Working with Oracle SQL

Chapter 6:

Data Definition Language to create and manage tables

Chapter Objectives

In this chapter, we will discuss:

- Creating and managing tables
- Altering tables
- Using sequence generators
- Managing integrity constraints
- Creating and managing views
- Working with indexes

Chapter Concepts

- **Schemas**
- Create and Manage Tables
- Alter a Table
- Creating Sequence Generators
- Enforcing Database Integrity
- Managing Views
- Managing Indexes
- Using SQL Developer
- Chapter Summary

Schema

- “Schema” is a relational database word meaning:
  - Collection of database objects
  - Including structures such as:
    - Tables
    - Views
    - Stored procedures
    - Indexes
- A schema has the name of the user who owns it
  - The terms schema, user, and account tend to be used interchangeably
  - `HR` and `SCOTT` are the users/schemas we have been using

Schema Objects

- Everything in the database *must* be owned by a user
  - The collection of objects owned by a user is called the schema
- A schema object is owned by the user account it is created in
  - Qualified by: `user_name.table_name`
  - Example: `hr.employees`
- Legal schema object names must conform to the Oracle standard
  - Maximum of 30 positions
  - Must start with an alpha character
  - Can include numbers and the special characters: $ _ #

Chapter Concepts

- Schemas
- **Create and Manage Tables**
- Alter a Table
- Creating Sequence Generators
- Enforcing Database Integrity
- Managing Views
- Managing Indexes
- Using SQL Developer
- Chapter Summary

CREATE TABLE

- The simplified syntax is:

```
CREATE TABLE table_name (
    column_name  datatype  [DEFAULT default_value] [NOT NULL]
    [, … ]
)
```

- A table must have at least one column
  - Each column must have a valid datatype
    - Can be defined as being mandatory
    - Can have a `DEFAULT` value set
- `CREATE TABLE` is a DDL command
  - DDL commands automatically commit
  - Any DDL command automatically commits any outstanding transaction

CREATE TABLE Example

- A table to store client information

```
CREATE TABLE clients (
    client_id          NUMBER(5) NOT NULL
  , client_name        VARCHAR2(20)
  , client_type        NUMBER(2) DEFAULT 01
  , client_start_date DATE
);

Table created.
```

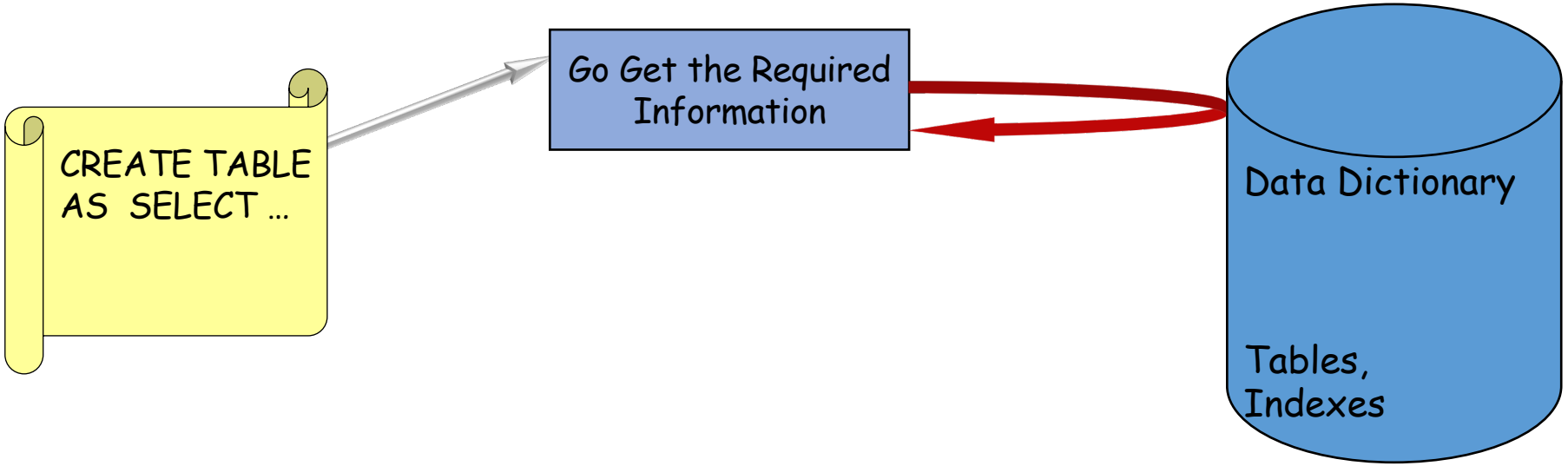
- NOT NULL is an integrity constraint specifying that the column is mandatory
- The client type is set to 01, **if** a value is not provided when a row is inserted
  - Using the key word DEFAULT in an UPDATE or INSERT will cause this value to be used

CREATE TABLE AS SELECT: CTAS

- IF you want to base the definition of a new table upon an existing one, the CTAS statement can be used
- Syntax:

```
CREATE TABLE table_name AS SELECT ….
```

