Database Objects

Object	Description
Table	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

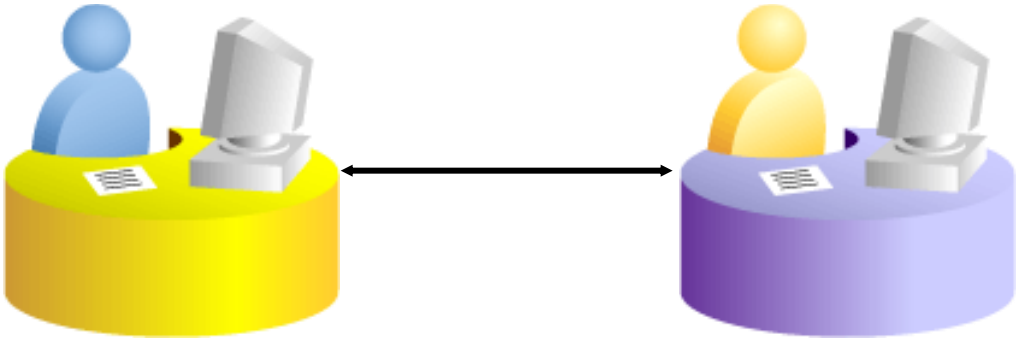
Naming Rules

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

Referencing Another User's Tables

- Tables belonging to other users are not in the user's schema.
- You should use the owner's name as a prefix to those tables.



USERA

```
SELECT *
FROM userB.employees;
```

USERB

```
SELECT *
FROM userA.employees;
```

DEFAULT Option

- Specify a default value for a column during an insert.

```
... 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);
CREATE TABLE succeeded.
```

Creating Tables

- Create the table:

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

```
CREATE TABLE succeeded.
```

- Confirm table creation:

```
DESCRIBE dept
```

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

4 rows selected
```

Data Types

Data Type	Description
VARCHAR2(*size*)	Variable-length character data
CHAR(*size*)	Fixed-length character data
NUMBER(*p*,*s*)	Variable-length numeric data
DATE	Date and time values
LONG	Variable-length character data (up to 2 GB)
CLOB	Character data (up to 4 GB)
RAW and LONG RAW	Raw binary data
BLOB	Binary data (up to 4 GB)
BFILE	Binary data stored in an external file (up to 4 GB)
ROWID	A base-64 number system representing the unique address of a row in its table

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

Including Constraints

- Constraints enforce rules at the table level.
- Constraints prevent the deletion of a table if there are dependencies.
- The following constraint types are valid:
  - `NOT NULL`
  - `UNIQUE`
  - `PRIMARY KEY`
  - `FOREIGN KEY`
  - `CHECK`

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, ...),
```

Defining Constraints

- 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

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));
```

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` Constraint: Keywords

- `FOREIGN KEY`: Defines the column in the child table at the table-constraint level
- `REFERENCES`: Identifies the table and column in the parent table
- `ON DELETE CASCADE`: Deletes the dependent rows in the child table when a row in the parent table is deleted
- `ON DELETE SET NULL`: Converts dependent foreign key values to null

CHECK Constraint

- 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),...
```

CREATE TABLE: Example

```
CREATE TABLE employees
    ( employee_id    NUMBER(6)
      CONSTRAINT     emp_employee_id   PRIMARY KEY
    , first_name     VARCHAR2(20)
    , last_name      VARCHAR2(25)
      CONSTRAINT     emp_last_name_nn  NOT NULL
    , email          VARCHAR2(25)
      CONSTRAINT     emp_email_nn      NOT NULL
      CONSTRAINT     emp_email_uk      UNIQUE
    , phone_number   VARCHAR2(20)
    , hire_date      DATE
      CONSTRAINT     emp_hire_date_nn  NOT NULL
    , job_id         VARCHAR2(10)
      CONSTRAINT     emp_job_nn        NOT NULL
    , salary         NUMBER(8,2)
      CONSTRAINT     emp_salary_ck     CHECK (salary>0)
    , commission_pct NUMBER(2,2)
    , manager_id     NUMBER(6)
         CONSTRAINT emp_manager_fk REFERENCES
          employees (employee_id)
    , department_id  NUMBER(4)
      CONSTRAINT     emp_dept_fk       REFERENCES
         departments (department_id));
```

