Chapter 2

Lecture (Lec 1 & 2) for Week 1

2.1 Basic Constructs of RDBMS

- **Schema** A schema is a collection of database objects. A schema is owned by a database user and has the same name as that user. Schema objects are the logical structures that directly refer to the database's data. Schema objects include structures like *tables, views, and indexes.*
- **Table** Tables are the basic unit of data storage in an Oracle database. Database tables hold all user-accessible data. Each table has *columns and rows.* A table that has an employee database, for example, can have a column called employee number, and each row in that column is an employee's number.
- **Indexes** Indexes are optional structures associated with tables. Indexes can be created to increase the performance of data retrieval.

  Indexes are created on one or more columns of a table. After it is created, an index is automatically maintained and used by Oracle. Changes to table data (such as adding new rows, updating rows, or deleting rows) are automatically incorporated into all relevant indexes with complete transparency to the users.
- **Views** Views are customized presentations of data in one or more tables or other views. A view can also be considered a stored query. Views do not actually contain data. Rather, they derive their data from the tables on which they are based, referred to as the base tables of the views.

  Views provide an additional level of table security by restricting access to a predetermined set of rows and columns of a table. They also hide data complexity and store complex queries.
- **Clusters** Clusters are groups of one or more tables physically stored together because they share common columns and are often used together. Because related rows are physically stored together, disk access time improves.

  Like indexes, clusters do not affect application design. Whether a table is part of a cluster is transparent to users and to applications. Data stored in a clustered table is accessed by SQL in the same way as data stored in a nonclustered table.

- **Synonyms** A synonym is an alias for any table, view, materialized view, sequence, procedure, function, package, type, Java class schema object, user-defined object type, or another synonym. Because a synonym is simply an alias, it requires no storage other than its definition in the data dictionary.

- **Data Dictionary** One of the most important parts of an Oracle database is its data dictionary, which is a read-only set of tables that provides information about the database. A data dictionary contains:

  – The definitions of all schema objects in the database (tables, views, indexes, clusters, synonyms, sequences, procedures, functions, packages, triggers, and so on)

  – How much space has been allocated for, and is currently used by, the schema objects

  – Integrity constraint information

  – Privileges and roles each user has been granted

  – Auditing information, such as who has accessed or updated various schema objects

- **DDL and DML** SQL statements are divided into two major categories: data definition language (DDL) and data manipulation language (DML).

  DDL: DDL statements are used to build and modify the structure of your tables and other objects in the database.

  DDL Example: CREATE TABLE , ALTER TABLE.

  DML:DML statements are used to work with the data in tables. When you are connected to most multi-user databases (whether in a client program or by a connection from a Web page script), you are in effect working with a private copy of your tables that cant be seen by anyone else until you are finished (or tell the system that you are finished).

  DML Example: The insert statement. (INSERT INTO ¡table name¿ VALUES (¡value 1¿, ... ¡value n¿);)

  The update statement. UPDATE ¡table name¿ SET ¡attribute¿ = ¡expression¿ WHERE ¡condition¿;

- **Cardinality** In data modelling terms, cardinality is how one table relates to another.

  – 1-1 (one row in table A relates to one row in tableB)

  – 1-Many (one row in table A relates to many rows in tableB)

  – Many-Many (Many rows in table A relate to many rows in tableB)

2.1.1 Tablespace, Datafiles and Objects

(source: https://docs.oracle.com/cd/A57673_01/DOC/server/doc/SCN73/ch4.htm)

The following block-diagram shows the relationship.