- The necessary column specifications are retrieved from the data dictionary



CTAS: Example 1

- An image copy of an existing table can be made
  - Notice that the rows of data are brought over

```
CREATE TABLE new_jobs AS
    SELECT *
    FROM jobs;

Table created.

SELECT *
FROM new_jobs;

JOB_ID     JOB_TITLE                          MIN_SALARY MAX_SALARY
---------- ---------------------------------- ---------- ----------
AD_PRES    President                               20000      40000
AD_VP      Administration Vice President           15000      30000
AD_ASST    Administration Assistant                 3000       6000
FI_MGR     Finance Manager                          8200      16000
...

19 rows selected.
```

CTAS: Example 1 (Continued)

- IF you wanted a copy of the definition of the table without any data, write the `WHERE` clause to exclude all data

```
CREATE TABLE new_jobs AS
    SELECT *
    FROM jobs
    WHERE 1 = 2;

Table created.

SELECT * FROM new_jobs;

No rows selected.
```

CTAS: Example 2

- We can also specify which columns we want and their names by providing a column list

```
CREATE TABLE new_jobs (title, job_number) AS
    SELECT job_title, job_id
    FROM jobs;

Table created.

SELECT * FROM new_jobs;

TITLE                               JOB_NUMBER
----------------------------------- ----------
President                           AD_PRES
Administration Vice President       AD_VP
Administration Assistant            AD_ASST
Public Relations Representative     PR_REP
. . .

19 rows selected.
```

CTAS: Example 3

- We can also limit the rows in the new table by specifying a `WHERE` clause

```
CREATE TABLE new_jobs (title, job_number) AS
    SELECT job_title, job_id
    FROM jobs
    WHERE job_id LIKE 'IT%';

Table created.

SELECT * FROM new_jobs;

TITLE                              JOB_NUMBER
---------------------------------- ----------
Programmer                         IT_PROG
```

CTAS: Example 4

- The table being created does not have to be based upon a single table
  - Any `SELECT` statement can be used

```
CREATE TABLE jobs_employees AS
    SELECT j.job_title, e.last_name
    FROM jobs j
    JOIN employees e
    USING (job_id)
    WHERE job_id LIKE 'IT%';

Table created.

SELECT * FROM jobs_employees;

JOB_TITLE                          LAST_NAME
---------------------------------- -------------------------
Programmer                         Hunold
Programmer                         Ernst
Programmer                         Austin
Programmer                         Pataballa
Programmer                         Lorentz
```

Other Table Commands

- Tables can be renamed

```
RENAME table_name TO new_table_name
RENAME jobs TO regional_jobs;
```

- Be careful:
  - Permissions and references to the original table are not automatically maintained
- Tables can have their rows quickly removed

```
TRUNCATE TABLE new_jobs;
```

  - Accomplishes the same result as `DELETE` without the `WHERE` clause
- Be careful:
  - `TRUNCATE` is a DDL command and cannot be rolled back
  - The reason why it is so efficient

Dropping Tables

- To remove the definition of the table (not just the rows):

```
DROP TABLE table_name
DROP TABLE new_jobs;
```

- This removes the table and all dependencies
  - Indexes, permissions, etc.
  - Code references become invalid

Restoring Dropped Tables

- Beginning in Oracle10g, the table is not actually dropped
  - The table is renamed and “held” in the recycle bin
    - Along with dependent objects, indexes, triggers, constraints
- The table can be restored using a `FLASHBACK` command
  - `FLASHBACK TABLE table_name TO BEFORE DROP`
  - `FLASHBACK new_jobs TO BEFORE DROP;`
    - Dependent objects, except for foreign keys, are also restored
- The table can also be renamed as a part of the flashback operation
  - `FLASHBACK new_jobs TO BEFORE DROP RENAME TO some_jobs;`
- Unless purged, the contents of the recycle bin will be available as long as space exists in the area where the original table was stored
- The recycle bin can be bypassed during the `DROP` by appending the `PURGE` option

```
DROP TABLE new_jobs PURGE;
```

Working with the Recycle Bin

- To examine the recycle bin

```
SELECT ORIGINAL_NAME, OBJECT_NAME, TYPE, DROPTIME, CAN_UNDROP FROM user_recyclebin;

ORIGINAL_NAME    OBJECT_NAME                     TYPE   DROPTIME            CAN_UNDROP
---------------- ------------------------------ ------ ------------------- ----------
NEW_JOBS         BIN$Y/ym7LmtSh2IYYBGAoBqoQ==$0 TABLE  2018-05-03:09:36:53 YES
```

  - Can access objects directly from the bin

```
SELECT * FROM "BIN$Y/ym7LmtSh2IYYBGAoBqoQ==$0";
```

- A specific table can be removed from the recycle bin
  - Or, if there was more than one matching table:

```
PURGE TABLE new_jobs;
```

```
PURGE TABLE "BIN$Y/ym7LmtSh2IYYBGAoBqoQ==$0";
```

- The entire contents of the recycle bin can be purged with a single command
  - Verify the contents first
  - Requires DBA privileges

```
PURGE DBA_RECYCLEBIN;
```

