Introduction to Data Definition Language

DDL
(Create, Alter ,Drop, Rename, Truncate)

Some Database Objects

Object	Description
Table	Is the basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative name to an object

Oracle Table Structures

- Tables can be created at any time, even when users are using the database.
- You do not need to specify the size of a table. The size is ultimately defined by the amount of space allocated to the database as a whole. It is important, however, to estimate how much space a table will use over time.
- Table structure can be modified online.

Note: More database objects are available, but are not covered in this course.

Naming Rules

Table names and column names must:

- Begin with a letter
- Be 1–30 characters long
- Contain only A–Z, a–z, 0–9, _, $, and #
- Not duplicate the name of another object owned by the same user
- Not be an Oracle server–reserved word

- Names must not be an Oracle server–reserved word.
  - You may also use quoted identifiers to represent the name of an object. A quoted identifier begins and ends with double quotation marks (“”). If you name a schema object using a quoted identifier, you must use the double quotation marks whenever you refer to that object. Quoted identifiers can be reserved words, although this is not recommended.

Naming Guidelines

Use descriptive names for tables and other database objects.

Note: Names are not case-sensitive. For example, `EMPLOYEES` is treated to be the same name as `eMPloyees` or `eMpLOYEES`. However, quoted identifiers are case-sensitive.

Data Types

Data Type	Description
VARCHAR2(*size*) ★	Variable-length character data (A maximum *size* must be specified: minimum *size* is 1.) Maximum size is: • 32767 bytes if MAX_SQL_STRING_SIZE = EXTENDED • 4000 bytes if MAX_SQL_STRING_SIZE = LEGACY
CHAR [(*size*)] ★	Fixed-length character data of length *size* bytes (Default and minimum *size* is 1; maximum *size* is 2,000.)
NUMBER [(*p*,*s*)] ★	Number having precision *p* and scale *s* (Precision is the total number of decimal digits and scale is the number of digits to the right of the decimal point; precision can range from 1 to 38, and scale can range from –84 to 127.)
DATE ★	Date and time values to the nearest second between January 1, 4712 B.C., and December 31, 9999 A.D.

https://docs.oracle.com/cd/B28359_01/server.111/b28318/datatype.htm

Data Types

Data Type	Description
LONG	Variable-length character data (up to 2 GB)
CLOB	A character large object containing single-byte or multibyte characters. Maximum size is (4 gigabytes - 1) * (DB_BLOCK_SIZE); stores national character set data.
NCLOB	A character large object containing Unicode characters. Both fixed-width and variable-width character sets are supported, both using the database national character set. Maximum size is (4 gigabytes - 1) * (database block size); stores national character set data.
RAW(size)	Raw binary data of length *size* bytes. You must specify *size* for a RAW value. Maximum *size* is: 32767 bytes if MAX_SQL_STRING_SIZE = EXTENDED 4000 bytes if MAX_SQL_STRING_SIZE = LEGACY
LONG RAW	Raw binary data of variable length up to 2 gigabytes
BLOB	A binary large object. Maximum size is (4 gigabytes - 1) * (DB_BLOCK_SIZE initialization parameter (8 TB to 128 TB)).
BFILE	Binary data stored in an external file (up to 4 GB)
ROWID	Base 64 string representing the unique address of a row in its table. This data type is primarily for values returned by the ROWID pseudocolumn

Guidelines

- A LONG column is not copied when a table is created using a subquery.
- A LONG column cannot be included in a GROUP BY or an ORDER BY clause.
- Only one LONG column can be used per table.
- No constraints can be defined on a LONG column.
- You might want to use a CLOB column rather than a LONG column.

Datetime Data Types

You can use several datetime data types:

Data Type	Description
TIMESTAMP	Date with fractional seconds
INTERVAL YEAR TO MONTH	Stored as an interval of years and months
INTERVAL DAY TO SECOND	Stored as an interval of days, hours, minutes, and seconds

Data Type	Description
TIMESTAMP	Enables storage of time as a date with fractional seconds. It stores the year, month, day, hour, minute, and the second value of the DATE data type, as well as the fractional seconds value. There are several variations of this data type such as WITH TIMEZONE and WITH LOCALTIMEZONE.
INTERVAL YEAR TO MONTH	Enables storage of time as an interval of years and months; used to represent the difference between two datetime values in which the only significant portions are the year and month
INTERVAL DAY TO SECOND	Enables storage of time as an interval of days, hours, minutes, and seconds; used to represent the precise difference between two datetime values

Note: These datetime data types are available with Oracle9*i* and later releases. The datetime data types are discussed in detail in the lesson titled "Managing Data in Different Time Zones" in the *Oracle Database: SQL Workshop II* course.

What is the Schema?

A schema is a collection of logical structures of data, or schema objects. A schema is owned by a database user and has the same name as that user. Each user owns a single schema. Schema objects can be created and manipulated with SQL.

