of all the red parts.

select sum (QTY) "Total Red"

from SP , P

where SP.P# = P.P# and Color = 'Red';

Total Red

---------

700

(d) GROUP BY

Groups the selected rows based on the values of expr for each row and returns a single row of summary information for each group.

Example. For each part supplied, get the part number and the total shipment quantity.

SELECT P#, SUM (QTY)

FROM SP

GROUP BY P#;

P# SUM(QTY)

----- ----------

P1 600

P2 1000

P3 400

P4 500

P5 500

P6 100

Get part numbers for all parts supplied by more than one supplier.

SELECT P#, count (P#) “count of P#”

FROM SP

GROUP BY P#

HAVING count(P#) > 1;

P# count of P#

----- -----------

P1 2

P2 4

P4 2

P5 2

Example. Get all parts numbers and their total quantity where total quantity is more than 500.

SELECT P#, SUM (QTY) "TOTAL SHIPMENT"

FROM SP

GROUP BY P#

HAVING SUM (QTY) > 500;

P# TOTAL SHIPMENT

----- --------------

P1 600

P2 1000

Note:

When using “GROUP BY”, each item in the SELECT\_list must be single-valued per group.

**8. Handling NULLs:**

NULL means information missing or unknown.

NULL is different from space (its ASCII hexadecimal value is 20) or zero (30).

If a row lacks value for a particular column, that column is said to be NULL.

(a) Retrieval involving NULL: SELECT

SELECT first\_name ||' '||last\_name "full Name"

FROM employees

WHERE manager\_id IS NULL ;

-- never use "= NULL" in the "where" clause.

(b) DISPLAY: the default is blank space, but you can change it using command

"set NULL something\_you\_want",

example: "set NULL NULL".

In SQL\* Developer

Tools  Preferences Database  Advanced  Display Null Value As.

(c) INSERT:

NULL must be present, you cannot leave blank space.

Example

INSERT INTO S VALUES ( 'S6', 'Thomas', *NULL*, 'Chicago') ;

WRONG : INSERT INTO S VALUES ('S6', 'Thomas', , 'Chicago') ;

WRONG : INSERT INTO S VALUES ('S6', 'Thomas', ' ' , 'Chicago');

(d) UPDATE

UPDATE S SET status **=** NULL WHERE s# = 'S5' ;

NVL (expr1, expr2) function: converts a null (expr1) to an actual value

-- expr1: the source that may contain null, returns expr1 if it is not null.

-- expr2: if expr1 is null, then function returns expr2.

The datatype of the return value is always the same as the datatype of *expr1*, unless *expr1* is character data. If expr1 and expr2 have different datatypes, then Oracle converts *expr2* to the datatype of *expr1* before comparing them.

Example:

SELECT empno , ename , sal + comm FROM emp;

-- (anything + null) result in null.

SELECT empno, ename, sal + NVL(comm, 0) “total income” FROM emp ;

-- if “comm” value is null, then convert it to zero.

**9. Subquery/ Nested Query**

A query (select) nested within another SQL statement is called subquery.

A subquery in the FROM clause of a SELECT statement is also called an inline view.

A subquery in the WHERE clause of a SELECT statement is also called a nested subquery.

1. **Unnesting of Nested subquery**

system optimizer may automatically unnest some (the uncorrelated) subquery.

Example. Get supplier numbers for suppliers whose status is greater than that

of supplier S1.

SELECT S#

FROM S

WHERE STATUS >

( SELECT STATUS

FROM S

WHERE S# = 'S1');

-- The nested (sub) query may return a single value, or multiple values,

S#

-----

S3

S5

We can rewrite the code using Join (a table with itself):

SELECT A.S#

FROM S A, S B

WHERE B.S# = 'S1' AND A.STATUS > B.STATUS AND A.S# > B.S#;

**(2) Correlated subquery**

Whenever a condition in the WHERE clause of a nested query references some attributes of a relation declared in the outer query, the two queries are said to be correlated.