Chapter Concepts

- Schemas
- Create and Manage Tables
- **Alter a Table**
- Creating Sequence Generators
- Enforcing Database Integrity
- Managing Views
- Managing Indexes
- Using SQL Developer
- Chapter Summary

Altering Tables

- A table, once created, can have many of its settings changed via the `ALTER` command
- For example, to add a new column to an existing table:

```
ALTER TABLE table_name
ADD (column_name  datatype  [DEFAULT default_value]  [NOT NULL]
  [, …] )

ALTER TABLE new_jobs
ADD (effective_date DATE);
Table altered.

DESCRIBE new_jobs
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------
 JOB_ID                                             VARCHAR2(10)
 JOB_TITLE                                 NOT NULL VARCHAR2(35)
 MIN_SALARY                                         NUMBER(6)
 MAX_SALARY                                         NUMBER(6)
 EFFECTIVE_DATE                                     DATE
```

Can only add a `NOT NULL` column if the table is empty, or there is a default specified

Altering Tables: Modifying Columns

- Using a similar syntax, existing columns can be modified
- Example:

```
ALTER TABLE new_jobs
MODIFY (max_salary  NUMBER(8));

Table altered.

DESCRIBE new_jobs
 Name                                      Null?    Type
 ----------------------------------------- -------- ---------------
 JOB_ID                                             VARCHAR2(10)
 JOB_TITLE                                 NOT NULL VARCHAR2(35)
 MIN_SALARY                                         NUMBER(6)
 MAX_SALARY                                         NUMBER(8)
 EFFECTIVE_DATE                                     DATE
```

Restrictions on Modifying Columns

- If rows already exist, then the pre-existing data must already agree with the modification
- Examples:
  - A column's size cannot be made smaller than the existing data
  - A column's datatype cannot be changed as long as there is existing data in the column
  - Any existing constraints must be adhered to

```
ALTER TABLE new_jobs
MODIFY (job_title VARCHAR2(4));
(job_title VARCHAR2(4) )
 *
ERROR at line 2:
ORA-01441: cannot decrease column length because some value is too big

ALTER TABLE new_jobs
MODIFY (job_id  NUMBER(6));
(job_id  NUMBER(6) )
 *
ERROR at line 2:
ORA-01439: column to be modified must be empty to change datatype
```

Altering Tables: Removing Columns

- Columns can be removed from the table in one of several ways
  - Remove the definition and the data

```
ALTER TABLE new_jobs
DROP (min_salary, max_salary) CASCADE CONSTRAINTS;

Table altered.

DESCRIBE new_jobs
Name                                 Null?    Type
------------------------------------ -------- --------------
JOB_ID                                        VARCHAR2(10)
JOB_TITLE                            NOT NULL VARCHAR2(35)
EFFECTIVE_DATE                                DATE
```

  - The `CASCADE` clause is needed when the column is a Foreign Key
    - In this example, the clause is unnecessary

Altering Tables: Setting Columns Unused

- The data dictionary can be modified without physically removing the column's data
  - The `SET UNUSED` clause

```
ALTER TABLE new_jobs
SET UNUSED (effective_date) CASCADE CONSTRAINTS;

Table altered.

DESCRIBE new_jobs
Name                                      Null?    Type
----------------------------------------- -------- ----------------
JOB_ID                                             VARCHAR2(10)
JOB_TITLE                                 NOT NULL VARCHAR2(35)
```

  - Allows the data to be physically removed when convenient

```
ALTER TABLE new_jobs
DROP UNUSED COLUMNS;
Table altered.
```

Altering Tables: Renaming Columns

- Existing columns can be renamed with the `RENAME` clause

```
ALTER TABLE new_jobs
RENAME COLUMN job_title TO title;

Table altered.

DESCRIBE new_jobs
Name                                      Null?    Type
----------------------------------------- -------- -------------
JOB_ID                                             VARCHAR2(10)
TITLE                                     NOT NULL VARCHAR2(35)
```

  - Any dependent constraints are maintained
  - Dependent code (triggers, views, stored procedures, etc.) are invalidated

Chapter Concepts

- Schemas
- Create and Manage Tables
- Alter a Table
- **Creating Sequence Generators**
- Enforcing Database Integrity
- Managing Views
- Managing Indexes
- Using SQL Developer
- Chapter Summary

Sequence Generator

- A sequence generator is a database object
  - It is an Oracle-managed series of numbers
  - Often used to provide the values for primary keys
    - Remember, primary keys must be unique
- Shortened (simplified) syntax:

```
CREATE SEQUENCE sequence_name
 INCREMENT BY interval
 START WITH start_value
```

- Example:

```
CREATE SEQUENCE seq_emp
 INCREMENT BY 1
 START WITH 8000;

Sequence created.
```

Sequence Generator Example

- We can now use this sequence to provide the data value for the primary key (`empno`) of the emp table
- Get the next sequence value by using `NEXTVAL`

```
INSERT INTO emp (empno, ename)
VALUES (seq_emp.NEXTVAL,'CHAVAS');

1 row created.
```