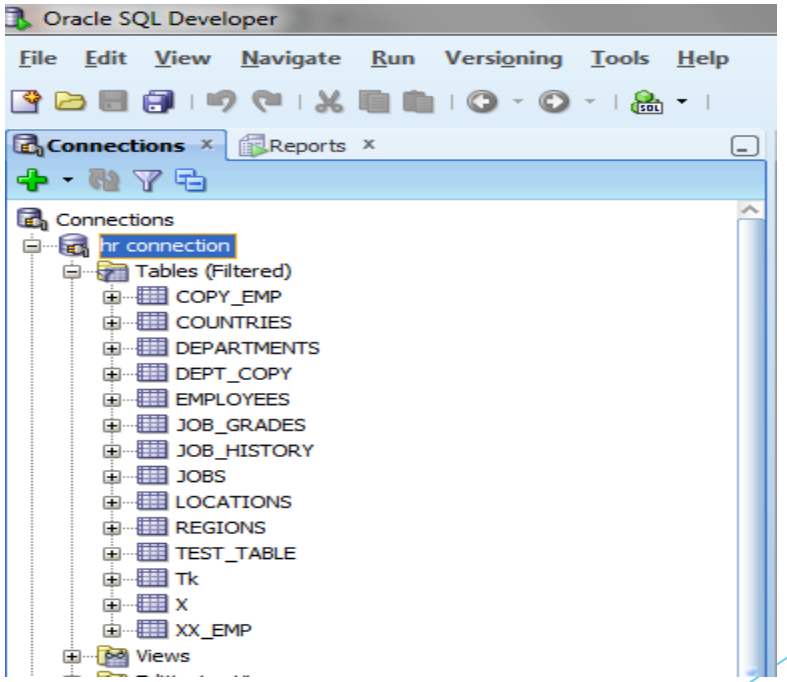
HR Schema / owend by user HR

Tables
---
emp
dpet
....

views
---
emp_v
dept_v
.....

indexes.....

procedures.....



CREATE TABLE Statement

- You must have:
  - The CREATE TABLE privilege
  - A storage area

```
CREATE TABLE [schema.]table
        (column datatype [DEFAULT expr][, ...]);
```

- You specify:
  - The table name
  - The column name, column data type, and column size

To create a table, a user must have the CREATE TABLE privilege and a storage area in which to create objects. The database administrator (DBA) uses data control language (DCL) statements to grant privileges to users.

In the syntax:

schema	Is the same as the owner's name
table	Is the name of the table
DEFAULT *expr*	Specifies a default value if a value is omitted in the INSERT statement
column	Is the name of the column
datatype	Is the column's data type and length

Note: The CREATE ANY TABLE privilege is needed to create a table in any schema other than the user's schema.

DEFAULT Option

- Specify a default value for a column during the CREATE table.

```
... hire_date DATE DEFAULT SYSDATE, ...
```

- Literal values, expressions, or SQL functions are legal values.
- Another column's name or a pseudocolumn are illegal values.
- The default data type must match the column data type.

```
CREATE TABLE hire_dates
        (id           NUMBER(8),
        hire_date DATE DEFAULT SYSDATE);
table HIRE_DATES created.
```

When you define a table, you can specify that a column should be given a default value by using the DEFAULT option. This option prevents null values from entering the columns when a row is inserted without a value for the column. The default value can be a literal, an expression, or a SQL function (such as SYSDATE or USER), but the value cannot be the name of another column or a pseudocolumn (such as NEXTVAL or CURRVAL). The default expression must match the data type of the column.

Consider the following examples:

```
INSERT INTO hire_dates values(45, NULL);
```

The preceding statement will insert the null value rather than the default value.

```
INSERT INTO hire_dates(id) values(35);
```

The preceding statement will insert SYSDATE for the HIRE_DATE column.

Creating Tables

- Create the table:

```
CREATE TABLE dept
        (deptno      NUMBER(2),
         dname       VARCHAR2(14),
         loc         VARCHAR2(13),
         create_date DATE DEFAULT SYSDATE);
table DEPT created.
```

- Confirm table creation:

```
DESCRIBE dept
```

```
DESCRIBE dept
Name        Null Type
----------- ---- ------------
DEPTNO           NUMBER(2)
DNAME            VARCHAR2(14)
LOC              VARCHAR2(13)
CREATE_DATE      DATE
```

Note: You can view the list of tables that you own by querying the data dictionary. For example:

```
select table_name from user_tables;
```

Using data dictionary views, you can also find information about other database objects such as views, indexes, and so on.

Including Constraints

The Oracle server uses constraints to prevent invalid data entry into tables.

You can use constraints to do the following:

- Enforce rules on the data in a table whenever a row is inserted, updated, or deleted from that table. The constraint must be satisfied for the operation to succeed.
- Prevent the dropping of a table if there are dependencies from other tables.
- Provide rules for Oracle tools, such as Oracle Developer.

Data Integrity Constraints

Constraint	Description
NOT NULL	Specifies that the column cannot contain a null value
UNIQUE	Specifies a column or combination of columns whose values must be unique for all rows in the table
PRIMARY KEY	Uniquely identifies each row of the table
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 condition that must be true

maxvlearn

Constraint Guidelines

- You can name a constraint or the Oracle server generates a name by using the `SYS_Cn` format.
- Create a constraint at either of the following times:
  - At the same time as the creation of the table
  - After the creation of the table
- Define a constraint at the column or table level.
- View a constraint in the data dictionary.

