

Monday, February 13, 2017 9:56 PM

DML Data Manipulation Language => SELECT, INSERT, UPDATE, DELETE, MERGE

DDL Data Definition Language => CREATE, ALTER, DROP, RENAME, TRUNCATE, COMMENT

DCL Data Control Language => GRANT, REVOKE

Transaction Control => COMMIT, ROLLBACK, SAVEPOINT

```
SELECT *                                => All Columns  
FROM departments;
```

```
SELECT department_id, location_id      => Specific Columns
FROM departments;
```

- SQL statements are not case sensitive
- Can have 1 or more lines
- Clauses are usually placed on different lines
- Indent for better readability
- Semicolons are optional on iSQL*Plus , but required at the end of multiple statements
- Semicolones are required on SQL*Plus

```
SELECT last_name | |job_id AS "Employees"
FROM employees
```

```
Links with literal strings
SELECT last_name | ' is a ' | job_id AS "Employees Details"
FROM employees
```

q Operator (just like the example above)

```
SELECT last_name ||  
       q' [, it's assigned Manager ID:]'  
       || manager_id  
       AS 'Department and Manager'  
FROM departments
```

$$+ \quad - \quad * \quad /$$

Using () parenthesis for complicated Equations if fine

Example:

```
SELECT salary + 300
FROM employees;
```

NULL is not the same as zero or blank space

ALIAS (use alias for better readability)

```
SELECT last_name AS Name, pct AS Percentage
```

SELECT last_name AS Last Name (this is an error
there is an space in the alias name)

```
SELECT last_name AS name => NAME
SELECT last_name AS "Name" => Name
```

DISTINCT will display repeated rows only 1 time, removing Duplicated rows

```
SELECT DISTINCT department_id
FROM employees;
```

DESCRIBE tablename

*will display table structure completely

Lesson 2 WHERE

Monday, February 13, 2017 10:37 PM

SELECT *

FROM table

WHERE condition(s)

....

WHERE last_name= "Whelan"

WHERE department_id = 90

The default Date format is DD-MON-RR

WHERE hire_date = '17-FEB-96'

Comparison and Condition Operators

= > >= < <= (this is easy)

<>	Not equal too
BETWEEN... AND ...	Inclusive
IN(set)	Match a list of values
LIKE	Match a character pattern
IS NULL	Is a null value

!= AND ^= are equivalent to NOT EQUAL too

WHERE manager_id IN (100,101,102,103)

WHERE salary BETWEEN 2500 AND 3500

Search conditions

% => denotes zero or many characters

_ => denotes one character

Example:

Getting all first name starting with S

WHERE first_name LIKE 'S%'

One character , "o" after and zero or many characters after

WHERE last_name LIKE '_o%'

ORDER BY with ALIAS

SELECT last_name (1), job_id (2), hire_date (3)

FROM employees

ORDER BY 2;

Using variables

SELECT employee_id

FROM employees

WHERE employee_id = &employee_num

To escape identifier

ESCAPE

WHERE job_id LIKE '%_ %'

In order to escape _

We do

WHERE job_id LIKE '%_%' ESCAPE '\'

Using NULL condition

WHERE manager_id IS NULL

Logical Operators

AND OR NOT

WHERE salary >= 10000

AND job_id LIKE '%MAN%'

WHERE salary >= 10000

OR job_id LIKE '%MAN%'

WHERE job_id

NOT IN('IT_PROG','ST_CLERK')

In order to override operators use ()

WHERE (job_id = 'SA_REP'

OR job_id = 'AS_PRES')

AND salary > 15000;

Order By is the last statement always

ORDER BY hire_date

ORDER BY hire_Date DESC

DEFAULT is ASC

&& => Use double ampersand to maintain
Column names

SELECT &&column_name

--

```
FROM employees  
WHERE employee_id = &employee_num
```

**this will prompt for an employee number

To use a character string use quotes

```
WHERE job_id = '&job_title'
```

It is ok to use it on Heading too
SELECT employee_id, &column_name

```
SELECT &&column_name
```

To remove it

```
UNDEFINE column_name
```

Using DEFINE

```
DEFINE employee_num = 200
```

```
SELECT .....
```

```
FROM .....
```

```
WHERE employee_id = &employee_num
```

```
UNDEFINE employee_num
```

Lesson 3 Using single Row Functions

Monday, February 13, 2017 11:06 PM

There are two types of functions

Single-row functions	one result per row
Multiple-row functions	one result per group of rows