- Now retrieve the rows

```
SELECT empno, ename FROM emp WHERE empno > 7900;
     EMPNO ENAME
---------- ----------
      7902 FORD
      7934 MILLER
      8000 CHAVAS
```

- A sequence number, once consumed, cannot be put back (even if rollback occurs)

IDENTITY Columns

- Sequences are normally combined with triggers to populate key fields automatically
  - Starting with Oracle 12c, IDENTITY columns make this functionality more accessible

```
GENERATED [ ALWAYS | BY DEFAULT [ ON NULL ] ] AS IDENTITY [ ( options ) ]
```

  - ALWAYS means it will always be auto-generated, specifying a value is an error
  - BY DEFAULT means it will be auto-generated unless specified
  - BY DEFAULT ON NULL means auto-generated if set to NULL
  - options are the same as for a SEQUENCE

```
CREATE TABLE clients (
    client_id          NUMBER GENERATED ALWAYS AS IDENTITY
  , client_name        VARCHAR2(20)
  , client_type        NUMBER(2) DEFAULT 01
  , client_start_date DATE
);

Table created.
```

Chapter Concepts

- Schemas
- Create and Manage Tables
- Alter a Table
- Creating Sequence Generators
- **Enforcing Database Integrity**
- Managing Views
- Managing Indexes
- Using SQL Developer
- Chapter Summary

Integrity Constraints

- Declarative method of enforcing business rules in the database
- Defined with the corresponding table using the `CREATE TABLE` or `ALTER TABLE` command
- Validated when constraints are defined and on each `INSERT`, `UPDATE`, and `DELETE`
- Five types
  - `NOT NULL`
  - Primary key
  - Unique key
  - Foreign key
  - Check

NOT NULL Constraints

- Enforces that a column contains a value for each row
- Usually defined in the CREATE TABLE command, as discussed in the previous section
- Example:
  - empno is mandatory

```
CREATE TABLE emp (
    empno NUMBER(4)    NOT NULL
  , ename VARCHAR2(10)
  , ...
);
```

- May be used with ALTER TABLE MODIFY after adding a new column to a populated table
  - Add column without the constraint
  - Populate the column
  - Alter the table to add the constraint

Primary Key Constraints

- Ensures that no two rows of a table have duplicate values in the key column(s)
  - All columns in the primary key must be mandatory
  - Only one primary key is allowed per table
- For ease of maintenance, use the `ALTER TABLE` command with the following syntax:

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
PRIMARY KEY (column_name, ..., column_name);
```

Primary Key Constraints (continued)

- An ideal primary key column is one whose value rarely changes
  - Updating a primary key usually impacts other tables
- A common naming convention is to end primary keys with the `_pk` suffix
  - Makes it easier to identify these keys when we query the data dictionary
- Example:
  - `empno` is used to uniquely identify employees

```
ALTER TABLE emp
ADD CONSTRAINT emp_pk
PRIMARY KEY (empno);
```

- Short numeric columns are preferred as primary keys because they offer the best performance and require the least amount of storage

Unique Key Constraints

- Same rules as for primary key, but allow optional columns
  - Allow an unlimited number per table
  - A common standard is to end unique keys with the `_uk` suffix
- For ease of maintenance, use the `ALTER TABLE` command with the following syntax:

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
UNIQUE (column_name, ..., column_name);
```

- Example:
  - `ename`, `job`, and `hiredate` must be unique

```
ALTER TABLE emp
ADD CONSTRAINT emp_uk1
UNIQUE (ename, job, hiredate);
```

Foreign Key Constraints

- Ensures that every value of a foreign key in the child table exists in the parent table
  - Prevents updates and deletes of parent table that would violate the constraint

```
ALTER TABLE child_table_name
ADD CONSTRAINT constraint_name
FOREIGN KEY (column_name, ..., column_name)
REFERENCES parent_table_name [(column_name, ..., column_name)]
[ ON DELETE CASCADE | ON DELETE SET NULL ];
```

- A foreign key can only reference existing primary or unique keys of the parent table
  - If no parent columns are specified, references primary key
- The `ON DELETE` options:
  - `CASCADE` automatically deletes child records on deletion of parent record
  - `SET NULL` automatically updates child columns to `NULL` on deletion of parent record
- A common naming convention is to end foreign keys with the `_fk` suffix

Foreign Key Constraint Examples

- Each employee must belong to a department that is defined in the `dept` table
  - A department cannot be deleted if employees work there

```
ALTER TABLE emp
ADD CONSTRAINT emp_dept_fk1
FOREIGN KEY (deptno) REFERENCES dept (deptno);
```

- Each employee must belong to a department that is defined in the `dept` table
  - If a department is being deleted, all of its employees should be deleted

```
ALTER TABLE emp
ADD CONSTRAINT emp_dept_fk1
FOREIGN KEY (deptno) REFERENCES dept (deptno)
ON DELETE CASCADE;
```

Make sure this is really what you want!