All constraints are stored in the data dictionary. Constraints are easy to reference if you give them a meaningful name. Constraint names must follow the standard object-naming rules, except that the name cannot be the same as another object owned by the same user. If you do not name your constraint, the Oracle server generates a name with the format `SYS_Cn`, where *n* is an integer so that the constraint name is unique.

Constraints can be defined at the time of table creation or after the creation of the table. You can define a constraint at the column or table level. Functionally, a table-level constraint is the same as a column-level constraint.

Defining Constraints

- Syntax:

```
CREATE TABLE [schema.]table
      (column datatype [DEFAULT expr]
      [column_constraint],
      ...
      [table_constraint][,...]);
```

- Column-level constraint syntax:

```
column [CONSTRAINT constraint_name] constraint_type,
```

- Table-level constraint syntax:

```
column,...
  [CONSTRAINT constraint_name] constraint_type
  (column, ...),
```

The slide gives the syntax for defining constraints when creating a table. You can create constraints at either the column level or table level. Constraints defined at the column level are included when the column is defined. Table-level constraints are defined at the end of the table definition, and must refer to the column or columns on which the constraint pertains in a set of parentheses. It is mainly the syntax that differentiates the two; otherwise, functionally, a column-level constraint is the same as a table-level constraint.

NOT NULL constraints must be defined at the column level.

Constraints that apply to more than one column must be defined at the table level.

- Example of a column-level constraint:

```
CREATE TABLE employees(
  employee_id  NUMBER(6)
    CONSTRAINT emp_emp_id_pk PRIMARY KEY,
  first_name   VARCHAR2(20),
  ...);
```

1

- Example of a table-level constraint:

```
CREATE TABLE employees(
  employee_id  NUMBER(6),
  first_name   VARCHAR2(20),
  ...
  job_id       VARCHAR2(10) NOT NULL,
  CONSTRAINT emp_emp_id_pk
    PRIMARY KEY (EMPLOYEE_ID));
```

2

Constraints are usually created at the same time as the table. Constraints can be added to a table after its creation and also be temporarily disabled.

Both examples in the slide create a primary key constraint on the `EMPLOYEE_ID` column of the `EMPLOYEES` table.

1. The first example uses the column-level syntax to define the constraint.
2. The second example uses the table-level syntax to define the constraint.

NOT NULL Constraint

Ensures that null values are not permitted for the column:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID	EMAIL	PHONE_NUMBER	HIRE_DATE
100	Steven	King	24000	(null)	90	SKING	515.123.4567	17-JUN-87
101	Neena	Kochhar	17000	(null)	90	NKOCHHAR	515.123.4568	21-SEP-89
102	Lex	De Haan	17000	(null)	90	LDEHAAN	515.123.4569	13-JAN-93
103	Alexander	Hunold	9000	(null)	60	AHUNOLD	590.423.4567	03-JAN-90
104	Bruce	Ernst	6000	(null)	60	BERNST	590.423.4568	21-MAY-91
107	Diana	Lorentz	4200	(null)	60	DLORENTZ	590.423.5567	07-FEB-99
124	Kevin	Mourgos	5800	(null)	50	KMOURGOS	650.123.5234	16-NOV-99
141	Trenna	Rajs	3500	(null)	50	TRAJS	650.121.8009	17-OCT-95
142	Curtis	Davies	3100	(null)	50	CDAVIES	650.121.2994	29-JAN-97
143	Randall	Matos	2600	(null)	50	RMATOS	650.121.2874	15-MAR-98
144	Peter	Vargas	2500	(null)	50	PVARGAS	650.121.2004	09-JUL-98
149	Eleni	Zlotkey	10500	0.2	80	EZLOTKEY	011.44.1344.429018	29-JAN-00
174	Ellen	Abel	11000	0.3	80	EABEL	011.44.1644.429267	11-MAY-96
176	Jonathon	Taylor	8600	0.2	80	JTAYLOR	011.44.1644.429265	24-MAR-98
178	Kimberely	Grant	7000	0.15	(null)	KGRANT	011.44.1644.429263	24-MAY-99
200	Jennifer	Whalen	4400	(null)	10	JWHALEN	515.123.4444	17-SEP-87
201	Michael	Hartstein	13000	(null)	20	MHARTSTE	515.123.5555	17-FEB-96
202	Pat	Fay	6000	(null)	20	PFAY	603.123.6666	17-AUG-97
205	Shelley	Higgins	12000	(null)	110	SHIGGINS	515.123.8080	07-JUN-94
206	William	Gietz	8300	(null)	110	WGIETZ	515.123.8181	07-JUN-94