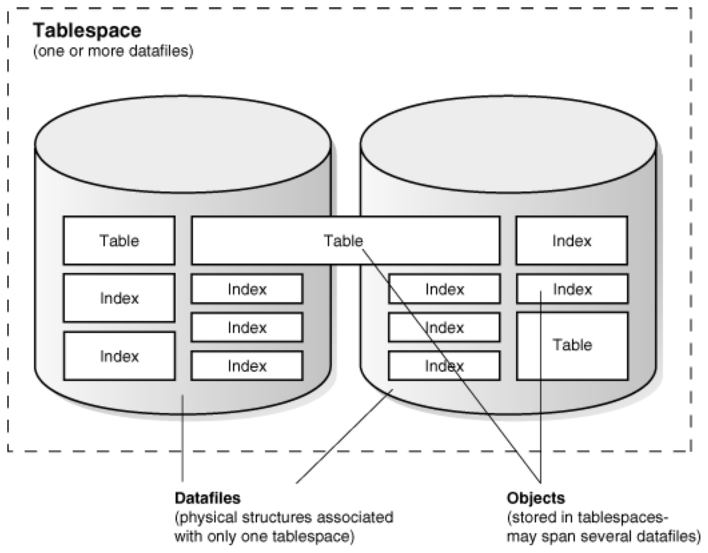

Figure 2.1: Tablespace, Datafiles and Objects

Although databases, tablespaces, datafiles, and segments are closely related, they have important differences:

- **Databases and tablespaces.** An Oracle database is comprised of one or more logical storage units called tablespaces. The database also has a lot more (background process). The database's data is collectively stored in the database's tablespaces.
- **Tablespaces and datafiles.** Each tablespace in an Oracle database is comprised of one or more operating system files called datafiles. A tablespace's datafiles physically store the associated database data on disk.
- **Databases and datafiles.**A database's data is collectively stored in the datafiles that constitute each tablespace of the database. For example, the simplest Oracle database would have one tablespace and one datafile. A more complicated database might have three tablespaces, each comprised of two datafiles (for a total of six datafiles).
- **Tablespace.** Tablespaces are the bridge between certain physical and logical components of the Oracle database. Tablespaces are where you store Oracle database objects such as tables, indexes and rollback segments.

  [A Rollback Segment is a database object containing before-images of data written to the database. Rollback segments are used to: i) Undo changes when a transaction is rolled back ii) Recover the database to a consistent state in case of failures]

  A database is divided into one or more logical storage units called tablespaces. A database administrator can use tablespaces to do the following:

– control disk space allocation for database data

– assign specific space quotas for database users

– control availability of data by taking individual tablespaces online or offline

2.1.1.1 Basic Operations on Tablespace

Basics of Space Management: Introduction to Data Blocks, Extents, and Segments.

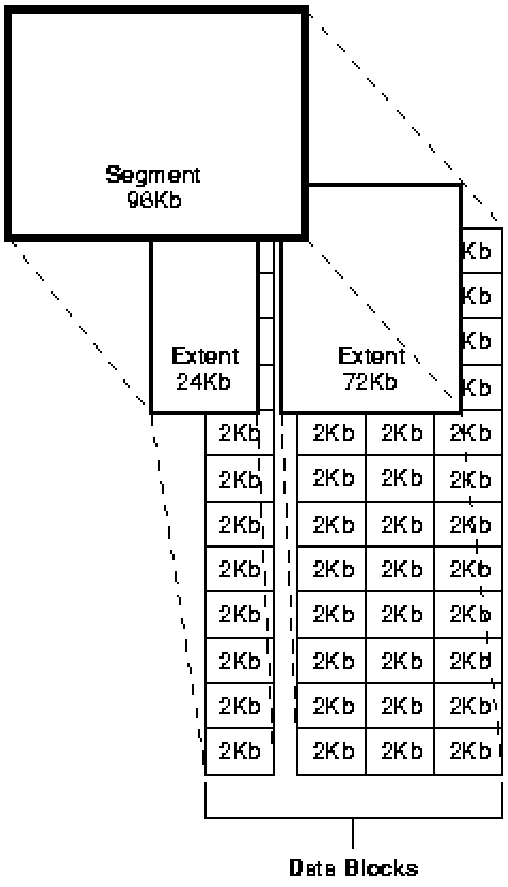

Figure 2.2: Oracle Storage Management

- **Data blocks.**At the finest level of granularity, Oracle stores data in data blocks (also called logical blocks, Oracle blocks, or pages). One data block corresponds to a specific number of bytes of physical database space on disk.