Check Constraints

- Enforces business rules on update or insert
  - Limited to simple validation based on constants or column values in the row

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
CHECK (condition);
```

- A common naming convention is to end check constraints with the `_ck` suffix
- E.g., Commission must always be less than 10% of salary

```
ALTER TABLE emp
ADD CONSTRAINT emp_comm_sal_ck
CHECK (comm < sal * .10);
```

Dropping Constraints

- Drop constraints using the following syntax:

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name;
```

  - For example, drop `emp_comm_sal_ck` check constraint

```
ALTER TABLE emp
DROP CONSTRAINT emp_comm_sal_ck;
```

- A simpler way to drop primary key constraints is using this syntax:

```
ALTER TABLE table_name
DROP PRIMARY KEY [CASCADE];
```

  - `CASCADE` option drops all the related foreign key constraints
  - For example, drop the primary key on `emp` table

```
ALTER TABLE emp
DROP PRIMARY KEY;
```

Enabling and Disabling Constraints

- Constraints can be made temporarily inactive and can later be reactivated using the following syntax:

```
ALTER TABLE table_name
DISABLE CONSTRAINT constraint_name;
```

```
ALTER TABLE table_name
ENABLE CONSTRAINT constraint_name;
```

- Examples:
  - Disable `emp_dept_fk1` foreign key constraint, then re-enable it

```
ALTER TABLE emp DISABLE CONSTRAINT emp_dept_fk1;

Table EMP altered

ALTER TABLE emp ENABLE CONSTRAINT emp_dept_fk1;

Table EMP altered
```

Constraints in the Dictionary

- Constraints are shown in the `user_constraints` dictionary view
  - Constraint type is “P” for primary key, “U” for unique key, “R” for referential constraint or foreign key, “C” for check constraint, and “V” for view with check option
  - Constraint status is either `ENABLED` or `DISABLED`
  - Constraint columns are stored in `user_cons_columns` view
- Example:

```
SELECT constraint_name, constraint_type, status
FROM user_constraints
WHERE table_name = 'EMP';

CONSTRAINT_NAME                C STATUS
------------------------------ - --------
EMP_PK                         P ENABLED
EMP_DEPTNO_FK1                 R ENABLED
```

Chapter Concepts

- Schemas
- Create and Manage Tables
- Alter a Table
- Creating Sequence Generators
- Enforcing Database Integrity
- **Managing Views**
- Managing Indexes
- Using SQL Developer
- Chapter Summary

Views

- A view is a stored definition of data
  - The data dictionary is updated when a view is created
  - The data is not physically stored in a view
    - It is read from the tables the view references
- Syntax:

```
CREATE OR REPLACE VIEW view_name (column_name, column_name …) AS
SELECT …
[WITH { READ ONLY } | { CHECK OPTION [CONSTRAINT constraint_name ] } ]
```

  - The `SELECT` statement can be any valid query
  - The view can specify the column names
  - There are two possible restrictions
    - `READ ONLY` means the view data cannot be updated
    - `CHECK OPTION` restricts updates to values that would be selected by the view

Reasons to Create Views

- Security
  - To restrict the columns that can be seen
  - To restrict the rows that can be read
- To simplify the accessing of information
  - Complex SQL statements do not have to be continually recreated
- To provide typical application images of the information
  - Including using a different column name
    - Preferred by different groups
- Isolate applications from table changes

VIEW Example

- Provide a restricted view of the jobs table for other accounts
  - Only three of the columns and some of the rows in the jobs table can be seen
  - So that a restricted user can only access the data in the view

```
CREATE OR REPLACE VIEW vw_jobs (max_sal, min_sal, title) AS
SELECT max_salary, min_salary, job_title
FROM jobs
WHERE max_salary < 8000
WITH CHECK OPTION CONSTRAINT ck_restrict_jobs;

View created.

SELECT * FROM vw_jobs;
   MAX_SAL    MIN_SAL TITLE
---------- ---------- -------------------------
      6000       3000 Administration Assistant
      5500       2500 Purchasing Clerk
      5000       2000 Stock Clerk
      5500       2500 Shipping Clerk
```

Updatable Views

- Views may be updatable if they contain a key-preserved table
  - A table where each row appears only once in the view results
- Each operation on an updatable view may only modify the data of one underlying table

```
UPDATE vw_jobs
SET max_sal = 7500
WHERE title = 'Administration Assistant';

1 row updated.

UPDATE vw_jobs
SET max_sal = 8500
WHERE title = 'Administration Assistant';

Error report -
ORA-01402: view WITH CHECK OPTION where-clause violation
```

Updatable Views (continued)

- To find out if a view column is updatable

```
SELECT table_name, column_name, updatable
FROM   user_updatable_columns
WHERE  table_name = 'VW_JOBS';

TABLE_NAME                     COLUMN_NAME                    UPD
------------------------------ ------------------------------ ---
VW_JOBS                        MAX_SAL                        YES
VW_JOBS                        MIN_SAL                        YES
VW_JOBS                        TITLE                          YES
```