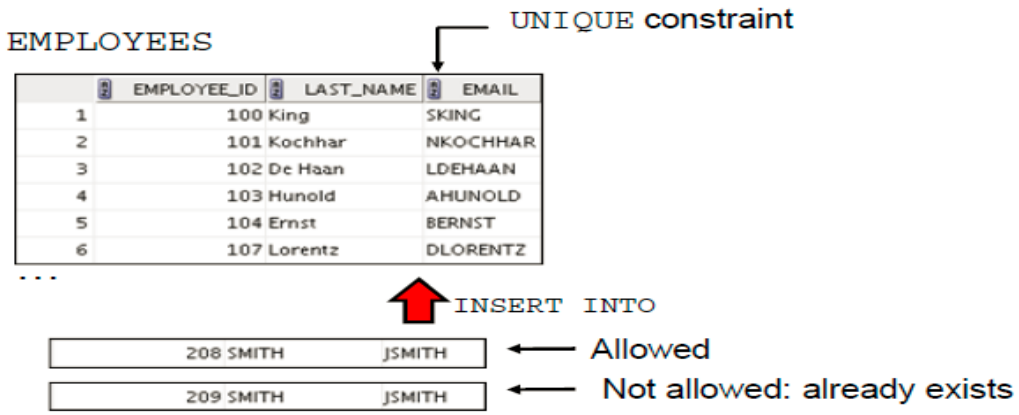
NOT NULL constraint (Primary Key enforces NOT NULL constraint.)

NOT NULL constraint

Absence of NOT NULL constraint (Any row can contain a null value for this column.)

The NOT NULL constraint ensures that the column contains no null values. Columns without the NOT NULL constraint can contain null values by default. NOT NULL constraints must be defined at the column level. In the EMPLOYEES table, the EMPLOYEE_ID column inherits a NOT NULL constraint because it is defined as a primary key.

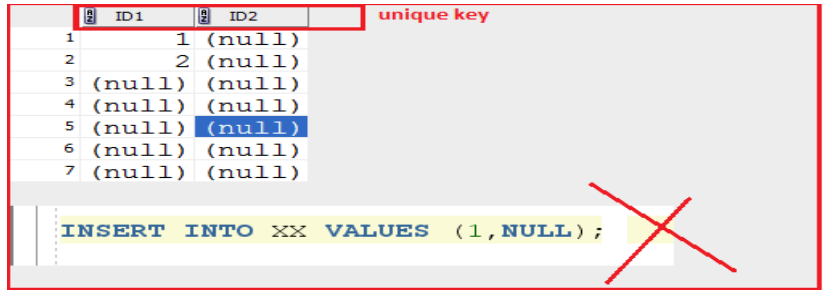
UNIQUE Constraint



A UNIQUE key integrity constraint requires that every value in a column or a set of columns (key) be unique—that is, no two rows of a table can have duplicate values in a specified column or a set of columns. The column (or set of columns) included in the definition of the UNIQUE key constraint is called the *unique key*. If the UNIQUE constraint comprises more than one column, that group of columns is called a *composite unique key*.

UNIQUE constraints enable the input of nulls unless you also define NOT NULL constraints for the same columns. In fact, any number of rows can include nulls for columns without the NOT NULL constraints because nulls are not considered equal to anything. A null in a column (or in all columns of a composite UNIQUE key) always satisfies a UNIQUE constraint.

Note: Because of the search mechanism for the UNIQUE constraints on more than one column, you cannot have identical values in the non-null columns of a partially null composite UNIQUE key constraint.