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.
```

Department 55 does not exist.

Violating Constraints

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.
```

Creating a Table Using a Subquery

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

```
CREATE TABLE succeeded.
```

```
DESCRIBE dept80
```

```
Name                          Null     Type
----------------------------- -------- -------------
EMPLOYEE_ID                            NUMBER(6)
LAST_NAME                     NOT NULL VARCHAR2(25)
ANNSAL                                 NUMBER
HIRE_DATE                     NOT NULL DATE
```

`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

Read-Only Tables

You can use the `ALTER TABLE` syntax to:

- Put a table into 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;
```

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

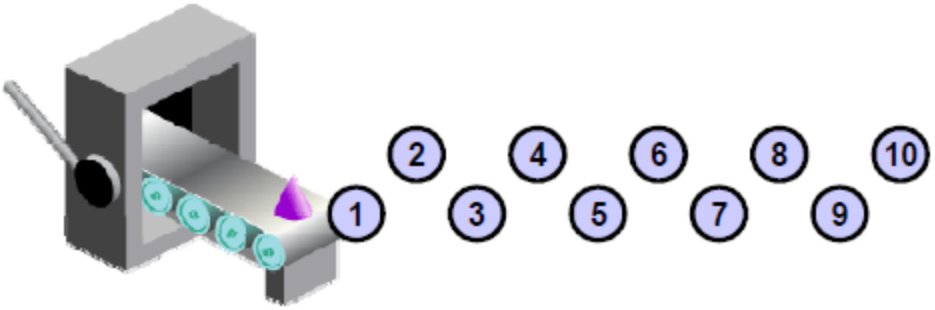
```
DROP TABLE dept80;
```

```
DROP TABLE dept80 succeeded.
```

Sequences

A sequence:

- Can automatically generate unique numbers
- Is a shareable object
- Can be used to create a primary key value
- Replaces application code
- Speeds up the efficiency of accessing sequence values when cached in memory



CREATE SEQUENCE Statement: Syntax

Define a sequence to generate sequential numbers automatically:

```
CREATE SEQUENCE [ schema. ] sequence
  [ { INCREMENT BY | START WITH } integer
  | { MAXVALUE integer | NOMAXVALUE }
  | { MINVALUE integer | NOMINVALUE }
  | { CYCLE | NOCYCLE }
  | { CACHE integer | NOCACHE }
  | { ORDER | NOORDER }
];
```

Creating a Sequence

- Create a sequence named `DEPT_DEPTID_SEQ` to be used for the primary key of the `DEPARTMENTS` table.
- Do not use the `CYCLE` option.

```
CREATE SEQUENCE dept_deptid_seq
                INCREMENT BY 10
                START WITH 280
                MAXVALUE 9999
                NOCACHE
                NOCYCLE;
```

```
sequence DEPT_DEPTID_SEQ created.
```

Using a Sequence

- Insert a new department named “Support” in location ID 2500:

```
INSERT INTO departments(department_id,
            department_name, location_id)
VALUES      (dept_deptid_seq.NEXTVAL,
            'Support', 2500);
1 rows inserted
```

- View the current value for the `DEPT_DEPTID_SEQ` sequence:

```
SELECT   dept_deptid_seq.CURRVAL
FROM     dual;
```

SQL Column defaulting using a Sequence

- SQL syntax for column defaults allow `<sequence>.nextval`, `<sequence>.currval` as a SQL column defaulting expression for numeric columns, where `<sequence>` is an Oracle database sequence.
- The `DEFAULT` expression can include the sequence pseudocolumns `CURRVAL` and `NEXTVAL`, as long as the sequence exists and you have the privileges necessary to access it.

```
CREATE SEQUENCE s1 START WITH 1;
CREATE TABLE emp (a1 NUMBER DEFAULT s1.NEXTVAL NOT
NULL, a2 VARCHAR2(10));
INSERT INTO emp (a2) VALUES ('john');
INSERT INTO emp (a2) VALUES ('mark');
SELECT * FROM emp;
```

```
sequence S1 created.
table EMP created.
1 rows inserted.
1 rows inserted.
A1 A2
-- ----------
 1 john
 2 mark
```

Modifying a Sequence

Change the increment value, maximum value, minimum value, cycle option, or cache option:

```
ALTER SEQUENCE dept_deptid_seq
               INCREMENT BY 20
               MAXVALUE 999999
               NOCACHE
               NOCYCLE;
```

```
sequence DEPT_DEPTID_SEQ altered.
```

Synonyms

A synonym

- Is a database object
- Can be created to give an alternative name to a table or to an other database object
- Requires no storage other than its definition in the data dictionary
- Is useful for hiding the identity and location of an underlying schema object

Creating and Removing Synonyms

- Create a shortened name for the DEPT_SUM_VU view:

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

```
synonym D_SUM created.
```

- Drop a synonym:

```
DROP SYNONYM d_sum;
```

```
synonym D_SUM dropped.
```

Synonym Information

```
DESCRIBE user_synonyms
```

```
DESCRIBE user_synonyms
Name         Null     Type
------------ -------- -------------
SYNONYM_NAME NOT NULL VARCHAR2(128)
TABLE_OWNER           VARCHAR2(128)
TABLE_NAME   NOT NULL VARCHAR2(128)
DB_LINK               VARCHAR2(128)
```

```
SELECT *
FROM   user_synonyms;
```

	SYNONYM_NAME	TABLE_OWNER	TABLE_NAME	DB_LINK
1	D_SUM	ORA21	DEPT_SUM_VU	(null)

Indexes

An index:

- Is a schema object
- Can be used by the Oracle Server to speed up the retrieval of rows by using a pointer
- Can reduce disk input/output (I/O) by using a rapid path access method to locate data quickly
- Is dependent on the table that it indexes
- Is used and maintained automatically by the Oracle Server

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: You can create unique or nonunique index on columns to speed up access to the rows.

Creating an Index

- Create an index on one or more columns:

```
CREATE [UNIQUE][BITMAP]INDEX index
ON table (column[, column]...);
```

- Improve the speed of query access to the LAST_NAME column in the EMPLOYEES table:

```
CREATE INDEX emp_last_name_idx
ON employees(last_name);
```

```
index EMP_LAST_NAME_IDX created.
```

CREATE INDEX with the CREATE TABLE Statement

```
CREATE TABLE NEW_EMP
(employee_id NUMBER(6)
             PRIMARY KEY USING INDEX
             (CREATE INDEX emp_id_idx ON
             NEW_EMP(employee_id)),
first_name   VARCHAR2(20),
last_name    VARCHAR2(25));
```

```
table NEW_EMP created.
```

```
SELECT INDEX_NAME, TABLE_NAME
FROM   USER_INDEXES
WHERE  TABLE_NAME = 'NEW_EMP';
```

	INDEX_NAME	TABLE_NAME
1	EMP_ID_IDX	NEW_EMP

Index Creation Guidelines

Create an index when:	
✓	A column contains a wide range of values
✓	A column contains a large number of null values
✓	One or more columns are frequently used together in a `WHERE` clause or a join condition
✓	The table is large and most queries are expected to retrieve less than 2% to 4% of the rows in the table
Do not create an index when:	
✗	The columns are not often used as a condition in the query
✗	The table is small or most queries are expected to retrieve more than 2% to 4% of the rows in the table
✗	The table is updated frequently
✗	The indexed columns are referenced as part of an expression

Example of Creating Multiple Indexes on the Same Set Of Columns

```
CREATE INDEX emp_id_name_ix1
ON employees(employee_id, first_name);
```

```
index EMP_ID_NAME_IX1 created.
```

```
ALTER INDEX emp_id_name_ix1 INVISIBLE;
```

```
index EMP_ID_NAME_IX1 altered.
```

```
CREATE BITMAP INDEX emp_id_name_ix2
ON employees(employee_id, first_name);
```

```
bitmap index EMP_ID_NAME_IX2 created.
```

Removing an Index

- Remove an index from the data dictionary by using the `DROP INDEX` command:

```
DROP INDEX index;
```

- Remove the `emp_last_name_idx` index from the data dictionary:

```
DROP INDEX emp_last_name_idx;

index EMP_LAST_NAME_IDX dropped.
```

- To drop an index, you must be the owner of the index or have the `DROP ANY INDEX` privilege.

You can drop an index using the `ONLINE` keyword.

