Advanced Database Systems (22MCA102)

BY

Dr. Anantha Murthy
Associate Professor
Dept. of MCA
NMAM.I.T, Nitte

Unit 2

Strcutured Query Langauge (SQL)

SQL - Overview

- SQL is a language to operate databases; it includes database creation, deletion, fetching rows, modifying rows, etc.
- SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.
- SQL is the standard language for Relational Database System.
- All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

What is RDBMS?

- RDBMS stands for Relational Database Management System.
- RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
- A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd in 1970.
- In RDBMs, the data is represented in the form of tables. i.e. in the form of rows and columns .

How the data is arranged in RDBMS?

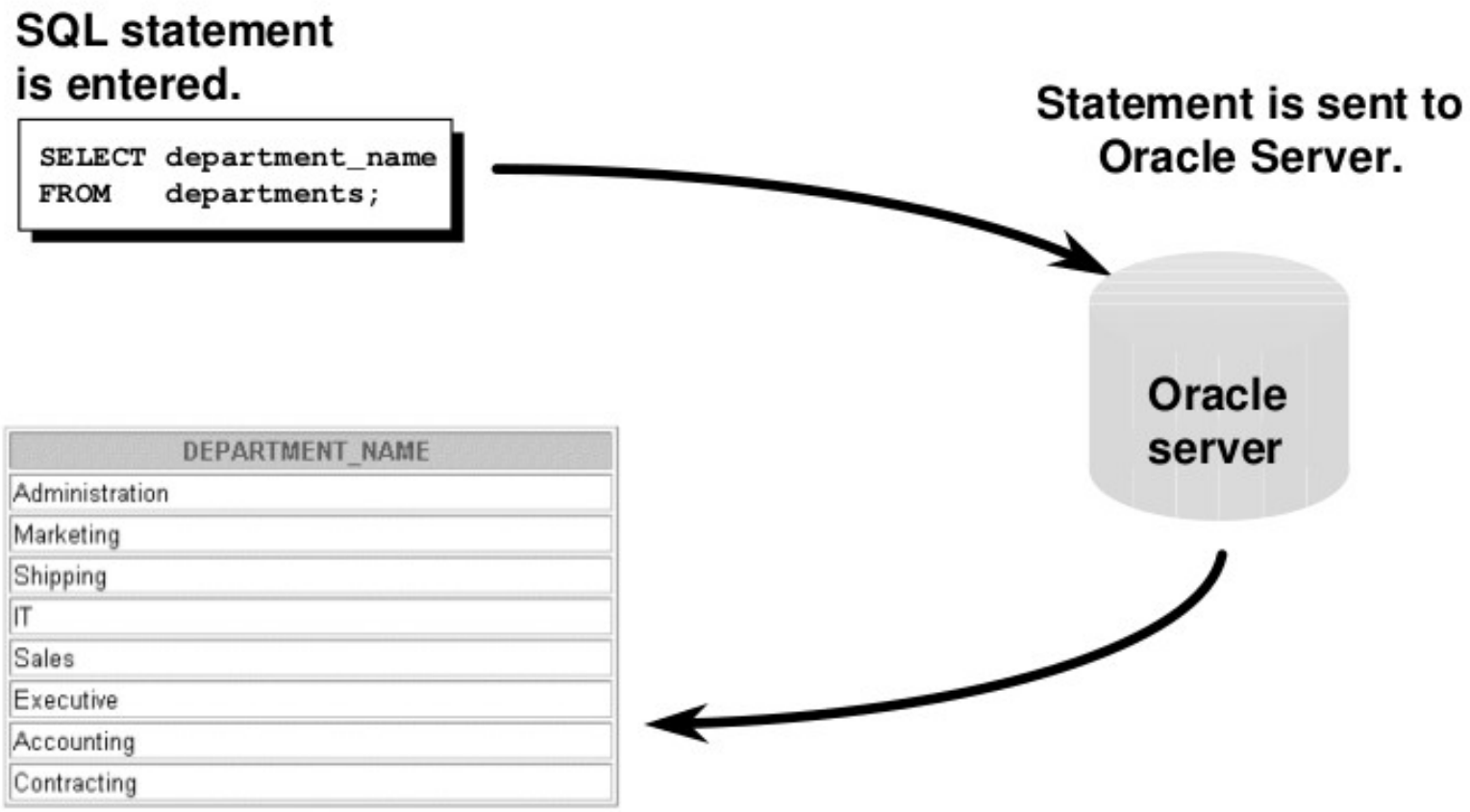
- The data in an RDBMS is stored in database objects which are called as tables. This table is basically a collection of related data entries and it consists of numerous columns and rows.
- Every table is broken up into smaller entities called fields.
- A record is also called as a row of data is each individual entry that exists in a table. A record is a horizontal entity in a table.
- A column is a vertical entity in a table that contains all information associated with a specific field in a table.

How the data is arranged in RDBMS?

Customer

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Communicating with a RDBMS Using SQL



SQL Statements

SELECT	Data retrieval
INSERT UPDATE DELETE MERGE	Data manipulation language (DML)
CREATE ALTER DROP RENAME TRUNCATE	Data definition language (DDL)
COMMIT ROLLBACK SAVEPOINT	Transaction control
GRANT REVOKE	Data control language (DCL)

SQL Statements (Continued)

Statement	Description
SELECT	Retrieves data from the database
INSERT UPDATE DELETE MERGE	Enters new rows, changes existing rows, and removes unwanted rows from tables in the database, respectively. Collectively known as *data manipulation language* (DML).
CREATE ALTER DROP RENAME TRUNCATE	Sets up, changes, and removes data structures from tables. Collectively known as *data definition language* (DDL).
COMMIT ROLLBACK SAVEPOINT	Manages the changes made by DML statements. Changes to the data can be grouped together into logical transactions.
GRANT REVOKE	Gives or removes access rights to both the Oracle database and the structures within it. Collectively known as *data control language* (DCL).

SQL Data Types

Data type	Description
VARCHAR2(*size*)	Variable-length character data (a maximum *size* must be specified: Minimum *size* is 1; maximum *size* is 4000)
CHAR [(*size*)]	Fixed-length character data of length *size* bytes (default and minimum *size* is 1; maximum *size* is 2000)
NUMBER [(*p*,*s*)]	Number having precision *p* and scale *s* (The precision is the total number of decimal digits, and the scale is the number of digits to the right of the decimal point; the precision can range from 1 to 38 and the 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.
LONG	Variable-length character data up to 2 gigabytes
CLOB	Character data up to 4 gigabytes

SQL Data Types (Continued)

Data type	Description
RAW(*size*)	Raw binary data of length *size* (a maximum *size* must be specified. maximum *size* is 2000)
LONG RAW	Raw binary data of variable length up to 2 gigabytes
BLOB	Binary data up to 4 gigabytes
BFILE	Binary data stored in an external file; up to 4 gigabytes
ROWID	A 64 base number system representing the unique address of a row in its table.

- 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 may want to use a CLOB column rather than a LONG column.

SQL Data Types (Continued)

Data Types	Description
Bit-string	Fixed length: `BIT(`*n*`)` Varying length: `BIT VARYING(`*n*`)`
Boolean	Values of TRUE, FALSE or NONE
TIME	Made up of hour, minute and seconds in the format hh:mm:ss
DATE	Ten Positions Components are YEAR, MONTH and DAY in the format YYYY-MM-DD
TIMESTAMP	Includes DATE and TIME fields plus a minimum of six positions for decimal fractions of seconds with optional WITH TIME ZONE qualifier.
INTERVAL	Specifies a relative value that can be used to increment or decrement an absolute value of a DATE, TIME or, TIMESTAMP.

SQL Constraints

- The Oracle Server uses constraints to prevent invalid data entry into tables.
- Constraints are the rules enforced on data columns on a table.
- These are used to limit the type of data that can go into a table.
- 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 deletion of a table if there are dependencies from other tables
- This ensures the accuracy and reliability of the data in the database.
- Constraints can either be column level or table level.
- Column level constraints are applied only to one column whereas, table level constraints are applied to the entire table.

SQL Constraints

Following are some of the most commonly used constraints available in SQL −

NOT NULL Constraint − Ensures that a column cannot have a NULL value.

DEFAULT Constraint − Provides a default value for a column when none is specified.

UNIQUE Constraint − Ensures that all the values in a column are different.

PRIMARY Key − Uniquely identifies each row/record in a database table.

FOREIGN Key − Uniquely identifies a row/record in any another database table.

CHECK Constraint − The CHECK constraint ensures that all values in a column satisfy certain conditions.

INDEX − Used to create and retrieve data from the database very quickly.

Defining Constraints

Column-level constraint

column [CONSTRAINT constraint_name] constraint_type,

Table level constraint

column,...

[CONSTRAINT constraint_name] constraint_type

(column, ...),

Defining Constraints (Continued)

In the syntax:

constraint_name is the name of the constraint

constraint_type is the type of the constraint

Constraints can be defined at one of two levels.

Constraint Level	Description
Column	References a single column and is defined within a specification for the owning column; can define any type of integrity constraint
Table	References one or more columns and is defined separately from the definitions of the columns in the table; can define any constraints except `NOT NULL`

SQL CREATE Table

The SQL CREATE TABLE statement is used to create a new table. Creating a basic table involves naming the table and defining its columns and each column's data type and constraints.

Syntax

```
CREATE TABLE table_name(
   column1 datatype column-level-constraint1,
   column2 datatype  column-level-constraint2,
   column3 datatype  column-level-constraint3,
   .....
   columnN datatype  column-level-constraintn,
   Table-level-constraint( one or more columns )
);
```

SQL CREATE Table (Continued)

```
CREATE TABLE Employee(
EmployeeID number(5) NOT NULL PRIMARY KEY,
FirstName varchar(255) NOT NULL,
LastName varchar(255),
City varchar(255)
);
```

Naming Rules

Name database tables and columns according to the standard rules for naming any Oracle database object:

- Table names and column names must begin with a letter and be 1–30 characters long.
- Names must contain only the characters A–Z, a–z, 0–9, _ (underscore), $, and # (legal characters, but their use is discouraged).
- Names must not duplicate the name of another object owned by the same Oracle server user.
- Names must not be an Oracle server reserved word.

Note: Use descriptive names for tables and other database objects.

Names are case insensitive. For example, EMPLOYEES is treated as the same name as eMPloyees or eMpLOYEES.