Functions

LOWER	Lowercase
UPPER	Uppercase
INITCAP	Capitalize
SUBSTR	Cut string

Example:

SUBSTR('Hello World', 1, 5) => Hello

CONCAT	Join together
--------	---------------

Example:

CONCAT('Hello','World') => HelloWorld

LENGTH	Length of the string
INSTR	Character positions

Example:

INSTR('Hello', 'e') => 2

LPAD	Left Padding
RPAD	Right Padding

Example:

LPAD(salary,10,'*') => *****24000

REPLACE	Replace values
---------	----------------

REPLACE('JACK','J','R') => RACK

TRIM	Remove whitespace or characters
------	---------------------------------

TRIM ('H' FROM 'HelloWorld') => elloWorld

ROUND(SYSDATE, 'MONTH')

Will round a July 25 to August 1

ROUND('SYSDATE','YEAR')

TRUNC('SYSDATE','MONTH')

TRUNC('SYSDATE','YEAR')

Conversion for numbers, chars and dates

TO_CHAR(number, 'format_model')

Example:

WHERE SUBSTR(last_name, -1, 1) = 'n'

-1 means 1 from the end, 1 means 1 space

Checking last character in other words

Number Functions

ROUND	Round to specified decimal
TRUNC	Truncate to specified decimal
MOD	Remainder of division (like % on C)

ROUND(45.926, 2) => 45.93

TRUNC(45.926, 2) => 45.92

MOD(1600, 300) => 100

Example

SELECT salary, round(salary, -3)

3100 => 3000

Get Date and Time from System

SELECT SYSDATE

FROM DUAL

Examples

(SYSDATE-hire_date)/7 AS WEEKS

MONTHS_BETWEEN	Number of months between two dates
ADD_MONTHS	Add calendar months to date
NEXT_DAY	Next day of the date specified
LAST_DAY	Last day of the month
ROUND	Round date
TRUNC	Truncate date

Conversion for Dates

TO_CHAR(hire_date, 'YYYY-Month-DD')

Elements

YYYY -> year in numbers

YEAR -> year in english

MM -> two digit month

MONTH -> month in english

MON -> three letter abbreviation

DY -> three letter abbreviation

DAY -> day in english

Conversion for numbers, chars and dates

```
TO_CHAR(number, 'format_model')
TO_NUMBER(char, 'format_model')
TO_DATE(char, 'format_model')
```

```
TO_CHAR(salary, '$99,999.00') as SALARY
Add the $ sign
```

Handling NULLs

```
NVL (expr1, expr2)
=> Converts a null value to an actual value
```

```
SELECT last_name, salary, salary* nvl(commission_pct,0)
NVL ( city , 'Unavailable' )
```

```
MONTH <- month in english
MON -> three letter abbreviation
DY -> three letter abbreviation
DAY -> day in english
DD -> numeric day
```

Nesting Functions

```
F3(F2(F1(col,arg1),arg2),arg3)
```

Using CASE

```
SELECT last_name, job_id, salary
      CASE job_id WHEN 'IT_PROG' THEN 1.10*salary
                  WHEN 'ST_CLERK' THEN 1.15*salary
                  ELSE salary END      "REVISED_SALARY"
FROM EMPLOYEES;
```

Lesson 4 Group Functions

Monday, February 13, 2017 11:58 PM

GROUP FUNCTIONS

AVG	Average Value (ignores NULL)
COUNT	Count rows
MAX	Maximum value
MIN	Minimum value
STDDEV	Standard deviation of n
SUM	Sum values
VARIANCE	Variance of n, ignoring nulls

DISTINCT => Makes the function consider only non-duplicates
Use **NVL** to substitute NULL for 0 for example

Examples:

```
SELECT COUNT (DISTINCT department_id)
FROM employees;
```

```
SELECT AVG (NVL (commission_pct , 0))
FROM employees;
```

GROUP BY

Example

```
SELECT DEPARTMENT_ID, AVG(SALARY)
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID;
```

****Usually we need an ORDER BY with GROUP BY**

```
SELECT      DEPARTMENT_ID, AVG(SALARY)
FROM        EMPLOYEES
GROUP BY    DEPARTMENT_ID
ORDER BY    DEPARTMENT_ID;
```