Chapter Concepts

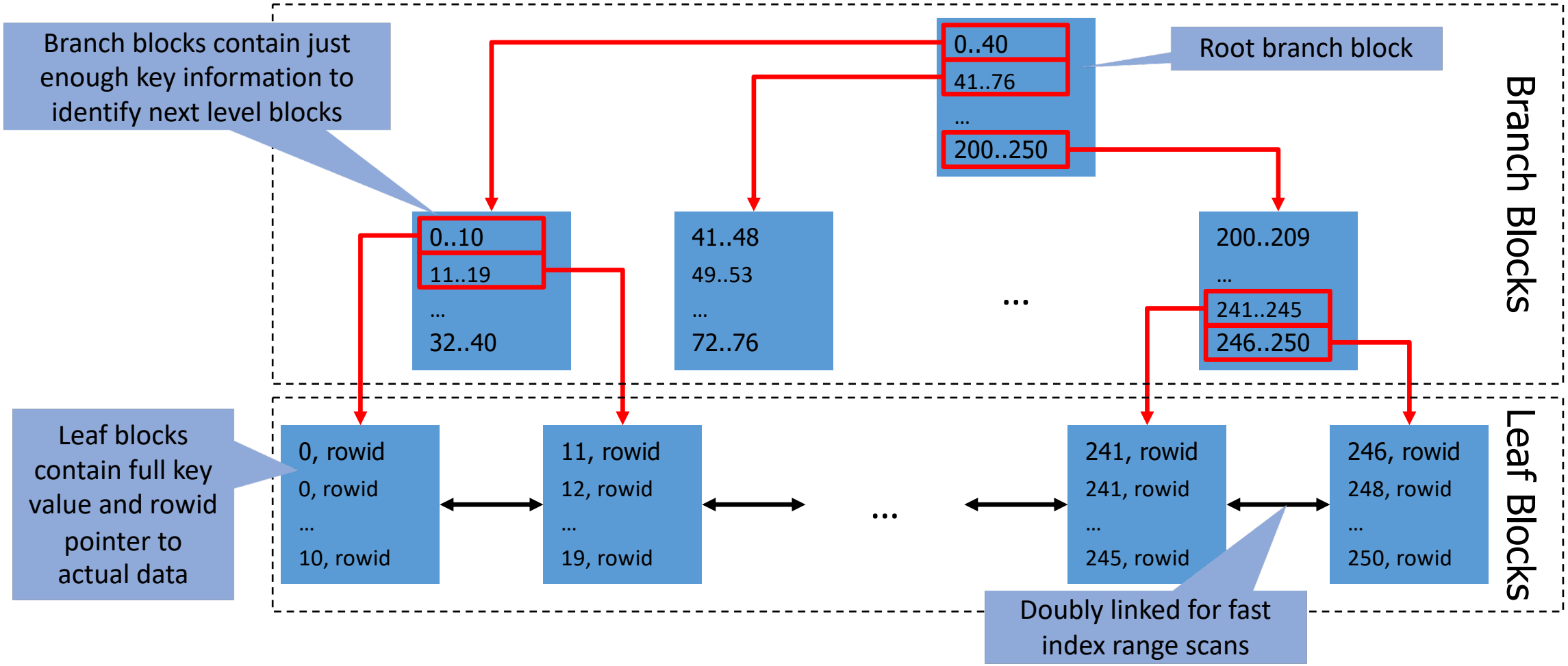
- Schemas
- Create and Manage Tables
- Alter a Table
- Creating Sequence Generators
- Enforcing Database Integrity
- Managing Views
- **Managing Indexes**
- Using SQL Developer
- Chapter Summary

Indexes

- An index is a schema object for improving query performance
  - Stored separately from the table
  - Not referenced in SQL, used by query optimizer
- Indexes work because keys are generally much smaller than the data they reference
  - Index stores only the key and a pointer to the actual data (rowid)
  - Index can be traversed much faster than the data
  - Oracle automatically creates indexes for primary key and unique constraints
- Default Oracle index type is B-Tree index
  - “Balanced” tree because each leaf node is the same number of steps away from the root
  - Accessing leaf data requires O(log N) block accesses (increases very slowly)

B-Tree Indexes

- Imagine a numeric index, e.g., foreign key index for `department_id` in `employees`



Unique Index

- An index with a one-to-one relationship to table rows
  - Efficient because each leaf entry points to one row in the table
  - Automatically generated for each primary key and unique key constraint
    - Unless there is already a covering index
  - They are dropped when the underlying constraint is dropped or disabled

```
ALTER TABLE emp
ADD CONSTRAINT emp_pk
PRIMARY KEY (empno);
```

```
ALTER TABLE emp
ADD CONSTRAINT emp_uk1
UNIQUE (ename, hiredate);
```

  - Or you can create your own

```
CREATE UNIQUE INDEX emp_pk_ix
ON emp (empno);
```

```
CREATE UNIQUE INDEX emp_uk1_ix
ON emp (ename, hiredate);
```

- Indexes are frequently named with a suffix `_ix`

Non-Unique Index

- A non-unique index has values that point to one or more table rows
  - More than one rowid per key value
  - They are generally less efficient than unique indexes
    - How much less efficient depends on the selectivity of the columns

```
CREATE INDEX index_name ON table_name (column_name, ..., column_name)
```

- Example:
  - Manager in the emp table is not unique
    - More than one employee can work for the same manager
  - If we frequently asked the question “who works for…”, we might create an index to improve performance

```
CREATE INDEX emp_mgr_ix
ON emp (mgr);
```