SQL CREATE Table (Continued)

```
CREATE TABLE Employee(
EmployeeID number(5) NOT NULL,
FirstName varchar(255) NOT NULL,
LastName varchar(255),
City varchar(255),
CONSTRAINT  PK_Employee PRIMARY KEY (EmployeeID, FirstName)
);
```

SQL DESC command

You can verify if your table has been created successfully by looking at the message displayed by the SQL server, otherwise you can use the DESC command as follows −

```
SQL> DESC EMPLOYEE;

+--------+-------------+--------+------+-----+---------+--------
| Field          | Type             | Null  | Key | Default | Extra |
+--------+-------------+--------+------+-----+---------+--------
| EmployeeID | NUMBER(5)  | NO   | PRI |           |         |
| FirstName    | varchar(255)  | NO   | PRI |           |         |
| LastName    | varchar(255)  | NO    |      |           |         |
| City varchar | char(255)       | YES  |       | NULL   |         |
+--------+-------------+-------+-------+-----+---------+--------

4 rows in set (0.00 sec)
```

The NOT Null Constraint

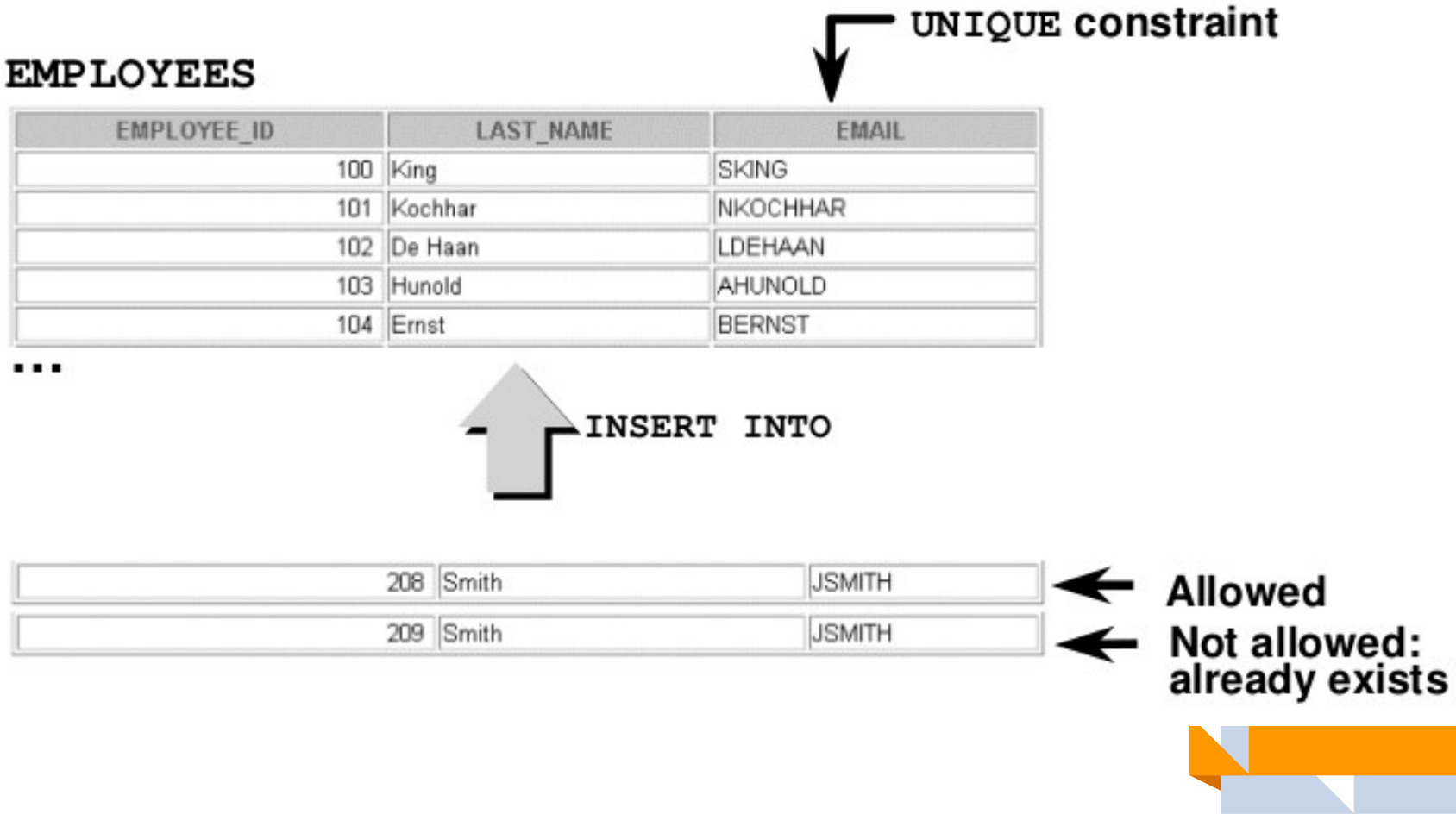
Is defined at the column level:

```
CREATE TABLE employees(
    employee_id     NUMBER(6),
    last_name       VARCHAR2(25) NOT NULL,
    salary          NUMBER(8,2),
    commission_pct  NUMBER(2,2),
    hire_date       DATE
                    CONSTRAINT emp_hire_date_nn
                    NOT NULL,
...
```

System named

User named

The UNIQUE Constraint



The UNIQUE Constraint (continued)

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

The UNIQUE Constraint (Continued)

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));
```

The DEFAULT Constraint

The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value.

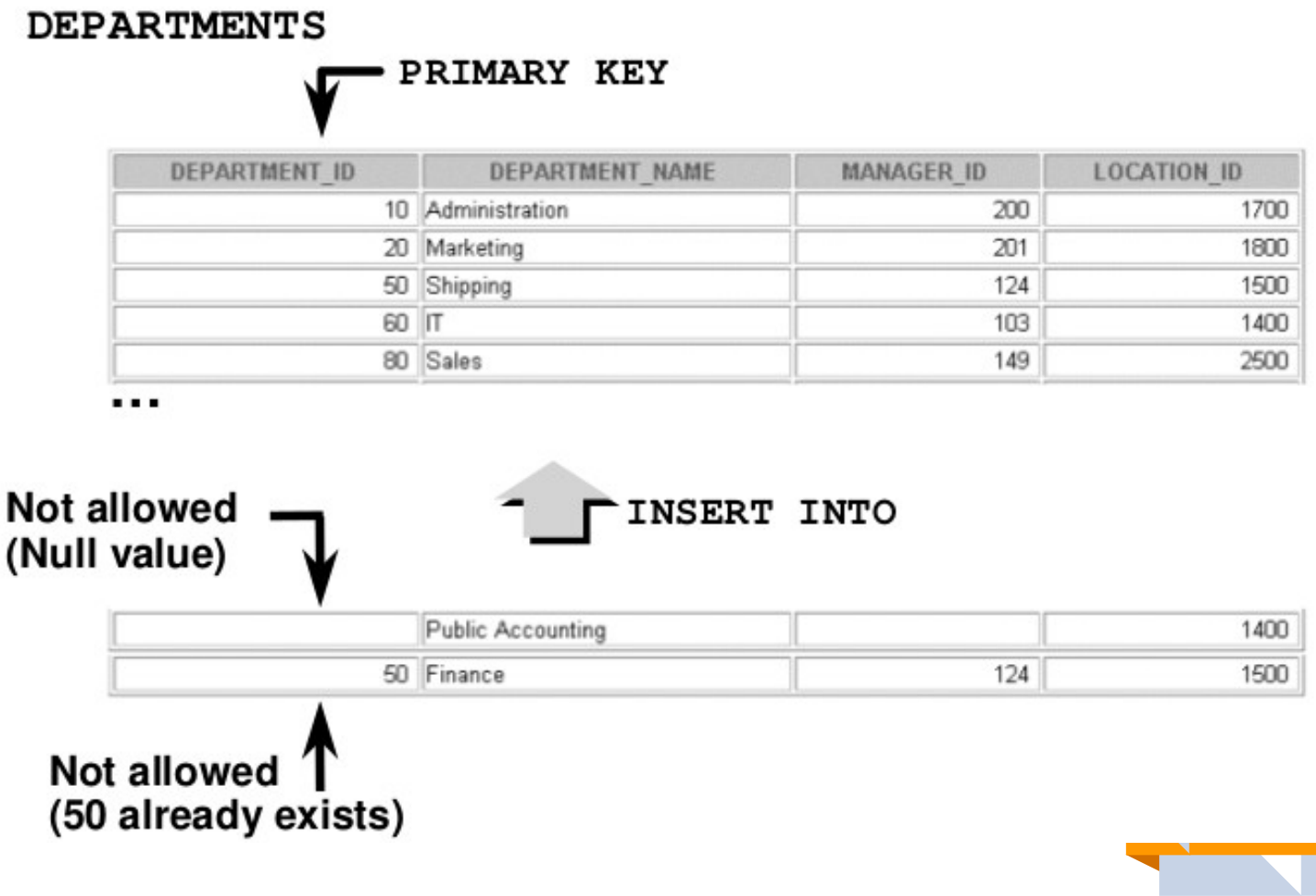
```
CREATE TABLE CUSTOMERS(
   ID   number(5)            NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  number(5)            NOT NULL,
   ADDRESS  CHAR (25) ,
   SALARY   number (18, 2) DEFAULT 5000.00,
   PRIMARY KEY (ID)
);
```

The above SQL creates a new table called CUSTOMERS and adds five columns. Here, the SALARY column is set to 5000.00 by default, so in case the INSERT INTO statement does not provide a value for this column, then by default this column would be set to 5000.00.

The PRIMARY KEY Constraint (Continued)

- 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 set of columns that uniquely identifies each row in a table.
- This constraint enforces uniqueness of the column or column combination and ensures that no column that is part of the primary key can contain a null value.
- PRIMARY KEY constraints can be defined at the column level or table level.
- A composite PRIMARY KEY is created by using the table-level definition.
- A table can have only one PRIMARY KEY constraint but can have several UNIQUE constraints.
- A UNIQUE index is automatically created for a PRIMARY KEY column.

The PRIMARY KEY Constraint



The PRIMARY KEY Constraint (Continued)

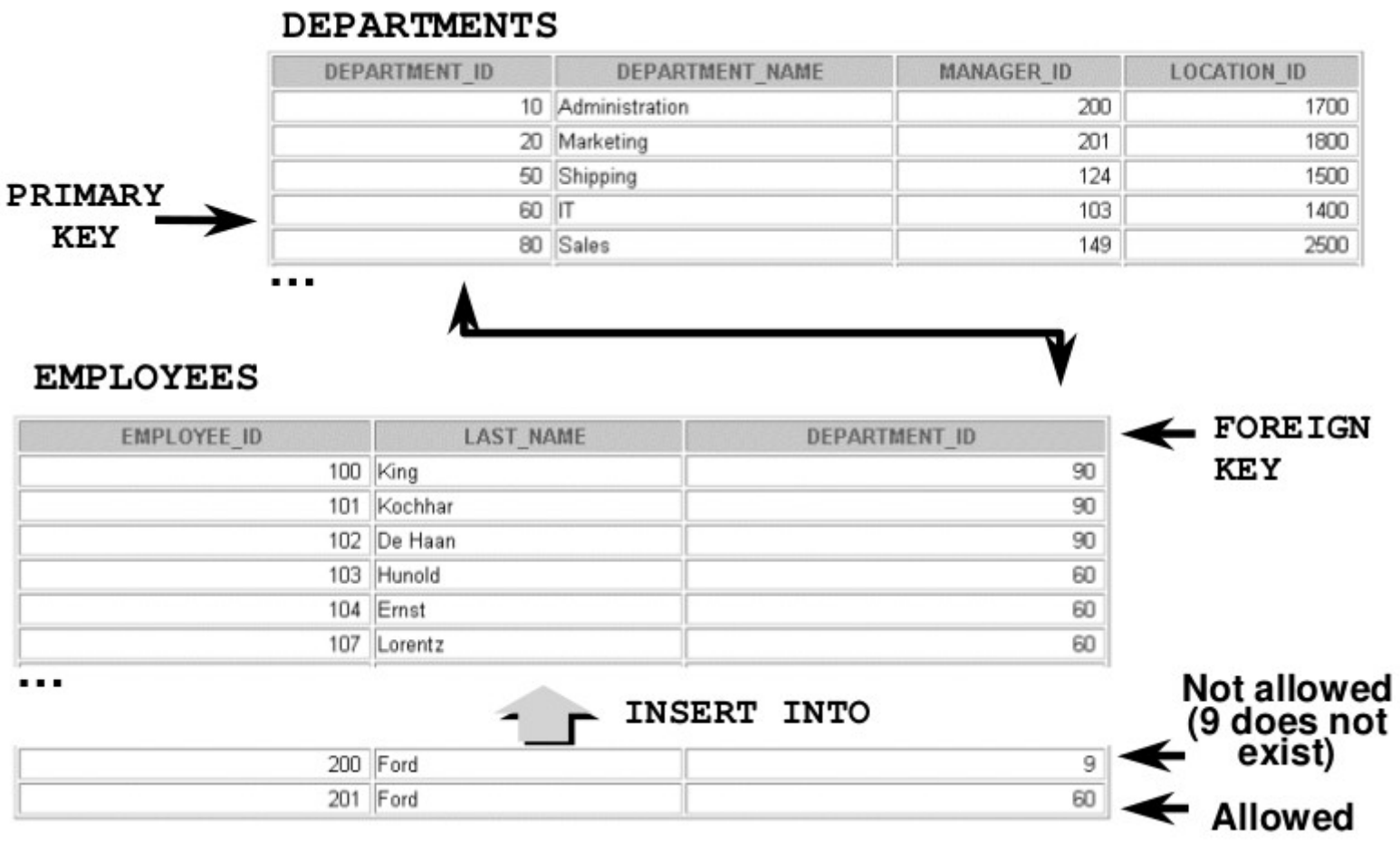
Defined at either the table level or the column level:

```
CREATE TABLE   departments(
    department_id          NUMBER(4),
    department_name        VARCHAR2(30)
      CONSTRAINT dept_name_nn NOT NULL,
    manager_id             NUMBER(6),
    location_id            NUMBER(4),
      CONSTRAINT dept_id_pk PRIMARY KEY(department_id));
```

The FOREIGN KEY Constraint

- The FOREIGN KEY, or referential integrity constraint, designates a column or combination of columns as a foreign key and establishes a relationship between a primary key or a unique key in the same table or a different table.
- A foreign key value must match an existing value in the parent table or be NULL.
- FOREIGN KEY constraints can be defined at the column or table level.
- A composite foreign key must be created by using the table-level definition.

The FOREIGN KEY Constraint (Continued)



The FOREIGN KEY Constraint (Continued)

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));
```

The FOREIGN KEY Constraint (Continued)

The foreign key can also be defined at the column level, provided the constraint is based on a single column.

For example:

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

The FOREIGN KEY Constraint Keywords

The foreign key is defined in the child table, and the table containing the referenced column is the parent table.

FOREIGN KEY is used to define the column in the child table at the table constraint level.

REFERENCES identifies the table and column in the parent table.

ON DELETE CASCADE indicates that when the row in the parent table is deleted, the dependent rows in the child table will also be deleted.

ON DELETE SET NULL converts foreign key values to null when the parent value is removed.

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.

The FOREIGN KEY Constraint Keywords (Continued)

Syntax

The syntax for creating a foreign key with set null on delete using a CREATE TABLE statement in SQL Server is:

```
CREATE TABLE child_table
(
  column1 datatype [ NULL | NOT NULL ],
  column2 datatype [ NULL | NOT NULL ],
  ...
  CONSTRAINT fk_name
    FOREIGN KEY (child_col1, child_col2, ... child_col_n)
    REFERENCES parent_table (parent_col1, parent_col2, ... parent_col_n)
    ON DELETE SET NULL
    [ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
);
```

The FOREIGN KEY Constraint Keywords (Continued)

The foreign key is defined in the child table, and the table containing the referenced column is the parent table.

FOREIGN KEY is used to define the column in the child table at the table constraint level.

REFERENCES identifies the table and column in the parent table.

ON DELETE CASCADE indicates that when the row in the parent table is deleted, the dependent rows in the child table will also be deleted.

ON DELETE SET NULL converts foreign key values to null when the parent value is removed.

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.

The FOREIGN KEY Constraint Keywords (Continued)

```
CREATE TABLE products
( product_id number(5) PRIMARY KEY,
  product_name VARCHAR(50) NOT NULL,
  category VARCHAR(25)
);


CREATE TABLE inventory
( inventory_id number(5) PRIMARY KEY,
  product_id number(5),
  quantity number(5),
  min_level number(5),
  max_level number(5),
  CONSTRAINT fk_inv_product_id FOREIGN KEY (product_id)
    REFERENCES products (product_id)ON DELETE SET NULL
);
```

The CHECK Constraint

The CHECK constraint defines a condition that each row must satisfy. The condition can use the same constructs as 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 which refer to the column in its definition. There is no limit to the number of CHECK constraints which you can define on a column.

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

```
CREATE TABLE employees
      (...
       salary NUMBER(8,2) CONSTRAINT emp_salary_min
                          CHECK (salary > 0),
      ...
```

Viewing Constraints

Query the `USER_CONSTRAINTS` table to view all constraint definitions and names.

```
SELECT    constraint_name, constraint_type,
          search_condition
FROM      user_constraints
WHERE     table_name = 'EMPLOYEES';
```

CONSTRAINT_NAME	C	SEARCH_CONDITION
EMP_LAST_NAME_NN	C	"LAST_NAME" IS NOT NULL
EMP_EMAIL_NN	C	"EMAIL" IS NOT NULL
EMP_HIRE_DATE_NN	C	"HIRE_DATE" IS NOT NULL
EMP_JOB_NN	C	"JOB_ID" IS NOT NULL
EMP_SALARY_MIN	C	salary > 0
EMP_EMAIL_UK	U	

...

Viewing Constraints (Continued)

Note:

- After creating a table, you can confirm its existence by issuing a DESCRIBE command.
- The only constraint that you can verify is the NOT NULL constraint. To view all constraints on your table, query the USER_CONSTRAINTS table.
- Constraints that are not named by the table owner receive the system-assigned constraint name.
- In constraint type, C stands for CHECK, P for PRIMARY KEY, R for referential integrity, and U for UNIQUE key.
- Notice that the NOT NULL constraint is really a CHECK constraint.

Viewing the Columns Associated with Constraints

You can view the names of the columns involved in constraints by querying the **USER_CONS_COLUMNS** data dictionary view.

This view is especially useful for constraints that use system-assigned names.

```
SELECT   constraint_name, column_name
FROM     user_cons_columns
WHERE    table_name = 'EMPLOYEES';
```

CONSTRAINT_NAME	COLUMN_NAME
EMP_DEPT_FK	DEPARTMENT_ID
EMP_EMAIL_NN	EMAIL
EMP_EMAIL_UK	EMAIL
EMP_EMP_ID_PK	EMPLOYEE_ID
EMP_HIRE_DATE_NN	HIRE_DATE
EMP_JOB_FK	JOB_ID
EMP_JOB_NN	JOB_ID

...

The ALTER TABLE Statement

After you create a table, you may need to change the table structure because: you omitted a column or constraint, your column definition needs to be changed, or you need to remove columns or constraints. You can do this by using the ALTER TABLE statement.

Use the ALTER TABLE statement to:

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

The ALTER TABLE Statement (Continued)

```
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);
```

DEFAULT expr specifies the default value for a new column

The ALTER TABLE Statement (Continued)

Adding a Column

Example: ALTER TABLE dept
ADD ((job_id VARCHAR2(9));

Note:

- You cannot specify where the column is to appear. The new column becomes the last column.
- If a table already contains rows when a column is added, then the new column is initially null for all the rows.

The ALTER TABLE Statement (Continued)

Modifying a Column

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

Example: ALTER TABLE dept
MODIFY (last_name VARCHAR2(30));

Note:

- You can increase the width or precision of a numeric column.
- You can increase the width of numeric or character columns.
- You can decrease the width of a column only if the column contains only null values or if the table has no rows.
- You can change the data type only if the column contains null values.
- 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.

The ALTER TABLE Statement (Continued)

Dropping a Column

You can drop a column from a table by using the ALTER TABLE statement with the DROP COLUMN clause. This is a feature available in Oracle8i and later.

Note:

- The column may or may not contain data.
- Using the ALTER TABLE 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.
- Once a column is dropped, it cannot be recovered.

Example:
```
ALTER TABLE dept
        DROP COLUMN job_id;
```

The ALTER TABLE Statement (Continued)

Use the `ALTER TABLE` statement to:

- Add or drop a constraint, but not modify its structure
- Enable or disable constraints
- Add a `NOT NULL` constraint by using the `MODIFY` clause

```
ALTER TABLE table
ADD [CONSTRAINT constraint] type (column);
```

The ALTER TABLE Statement (Continued)

Adding a Constraint

Guidelines

- You can add, drop, enable, or disable a constraint, but you cannot modify its structure.
- You can add a NOT NULL constraint to an existing column by using the MODIFY clause of the ALTER TABLE statement.
- You can define a NOT NULL column only if the table is empty or if the column has a value for every row.
- The constraint name in the syntax is optional, although recommended. If you do not name your constraints, the system will generate constraint names.

The ALTER TABLE Statement (Continued)

Examples:

1. **Add a FOREIGN KEY constraint to the EMPLOYEES table indicating that a manager must already exist as a valid employee in the EMPLOYEES table.**

```
ALTER TABLE employees
ADD CONSTRAINT emp_manager_fk FOREIGN KEY(manager_id) REFERENCES
employees(employee_id);
```

2. **To add a NOT NULL constraint, use the ALTER TABLE MODIFY syntax:**

```
ALTER TABLE employees
MODIFY (salary CONSTRAINT emp_salary_nn NOT NULL);
```

The ALTER TABLE Statement (Continued)

Dropping a Constraint

To drop a constraint, you can identify the constraint name from the USER_CONSTRAINTS and USER_CONS_COLUMNS data dictionary views. Then use the ALTER TABLE statement with the DROP clause. The CASCADE option of the DROP clause causes any dependent constraints also to be dropped.

Syntax

```
ALTER TABLE table
DROP PRIMARY KEY | UNIQUE (column) |
CONSTRAINT constraint [CASCADE];
```

When you drop an integrity constraint, that constraint is no longer enforced by the Oracle server and is no longer available in the data dictionary.

The ALTER TABLE Statement (Continued)

Dropping a Constraint

- Remove the manager constraint from the EMPLOYEES table.

```
ALTER TABLE       employees
DROP CONSTRAINT   emp_manager_fk;
Table altered.
```

- Remove the PRIMARY KEY constraint on the DEPARTMENTS table and drop the associated FOREIGN KEY constraint on the EMPLOYEES.DEPARTMENT_ID column.

```
ALTER TABLE   departments
DROP PRIMARY KEY CASCADE;
Table altered.
```

The DROP TABLE Statement

The DROP TABLE statement removes the definition of an Oracle 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

Example: DROP TABLE dept;

Note :

- 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.
- The DROP TABLE statement, once executed, is irreversible.

The Data Manipulation Language

- Data manipulation language (DML) is a core part of SQL.
- When you want to add, update, or delete data in the database, you execute a DML statement.
- Every table has INSERT, UPDATE, and DELETE privileges associated with it.
- These privileges are automatically granted to the creator of the table, but in general they must be explicitly granted to other users.
- A collection of DML statements that form a logical unit of work is called a transaction.

The Data Manipulation Language

When Does a Transaction Start and End?

A transaction begins when the first DML statement is encountered and ends when one of the following occurs:

- A COMMIT or ROLLBACK statement is issued
- A DDL statement, such as CREATE, is issued
- A DCL statement is issued
- The user exits SQL
- A machine fails or the system crashes

After one transaction ends, the next executable SQL statement automatically starts the next transaction.

A DDL statement or a DCL statement is automatically committed and therefore implicitly ends a transaction.

The INSERT statement (Continued)

70	Public Relations	100	1700

New row

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

...insert a new row into the DEPARMENTS table...

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700
70	Public Relations	100	1700

The INSERT statement (Continued)

Syntax :

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

Note: This statement with the VALUES clause adds only one row at a time to a table.

- Because you can insert a new row that contains values for each column, the column list is not required in the INSERT clause.
- However, if you do not use the column list, the values must be listed according to the default order of the columns in the table, and a value must be provided for each column.