HAVING

Example:

```
SELECT department_id, MAX(salary)
FROM employees
GROUP BY department_id
HAVING MAX(salary)>10000;
```

Lesson 5 Join Tables

Monday, February 13, 2017 11:58 PM

CUSTOMERS

PID	PNAME	PEMAIL
1	John Smith	John.Smith@yahoo.com
2	Steven Goldfish	goldfish@fish.net
3	Paula Brown	pb@domain.org
4	James Smith	jim@sup.co.uk
5	Uncle Joe	UNK@sympatico.ca

ORDERS

OID	ODATE	AMOUNT	PID
2	06-MAY-10	100.22	2
1	07-MAY-10	99.95	1
3	07-MAY-10	122.95	3
3	13-MAY-10	100	3
4	22-MAY-10	555.55	4
5	22-MAY-10	999.99	9

SAMPLE JOIN

```
SELECT DEPARTMENT_ID, DEPARTMENT_NAME, D.LOCATION_ID, CITY
FROM DEPARTMENTS D,
     LOCATIONS L
WHERE D.LOCATION_ID = L.LOCATION_ID
```

JOINING COLUMN NAMES -- USING

```
SELECT EMPLOYEES.EMPLOYEE_ID,
       EMPLOYEES.LAST_NAME,
       DEPARTMENTS.LOCATION_ID,
       DEPARTMENT_ID
FROM EMPLOYEES JOIN DEPARTMENTS
USING (DEPARTMENT_ID);
```

INNER JOIN (Default Join also)

The INNER JOIN will select all rows from both tables → as long as there is a match between the columns we are matching on.

```
SELECT employee_id, last_name, department_name
FROM employees INNER JOIN departments
ON employees.Department_ID = departments.Department_ID;
```

Another way

```
SELECT employee_id, last_name, department_name
FROM employees, departments
WHERE employees.Department_ID = departments.Department_ID;
```

How much did a customer purchase?

```
SELECT      pname,
            Amount AS "Sales Per Customer"
FROM        Customers, Orders
WHERE       Customers.pid = orders.pid;
```

NATURAL JOIN

```
SELECT DEPARTMENT_ID, DEPARTMENT_NAME, LOCATION_ID,
       CITY
FROM DEPARTMENTS
NATURAL JOIN LOCATIONS;
```

Natural join, naturally check headers names without the need of letters to implicitly declare tables

Creating Joins with the USING Clause

```
SELECT l.city, D.department_name
FROM locations L JOIN departments D USING (location_id)
WHERE location_id = 1400;
```

Creating JOINS with the ON clause

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM employees e JOIN departments d
ON (e.department_id = d.department_id);
```

Three way joins

```
SELECT employee_id, city, department_name
FROM employees e
JOIN departments d
ON d.department_id = e.department_id
JOIN locations l
ON d.location_id = l.location_id;
```

3 Types of OUTER JOINS

LEFT => Includes left side even if they don't match
RIGHT => Includes right
FULL => Includes both (everything)

```
SELECT pname,
       SUM(Amount) AS SalesPerCustomer
FROM Customers LEFT JOIN Orders
ON Customers.pid = orders.pid
GROUP BY PNAME
```

Lesson 6 Using SubQueries

Monday, February 13, 2017 11:58 PM

Subquery Syntax

```
SELECT select_list
FROM table
WHERE expr operator
      (SELECT select_list
       FROM table);
```

A Subquery is a SELECT statement that is imbedded in a clause of another SELECT statement.

Useful when you need to select rows from a table with a condition that depend so on data from the same table or other tables.

Example:

```
SELECT last_name
FROM employees
WHERE salary > ( SELECT salary
                 FROM employees
                 WHERE last_name = "Abel")
```

Single-Row Subqueries

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id =
      (SELECT job_id
       FROM employees
       WHERE employee_id = 141)
AND salary >
      (SELECT salary
       FROM employees
       WHERE employee_id = 143)
```

Multiple-Row Subqueries

```
SELECT department_id, employee_id, last_name, salary
FROM employees
WHERE salary IN
      (SELECT min (salary)
       FROM employees
       GROUP BY department_id)
```

Using the ANY Operator in Multiple Row Subqueries

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary < ANY
```

Guidelines for using Subqueries:

→ A Subquery must be enclosed in parenthesis.

→ Place the Subquery on the right side of the comparison operator for readability

You can do it the other way