Example Get supplier names for suppliers who supply part P2.

SELECT DISTINCT S.SNAME

FROM S

WHERE s.s# =

( SELECT sp.s#

from SP

where *SP.S# = S.S#* AND P# = 'P2');

**(3) IN/NOT IN**

expr in (.., ...) *equals to* expr = ...or expr = ... or

expr not in (.., ...) *equals to* expr != ..and expr != ... and

Example Get all parts information if their colors are either red or blue.

SELECT \*

FROM P

WHERE COLOR IN ('Red', 'Blue');

-- same as WHERE COLOR = 'Red' OR COLOR ='Blue'

Example Get supplier names for suppliers who supply at least one red part.

SELECT S.SNAME

FROM S

WHERE S.S# IN

( SELECT SP.S#

from SP

where SP.P# IN

( SELECT P.P#

from P

where P.color = 'Red'));

SNAME

----------

Smith

Jones

Clark

-- method #2

SELECT DISTINCT SNAME

FROM S, P, SP

WHERE S.S# = SP.S# and P.P# = SP.P# and P.color = 'Red';

Example Get supplier names for suppliers who supply part P2.

SELECT DISTINCT S.SNAME

FROM S

WHERE S.S# in

( SELECT S.S#

from SP

where P# = 'P2');

SNAME

----------

Adams

Blake

Clark

Jones

Smith

**(4) EXISTS/NOT EXISTS**

Exists: is a Boolean function that returns TRUE or FALSE.

Result of EXISTS is TRUE if there is at least one tuple in the nested query, False if there is

no tuple returns in nested query.

NOT ExistS is TRUE if there is no tuple in the result of nested query (returned set is empty).

Example: Get supplier names for suppliers who supply part P2 (same as above).

SELECT DISTINCT S.SNAME

FROM S

WHERE EXISTS

( SELECT \*

from SP

where SP.S# = S.S# AND P# = 'P2');

Using JOIN:

SELECT DISTINCT S.SNAME

FROM S, SP

WHERE SP.S# = S.S# AND P# = 'P2';

Example. Get supplier names for suppliers who do **NOT** supply part P2.

SELECT DISTINCT S.SNAME

FROM S

WHERE NOT EXISTS

( SELECT \*

from SP

where SP.P# = 'P2' and SP.S# = S.S#);

SNAME

----------

Adams

REMARK method #2

SELECT DISTINCT S.SNAME

FROM S

WHERE S.S# not IN

( SELECT SP.S#

from SP

where SP.P# = 'P2');

**10. SET OPERATIONS**

Set operators combine the results of two queries into a single result.

(a) UNION

Returns all distinct rows selected by either query.

Example:

Get part numbers for parts that either weigh more than 16 pounds

or are supplied by supplier S2, or both.

SELECT P#

FROM P

WHERE WEIGHT > 16

UNION

SELECT SP.P#

FROM SP

WHERE SP.S# = 'S2’;

P#

-----

P1

P2

P3

P6

UNION (ALL) : returns all rows selected by either query, including all duplicates

If UNION ALL is used in the above sample, then P2 will be shown twice (duplicated).

Note: select-list (of columns) is not necessarily the same set of names for the two "SELECT"s.

(b) MINUS

Returns all distinct rows selected by first query but not the second.

Example:

Get supplier names that do not supply part P2.

SELECT SNAME

FROM S

MINUS

SELECT DISTINCT SNAME

FROM S, SP

WHERE SP.S# = S.S# AND P# = 'P2 ';

SNAME

----------

Adams

(c) INTERSECT

Returns all distinct rows selected by both queries.

Example

Get supplier numbers for suppliers whose city is London and supply part P2.

SELECT S.S#

FROM S

WHERE UPPER (CITY) = 'LONDON '

INTERSECT

SELECT DISTINCT SP.S#

FROM SP

WHERE P# = 'P2 ';

S#

-----

S1

S4

1. **System Catalog (Data Dictionary)**
2. Definition:

System catalog is a group of tables and views that incorporate vital details regarding a database. The information specifies the framework of the database. Sometimes, it is called as metadata (data about data).