The INSERT statement (Continued)

Inserting New Rows

- You can insert a new row that contains values for each column, the column list is not required in the INSERT clause.
- If you do not use the column list, the values must be listed according to the default order of the columns in the table, and a value must be provided for each column.
- For clarity, use the column list in the INSERT clause.
- Enclose character and date values within single quotation marks; it is not recommended to enclose numeric values within single quotation marks.
- Number values should not be enclosed in single quotes, because implicit conversion may take place for numeric values assigned to NUMBER data type columns if single quotes are included.

The INSERT statement (Continued)

Inserting New Rows

```
DESCRIBE  departments
```

Name	Null?	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

```
INSERT INTO departments(department_id, department_name,
                        manager_id, location_id)
VALUES      (70, 'Public Relations', 100, 1700);
1 row created.
```

The INSERT statement (Continued)

Inserting Rows with Null Values

Method	Description
Implicit	Omit the column from the column list.
Explicit	Specify the `NULL` keyword in the `VALUES` list, specify the empty string (`''`) in the `VALUES` list for character strings and dates.

Be sure that you can use null values in the targeted column by verifying the `Null?` status with the `DESCRIBE` command.

The Oracle Server automatically enforces all data types, data ranges, and data integrity constraints. Any column that is not listed explicitly obtains a null value in the new row.

Common errors that can occur during user input:

- Mandatory value missing for a `NOT NULL` column
- Duplicate value violates uniqueness constraint
- Foreign key constraint violated
- `CHECK` constraint violated
- Data type mismatch
- Value too wide to fit in column

The INSERT statement (Continued)

Inserting Multiple Rows (Creating a Script)

- Use & substitution in a SQL statement to prompt for values.
- & is a placeholder for the variable value.

```
INSERT INTO departments
            (department_id, department_name, location_id)
VALUES      (&department_id, '&department_name',&location);
```

The UPDATE statement

- **Modify existing rows with the `UPDATE` statement.**

```
UPDATE      table
SET         column = value [, column = value, ...]
[WHERE      condition];
```

- **Update more than one row at a time, if required.**

Note:

In general, use the primary key to identify a single row. Using other columns can unexpectedly cause several rows to be updated. For example, identifying a single row in the EMPLOYEES table by name is dangerous, because more than one employee may have the same name.

The UPDATE statement (Continued)

- Specific row or rows are modified if you specify the `WHERE` clause.

```
UPDATE employees
SET    department_id = 70
WHERE  employee_id = 113;
1 row updated.
```

- All rows in the table are modified if you omit the `WHERE` clause.

```
UPDATE   copy_emp
SET      department_id = 110;
22 rows updated.
```

The DELETE statement

- You can remove existing rows by using the DELETE statement.
- The DELETE statement does not ask for confirmation. However, the delete operation is not made permanent until the data transaction is committed.
- Therefore, you can undo the operation with the ROLLBACK statement if you make a mistake.
- In the Syntax, ***condition*** identifies the rows to be deleted and is composed of column names, expressions, constants, subqueries, and comparison operators.

```
DELETE [FROM]   table
[WHERE          condition];
```

The DELETE statement (Continued)

- Specific rows are deleted if you specify the `WHERE` clause.

```
 DELETE FROM departments
 WHERE  department_name = 'Finance';
1 row deleted.
```

- All rows in the table are deleted if you omit the `WHERE` clause.

```
DELETE FROM  copy_emp;
22 rows deleted.
```

Data Query Language Statement (DQL)
The SELECT statement

The SQL **SELECT** statement is used to fetch the data from a database table which returns this data in the form of a result table. These result tables are called result-sets.

Syntax

The basic syntax of the SELECT statement is as follows −

```
SELECT column1, column2, columnN FROM table_name;
```

Here, column1, column2... are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field, then you can use the following syntax.

```
SELECT * FROM table_name;
```

The SELECT statement (Continued)

Example

Consider the CUSTOMERS table having the following records −

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

```
1. SELECT ID, NAME, SALARY FROM CUSTOMERS;

2. SELECT * FROM CUSTOMERS;
```

Eliminating Duplicate Rows

Eliminate duplicate rows by using the DISTINCT keyword in the SELECT clause.

Example:

```
SELECT DISTINCT name  FROM  employees;
```

```
SELECT DISTINCT Country FROM Customers;
```

The SELECT statement with WHERE clause

- The SQL WHERE clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables.
- If the given condition is satisfied, then only it returns a specific value from the table. You should use the WHERE clause to filter the records and fetching only the necessary records.
- The WHERE clause is not only used in the SELECT statement, but it is also used in the UPDATE, DELETE statement, etc.,
- You can specify a condition using the comparison or logical operators like >, <, =, LIKE, NOT, etc.

The SELECT statement with WHERE clause

Syntax

The basic syntax of the SELECT statement with the WHERE clause is as shown below.

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
```

```
SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE SALARY >
     2000;
```

```
SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE NAME =
     'Hardik';
```

The SQL Operators

What is an Operator in SQL?

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations. These Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

- Arithmetic operators
- Comparison operators
- Logical operators
- Operators used to negate conditions

The SQL Expressions

An expression is a combination of one or more values, operators and SQL functions that evaluate to a value. These SQL EXPRESSIONs are like formulae and they are written in query language. You can also use them to query the database for a specific set of data.

Syntax

```
SELECT column1, column2, columnN
FROM table_name
WHERE [CONDITION|EXPRESSION];
```

The SQL Expressions

Boolean Expressions

SQL Boolean Expressions fetch the data based on matching a single value.

Syntax −

```
SELECT column1, column2, columnN FROM table_name
WHERE SINGLE VALUE MATCHING EXPRESSION;
```

Example: SELECT * FROM CUSTOMERS WHERE SALARY = 10000;

The SQL Expressions

Numeric Expression

These expressions are used to perform any mathematical operation in any query.

Syntax

```
SELECT numerical_expression as  OPERATION_NAME
[FROM table_name
WHERE CONDITION] ;
```

Example: SELECT (15 + 6) AS ADDITION

```
SELECT COUNT(*) AS "RECORDS" FROM CUSTOMERS;
```

The SQL Expressions

Date Expressions

Date Expressions return current system date and time values −

Example:

```
SQL>  SELECT CURRENT_TIMESTAMP
+----------------------+
| Current_Timestamp    |
+----------------------+
| 2009-11-12 06:40:23 |
+----------------------+
1 row in set (0.00 sec)
```

```
SQL>  SELECT  GETDATE();;
+--------------------------+
| GETDATE                  |
+--------------------------+
| 2009-10-22 12:07:18.140 |
+--------------------------+
1 row in set (0.00 sec)
```

The SQL Logical Operators

Sr.No.	Operator & Description
1	**ALL** The ALL operator is used to compare a value to all values in another value set.
2	**AND** The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
3	**ANY** The ANY operator is used to compare a value to any applicable value in the list as per the condition.

The SQL Logical Operators

4	**BETWEEN** The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
5	**EXISTS** The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.
6	**IN** The IN operator is used to compare a value to a list of literal values that have been specified.
7	**LIKE** The LIKE operator is used to compare a value to similar values using wildcard operators.

The SQL Logical Operators

9	**OR** The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
10	**IS NULL** The NULL operator is used to compare a value with a NULL value.
11	**UNIQUE** The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

The SQL Logical Operators

Consider the CUSTOMERS table having the following records −

```
SQL> SELECT * FROM CUSTOMERS;
```

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
7 rows in set (0.00 sec)
```

The SQL Logical Operators

Examples

- SELECT * FROM CUSTOMERS WHERE AGE >= 25 AND SALARY >= 6500;
- SELECT * FROM CUSTOMERS WHERE AGE >= 25 OR SALARY >= 6500;
- SELECT * FROM CUSTOMERS WHERE AGE IS NOT NULL;
- SELECT * FROM CUSTOMERS WHERE NAME LIKE 'Ko%';
- SELECT * FROM CUSTOMERS WHERE AGE IN (25, 27);
- SELECT * FROM CUSTOMERS WHERE AGE BETWEEN 25 AND 27;

The SQL IN Operator

Examples:

- SELECT * FROM Customers

  WHERE Country IN ('Germany', 'France', 'UK');
- SELECT * FROM Customers

  WHERE Country NOT IN ('Germany', 'France', 'UK');
- SELECT * FROM Customers

  WHERE Country IN (SELECT Country FROM Suppliers);

The SQL IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.
- The IN operator is a shorthand for multiple OR conditions.

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

or:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

The SQL LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character

The percent sign and the underscore can also be used in combinations!

LIKE Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

The SQL LIKE Operator

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

The SQL NOT Operator

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id
       NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP');
```

The NOT Truth Table

The following table shows the result of applying the NOT operator to a condition:

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

Note: The NOT operator can also be used with other SQL operators, such as BETWEEN, LIKE, and NULL.

```
... WHERE  job_id      NOT  IN ('AC_ACCOUNT', 'AD_VP')
... WHERE  salary      NOT  BETWEEN  10000 AND  15000
... WHERE  last_name NOT  LIKE '%A%'
... WHERE  commission_pct  IS   NOT  NULL
```

The Concatenation Operator

A concatenation operator:

Concatenates columns or character strings to other columns

Is represented by two vertical bars (||)

Creates a resultant column that is a character expression

Example: SELECT first_name || last_name AS "Employees" FROM employees;

The SELECT statement with ORDER BY clause

- The order of rows returned in a query result is undefined. The ORDER BY clause can be used to sort the rows.
- If you use the ORDER BY clause, it must be the last clause of the SQL statement. You can specify an expression, or an alias, or column position as the sort condition.
- If the ORDER BY clause is not used, the sort order is undefined, and the Oracle server may not fetch rows in the same order for the same query twice.
- Use the ORDER BY clause to display the rows in a specific order.

The SELECT statement with ORDER BY clause

Syntax