```
SELECT * from employees
```

```
WHERE (select salary from employees where last_name = 'Abel') < salary
```

→ ORDER BY clause in the Subquery is only needed when performing TOP-N analysis

- Normally the order by clause is only found at the end of the SQL statement.

- TOP-N analysis refers two finding the top number of rows.

- Example top seven salaries

→ 2 types of Subqueries are used:

- Single-row operators

- Multiple-row operators

Group Functions in a Subquery

```
SELECT LAST_NAME, JOB_ID, SALARY
FROM EMPLOYEES
WHERE SALARY = ( SELECT MIN (SALARY)
                 FROM EMPLOYEES);
```

**** Oracle executes subqueries first**

**** Oracle return results into the HAVING Clause of the main query**

```
SELECT job_id, AVG (salary)
FROM employees
GROUP BY job_id
HAVING AVG (salary) = (SELECT MIN ( AVG
(salary) )
                       FROM employees
                       GROUP BY job_id );
```

Using the ALL Operator in Multiple Row Subqueries

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary < ALL
      (SELECT salary
       FROM employees
       WHERE job_id = 'IT_PROG')
AND job_id != 'IT_PROG'
```



```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary < ANY
      (SELECT salary
       FROM employees
       WHERE job_id = 'IT_PROG')
AND job_id != 'IT_PROG'
```

NOTE:

< ANY -- less than any will mean less than the maximum return
> ANY -- greater than any means more than the minimum
value returned
= ANY -- equal to any is the equivalent of the IN operator

```
WHERE job_id = 'IT_PROG')
AND job_id != 'IT_PROG'
```

NOTE:

> ALL -- greater than all means more than the maximum
< ALL -- less than all means less than the minimum

The NOT operator can be used with any of these. Caution is recommended the use of the not operator just as it was in other programming languages.

Lesson 7 Data Modelling

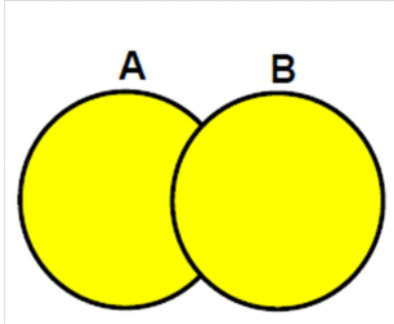
Tuesday, February 14, 2017 1:18 AM

Self-Study ERD Diagram

Lesson 8 Set Operators

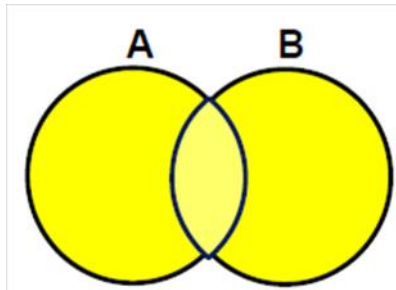
Tuesday, February 14, 2017 1:19 AM

UNION



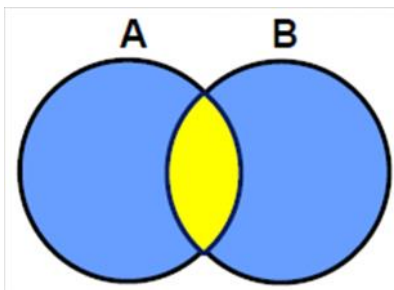
UNION of all the rows in A
With ALL the rows in B
With NO DUPLICATES

UNION ALL



UNION of ALL the rows in A and B
including duplicates

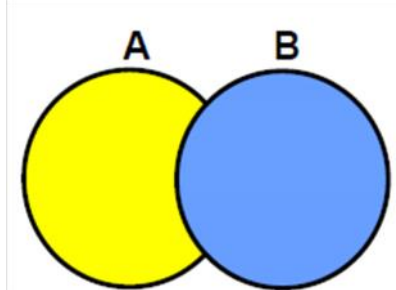
INTERSECT



The rows in common to both tables only

A intersect B
same as
B intersect A

MINUS



Rows in the first query A
That are not in second query B

PRECEDENCE – equal – evaluated left to right

Caution recommended. Use brackets with
INTERSECT

```
SELECT employee_id, job_id, salary
FROM employees
UNION
SELECT employee_id, job_id, 0
FROM job_history;
```

→ Matching columns
If no salary will show 0

```
SELECT department_id, TO_NUMBER (null) as
location, hire_date
FROM employees
UNION
SELECT department_id, location_id, TO_DATE
(null)
FROM departments;
```

→ Note the location because
TO_NUMBER (null) does not
make a good column heading

```
SELECT employee_id, job_id, salary
FROM employees
UNION
SELECT employee_id, job_id, 0
FROM job_history
ORDER BY 2; ⬅ to change default of sorting
on employee-id
```

The ORDER BY clause can appear only once at the end of the compound query. Same as before – at the end

Lesson 9A DML-Insert

Tuesday, February 14, 2017 1:20 AM

DML – Data Manipulation Language

What is it?