Dropping Indexes

- Any index may be dropped unless it is being used to enforce a constraint

```
DROP INDEX emp_mgr_ix;
```

- Automatically created indexes for primary keys and unique constraints are automatically dropped if the constraint is dropped
  - You cannot usually drop them manually
- Any index you create may be used by a constraint that is created later
  - In this case, you will not be able to drop the index without first dropping the constraint

Function-Based Indexes

- You can create an index that includes a function call
  - SQL scalar function
  - User-defined function
  - Package function
- Oracle pre-computes the function return value and stores it in the index

```
CREATE INDEX emp_fname_uppercase_ix
ON employees (UPPER(first_name));
```

- What do you think the performance impact of this will be:
  - For queries?
  - For data maintenance (insert and update)?

Indexing Guidelines

- Deciding WHAT indexes to create is driven by HOW the data is accessed
  - This can change over time
- Columns to consider for an index
  - All columns that are part of primary and unique keys are indexed automatically
  - Foreign keys that are commonly used for joins
  - Frequently queried columns that are relatively unique (high selectivity)
    - Return less than 10 percent of the rows when queried
- Columns not to index
  - Columns that are frequently updated
  - Columns in small tables
  - Columns that contain few unique values (low selectivity)
    - Reason: likely to be retrieving many rows

Exercise 6.1: Table Management

45 min

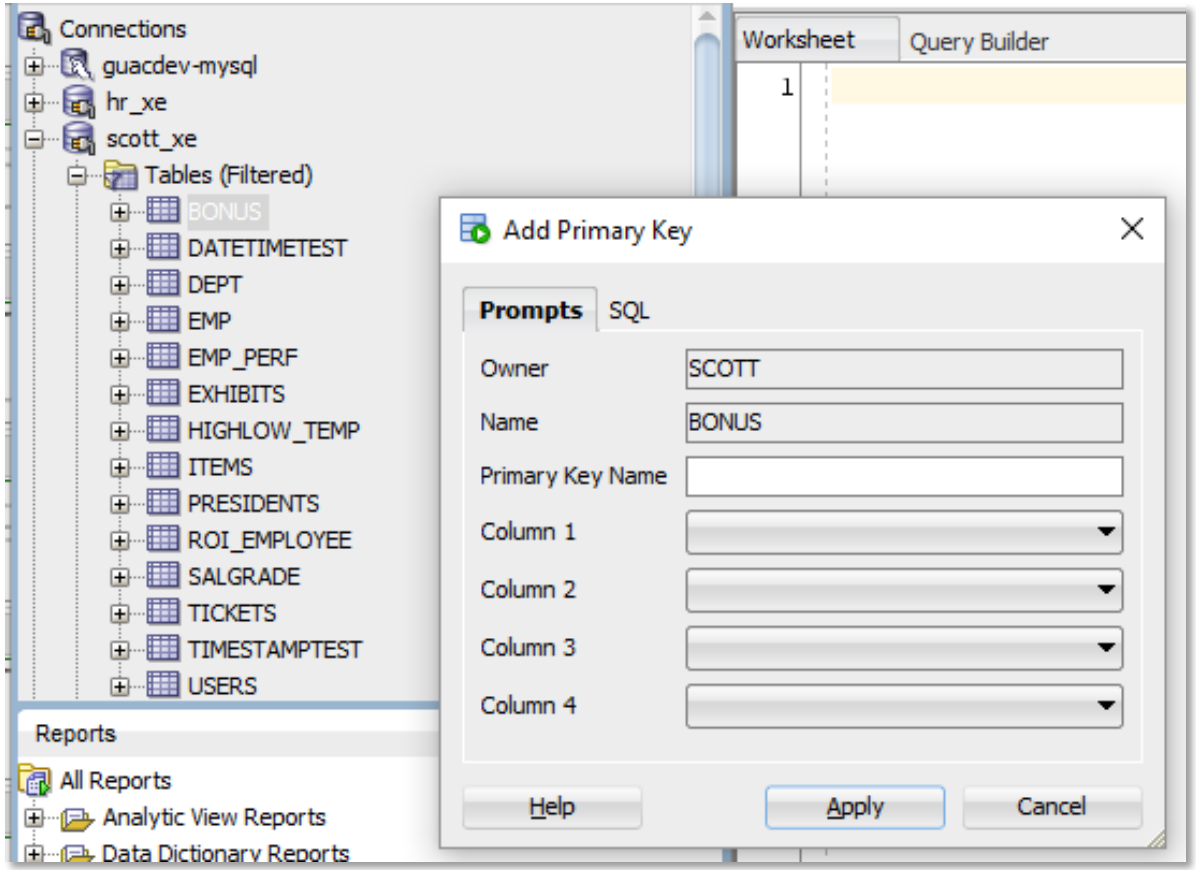
- Please turn to the Exercise Manual and complete this exercise

Chapter Concepts

- Schemas
- Create and Manage Tables
- Alter a Table
- Creating Sequence Generators
- Enforcing Database Integrity
- Managing Views
- Managing Indexes
- **Using SQL Developer**
- Chapter Summary