```
SELECT      *|{[DISTINCT] column|expression [alias],...}
FROM        table
[WHERE      condition(s)]
[ORDER BY   {column, expr, alias} [ASC|DESC]];
```

In the syntax:

ORDER BY	specifies the order in which the retrieved rows are displayed
ASC	orders the rows in ascending order (this is the default order)
DESC	orders the rows in descending order

Internally, the order of execution for a SELECT statement is as follows:
FROM clause
WHERE clause
SELECT clause
ORDER BY clause

The SELECT statement with ORDER BY clause

Default Ordering of Data

The default sort order is ascending:

- Numeric values are displayed with the lowest values first—for example, 1–999.
- Date values are displayed with the earliest value first—for example, 01-JAN-2021 before 01-JAN-2021.
- Character values are displayed in alphabetical order—for example, A first and Z last.
- Null values are displayed last for ascending sequences and first for descending Sequences.

To reverse the order in which rows are displayed, specify the *DESC* keyword after the column name in the ORDER BY clause.

The SELECT statement with ORDER BY clause

Default Ordering of Data

The default sort order is ascending:

- Numeric values are displayed with the lowest values first—for example, 1–999.
- Date values are displayed with the earliest value first—for example, 01-JAN-2021 before 01-JAN-2021.
- Character values are displayed in alphabetical order—for example, A first and Z last.
- Null values are displayed last for ascending sequences and first for descending Sequences.

To reverse the order in which rows are displayed, specify the *DESC* keyword after the column name in the ORDER BY clause.

The SELECT statement with ORDER BY clause

Defining a Column Alias

A column alias:

- Renames a column heading
- Is useful with calculations
- Immediately follows the column name - there can also be the optional AS keyword between the column name and alias
- Requires double quotation marks if it contains spaces or special characters or is case sensitive
- By default, alias headings appear in uppercase.

The SELECT statement with ORDER BY clause

Examples:

SELECT last_name, job_id, department_id, hire_date FROM employees
ORDER BY hire_date DESC ;

SELECT FROM ORDER BY last_name, salary employees 2 DESC; (using column number)

SELECT last_name AS name, commission_pct comm
FROM employees ORDER BY name; (using column aliases)

SELECT employee_id, last_name, salary*12 annsal FROM employees
ORDER BY annsal; (using column aliases)

SELECT last_name, department_id, salary FROM employees
ORDER BY department_id, salary DESC; (Using multiple columns)

The Concatenation Operator

A concatenation operator:

Concatenates columns or character strings to other columns

Is represented by two vertical bars (||)

Creates a resultant column that is a character expression

Example: SELECT first_name || last_name AS "Employees" FROM employees;

Using Literal Character Strings

SELECT last_name ||’ has a ’|| job_id AS "Employee Details" FROM employees;

The SQL GROUP BY Statement

- The GROUP BY clause is a SQL command that is used to group rows that have the same values.
- The GROUP BY clause is used in the SELECT statement. Optionally it is used in conjunction with aggregate functions to produce summary reports from the database.
- The queries that contain the GROUP BY clause are called grouped queries and only return a single row for every grouped item.
- The SELECT statement used in the GROUP BY clause can only be used to contain column names, aggregate functions, constants and expressions.
- SQL Having Clause is used to restrict the results returned by the GROUP BY clause.

The SQL GROUP BY Statement

Syntax

SELECT statements... GROUP BY column_name1[,column_name2,...]

[HAVING condition];

Here

- “SELECT statements…” is the standard SQL SELECT command query.
- “GROUP BY column_name1” is the clause that performs the grouping based on column_name1. “[,column_name2,…]” is optional; represents other column names when the grouping is done on more than one column.
- “[HAVING condition]” is optional; it is used to restrict the rows affected by the GROUP BY clause. It is similar to the WHERE clause.

The SQL GROUP BY Statement

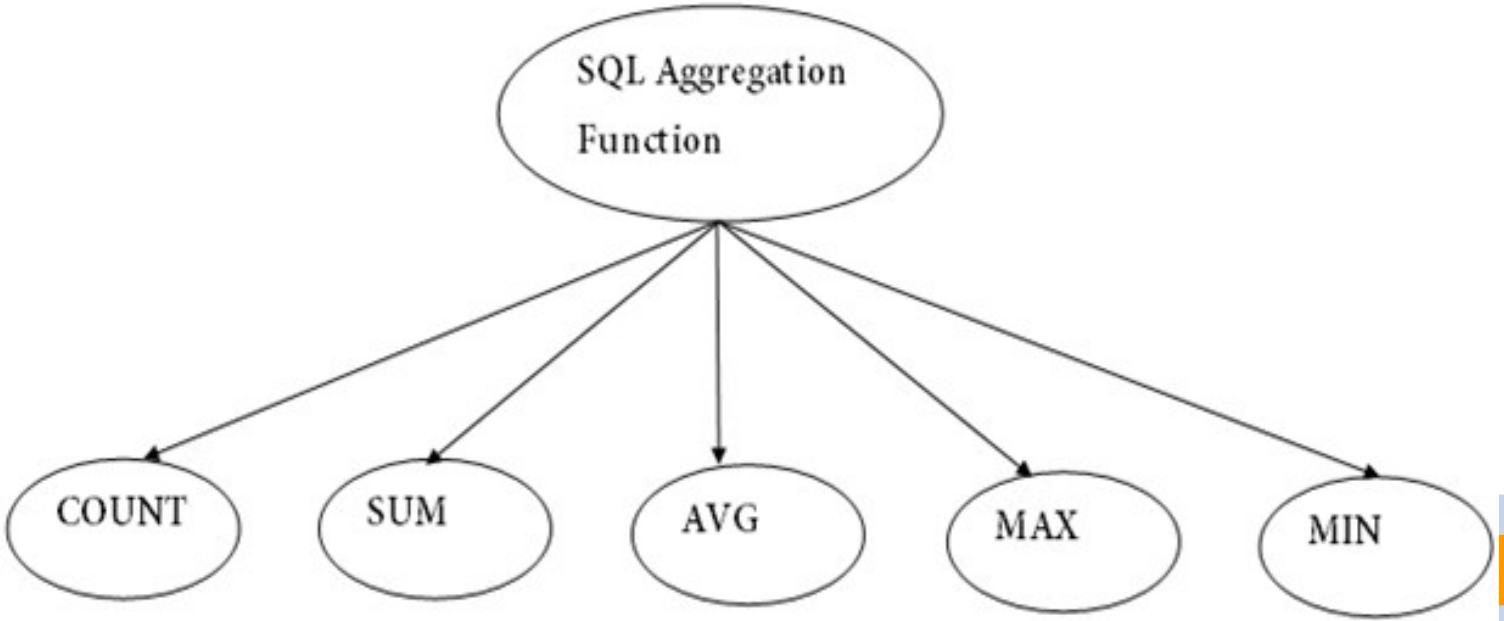
Examples:

```
SELECT gender FROM Members GROUP BY gender ;


SELECT category_id, year_released FROM Movies
GROUP BY category_id, year_released;


SELECT * FROM movies GROUP BY category_id, year_released
HAVING category_id = 8;
```

Aggregate functions in SQL

In database management an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

Types of SQL Aggregation Functions



Aggregate functions in SQL

COUNT FUNCTION

- COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.
- COUNT(*) returns the count of all the rows in a specified table. COUNT(*) considers duplicate and Null.

Syntax

COUNT(*)

or

COUNT([ALL|DISTINCT] expression)

Aggregate functions in SQL

COUNT FUNCTION Examples

- SELECT COUNT(*) FROM PRODUCT_MAST;
- SELECT COUNT(*) FROM PRODUCT_MAST WHERE RATE>=20;
- SELECT COUNT(DISTINCT COMPANY) FROM PRODUCT_MAST;
- SELECT COMPANY, COUNT(*) FROM PRODUCT_MAST
  GROUP BY COMPANY;
- SELECT COMPANY, COUNT(*) FROM PRODUCT_MAST GROUP BY COMPANY HAVING COUNT(*)>2;
- SELECT Pnumber, Pname, COUNT (*) FROM PROJECT, WORKS_ON WHERE Pnumber=Pno GROUP BY Pnumber, Pname HAVING COUNT (*) > 2;

(For each project on which more than two employees work, retrieve the project number, the project name, and the number of employees who work on the project.)

Aggregate functions in SQL

SUM FUNCTION

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax

SUM()

or

SUM([ALL|DISTINCT] expression)

Aggregate functions in SQL

SUM FUNCTION Examples

- SELECT SUM(COST) FROM PRODUCT_MAST;
- SELECT SUM(COST) FROM PRODUCT_MAST WHERE QTY>3;
- SELECT SUM(COST) FROM PRODUCT_MAST WHERE QTY>3
  GROUP BY COMPANY;
- SELECT COMPANY, SUM(COST) FROM PRODUCT_MAST
  GROUP BY COMPANY HAVING SUM(COST)>=170;

Aggregate functions in SQL

AVG FUNCTION

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax

AVG()

or

AVG([ALL|DISTINCT] expression)

Example : SELECT AVG(COST) FROM PRODUCT_MAST;

Aggregate functions in SQL

MAX FUNCTION

MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Syntax

MAX()

or

MAX([ALL|DISTINCT] expression)

Example: SELECT MAX(RATE) FROM PRODUCT_MAST;

Aggregate functions in SQL

MIN FUNCTION

MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

Syntax

MIN()

or

MIN([ALL|DISTINCT] expression)

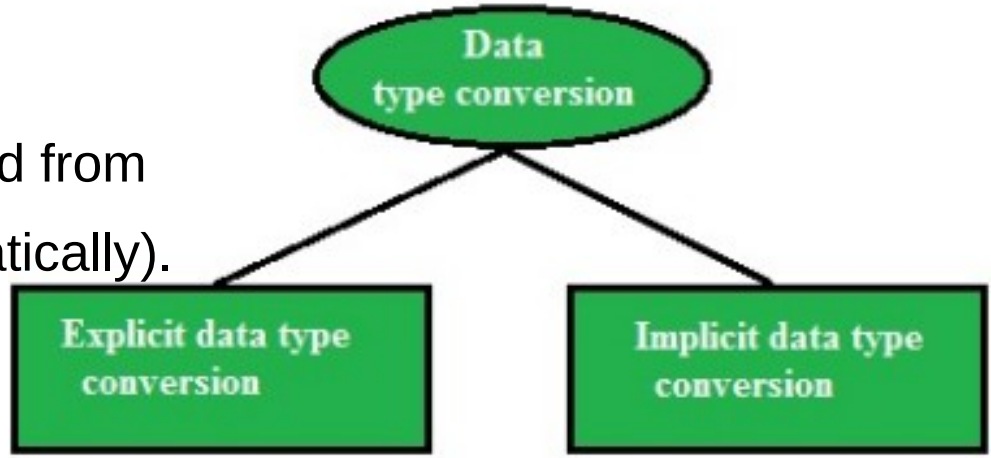
Example: SELECT MIN(RATE) FROM PRODUCT_MAST;

Aggregate functions in SQL

Conversion Function

Implicit Data-Type Conversion :

In this type of conversion the data is converted from one type to another implicitly (by itself/automatically).



From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE
DATE	VARCHAR2
NUMBER	VARCHAR2

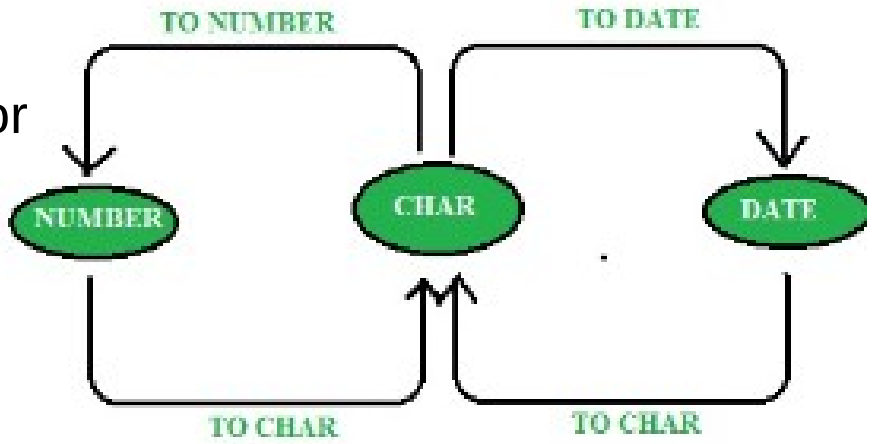
```
SELECT employee_id,first_name,salary
FROM employees
WHERE salary > '15000';
```

Aggregate functions in SQL

Explicit Data-Type Conversion :

TO_CHAR Function :

TO_CHAR function is used to typecast a numeric or date input to character type with a format model (optional).



Explicit Data Type Converson

SYNTAX :

TO_CHAR(number1, [format], [nls_parameter])

Aggregate functions in SQL

Using the TO_CHAR Function with Dates :

SYNTAX :

TO_CHAR(date, 'format_model')

The format model:

- Must be enclosed in single quotation marks and is case sensitive
- Can include any valid date format element
- Has an fm element to remove padded blanks or suppress leading zeros
- Is separated from the date value by a comma

Example:

```
SELECT employee_id, TO_CHAR(hire_date, 'MM/YY') Month_Hired
FROM employees WHERE last_name = 'Ram';
```

Aggregate functions in SQL

Elements of the Date Format Model :