The SQL that manipulates data.
Data can be added, changed or deleted

A DML statement is executed when you:

- Add new rows to a table
- Modify existing rows in a table
- Remove existing rows from a table

- A transaction consists of a collection of DML statements that form a logical unit of work – such as inserting students registering

INSERT – by COPYING FROM ANOTHER TABLE

Use a subquery

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
SELECT employee_id, last_name, salary, commission_pct
FROM employees
WHERE job_id LIKE '%REP%';
```

- > No VALUES clause
- > Number of columns must match
- > Data type must match

TRUNCATE statement

Removes ALL rows from a table, but leaves the table structure

```
TRUNCATE employees;
```

WHY USE?

More efficient than DELETE

Delete checks all delete triggers

Truncate is a DDL statement and does not create a copy to allow for ROLLBACK

If the table is the parent you need to drop constraint of the FK to be able to do this

Example

```
INSERT INTO table [(column [, column...])]
VALUES (value [, value...]);
```

```
INSERT INTO departments
VALUES (100, 'Finance', NULL, NULL);
```

SYSDATE => Server current date and time

Update employee 113

- A) JOB_ID SAME AS EMPLOYEE 205
- B) SALARY SAME AS 205

Method 1:

```
UPDATE employees
SET job_id = (SELECT job_id
FROM employees
WHERE employee_id = 205),
salary = (SELECT salary
FROM employees
WHERE employee_id = 205)
WHERE employee_id = 113;
```

Method 2:

```
UPDATE employees
SET (job_id, salary) = (SELECT job_id, salary
FROM employees
WHERE employee_id = 205)
WHERE employee_id = 113;
```

COMMIT and ROLLBACK Statements

With COMMIT and ROLLBACK statements, you can:

- Ensure data consistency
- Preview data changes before making changes permanent
- Group logically-related operations

UPDATE...

SAVEPOINT update_done; <- receive a message

SAVEPOINT update_done succeeded

INSERT...

ROLLBACK TO update_done; <- receive a message ROLLBACK succeeded

State of data after a ROLLBACK

Example

1. Remove departments 290 and 300 in the DEPARTMENTS table
2. Update a row in the EMPLOYEES table.
3. Save the data change.

```
1
DELETE FROM departments
WHERE department_id IN (290, 300);
```

```
2
UPDATE employees
SET department_id = 80
WHERE employee_id = 206;
```

```
3
COMMIT;
```

State of data after a ROLLBACK

Discard all pending changes by using the ROLLBACK statement:

- Data changes are undone.
- Previous state of the data is restored.
- Locks on the affected rows are released.

```
DELETE FROM employees;
ROLLBACK;
```

Lesson 10 Create & Manage Tables

Tuesday, February 14, 2017 1:18 AM

```
CREATE TABLE dept
(
    deptno    NUMBER(2),
    dname     VARCHAR2(14),
    loc       VARCHAR2(13),
    create_date DATE);
```

Data types available

VARCHAR2 (size) Maximum size need to be specified (up to 4000)
CHAR Fixed Length size to maximum 2000
NUMBER (p, s) P is precision or total number of decimal digit and S is scale or number of digits to the right of the decimal point
EX: **NUMBER (5, 2)** means 5 all together and 2 decimal places
The value 1000 will be rejected by the server as that is 6 wide

DATE Date and Time value to the nearest second
Range: Jan 1, 4712 BC and Dec 31, 9999

=====

LONG data type is variable length up to @GB

CLOB character data up to 4GB

UNIQUE CONSTRAINT

Example: At the TABLE LEVEL
CREATE TABLE employees(
employee_id NUMBER(6),
last_name VARCHAR2(25) NOT NULL,
email VARCHAR2(25),
salary NUMBER(8,2),
commission_pct NUMBER(2,2),
hire_date DATE NOT NULL,
...
CONSTRAINT emp_email_uk UNIQUE(email));

CHECK CONSTRAINT

Defines a condition that each row must satisfy in order to be added to the table

EXAMPLE:

```
CREATE TABLE EMPLOEES
( .... other columns

salary NUMBER(2)
CONSTRAINT employees_salary_min CHECK
(salary > 0),
...
)
```

DATETIME data types

TIMESTAMP - By default this is in microsecond
- EX: 12-MAR-15 08:45.23.123456 ⬅ 6 decimals for microseconds

TIMESTAMP (0) - removes part seconds
TIMESTAMP (9) - can go to nanoseconds

• Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints

NOT NULL	- SPECIFY DATA CANNOT BE null
UNIQUE	- PREVENTS DUPLICATION OF DATA IN THAT ROW
PRIMARY KEY	Unique identifier for each row in a table Aside: It is both NOT NULL and UNIQUE
FOREIGN KEY	Establishes and enforces a referential integrity between the column and a column of the referenced table such that values in one table match values in another table
CHECK	Specifies a TRUE condition

Foreign Key – table level

```
CREATE TABLE employees(
employee_id NUMBER(6),
last_name VARCHAR2(25) NOT NULL,
email VARCHAR2(25),
salary NUMBER(8,2),
commission_pct NUMBER(2,2),
hire_date DATE NOT NULL,
...
department_id NUMBER(4),
CONSTRAINT emp_dept_fk FOREIGN KEY
(department_id)
REFERENCES departments(department_id),
CONSTRAINT emp_email_uk UNIQUE(email));
```

Foreign Key – column level

```
CREATE TABLE employees
(
...
department_id NUMBER(4) CONSTRAINT
emp_deptid_fk
REFERENCES departments(department_id),
...
)
NOTE:
Foreign Key not stated as the reference handles that concept
Still need to name it and specify what it references
```

Lesson 11 Alter

Tuesday, February 14, 2017 1:19 AM

SYNTAX

ALTER

TABLE - name of the table

ADD – MODIFY – DROP is the type of modification

COLUMN -- name of column effected

DATATYPE -- datatype and length of the column

DEFAULT expr – specifies the default value for a column

INITIALLY DEFERRED	Waits to check the constraint until the transaction ends
INITIALLY IMMEDIATE	Checks the constraint at the end of the statement execution

PUBLIC SYNONYM – created by DBA

CREATE PUBLIC SYNONYM STUDLIST

FOR registration.STUDENT;

Allows access to table STUDENT owned by user REGISTRATION.

SYNONYM

PURPOSE

- 1 To shorten lengthy object names
- 2 Refer to table owned by another user – really the same as 1

CREATING SYNONYM

```
CREATE SYNONYM d_sum  
FOR dept_sum_vu;
```

REMOVING SYNONYM

```
DROP SYNONYM d_sum;
```


Lesson 12 Views

Tuesday, February 14, 2017 1:18 AM

Generic Syntax

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
[(alias [, alias]...)]
AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

VIEW – Examples

```
CREATE VIEW empvu80
AS SELECT employee_id,
last_name,
salary
FROM employees
WHERE department_id = 80;
```

DESCRIBE empvu80

VIEW – Examples – with aliases

Column aliases

```
CREATE VIEW salvu50
AS SELECT employee_id ID_NUMBER,
last_name NAME,
salary*12 ANN_SALARY
FROM employees
WHERE department_id = 50;
```

REMOVING A VIEW

Removing a view does not remove the data

```
DROP VIEW empvu80;
```

CREATE SEQUENCE Statement

Generic syntax

```
CREATE SEQUENCE sequence          <- name of sequence
[INCREMENT BY n]                  <- specifies increment value
[START WITH n]                    <- Starting (default 1 if omitted)
[{MAXVALUE n | NOMAXVALUE}]       <- maximum value – default is nomax
[{MINVALUE n | NOMINVALUE}]       <- this is default if not stated
[{CYCLE | NOCYCLE}]               <- allows recycling of numbers–reuse
[{CACHE n | NOCACHE}];           <- allows caching x values-faster
```

View – Retrieving Data

```
SELECT *
FROM salvu50;
```

MODIFY – CHANGE a VIEW



Requires

CREATE OR REPLACE ← it saves deleting and creating – and re granting privileges

Example: Modify the previous empvu80 to add aliases