- **Extent.** The next level of logical database space is an extent. An extent is a specific number of contiguous data blocks allocated for storing a specific type of information.

- **Segment.** The level of logical database storage above an extent is called a segment. A segment is a set of extents, each of which has been allocated for a specific data structure and all of which are stored in the same tablespace. For example, each table's data is stored in its own data segment, while each index's data is stored in its own index segment. If the table or index is partitioned, each partition is stored in its own segment.

*Oracle allocates space for segments in units of one **extent**. When the existing extents of a segment are full, Oracle allocates another extent for that segment.*

Step 1: Create a Tablespace first.

```
CREATE TABLESPACE mytspace
DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
EXTENT MANAGEMENT LOCAL AUTOALLOCATE;
```

AUTOALLOCATE causes the tablespace to be system managed with a minimum extent size of 64K.

Step 2 (a): Create an user and assign that user to a specific tablespace.

```
 CREATE USER sidney
    IDENTIFIED BY test123
    DEFAULT TABLESPACE mytspace
```

Specify the default tablespace for objects that the user creates. If you omit this clause, then the user's objects are stored in the database default tablespace.

If no default tablespace has been specified for the database, then the user's objects are stored in the SYSTEM tablespace.

There are a number of other parameters which we are now ignoring as not part of our study objective.

Step 2 (b): Create a specific table and assign a tablespace with it (this will overrule previous).

```
 create table intel
(dt varchar2(20),
tm varchar2(28),
ep number,
moteid number,
temperature number(12,5),
humidity number(20,8),
light number(12,6),
voltage number(10,6)
) tablespace NEW_TBSPACE;
```

Step 3: How to get information about free available space for a tablespace. Use `DBA_FREE_SPACE` data-dictionary.

```
SELECT TABLESPACE_NAME,SUM(BYTES)/1024/1024/1024 "FREE SPACE(GB)"
FROM DBA_FREE_SPACE GROUP BY TABLESPACE_NAME;
```

2.1.1.2 More Operations on Tablespace

Will be covered at the Admin and Tuning Sections W13 to W15

- The SYSTEM Tablespace
- Allocating More Space for a Database
- Online and Offline Tablespaces
- Read-Only Tablespaces

Chapter 3

Lecture (Lec 3 & 4) for Week 2

3.1 Basic Parts of Speech of SQL

Before you start: SQL*Plus tells you how many rows it found in One table after a valid query on it. (Notice the 14 rows selected notation at the bottom of the display.) This is called *feedback*.

It can be controlled in the following way:

```
set feedback off

set feedback 25

show feedback
```

3.1.1 Select, from, where, and order by

Run an SQL query similar to the following:

```
select Feature, Section, Page
from NEWSPAPER
where Section = 'F'
order by Page desc, Feature;
```

Default is asc in order by clause.

Logical Operators

```
Equal, Greater Than, Less Than, Not Equal
Page= 6 Page is equal to 6.
Page> 6 Page is greater than 6.
Page>= 6 Page is greater than or equal to 6.
```

```
Page< 6 Page is less than 6.
Page<= 6 Page is less than or equal to 6.
Page!= 6 Page is not equal to 6.
Page^= 6 Page is not equal to 6.
Page<> 6 Page is not equal to 6.
```

LIKE

```
Feature LIKE 'Mo%' Feature begins with the letters Mo.

Feature LIKE '_ _ I%' Feature has an I in the third position.

Feature LIKE '%o%o%' Feature has two os in it.
```

LIKE performs pattern matching. An underline character (_) represents exactly one character. A percent sign (%) represents any number of characters, including zero characters.