```
SELECT last_name,
TO_CHAR(hire_date, 'fmDD Month YYYY')
AS HIREDATE FROM employees;
```

OUTPUT :

LASTNAME	HIIREDATE
Austin	25 January 2005
Shubham	20 June 2004

YYYY	Full year in Numbers
YEAR	Year spelled out
MM	Two digit value for month
MONTH	Full name of the month
MON	Three Letter abbreviation of the month
DY	Three letter abbreviation of the day of the week
DAY	Full Name of the of the week
DD	Numeric day of the month

Aggregate functions in SQL

Using the TO_CHAR Function with Numbers :

SYNTAX :

TO_CHAR(number, 'format_model')

Example:

```
SELECT TO_CHAR(salary, '$99,999.00')
SALARY
FROM employees
WHERE last_name = 'Rao';
```

9	Represent a number
0	Forces a zero to be displayed
$	places a floating dollar sign
L	Uses the floating local currency symbol
.	Print a decimal point
,	Prints a Thousand indicator

Aggregate functions in SQL

Using the TO_NUMBER and TO_DATE Functions :

Convert a character string to a number format using the TO_NUMBER function :

TO_NUMBER(char[, ’format_model’])

Convert a character string to a date format using the TO_DATE function:

TO_DATE(char[, ’format_model’])

These functions have an fx modifier.

This modifier specifies the exact matching for the character argument and date format model of a TO_DATE function.

Aggregate functions in SQL

Using TO_DATE Function :

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date = TO_DATE('May 24, 1999', 'fxMonth DD, YYYY');
```

OUTPUT :

LASTNAME	HIREDATE
Kumar	24-MAY-99

Aggregate functions in SQL

Using TO_NUMBER Function (A string function that converts a string expression to a value of NUMERIC data type**):**

```
SELECT TO_NUMBER('5467.12') FROM DUAL;  --  5467.12

SELECT TO_NUMBER('4687841', '9999999') FROM DUAL; -- 4687841

SELECT TO_NUMBER('$65.169', 'L99.999') FROM DUAL; -- 65.169

SELECT TO_NUMBER('-15 degrees F') -- Display Mode: -15.00
```

How to Join Two Tables in SQL

Relational databases are built with multiple tables that refer to each other. Rows from one table refer to specific rows in another table, which are connected by some ID column(s).

`product`

id	product_name	price	category_id
1	smartwatch	235.00	2
2	bricks	26.70	3
3	lamp	128.00	2
4	sofa	3200.00	1
5	desk	1350.00	1
6	power strip	29.00	2

`category`

id	category_name
1	furniture
2	electronics
3	toys

How to Join Two Tables in SQL

Let’s say you need some details from this warehouse database, like the name of the products, the price and their respective categories.

```
SELECT
product.product_name,
product.price,
category.category_name
FROM product, category
WHERE
product.category_id =
category.id ;
```

product_name	price	category_name
smartwatch	235.00	electronics
bricks	26.70	toys
lamp	128.00	electronics
sofa	3200.00	furniture
desk	1350.00	furniture
power strip	29.00	electronics

How to Join Two Tables in SQL

Syntax : SELECT * FROM <table1>, <table2> WHERE <condition>

</condition></table2></table1>

Example:

```
SELECT artist_name, album_name, year_recorded  FROM artist, album
WHERE artist.id = album.artist_id;
```

Syntax: SELECT * FROM <table1> JOIN <table2> ON/USING <condition>

Example:

```
SELECT  artist_name, album_name, year_recorded FROM artist
JOIN album  ON artist.id = album.artist_id;
```

Types of Joins in SQL

SQL Join statement is used to combine data or rows from two or more tables based on a common field between them.

Different types of Joins are as follows:

- CARTESIAN JOIN
- SELF JOIN
- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

OUTER JOINS

Types of Joins in SQL

CARTESIAN JOIN

- The CARTESIAN JOIN is also known as CROSS JOIN.
- In a CARTESIAN JOIN there is a join for each row of one table to every row of another table. This usually happens when the matching column or WHERE condition is not specified.
- In the absence of a WHERE condition the CARTESIAN JOIN will behave like a CARTESIAN PRODUCT . i.e., the number of rows in the result-set is the product of the number of rows of the two tables.
- In the presence of WHERE condition this JOIN will function like a INNER JOIN.
- In General, Cross join is similar to an inner join where the join-condition will always evaluate to True.

Types of Joins in SQL

Consider the following tables

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
4	SURESH	Delhi	XXXXXXXXXX	18

StudentCourse

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4

Types of Joins in SQL

CARTESIAN JOIN (Syntax and Example)

Syntax:

```
SELECT table1.column1 , table1.column2, table2.column1...
FROM table1
CROSS JOIN table2;
```

```
SELECT Student.NAME, Student.AGE, StudentCourse.COURSE_ID
FROM Student
CROSS JOIN StudentCourse;
```

NAME	AGE	COURSE_ID
Ram	18	1
Ram	18	2
Ram	18	2
Ram	18	3
RAMESH	18	1
RAMESH	18	2
RAMESH	18	2
RAMESH	18	3
SUJIT	20	1
SUJIT	20	2
SUJIT	20	2
SUJIT	20	3
SURESH	18	1
SURESH	18	2
SURESH	18	2
SURESH	18	3

Types of Joins in SQL

SELF JOIN

- In SELF JOIN a table is joined to itself.
- That is, each row of the table is joined with itself and all other rows depending on some conditions.
- In other words we can say that it is a join between two copies of the same table.
- Syntax:

```
SELECT a.coulmn1 , b.column2
FROM table_name a, table_name b
WHERE some_condition;
```

Types of Joins in SQL

SELF JOIN Example

```
SELECT a.ROLL_NO , b.NAME
FROM Student a, Student b
WHERE a.ROLL_NO < b.ROLL_NO;
```

ROLL_NO	NAME
1	RAMESH
1	SUJIT
2	SUJIT
1	SURESH
2	SURESH
3	SURESH

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
4	SURESH	Delhi	XXXXXXXXXX	18

Types of Joins in SQL

SELF JOIN Example

SQL> desc studentself;

```
Name            Null?    Type
--------------- -------- ----------
ROLL_NO                  NUMBER(1)
NAME                     VARCHAR2(10)
LEADER                   NUMBER(3)
```

SQL> select * from studentself;

```
  ROLL_NO NAME       LEADER
---------- ---------- ----------
         1 ram
         2 ramesh            1
         3 sujit             1
         4 suresh            2
```

SQL> select a.name as Leader,b.name from studentself a,studentself b where a.roll_no=b.leader;

```
LEADER     NAME
---------- ----------
ram        ramesh
ram        sujit
ramesh     suresh
```

Types of Joins in SQL

INNER JOIN

The INNER JOIN keyword selects all rows from both the tables as long as the condition is satisfied.

This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.

Note: We can also write JOIN instead of INNER JOIN. JOIN is same as INNER JOIN.

Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```

Types of Joins in SQL

Consider the following tables

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

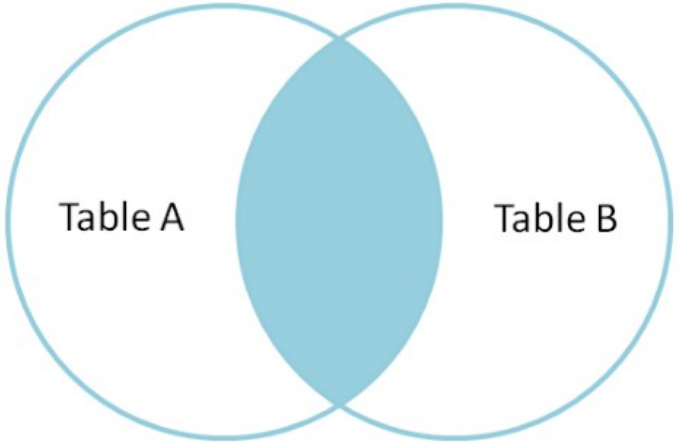
StudentCourse

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

Types of Joins in SQL

INNER JOIN Example:

```
SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE FROM
Student
INNER JOIN StudentCourse
ON Student.ROLL_NO = StudentCourse.ROLL_NO;
```



Output:

COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

Types of Joins in SQL

LEFT JOIN

This join returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join.

For the rows for which there is no matching row on the right side, the result-set will contain null.

LEFT JOIN is also known as LEFT OUTER JOIN.

Note: We can also use LEFT OUTER JOIN instead of LEFT JOIN, both are the same.

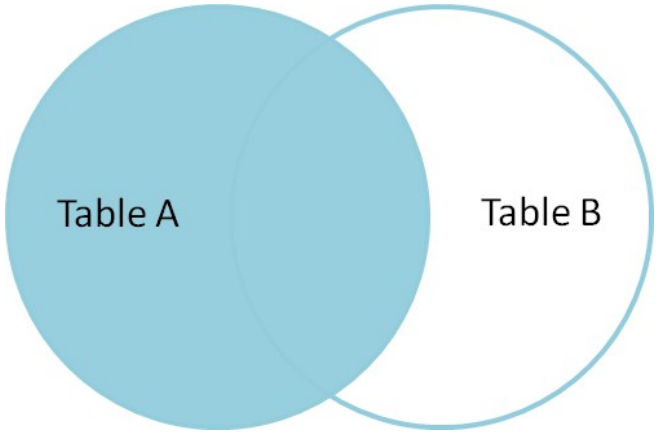
Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
```

Types of Joins in SQL

LEFT JOIN Example

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
LEFT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```



NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*

Types of Joins in SQL

RIGHT JOIN

This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join.

For the rows for which there is no matching row on the left side, the result-set will contain null.

RIGHT JOIN is also known as RIGHT OUTER JOIN.

Note: We can also use RIGHT OUTER JOIN instead of RIGHT JOIN, both are the same.

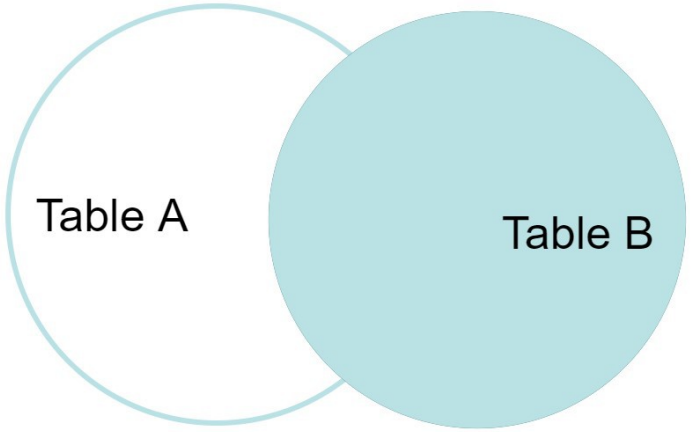
Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```

Types of Joins in SQL

RIGHT JOIN Example

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
RIGHT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```



NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

Types of Joins in SQL

FULL JOIN

FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN.

The result-set will contain all the rows from both tables.

For the rows for which there is no matching, the result-set will contain NULL values.

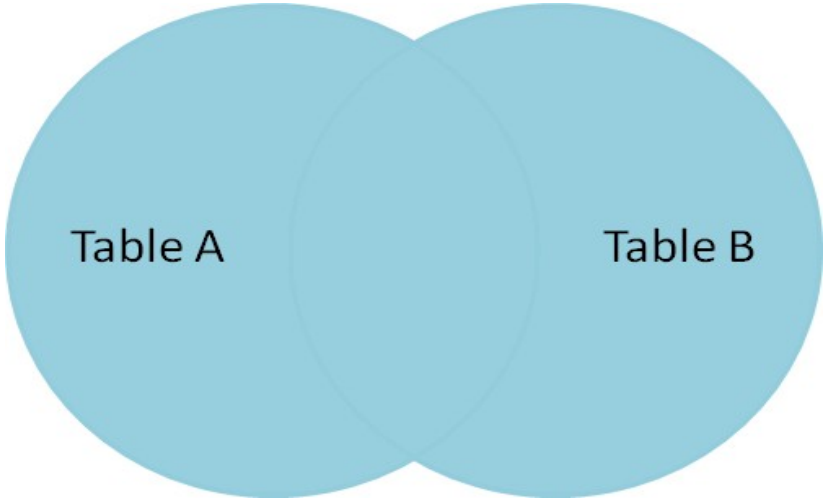
Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
FULL JOIN table2
ON table1.matching_column = table2.matching_column;
```