```
CREATE OR REPLACE VIEW empvu80
(id_number, name, sal, department_id)
AS SELECT employee_id,
first_name || ' ' || last_name,
salary,
department_id
FROM employees
WHERE department_id = 80;
```

Rules for Performing DML Operations on a View

- You can usually perform DML operations on simple views. 
- You cannot remove a row if the view contains the following: 

- Group functions
- A GROUP BY clause
- The DISTINCT keyword
- The pseudocolumn ROWNUM keyword

Example:

```
CREATE SEQUENCE dept_deptid_seq ←
note the naming convention
INCREMENT BY 10
START WITH 120
MAXVALUE 9999
NOCACHE
NOCYCLE;
```

NEXTVAL and CURRVAL Pseudo columns

- NEXTVAL
 - used to extract successive sequence number
 - returns the next available sequence value.It returns a unique value every time it is referenced, even for different users.

Specify NEXTVAL and the sequence name

- CURRVAL obtains the current sequence value.
- NEXTVAL must be issued for that sequence before CURRVAL contains a value.

Create and DROP index

How Are Indexes Created?

Automatically:

A unique index is created automatically when you define a

- PRIMARY KEY or
- UNIQUE constraint in a table definition.

Manually:

Developers can create nonunique indexes on other columns to speed up access to rows.

```
CREATE INDEX emp_last_name_idx <- note naming convention
ON    employees (last_name);
```

INSERT INTO departments

(department_id, department_name, location_id)

```
VALUES (dept_deptid_seq.NEXTVAL, 'Support',
2500);
```

You can also ALTER or DROP Sequences

```
DROP SEQUENCE dept_deptid_seq;
```

SOME RULES ON INDEXES

- 1 The column is used often in a where clause and the table is large.
- 2 The table is very large and most retrievals display a small amount of data.

REMOVE INDEX

```
DROP INDEX emp_last_name_idx;
```

Lesson 13 Creating Indexes and Sequences + Practice

Tuesday, February 14, 2017 1:18 AM

CLASS EXERCISE, CHAPTER 10 and 11 -- CREATING INDEXES and SEQUENCES plus DATA DICTIONARY VIEWS

=====

REMOVING and RESTORING TABLES

*Firstly, we will create two tables to play with *

```
SQL> CREATE TABLE STAFF AS
  SELECT employee_id, last_name, hire_date, job_id,
         salary, department_id
  FROM   employees;
```

Table created.

```
SQL> CREATE TABLE MINISTAFF AS
  SELECT employee_id, last_name, hire_date, job_id, salary
  FROM   employees
  WHERE  department_id IN (10,20,60,80);
```

Table created.

```
SQL> SELECT * FROM ministaff;
```

EMPLOYEE_ID	LAST_NAME	HIRE_DAT	JOB_ID	SALARY
200	Whalen	87-09-17	AD_ASST	4400
201	Hartstein	96-02-17	MK_MAN	13000
202	Fay	97-08-17	MK_REP	6000
103	Hunold	90-01-03	IT_PROG	9000
104	Ernst	91-05-21	IT_PROG	6000
107	Lorentz	99-02-07	IT_PROG	4200
149	Zlotkey	00-01-29	SA_MAN	10500
174	Abel	96-05-11	SA_REP	11000
176	Taylor	98-03-24	SA_REP	8600

9 rows selected.

```
SQL> DROP TABLE STAFF;
```

Table dropped. → this was temporary removal to recyclebin

```
SQL> SELECT original_name, droptime
FROM   recyclebin ;
```

ORIGINAL_NAME	DROPTIME
STAFF	2006-12-03:11:13:47

```
SQL> DESC staff
```

ERROR:
ORA-04043: object staff does not exist

```
SQL> FLASHBACK TABLE staff TO BEFORE DROP;
```

SEQUENCES

```
SQL> CREATE SEQUENCE staff_empid_seq
  START WITH 111
  MAXVALUE 200
  NOCACHE ; → Default value for CACHE is 20
values
```

Sequence created.