NULL and NOT NULL

IS NULL essentially instructs Oracle to identify columns in which the *data is missing.*

Tests Against a List of Values.

For Numbers

```
Page IN (1,2,3) => Page is in the list (1,2,3)

Page NOT IN (1,2,3) => Page is not in the list (1,2,3)

Page BETWEEN 6 AND 10 => Page is equal to 6, 10, or anything in between

Page NOT BETWEEN 6 AND 10 => Page is below 6 or above 10
```

For Strings

```
Section IN ('A','C','F') => Section is in the list ('A','C','F')

Section NOT IN ('A','C','F'))=>  Section is not in the list ('A','C','F')

Section BETWEEN 'B' AND 'D' =>Section is equal to 'B', 'D', or anything
in between (alphabetically)

Section NOT BETWEEN 'B' AND 'D' => Section is below 'B' or above 'D' (alphabetically)
```

AND , OR Similar.

3.1.2 Sub-queries.

Example in a real-life scenario:

A subquery answers multiple-part questions. For example, to determine who works in Taylor's department, you can first use a subquery to determine the department in which Taylor works. You can then answer the original question with the parent SELECT statement.

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

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

(a) Subquery in WHERE clause: Example

The following statement *returns data about employees whose salaries exceed their department average.* The following statement assigns an alias to employees, the table containing the salary information, and then uses the alias in a correlated subquery:

```
SELECT department_id, last_name, salary
  FROM employees x
  WHERE salary > (SELECT AVG(salary)
     FROM employees
     WHERE x.department_id = department_id)
  ORDER BY department_id;
```

(b) Subquery in FROM clause : Example

A subquery can also be found in the FROM clause. These are called inline views.

Display the top five earner names and salaries from the EMPLOYEES table:

```
SELECT ROWNUM as RANK, last_name, salary
FROM (SELECT last_name, salary
 FROM employees
 ORDER BY salary DESC)
 WHERE ROWNUM <= 5;
```

3.2 Joins,UNION, INTERSECT, and MINUS

3.2.1 Joins

3.2.1.1 Inner Join or Natural Join

(a) New format using *natural* keyword

You can use the natural keyword to indicate that a join should be performed based on all columns that have the same name in the two tables being joined. For example, what titles in `BOOK_ORDER` match those already in BOOKSHELF?

```
 select Title
from BOOK_ORDER natural join BOOKSHELF;
```

The natural join returned the results as if you had typed in the following:

```
 select BO.Title
from BOOK_ORDER BO, BOOKSHELF
where BO.Title = BOOKSHELF.Title
and BO.Publisher = BOOKSHELF.Publisher
and BO.CategoryName = BOOKSHELF.CategoryName;
```

Note: The main difference is that INNER JOIN is used in real life; NATURAL JOIN is only used in textbooks.

INNER JOIN Note that they support the on and using clauses, so you can specify your join criteria as shown in the following listing:

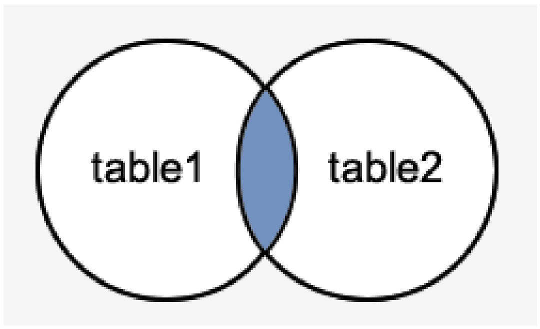

Figure 3.1: Inner Join

New Syntax:

```
 SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date
FROM suppliers
INNER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;
```

Old Syntax:

```
 SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date
FROM suppliers, orders
WHERE suppliers.supplier_id = orders.supplier_id;
```

(b) Classical format

```
 SELECT ...
FROM   dataset_one d1
,      dataset_two d2
WHERE  d1.column(s) = d2.column(s)
AND    ...
```