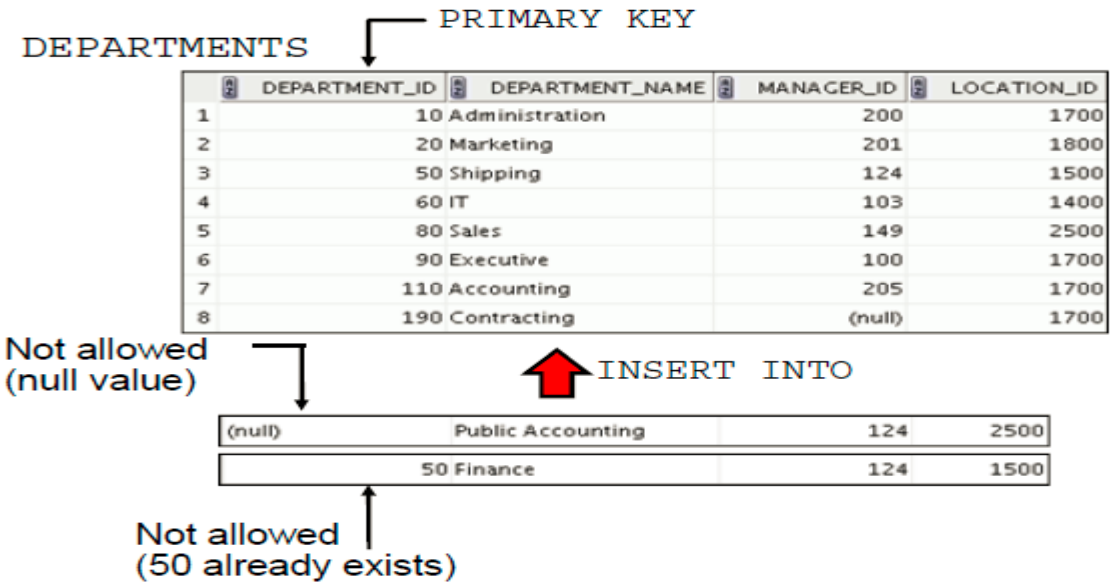
UNIQUE Constraint

Defined at either the table level or the column 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));
```

Note: The Oracle server enforces the `UNIQUE` constraint by implicitly creating a unique index on the unique key column or columns.

PRIMARY KEY Constraint



A PRIMARY KEY constraint creates a primary key for the table. Only one primary key can be created for each table. The PRIMARY KEY constraint is a column or a set of columns that uniquely identifies each row in a table. This constraint enforces the uniqueness of the column or column combination, and ensures that no column that is part of the primary key can contain a null value.

Note: Because uniqueness is part of the primary key constraint definition, the Oracle server enforces the uniqueness by implicitly creating a unique index on the primary key column or columns.

FOREIGN KEY Constraint

DEPARTMENTS

PRIMARY KEY

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500

...

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
4	205	Higgins	110
5	206	Gietz	110

...

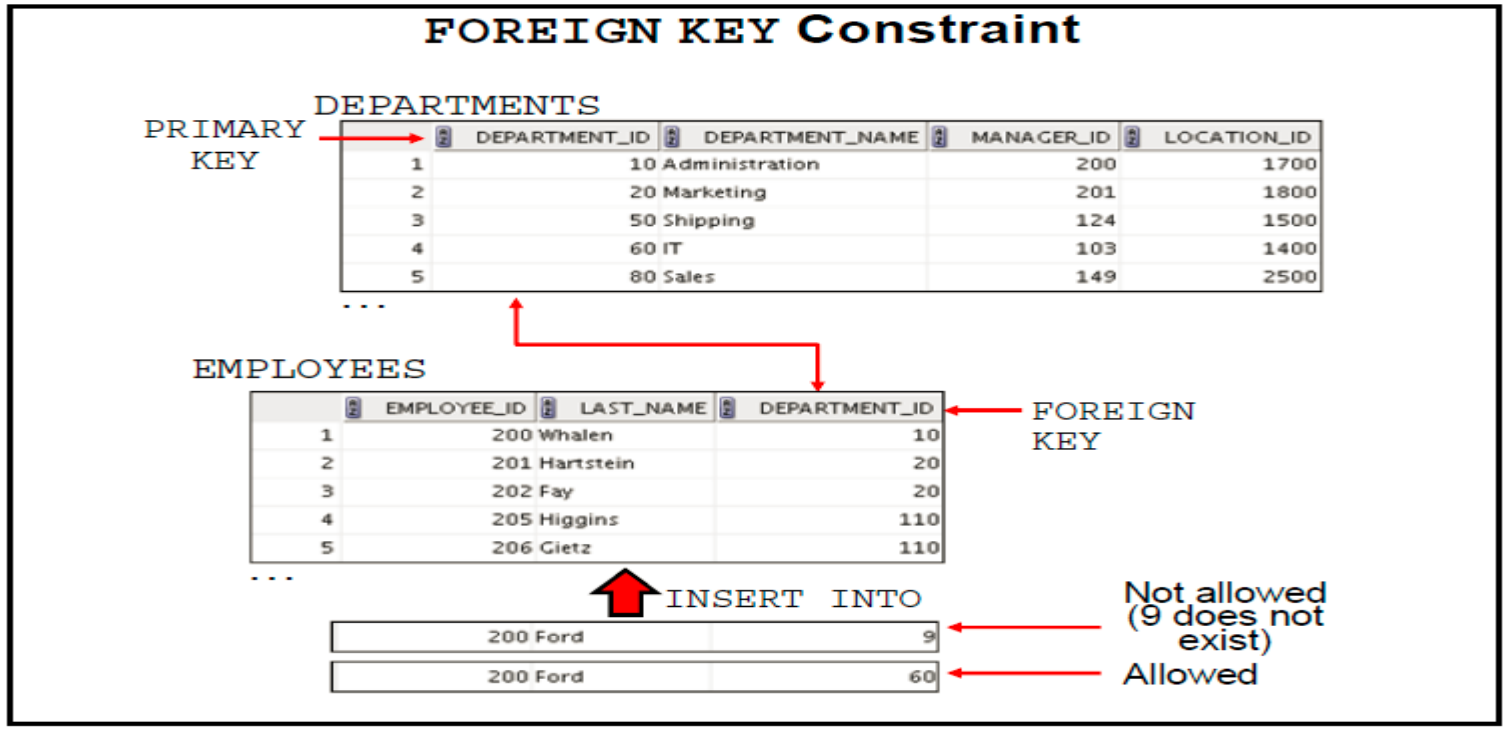
FOREIGN KEY

INSERT INTO

200	Ford	9
200	Ford	60

Not allowed (9 does not exist)

Allowed



The FOREIGN KEY (or referential integrity) constraint designates a column or a combination of columns as a foreign key, and establishes a relationship with a primary key or a unique key in the same table or a different table.

In the example in the slide, DEPARTMENT_ID has been defined as the foreign key in the EMPLOYEES table (dependent or child table); it references the DEPARTMENT_ID column of the DEPARTMENTS table (the referenced or parent table).

Guidelines

- A foreign key value must match an existing value in the parent table or be NULL.
- Foreign keys are based on data values and are purely logical, rather than physical, pointers.

FOREIGN KEY Constraint

Defined at either the table level or the column 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 constraints can be defined at the column or table constraint level. A composite foreign key must be created by using the table-level definition.

The example in the slide defines a FOREIGN KEY constraint on the DEPARTMENT_ID column of the EMPLOYEES table, using table-level syntax. The name of the constraint is EMP_DEPT_FK.

The foreign key can also be defined at the column level, provided that the constraint is based on a single column. The syntax differs in that the keywords FOREIGN KEY do not appear. For example:

```
CREATE TABLE employees
(...
department_id NUMBER(4) CONSTRAINT emp_deptid_fk
REFERENCES departments(department_id),
...
)
```

FOREIGN KEY Constraint: Keywords

The foreign key is defined in the child table and the table containing the referenced column is the parent table. The foreign key is defined using a combination of the following keywords:

- `FOREIGN KEY` is used to define the column in the child table at the table-constraint level.
- `REFERENCES` identifies the table and the column in the parent table.
- `ON DELETE CASCADE` indicates that when a row in the parent table is deleted, the dependent rows in the child table are also deleted.
- `ON DELETE SET NULL` indicates that when a row in the parent table is deleted, the foreign key values are set to null.

The default behavior is called the *restrict rule*, which disallows the update or deletion of referenced data.

Without the `ON DELETE CASCADE` or the `ON DELETE SET NULL` options, the row in the parent table cannot be deleted if it is referenced in the child table. And these keyword cannot be used in column-level syntax.

CHECK Constraint

- It defines a condition that each row must satisfy.
- The following expressions are not allowed:
  - References to CURRVAL, NEXTVAL, LEVEL, and ROWNUM pseudocolumns
  - Calls to SYSDATE, UID, USER, and USERENV functions
  - Queries that refer to other values in other rows

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

The CHECK constraint defines a condition that each row must satisfy. The condition can use the same constructs as the query conditions, with the following exceptions:

- References to the CURRVAL, NEXTVAL, LEVEL, and ROWNUM pseudocolumns
- Calls to SYSDATE, UID, USER, and USERENV functions
- Queries that refer to other values in other rows

A single column can have multiple CHECK constraints that refer to the column in its definition. There is no limit to the number of CHECK constraints that you can define on a column.

CHECK constraints can be defined at the column level or table level.

Violating Constraints

```
UPDATE employees
SET    department_id = 55
WHERE  department_id = 110;
```

```
Error starting at line 1 in command:
UPDATE employees
SET    department_id = 55
WHERE  department_id = 110
Error report:
SQL Error: ORA-02291: integrity constraint (ORA1.EMP_DEPT_FK) violated - parent key not found
02291. 00000 - "integrity constraint (%s.%s) violated - parent key not found"
*Cause:    A foreign key value has no matching primary key value.
*Action:   Delete the foreign key or add a matching primary key.
```

ORA-02291

You cannot delete a row that contains a primary key that is used as a foreign key in another table.

```
DELETE FROM departments
WHERE department_id = 60;
```

```
Error starting at line 1 in command:
DELETE FROM departments
WHERE department_id = 60
Error report:
SQL Error: ORA-02292: integrity constraint (ORA1.JHIST_DEPT_FK) violated - child record found
02292. 00000 - "integrity constraint (%s.%s) violated - child record found"
*Cause:    attempted to delete a parent key value that had a foreign
           dependency.
*Action:   delete dependencies first then parent or disable constraint.
```

ORA-02292

maxvlearn

Creating a Table Using a Subquery

- Create a table and insert rows by combining the CREATE TABLE statement and the AS *subquery* option.

```
CREATE TABLE table
          [(column, column...)]
AS subquery;
```

- Match the number of specified columns to the number of subquery columns.
- Define columns with column names and default values.

Guidelines

- The table is created with the specified column names, and the rows retrieved by the SELECT statement are inserted into the table.
- The column definition can contain only the column name and default value.
- If column specifications are given, the number of columns must equal the number of columns in the subquery SELECT list.
- If no column specifications are given, the column names of the table are the same as the column names in the subquery.
- The column data type definitions and the NOT NULL constraint are passed to the new table. Note that only the explicit NOT NULL constraint will be inherited. The PRIMARY KEY column will not pass the NOT NULL feature to the new column. Any other constraint rules are not passed to the new table. However, you can add constraints in the column definition.

Creating a Table Using a Subquery

When you use expression you should use
Column alias