Types of Joins in SQL

FULL JOIN Example

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
FULL JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

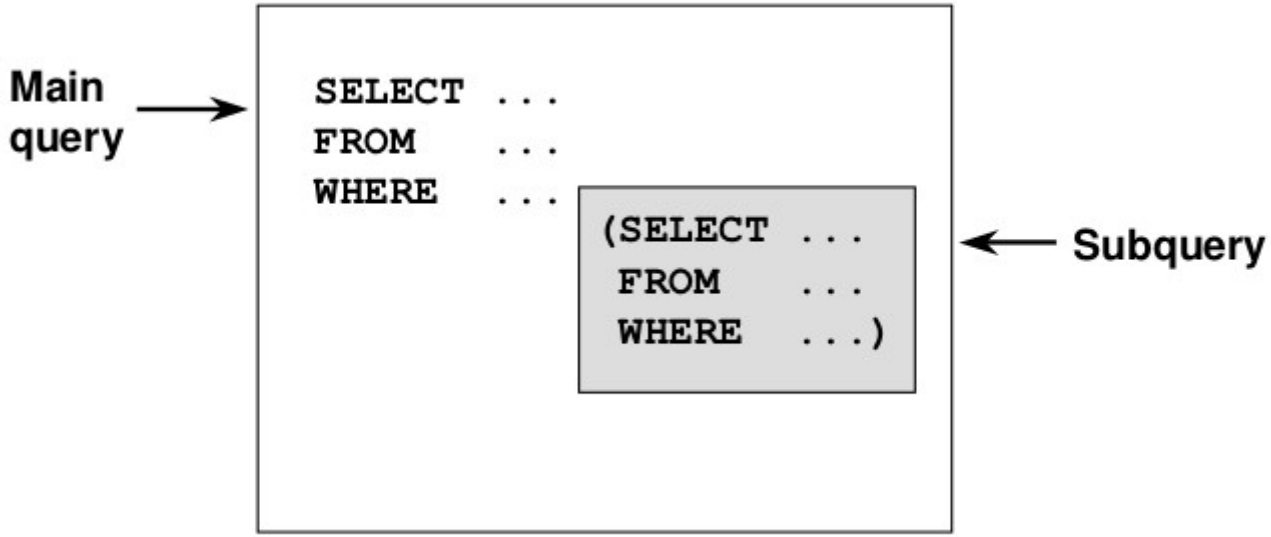


NAME	COURSE_ID
HARSH	1
PRACTICAL	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	NULL
ROHIT	NULL
NIRAJ	NULL
NULL	4
NULL	5
NULL	4

Subquery

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.



Subquery

Important rules for Subqueries:

- You can place the Subquery in a number of SQL clauses: WHERE clause, HAVING clause, FROM clause.
- Subqueries can be used with SELECT, UPDATE, INSERT, DELETE statements along with expression operator.
- It could be equality operator (IN, NOT IN etc) or comparison operator such as =, >, =, <= and Like operator.
- The subquery generally executes first, and its output is used to complete the query condition for the main or outer query.

Subquery

Important rules for Subqueries:

- Subquery must be enclosed in parentheses.
- Subqueries are on the right side of the comparison operator.
- Use single-row operators with singlerow Subqueries. Use multiple-row operators with multiple-row Subqueries.

Creating a Table from an Existing Table

- A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement.
- The new table has the same column definitions.
- All columns or specific columns can be selected.
- When you create a new table using the existing table, the new table would be populated using the existing values in the old table.

Syntax

```
CREATE TABLE NEW_TABLE_NAME AS
   SELECT [ column1, column2...columnN ]
   FROM EXISTING_TABLE_NAME
   [ WHERE ]
```

Example

```
SQL> CREATE TABLE SALARY AS
   SELECT ID, SALARY
   FROM CUSTOMERS;
```

Subqueries with the SELECT Statement

Syntax:

```
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
      (SELECT column_name [, column_name ]
      FROM table1 [, table2 ]
      [WHERE])
```

Subqueries with the SELECT Statement

Consider the CUSTOMERS table having the following record

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

```
SELECT *
FROM CUSTOMERS
WHERE ID IN (SELECT ID
      FROM CUSTOMERS
      WHERE SALARY > 4500) ;
```

ID	NAME	AGE	ADDRESS	SALARY
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
7	Muffy	24	Indore	10000.00

Subqueries with the INSERT Statement

- The INSERT statement uses the data returned from the subquery to insert into another table.
- The selected data in the subquery can be modified with any of the character, date or number functions.

Syntax:

```
INSERT INTO table_name [ (column1 [, column2 ]) ]
SELECT [ *|column1 [, column2 ]
FROM table1 [, table2 ]
[ WHERE]
```

Subqueries with the INSERT Statement

Example:

- Consider a table CUSTOMERS_BKP with similar structure as CUSTOMERS table. Now to copy the complete CUSTOMERS table into the CUSTOMERS_BKP table, you can use the following syntax.

```
INSERT INTO CUSTOMERS_BKP
SELECT * FROM CUSTOMERS
WHERE ID IN (SELECT ID FROM CUSTOMERS) ;
```

Subqueries with the UPDATE Statement

- The subquery can be used in conjunction with the UPDATE statement.
- Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

Syntax:

```
UPDATE table
SET column_name = new_value
[ WHERE OPERATOR [ VALUE ]
   (SELECT COLUMN_NAME
   FROM TABLE_NAME)
   [ WHERE}) ]
```

Subqueries with UPDATE Statement

Example:

- Assuming, we have CUSTOMERS_BKP table available which is backup of CUSTOMERS table. The following example updates SALARY by 0.25 times in the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

```
UPDATE CUSTOMERS
   SET SALARY = SALARY * 0.25
   WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
     WHERE AGE >= 27 );
```

Subqueries with the DELETE Statement

Syntax:

```
DELETE FROM TABLE_NAME
[ WHERE OPERATOR [ VALUE ]
   (SELECT COLUMN_NAME
   FROM TABLE_NAME)
   [ WHERE) ]
```

Example:

```
DELETE FROM CUSTOMERS
   WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
      WHERE AGE >= 27 );
```

Using a Subquery in the FROM Clause

```
SELECT   a.last_name, a.salary,
         a.department_id, b.salavg
FROM     employees a, (SELECT   department_id,
                       AVG(salary) salavg
                       FROM     employees
                       GROUP BY department_id) b
WHERE    a.department_id = b.department_id
AND      a.salary > b.salavg;
```

LAST_NAME	SALARY	DEPARTMENT_ID	SALAVG
Hartstein	13000	20	9500
Mourgos	5800	50	3500
Hunold	9000	60	6400
Zlotkey	10500	80	10033.3333
Abel	11000	80	10033.3333
King	24000	90	19333.3333
Higgins	12000	110	10150

HAVING clause with subqueries

Find the designation with lowest average salary

```
SELECT  job, AVG(sal)
  FROM emp
  GROUP BY  job
  HAVING   AVG(sal) =
                        ( SELECT MIN(AVG(sal))
                          FROM emp
                          GROUP BY job );
```

Multilevel Subquery

Display departments that have minimum salary greater than that of department 1

```
SELECT DNAME FROM DEPARTMENT
WHERE DNO IN
                (SELECT DNO FROM EMPLOYEE
                GROUP BY DNO
                HAVING MIN(SALARY) >
                                (SELECT MIN(SALARY)
                                FROM EMPLOYEE
                                WHERE DNO=1));
```

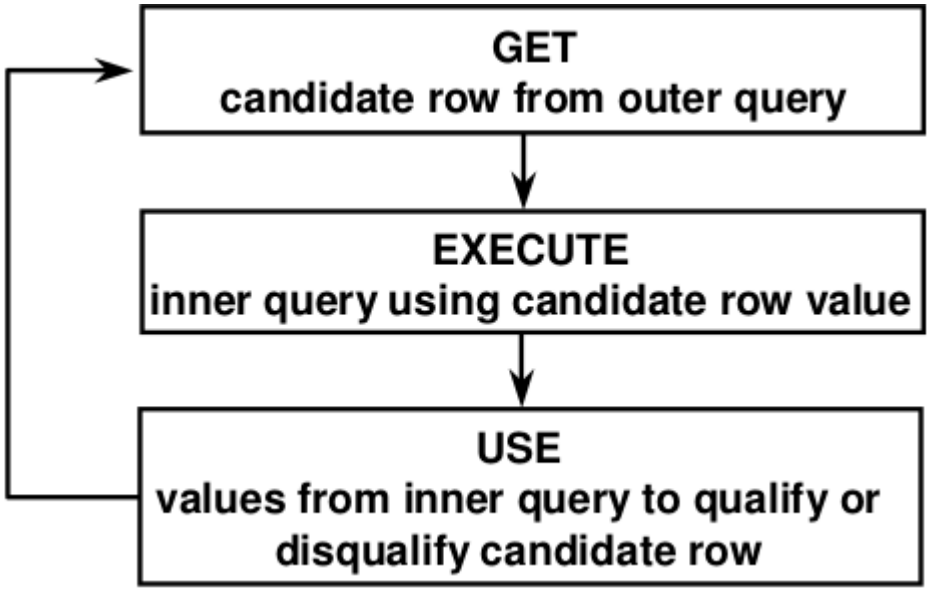
Subquery returning multiple columns

Display details of employees who are having same dno and salary as that of employee with empno=101

```
SELECT * FROM EMPLOYEE
WHERE (DNO,SALARY) IN
                    (SELECT DNO,SALARY
                     FROM EMPLOYEE
                     WHERE ENO=101);
```

Correlated Subqueries

- Correlated subqueries are used for row-by-row processing.
- Each subquery is executed once for every row of the outer query.
- A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a SELECT, UPDATE, or DELETE statement.



Correlated Subqueries

```
SELECT column1, column2, ....
FROM table1 outer
WHERE column1 operator
                    (SELECT column1, column2
                     FROM table2
                     WHERE expr1 =
                                outer.expr2);
```

You can use a correlated subquery to answer a multipart question whose answer depends on the value in each row processed by the parent statement.

Note: You can use the ANY and ALL operators in a correlated subquery.

Correlated Subqueries

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

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

Correlated Subqueries

EXAMPLE : Display details of those employees who have switched jobs at least twice.

```
SELECT e.Employee_id, last_name,e.job_id
FROM
Employees e
      WHERE 2 <= (SELECT COUNT(*)
                        FROM  job_history
                        WHERE Employee_id = e.Employee_id);
```

The ANY and ALL operators

- The ANY and ALL operators are used with a WHERE or HAVING clause.
- The ANY operator returns true if any of the subquery values meet the condition.

SQL ANY compares a value of the first table with all values of the second table and returns the row if there is a match with any value.

- The ALL operator returns true if all of the subquery values meet the condition.

SQL ALL compares a value of the first table with all values of the second table and returns the row if there is a match with all values.

The ANY operator

Condition	Meaning
x = ANY (...)	The values in column c must match one or more values in the set to evaluate to true.
x != ANY (...)	The values in column c must not match one or more values in the set to evaluate to true.
x > ANY (...)	The values in column c must be greater than the smallest value in the set to evaluate to true.
x < ANY (...)	The values in column c must be smaller than the biggest value in the set to evaluate to true.
x >= ANY (...)	The values in column c must be greater than or equal to the smallest value in the set to evaluate to true.
x <= ANY (...)	The values in column c must be smaller than or equal to the biggest value in the set to evaluate to true.