```
SQL> INSERT INTO staff VALUES
(staff_empid_seq.NEXTVAL,'Moore',sysdate,'IT_PROG',8000,60);
```

1 row created.

→ We used AUTO option for generation of UNIQUE integer values with SEQUENCE.NEXTVAL here

```
SQL> SELECT * FROM staff
  WHERE hire_date = sysdate;
```

No rows selected. -- be careful when equalling dates

```
SQL> SELECT * FROM staff
  WHERE to_date(hire_date,'RR-MM-DD') =
        to_date(sysdate, 'RR-MM-DD');
```

EMPLOYEE_ID	LAST_NAME	HIRE_DAT	JOB_ID	SALARY	DEPARTMENT_ID
111	Moore	06-12-03	IT_PROG	8000	60

```
SQL> SELECT sequence_name, last_number
FROM   user_sequences ;
```

SEQUENCE_NAME	LAST_NUMBER
DEPARTMENTS_SEQ	280
EMPLOYEES_SEQ	207
LOCATIONS_SEQ	3300
STAFF_EMPID_SEQ	112

→ Column Last_Number means actually NEXT available number (if NOCACHE option is used)

ORA-04043: object staff does not exist

SQL> FLASHBACK TABLE staff TO BEFORE DROP;

Flashback complete. → this was restore from recyclebin

SQL> DESC staff

Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
DEPARTMENT_ID		NUMBER(4)

SQL> DROP TABLE ministaff PURGE;

Table dropped. → this was permanent removal (no recyclebin)

SQL> SELECT original_name, droptime
FROM recyclebin ;

no rows selected

SQL> FLASHBACK TABLE ministaff TO BEFORE DROP;

FLASHBACK TABLE ministaff TO BEFORE DROP
*

ERROR at line 1:

ORA-38305: object not in RECYCLE BIN

→ we could not restore this table, it was not in the recycle bin after PURGE option

INDEXES

SQL> CREATE INDEX staff_salary_idx ON staff(salary);

Index created. → We created a SINGLE index

SQL> CREATE INDEX staff_lname_idx ON staff(last_name);

Index created.

SQL> DROP INDEX staff_lname_idx;

Index dropped.

SQL> CREATE INDEX staff_lname_salary_idx
ON staff(last_name, salary);

Index created.

→ In order to modify an Index we need to drop it and re-create it again. Here we created a COMPOSITE Index that will serve a dual purpose: for two columns and for the first mentioned one (that is why we do NOT

→ Column Last_Number means actually NEXT available number (if NOCACHE option is used)

SQL> ALTER SEQUENCE staff_empid_seq
MAXVALUE 140
CACHE 10;

Sequence altered.

SQL> SELECT sequence_name, last_number,
cache_size
FROM user_sequences
WHERE sequence_name LIKE 'STAFF%';

SEQUENCE_NAME	LAST_NUMBER	CACHE_SIZE
---------------	-------------	------------

STAFF_EMPID_SEQ	122	10
-----------------	-----	----

→ Column Last_Number means actually FIRST number from the NEXT set of cached values (if CACHE option is used)

SQL> INSERT INTO staff VALUES
(staff_empid_seq.NEXTVAL,'Dunn',sysdate,'IT_PROG',7000,60);

1 row created.

SQL> ROLLBACK;

Rollback complete.

SQL> INSERT INTO staff VALUES
(staff_empid_seq.NEXTVAL,'Markov',sysdate,'IT_PROG',11000,60);

1 row created.

SQL> SELECT * FROM staff
WHERE to_date(hire_date,'RR-MM-DD') =
to_date(sysdate, 'RR-MM-DD');

EMPLOYEE_ID	LAST_NAME	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
111	Moore	06-12-03	IT_PROG	8000 60	
113	Markov	06-12-03	IT_PROG	11000 60	

→ So, if we perform any rollback, then we create gaps in the sequence values (employee Dunn got number 112 and that number was lost after rollback)

need an index just for the last name anymore, it is given with this composite one)

```
SQL> SELECT index_name, uniqueness FROM user_indexes  
WHERE table_name = 'STAFF' ;
```

INDEX_NAME	UNIQUENESS
STAFF_SALARY_IDX	NONUNIQUE
STAFF_LNAME_SALARY_IDX	NONUNIQUE

```
SQL> SELECT index_name, column_name, column_position  
FROM user_ind_columns  
WHERE table_name = 'STAFF' ;
```

→ In order to see column name that is indexed and their relative position (if index is a composite one) use user_ind_columns view

INDEX_NAME	COLUMN_NAME	COLUMN_POSITION
STAFF_SALARY_IDX	SALARY	1
STAFF_LNAME_SALARY_IDX	SALARY	2
STAFF_LNAME_SALARY_IDX	LAST_NAME	1