DDL Through SQL Developer

- SQL Developer provides a method of doing most DDL through the GUI
- ***Do not use this feature***
- Get in the habit of typing the commands:
  - It helps you learn
  - It means there is a clear history
  - It allows you to maintain a DDL file
- If you are unsure of the syntax, use SQL Developer and then immediately export the DDL to examine it and to keep a history



Maintaining DDL Files

- You should maintain a series of DDL files for your project
  - You should be able to completely recreate the database from these scripts
  - Separate test data from schema structure
  - Consider separating schema objects into separate files and including them using `@@`

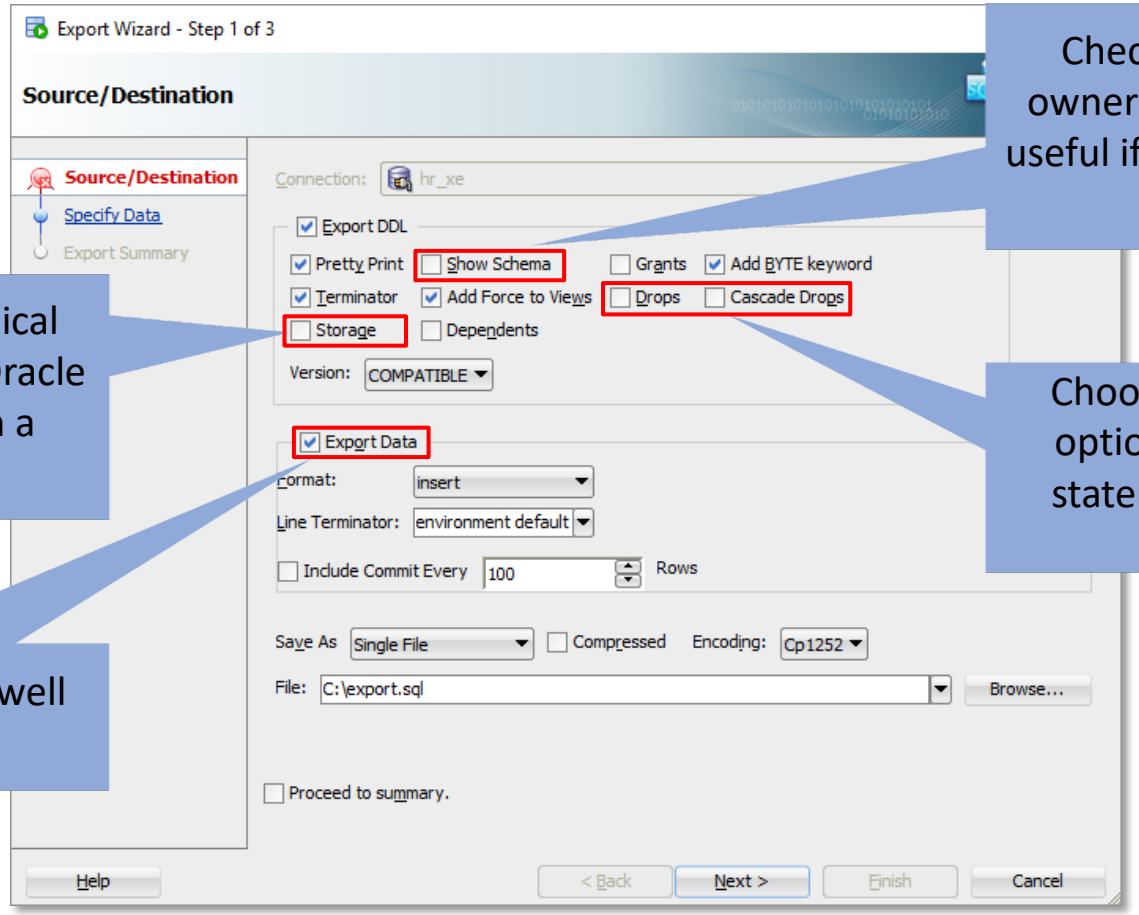
```
@@client_ddl.sql
@@staff_ddl.sql

@@populate_data.sql
```

https://docs.oracle.com/en/database/oracle/oracle-database/12.2/sqpug/SQL-Plus-command-reference.html

Exporting from SQL Developer

- You can generate a starting point by exporting from SQL Developer
  - Experiment with the options



Checking this option shows schema ownership for each object, which is not useful if you want to re-create the objects in another schema

Adds information about the physical storage that you usually leave to Oracle and makes it harder to apply on a different server

Choosing one of these options creates `DROP` statements before the `CREATE`s

If you want the data as well as the structure

Chapter Concepts

- Schemas
- Create and Manage Tables
- Alter a Table
- Creating Sequence Generators
- Enforcing Database Integrity
- Managing Views
- Managing Indexes
- Using SQL Developer
- **Chapter Summary**

Chapter Summary

In this chapter, we have discussed:

- Creating and managing tables
- Altering tables
- Using sequence generators
- Managing integrity constraints
- Creating and managing views
- Working with indexes