```
CREATE TABLE  dept80
  AS
    SELECT  employee_id, last_name,
            salary*12 ANNSAL,
            hire_date
    FROM    employees
    WHERE   department_id = 80;
```

However, be sure to provide a column alias when selecting an expression. The expression SALARY*12 is given the alias ANNSAL. Without the alias, the following error is generated:

```
Error starting at line 1 in command:
CREATE TABLE  dept80
  AS
    SELECT  employee_id, last_name,
            salary*12,
            hire_date
    FROM    employees
    WHERE   department_id = 80
Error at Command Line:4 Column:18
Error report:
SQL Error: ORA-00998: must name this expression with a column alias
00998. 00000 -  "must name this expression with a column alias"
```

ALTER TABLE Statement

Use the ALTER TABLE statement to:

- Add a new column
- Modify an existing column definition
- Define a default value for the new column
- Drop a column
- Rename a column
- Change table to read-only status

Also we use alter table to:

- Add new constraint to the table
- Disable constraint in the table
- Drop constraint from the table

ALTER TABLE Statement

Use the ALTER TABLE statement to add, modify, or drop columns:

```
ALTER TABLE table
ADD         (column datatype [DEFAULT expr]
            [, column datatype]...);
```

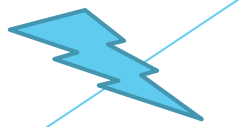
```
ALTER TABLE table
MODIFY      (column datatype [DEFAULT expr]
            [, column datatype]...);
```

```
ALTER TABLE table
DROP (column [, column] …);
```

Another syntax for drop is using DROP column

```
ALTER TABLE E_EMP
drop column CREATED_BY;
```

(one column at a time)



Adding a Column

- You use the ADD clause to add columns:

```
ALTER TABLE dept80
ADD         (job_id VARCHAR2(9));
```

```
table DEPT80 altered.
```

- The new column becomes the last column:

	EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE	JOB_ID
1	149	Zlotkey	10500	29-JAN-08	(null)
2	174	Abel	11000	11-MAY-04	(null)
3	176	Taylor	8600	24-MAR-06	(null)

Guidelines for Adding a Column

- You cannot specify where the column is to appear. The new column becomes the last column.

The example in the slide adds a column named JOB_ID to the DEPT80 table. The JOB_ID column becomes the last column in the table.

Note: If a table already contains rows when a column is added, the new column is initially null or takes the default value for all the rows. You can add a mandatory NOT NULL column to a table that contains data in the other columns only if you specify a default value. You can add a NOT NULL column to an empty table without the default value.

Modifying a Column

- You can change a column's data type, size, and default value.

```
ALTER TABLE dept80
MODIFY      (last_name VARCHAR2(30));
```

```
table DEPT80 altered.
```

- A change to the default value affects only subsequent insertions to the table.

You can modify a column definition by using the ALTER TABLE statement with the MODIFY clause. Column modification can include changes to a column's data type, size, and default value.

Guidelines

- You can increase the width or precision of a numeric column.
- You can increase the width of character columns.
- You can decrease the width of a column if:
  - The column contains only null values
  - The table has no rows
  - The decrease in column width is not less than the existing values in that column
- You can change the data type if the column contains only null values. The exception to this is CHAR-to-VARCHAR2 conversions, which can be done with data in the columns.
- You can convert a CHAR column to the VARCHAR2 data type or convert a VARCHAR2 column to the CHAR data type only if the column contains null values or if you do not change the size.
- A change to the default value of a column affects only subsequent insertions to the table.

Dropping a Column

Use the DROP COLUMN clause to drop columns that you no longer need from the table:

```
ALTER TABLE  dept80
DROP (job_id);
```

```
table DEPT80 altered.
```

	EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE
1	149	Zlotkey	10500	29-JAN-08
2	174	Abel	11000	11-MAY-04
3	176	Taylor	8600	24-MAR-06

Guidelines

- The column may or may not contain data.
- Using the ALTER TABLE DROP COLUMN statement, only one column can be dropped at a time.
- The table must have at least one column remaining in it after it is altered.
- After a column is dropped, it cannot be recovered.
- A primary key that is referenced by another column cannot be dropped, unless the cascade option is added.
- Dropping a column can take a while if the column has a large number of values. In this case, it may be better to set it to be unused and drop it when there are fewer users on the system to avoid extended locks.

Note: Certain columns can never be dropped, such as columns that form part of the partitioning key of a partitioned table or columns that form part of the PRIMARY KEY of an index-organized table. For more information about index-organized tables and partitioned tables, refer to *Oracle Database Concepts* and *Oracle Database Administrator's Guide*.

SET UNUSED Option

- You use the SET UNUSED option to mark one or more columns as unused.
- You use the DROP UNUSED COLUMNS option to remove the columns that are marked as unused.
- You can specify the ONLINE keyword to indicate that DML operations on the table will be allowed while marking the column or columns UNUSED.