* Stores the information for the DBMS to operate.
* Includes all the structures that make up the objects (such as tables, views, indexes, users, security rules, etc., ...) inside the database
* Includes v$ (in Oracle) dynamic performance view and tables.

1. Catalog consists of tables (system tables)

* the information is stored as tables.
* also including the information of itself (self-descriptive).
* the catalogs are updated by DDL statements.

For example, the statement Create table S (S#..., Sname, Status, City, ...);

will insert one row in *all\_tables* and four rows in *all\_columns*.

1. Querying system catalog

A user may look up the system catalog to gain information regarding his own objects as well as privileges; while the database admin must be capable of inquiring about any event or structure inside the database, it is extremely important and useful for DBA.

* User Views (User\_ ) Hold information about the current user’s objects.
* All Views (All\_ ) Hold information about all the objects the user has access to.
* DBA Views (DBA\_ ) Hold information about all the objects in the database.
* Dictionary (DICT -- synonym for dictionary)

Desc dictionary

Name Null? Type

------------------------------ ----- --------------

TABLE\_NAME varchar2(128)

COMMENTS varchar2(4000)

Column table\_name format A30;

Column comments format A45;

SELECT \* FROM dict;

-- 1,067 rows selected.

Some useful tables/views:

Tab tname, tabtype (table, view), clusterid

Cat table\_name, table\_type ( table, view, sequence, …)

User\_indexes index\_name, index\_type, table\_name, …..

user\_constraints owner, constraint\_name, constraint\_type, table\_name, …

user\_cons\_columns owner, constraint\_name, table\_name, column\_name, ...

user\_objects object\_name, subobject\_name, object\_id, ….

user\_synonyms syn

user\_tables tabs

user\_tab\_columns user\_views

user\_tab\_privs grantee, owner, table\_name, grantor, privilege, grantable, ..

user\_sys\_privs privilege, admin\_option

Some samples:

purge recyclebin;

COLUMN TNAME FORMAT A25

SELECT \* FROM tab order by 2, 1; (including TABLE, VIEW)

COLUMN TABLE\_NAME FORMAT A25

SELECT \* FROM cat ORDER BY 2, 1; (including TABLE, VIEW, SEQUENCE)

SELECT username FROM user\_users;

SELECT user FROM DUAL;

SHOW user;

column index\_name format A20

column index\_type format A20

column table\_name format A20

SELECT index\_name, index\_type, table\_name FROM user\_indexes;

select table\_name from dictionary where table\_name like '%V$%';

104 rows selected.

set pagesize 55

column constraint\_name format A22

column table\_name format A15

column column\_name format A15

column owner format A10

-- drop view my\_pk;

CREATE VIEW my\_pk AS

SELECT a.owner, a.constraint\_name, a.table\_name, b.column\_name, constraint\_type t

FROM user\_constraints a , user\_cons\_columns b

WHERE a.constraint\_name = b.constraint\_name and a.owner=b.owner;

SELECT \* FROM my\_pk ORDER BY 1, 5, 2, 3, 4;

Some samples of querying the system catalog.

column TABLESPACE\_NAME format A20

column BYTES format 999,999,999

desc user\_free\_space

SELECT \* FROM user\_free\_space;

desc user\_users

COLUMN username FORMAT A25

SELECT username, user\_Id, created FROM user\_users;

desc user\_resource\_limits

SELECT \* FROM user\_resource\_limits;

desc user\_tab\_privs

Column GRANTEE format A10

Column GRANTOR format A10

Column OWNER format A10

Column TABLE\_NAME format A12

Column PRIVILEGE format A15

SELECT \* FROM user\_tab\_privs;

desc dictionary

column comments format A45

column TABLE\_NAME format A35

SELECT table\_name, comments

FROM dictionary

WHERE table\_name LIKE '%\_AUDIT%';