```
DROP INDEX emp_indx ONLINE;
```

`ONLINE`: Specify `ONLINE` to indicate that DML operations on the table are allowed while dropping the index.

What Is a View?

EMPLOYEES table

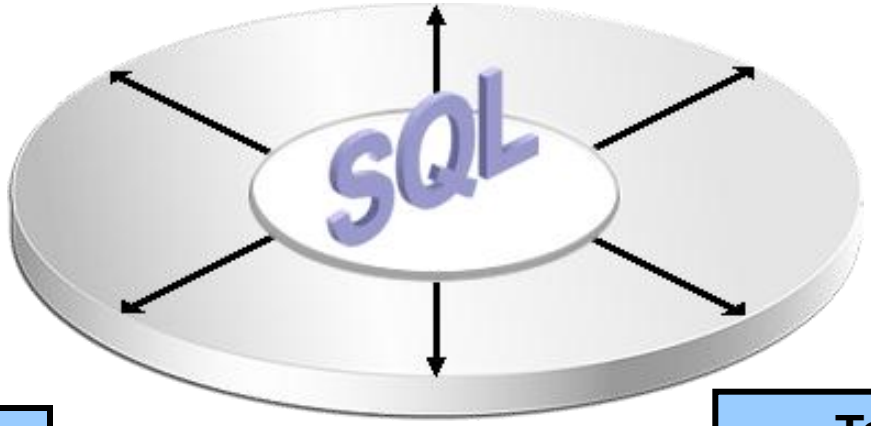
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000
104	Br					ROG	6000
1						ROG	4200
						ACCOUNT	6900
						MAN	5800
						CLERK	3500
						CLERK	3100
						CLERK	2600
						CLERK	2500
						SA_MAN	10500
					6	SA_REP	11000
					AR-98	SA_REP	8600
					4-MAY-99	SA_REP	7000
					17-SEP-87	AD_ASST	4400
					17-FEB-96	MK_MAN	13000
				6666	17-AUG-97	MK_REP	6000
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	8300

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
100	Steven	King	24000
101	Neena	Kochhar	17000
102	Lex	De Haan	17000
103	Alexander	Hunold	9000
104	Bruce	Ernst	6000

Advantages of Views

To restrict data access

To make complex queries easy



To provide data independence

To present different views of the same data

Simple Views and Complex Views

Feature	Simple Views	Complex Views
Number of tables	One	One or more
Contain functions	No	Yes
Contain groups of data	No	Yes
DML operations through a view	Yes	Not always

Creating a View

- Create the `EMPVU80` view, which contains details of the employees in department 80:

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

- Describe the structure of the view by using the SQL*Plus `DESCRIBE` command:

```
DESCRIBE empvu80
```

Creating a View

- Create a view by using column aliases in the subquery:

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

- Select the columns from this view by the given alias names.

Retrieving Data from a View

```
SELECT *
FROM   salvu50;
```

	ID_NUMBER	NAME	ANN_SALARY
1	124	Mourgos	69600
2	141	Rajs	42000
3	142	Davies	37200
4	143	Matos	31200
5	144	Vargas	30000

Modifying a View

- Modify the `EMPVU80` view by using a `CREATE OR REPLACE VIEW` clause. Add an alias for each column name:

```
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;
CREATE OR REPLACE VIEW succeeded.
```

- Column aliases in the `CREATE OR REPLACE VIEW` clause are listed in the same order as the columns in the subquery.

Creating a Complex View

Create a complex view that contains group functions to display values from two tables:

```
CREATE OR REPLACE VIEW dept_sum_vu
  (name, minsal, maxsal, avgsal)
AS SELECT   d.department_name, MIN(e.salary),
            MAX(e.salary),AVG(e.salary)
   FROM     employees e JOIN departments d
   ON       (e.department_id = d.department_id)
   GROUP BY d.department_name;
CREATE OR REPLACE VIEW succeeded.
```

Using the WITH CHECK OPTION Clause

- You can ensure that DML operations performed on the view stay in the domain of the view by using the WITH CHECK OPTION clause:

```
CREATE OR REPLACE VIEW empvu20
AS SELECT     *
   FROM      employees
   WHERE     department_id = 20
   WITH CHECK OPTION CONSTRAINT empvu20_ck ;
```

```
CREATE OR REPLACE VIEW succeeded.
```

- Any attempt to INSERT a row with a department_id other than 20, or to UPDATE the department number for any row in the view fails because it violates the WITH CHECK OPTION constraint.

Denying DML Operations

```
CREATE OR REPLACE VIEW empvu10
    (employee_number, employee_name, job_title)
AS SELECT    employee_id, last_name, job_id
   FROM     employees
   WHERE    department_id = 10
   WITH READ ONLY ;
```

```
CREATE OR REPLACE VIEW succeeded.
```

Removing a View

You can remove a view without losing data because a view is based on underlying tables in the database.