```
ALTER TABLE <table name>
SET   UNUSED(<column name> [ , <column_name>]);
OR
ALTER TABLE  <table_name>
SET   UNUSED COLUMN <column_name> [,<column_name>];
```

```
ALTER TABLE <table_name>
DROP  UNUSED COLUMNS;
```

The SET UNUSED option marks one or more columns as unused so that they can be dropped when the demand on system resources is lower. Specifying this clause does not actually remove the target columns from each row in the table (that is, it does not restore the disk space used by these columns). Therefore, the response time is faster than if you executed the DROP clause. Unused columns are treated as if they were dropped, even though their column data remains in the table's rows. After a column has been marked as unused, you have no access to that column. A SELECT * query will not retrieve data from unused columns. In addition, the names and types of columns marked as unused will not be displayed during a DESCRIBE statement, and you can add to the table a new column with the same name as an unused column. The SET UNUSED information is stored in the USER_UNUSED_COL_TABS dictionary view.

You can specify the ONLINE keyword to indicate that DML operations on the table will be allowed while marking the column or columns UNUSED. The code example shows the use of SET UNUSED COLUMN that sets a column unused forever using the ONLINE keyword.

```
ALTER TABLE dept80 SET UNUSED(hire_date)ONLINE;
```

Note: The guidelines for setting a column to be UNUSED are similar to those for dropping a column.

DROP UNUSED COLUMNS Option

DROP UNUSED COLUMNS removes from the table all columns that are currently marked as unused. You can use this statement when you want to reclaim the extra disk space from the unused columns in the table. If the table contains no unused columns, the statement returns with no errors.

```
ALTER TABLE  dept80
SET   UNUSED (last_name);
table DEPT80 altered.

ALTER TABLE  dept80
DROP  UNUSED COLUMNS;
table DEPT80 altered.
```

Note: You cannot specify the ONLINE clause when marking a column with a DEFERRABLE constraint as unused. A subsequent DROP UNUSED COLUMNS will physically remove all unused columns from a table, similar to a DROP COLUMN.

Read-Only Tables

You can use the ALTER TABLE syntax to:

- Put a table in read-only mode, which prevents DDL or DML changes during table maintenance
- Put the table back into read/write mode

```
ALTER TABLE employees READ ONLY;

-- perform table maintenance and then
-- return table back to read/write mode

ALTER TABLE employees READ WRITE;
```

With Oracle Database 11*g*, you can specify READ ONLY to place a table in read-only mode. When the table is in READ ONLY mode, you cannot issue any DML statements that affect the table or any SELECT . . . FOR UPDATE statements. You can issue DDL statements as long as they do not modify any data in the table. Operations on indexes associated with the table are allowed when the table is in READ ONLY mode.

Specify READ/WRITE to return a read-only table to read/write mode.

Note: You can drop a table that is in READ ONLY mode. The DROP command is executed only in the data dictionary, so access to the table contents is not required. The space used by the table will not be reclaimed until the tablespace is made read/write again, and then the required changes can be made to the block segment headers, and so on.

Dropping a Table

- Moves a table to the recycle bin
- Removes the table and all its data entirely if the PURGE clause is specified
- Invalidates dependent objects and removes object privileges on the table

```
DROP TABLE dept80;

table DEPT80 dropped.
```

The DROP TABLE statement moves a table to the recycle bin or removes the table and all its data from the database entirely. Unless you specify the PURGE clause, the DROP TABLE statement does not result in space being released back to the tablespace for use by other objects, and the space continues to count toward the user's space quota. Dropping a table invalidates the dependent objects and removes object privileges on the table.

When you drop a table, the database loses all the data in the table and all the indexes associated with it.

Syntax

```
DROP TABLE table [PURGE]
```

In the syntax, *table* is the name of the table.

Guidelines

- All data is deleted from the table.
- Any views and synonyms remain, but are invalid.
- Any pending transactions are committed.
- Only the creator of the table or a user with the DROP ANY TABLE privilege can remove a table.

Note: Use the FLASHBACK TABLE statement to restore a dropped table from the recycle bin. This is discussed in detail in the course titled *Oracle Database: SQL Workshop II*.

Oracle Database 10g introduces a new feature for dropping tables.
When you drop a table, the database does not immediately release the space associated with the table. Rather, the database renames the table and places it in a recycle bin, where it can later be recovered with the FLASHBACK TABLE statement

If you want to immediately release the space associated with the table at the time you issue the DROP TABLE statement, then include the PURGE clause as follows. DROP TABLE employees PURGE; Specify PURGE only if you want to drop the table and release the space associated with it in a single step. If you specify PURGE, then the database does not place the table and its dependent objects into the recycle bin. NOTE: You cannot roll back a DROP TABLE statement with the PURGE clause, and you cannot recover the table if you drop it with the PURGE clause. This feature was not available in earlier releases.

https://docs.oracle.com/cd/B28359_01/server.111/b28286/statements_9012.htm

Quiz

To do which three of the following can you use constraints?

a. Enforce rules on the data in a table whenever a row is inserted, updated, or deleted.
b. Prevent the dropping of a table.
c. Prevent the creation of a table.
d. Prevent the creation of data in a table.

Answer: a, b, d

Thank You