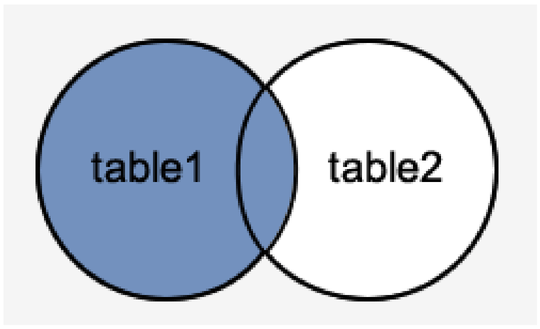

Figure 3.2: Left Outer Join

3.2.1.2 Outer Join

```
 The New syntax for the Oracle LEFT OUTER JOIN is:

 SELECT columns
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;

 Example:

 SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date
FROM suppliers
LEFT OUTER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;


Example Old:

 SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date
FROM suppliers, orders
WHERE suppliers.supplier_id = orders.supplier_id(+);


Right Outer:
New:

SELECT orders.order_id, orders.order_date, suppliers.supplier_name
FROM suppliers
RIGHT OUTER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;

Right Outer:
Old:
```

```
SELECT orders.order_id, orders.order_date, suppliers.supplier_name
FROM suppliers, orders
WHERE suppliers.supplier_id(+) = orders.supplier_id;



Full Outer :

SELECT columns
FROM table1
FULL [OUTER] JOIN table2
ON table1.column = table2.column;

New:

SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_date
FROM suppliers
FULL OUTER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;

Old:
As a final note, it is worth mentioning that the FULL OUTER JOIN example above could
written in the old syntax without using a UNION query.
```

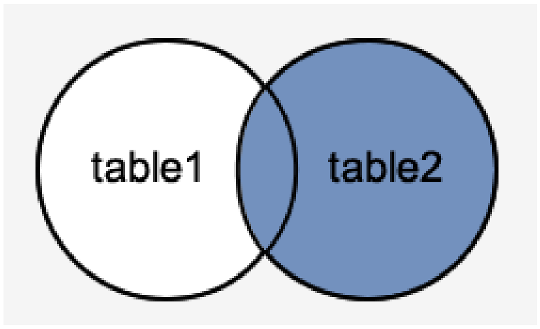

Figure 3.3: Right Outer Join

(a) Pre-Oracle9i Syntax for Outer Joins

```
 select B.Title, MAX(BC.ReturnedDate - BC.CheckoutDate)
"Most Days Out"
from BOOKSHELF_CHECKOUT BC, BOOKSHELF B
where BC.Title (+) = B.Title
group by B.Title;
```

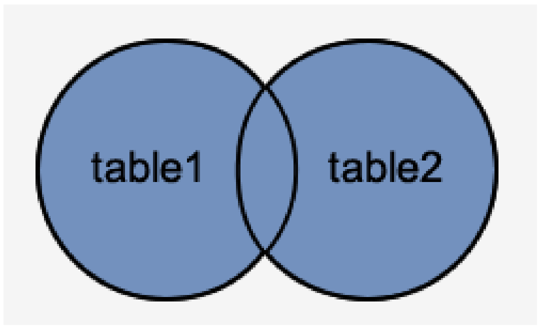

Figure 3.4: Full Outer Join

(b) Modern Syntax for Outer Joins

```
select B.Title, MAX(BC.ReturnedDate - BC.CheckoutDate)
"Most Days Out"
from BOOKSHELF_CHECKOUT BC right outer join BOOKSHELF B
on BC.Title = B.Title
group by B.Title;
```

Left and Right Outer: So, what is the difference between the right and left outer joins? The difference is simple in a left outer join, all of the rows from the left table will be displayed, regardless of whether there are any matching columns in the right table. In a right outer join, all of the rows from the right table will be displayed, regardless of whether there are any matching columns in the left table. Hopefully the example that we gave above help clarified this as well.