```
DROP VIEW view;
```

```
DROP VIEW empvu80;
```

```
DROP VIEW empvu80 succeeded.
```

`ALTER TABLE` Statement

Use the `ALTER TABLE` statement to:

- Add a new column
- Modify an existing column
- Define a default value for the new column
- Drop a column

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] …);
```

Adding a Column

- You use the `ADD` clause to add columns:

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

```
ALTER TABLE dept80 succeeded.
```

- The new column becomes the last column:

	EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE	JOB_ID
1	145	Russell	14000	01-OCT-96	(null)
2	146	Partners	13500	05-JAN-97	(null)
3	147	Errazuriz	12000	10-MAR-97	(null)
4	148	Cambrault	11000	15-OCT-99	(null)
5	149	Zlotkey	10500	29-JAN-00	(null)

Modifying a Column

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

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

```
ALTER TABLE dept80 succeeded.
```

- A change to the default value 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 COLUMN  job_id;
```

```
ALTER TABLE dept80 succeeded.
```

	EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE
1	145	Russell	14000	01-OCT-96
2	146	Partners	13500	05-JAN-97
3	147	Errazuriz	12000	10-MAR-97
4	148	Cambrault	11000	15-OCT-99
5	149	Zlotkey	10500	29-JAN-00

ON DELETE Clause

- Use the `ON DELETE CASCADE` clause to delete child rows when a parent key is deleted:

```
ALTER TABLE emp2 ADD CONSTRAINT emp_dt_fk
FOREIGN KEY (Department_id)
REFERENCES departments(department_id) ON DELETE CASCADE;
```

```
ALTER TABLE Emp2 succeeded.
```

- Use the `ON DELETE SET NULL` clause to set the child rows value to null when a parent key is deleted:

```
ALTER TABLE emp2 ADD CONSTRAINT emp_dt_fk
FOREIGN KEY (Department_id)
REFERENCES departments(department_id) ON DELETE SET NULL;
```

```
ALTER TABLE Emp2 succeeded.
```

Dropping a Constraint

- Remove the manager constraint from the `EMP2` table:

```
ALTER TABLE emp2
DROP CONSTRAINT emp_mgr_fk;
```

```
ALTER TABLE Emp2 succeeded.
```

Disabling Constraints

- **Execute the `DISABLE` clause of the `ALTER TABLE` statement to deactivate an integrity constraint.**
- **Apply the `CASCADE` option to disable dependent integrity constraints.**

```
ALTER TABLE emp2
DISABLE CONSTRAINT emp_dt_fk;
```

```
ALTER TABLE Emp2 succeeded.
```

Cascading Constraints

Example:

```
ALTER TABLE emp2
DROP COLUMN employee_id CASCADE CONSTRAINTS;
```

```
ALTER TABLE Emp2 succeeded.
```

```
ALTER TABLE test1
DROP (col1_pk, col2_fk, col1) CASCADE CONSTRAINTS;
```

```
ALTER TABLE test1 succeeded.
```

Using the FLASHBACK TABLE Statement

```
DROP TABLE emp2;
```

```
DROP TABLE emp2 succeeded.
```

```
SELECT original_name, operation, droptime FROM
recyclebin;
```

ORIGINAL_NAME	OPERATION	DROPTIME
EMP2	DROP	2009-05-20:18:00:39

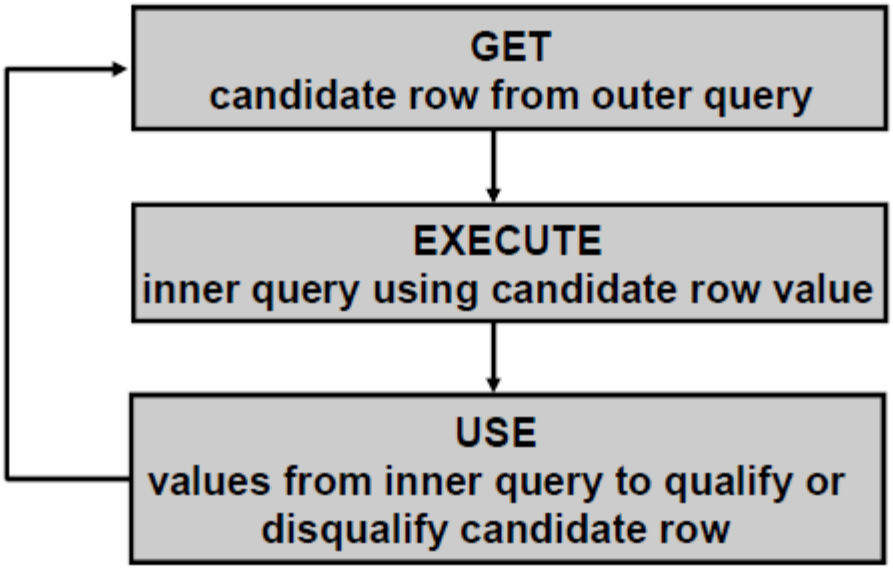
...

```
FLASHBACK TABLE emp2 TO BEFORE DROP;
```

```
FLASHBACK TABLE succeeded.
```

Correlated Subqueries

Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query.



Using Correlated Subqueries

Find all employees who earn more than the average salary in their department.

```
SELECT last_name, salary, department_id
FROM   employees outer_table
WHERE  salary >
       (SELECT AVG(salary)
        FROM   employees inner_table
        WHERE inner_table.department_id =
        outer_table.department_id);
```

Each time a row from the outer query is processed, the inner query is evaluated.

New Features
Truncate Table

- When you truncate a parent table with child tables, you get:

```
ORA-02266: unique/primary keys in table
referenced by enabled foreign keys
```

- In Oracle 12c, you can use:

```
truncate table <Parent> cascade;
```

- Must have defined the FK as ON DELETE CASCADE.
- Otherwise `ORA-14705: unique or primary keys referenced by enabled foreign keys in table will result`

Enhanced DDL capabilities using ONLINE

- DDLs do not need lock.
- DML continues as usual
  - drop index i1 online
  - alter index i1 unusable online
  - alter table t1 set unused columns online
  - alter table t1 drop column c1 online

Top N Queries

- First 10, second 10 rows, etc.

```
select … from (select … from … order by …) where rownum <= 10
```

- 12c way:

```
select *
from sales_fact
order by year, week, country, region, product
fetch first 10 rows only;
```

- Next 10 rows
  - `offset 10 rows fetch first 10 rows only`
  - `offset 10 rows fetch first 0.1 percent rows only`
  - `offset 10 rows fetch first 0.1 percent rows with ties`

Invisible Columns

```
SQL> create table t4 (col1 number, col2 number invisible);
SQL> desc t4
Name  Null? Type
----- ----- -----
COL1      NUMBER
SQL> insert into t4 values (1);
1 row created.
SQL> select * from t4;
      COL1
----------
         1
SQL> select col1, col2 from t4;
      COL1       COL2
---------- ----------
         1
SQL> insert into t4 (col1,col2) values (2,2);
1 row created.
```

```
SQL> set colinvisible on
SQL> desc t4
 Name                 Null?    Type
 ---------------- -------- ------------
 COL1                          NUMBER
 COL2 (INVISIBLE)              NUMBER

SQL> create index in_t4 on t4(col2);
Index created.
```

DEFAULT Values

```
SQL> create table t5 (col1 number, col2 number default on null 0);
Table created.

SQL> desc t5
 Name                    Null?    Type
 ----------------- -------- ------
 COL1                             NUMBER
 COL2                     NOT NULL NUMBER

SQL> insert into t5 values (1, null);
SQL> insert into t5 values (2,2);

SQL> select * from t5;
      COL1       COL2
---------- ----------
         1          0
         2          2
```

Defva

Identity columns

```
SQL> create table t9 (col1 number, col2 number generated by default as identity);
SQL> insert into t9 values (9,9);
SQL> insert into t9 values (10,default);
SQL> insert into t9 (col1) values (11);
SQL> select * from t9;
      COL1       COL2
---------- ----------
         9          9
        10          2
```