The ANY operator

EXAMPLE : Consider the following tables

Table: Teachers

id	name	age
1	Peter	32
2	Megan	43
3	Rose	29
4	Linda	30
5	Mary	41

Table: Students

id	name	age
1	Harry	23
2	Jack	42
3	Joe	32
4	Dent	23
5	Bruce	40

select * from Teachers

where age= ANY (select age from Students)

id	name	age
1	Peter	32

The ANY operator

EXAMPLE : Consider the following tables

Table: Teachers

id	name	age
1	Peter	32
2	Megan	43
3	Rose	29
4	Linda	30
5	Mary	41

Table: Students

id	name	age
1	Harry	23
2	Jack	42
3	Joe	32
4	Dent	23
5	Bruce	40

select * from Teachers

where age < ANY (select age from Students)

id	name	age
1	Peter	32
3	Rose	29
4	Linda	30
5	Mary	41

The ANY operator

EXAMPLE :

Find all employees whose salaries are equal to the average salary of their department, you use the following query:

```
SELECT first_name, last_name, salary FROM employees WHERE salary = ANY ( SELECT AVG(salary) FROM employees GROUP BY department_id) ORDER BY first_name, last_name, salary;
```

Find all employees whose salaries are greater than the average salary in every department:

```
SELECT first_name, last_name, salary FROM employees WHERE salary > ANY (SELECT AVG(salary) FROM employees GROUP BY department_id) ORDER BY salary;
```

The ALL operator

EXAMPLE : Consider the following tables

Table: Teachers

id	name	age
1	Peter	32
2	Megan	43
3	Rose	29
4	Linda	30
5	Mary	41

Table: Students

id	name	age
1	Harry	23
2	Jack	42
3	Joe	32
4	Dent	23
5	Bruce	40

select * from Teachers

where age > ALL (select age from Students)

id	name	age
2	Megan	43

The ANY and ALL operators

They are used to compare a single value with each of the individual values in the set returned by the subquery.

```
SELECT   Emp_Name
FROM     Emp
WHERE    Salary  >ANY
            ( SELECT   Salary
              FROM     Emp
              WHERE    Dept_No  =  'D1' ) ;
```

If '>' is *true* for ***at least one*** value, **ANY** returns *true*, else *false*.

```
SELECT   Emp_Name
FROM     Emp
WHERE    Salary  >ALL
            ( SELECT   Salary
              FROM     Emp
              WHERE    Dept_No  =  'D1' ) ;
```

If '>' is *true* for ***every*** value, **ALL** returns *true*, else *false*.

The Example for ANY

Display the employees whose salary is less than any clerk and who are not clerks

```
SELECT empno, ename, job
FROM emp
WHERE sal < ANY
            ( SELECT sal
              FROM emp
              WHERE job = 'CLERK' )
AND job <> 'CLERK' ;
```

The Example for ALL

Display the employees whose salary is greater than the average salaries of all the departments

```
SELECT empno, ename, job
  FROM emp
  WHERE sal > ALL
                ( SELECT AVG(sal)
                  FROM emp
                 GROUP BY deptno ) ;
```

Nested Subqueries Versus Correlated Subqueries

Nested Subquery Execution

- The inner query executes first and finds a value.
- The outer query executes once, using the value from the inner query.

"With a normal nested subquery, the inner SELECT query runs first and executes once, returning values to be used by the main query".

Correlated Subquery Execution

- Get a candidate row (fetched by the outer query).
- Execute the inner query using the value of the candidate row.
- Use the values resulting from the inner query to qualify or disqualify the candidate.
- Repeat until no candidate row remains.

"A correlated subquery, however, executes once for each candidate row considered by the outer query. In other words, the inner query is driven by the outer query".

Using the EXISTS Operator

- The EXISTS operator tests for existence of rows in the results set of the subquery.
- **If a subquery row value is found:**
  - –The search does not continue in the inner query
  - –The condition is flagged TRUE
- **If a subquery row value is not found:**
  - – The condition is flagged FALSE
  - – The search continues in the inner query

Using the EXISTS Operator

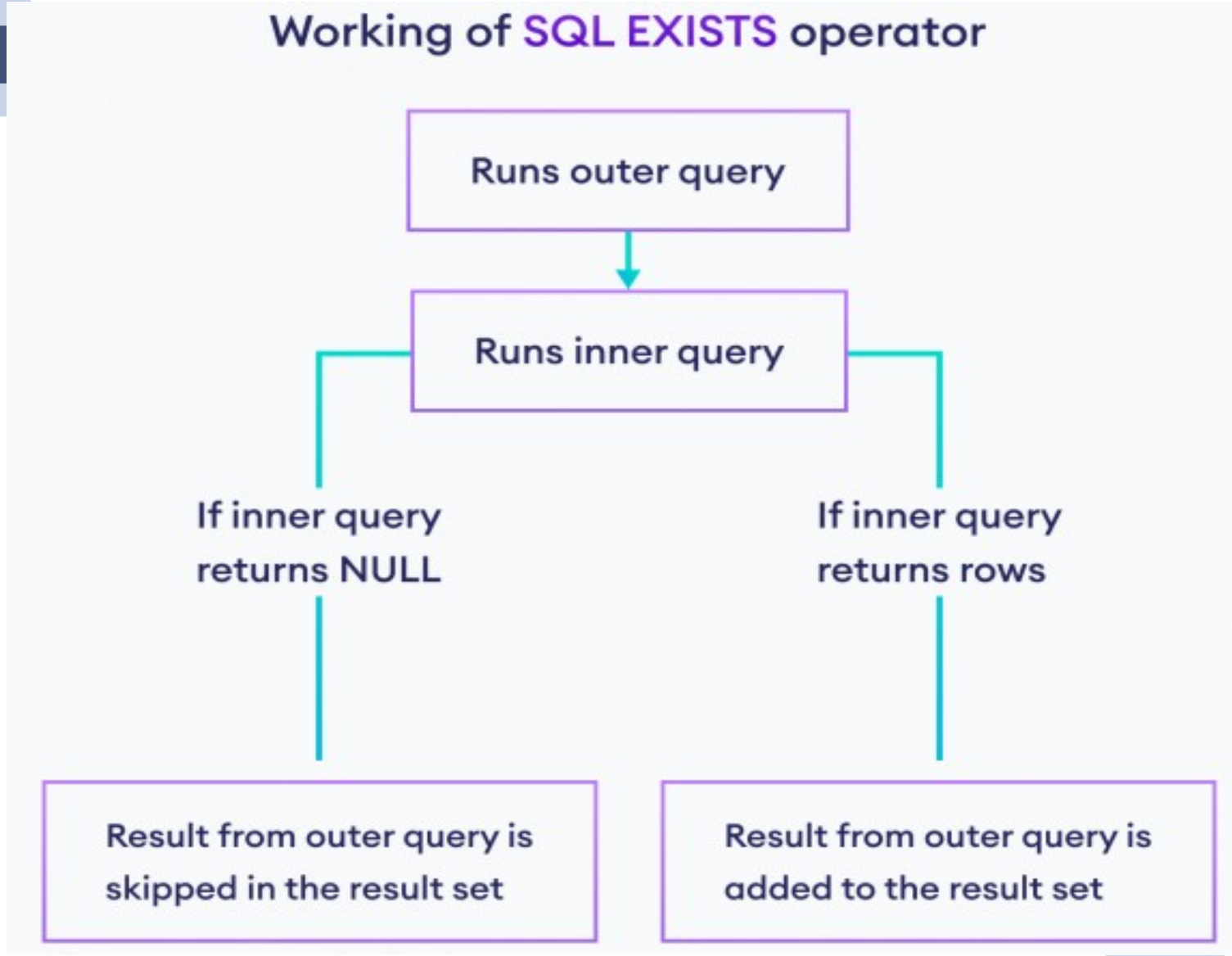
The EXISTS condition in SQL is used to check whether the result of a correlated nested query is empty (contains no tuples) or not.

The result of EXISTS is a boolean value True or False.

It can be used in a SELECT, UPDATE, INSERT or DELETE statement.

EXISTS Syntax

```
SELECT column_name(s) FROM table_name
WHERE EXISTS (SELECT column_name(s)
FROM table_name WHERE condition);
```


Working of SQL EXISTS operator
Runs outer query
Runs inner query
If inner query returns NULL
If inner query returns rows
Result from outer query is skipped in the result set
Result from outer query is added to the result set

Using the EXISTS Operator (Example)

Customers

customer_id	lname	fname	website
401	Singh	Dolly	abc.com
402	Chauhan	Anuj	def.com
403	Kumar	Niteesh	ghi.com
404	Gupta	Shubham	jkl.com
405	Walecha	Divya	abc.com
406	Jain	Sandeep	jkl.com
407	Mehta	Rajiv	abc.com
408	Mehra	Anand	abc.com

Orders

order_id	c_id	order_date
1	407	2017-03-03
2	405	2017-03-05
3	408	2017-01-18
4	404	2017-02-05

Using the EXISTS Operator (Example)

1. To fetch the first and last name of the customers who placed atleast one order.

```
SELECT fname, lname

FROM Customers
WHERE EXISTS (SELECT *
              FROM Orders
              WHERE Customers.customer_id = Orders.c_id);
```

fname	lname
Shubham	Gupta
Divya	Walecha
Rajiv	Mehta
Anand	Mehra

Using the EXISTS Operator (Example)

2. Using NOT with EXISTS

Fetch last and first name of the customers who has not placed any order.

```
SELECT lname, fname
FROM Customer
WHERE NOT EXISTS (SELECT *
                  FROM Orders
                  WHERE Customers.customer_id = Orders.c_id);
```

lname	fname
Singh	Dolly
Chauhan	Anuj
Kumar	Niteesh
Jain	Sandeep

Using the EXISTS Operator (Example)

3.Using EXISTS condition with DELETE statement

Delete the record of all the customer from Order Table whose last name is ‘Mehra’.

```
DELETE
FROM Orders
WHERE EXISTS (SELECT *
        FROM customers
        WHERE Customers.customer_id = Orders.cid
        AND Customers.lname = 'Mehra');
```

```
SELECT * FROM Orders;
```

order_id	c_id	order_date
1	407	2017-03-03
2	405	2017-03-05
4	404	2017-02-05

Using the EXISTS Operator (Example)

4.Using EXISTS condition with UPDATE statement

Update the lname as ‘Kumari’ of customer in Customer Table whose customer_id is 401.

```
UPDATE Customers
SET lname = 'Kumari'
WHERE EXISTS (SELECT *
        FROM Customers
        WHERE customer_id = 401);

SELECT * FROM Customers;
```

customer_id	lname	fname	website
401	Kumari	Dolly	abc.com
402	Chauhan	Anuj	def.com
403	Kumar	Niteesh	ghi.com
404	Gupta	Shubham	jkl.com
405	Walecha	Divya	abc.com
406	Jain	Sandeep	jkl.com
407	Mehta	Rajiv	abc.com
408	Mehra	Anand	abc.com

UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order

Syntax

```
SELECT column_name(s) FROM table1 [WHERE condition]
UNION [ALL]
SELECT column_name(s) FROM table2 [WHERE condition];
```

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL

UNION Operator Examples

Note: The column names in the result-set are usually equal to the column names in the first SELECT statement.

- SELECT City FROM Customers UNION SELECT City FROM Suppliers ORDER BY City; (no duplicates)
- SELECT City FROM Customers UNION ALL SELECT City FROM Suppliers ORDER BY City; (with duplicates)
- SELECT City, Country FROM Customers WHERE Country='Germany'
  UNION
  SELECT City, Country FROM Suppliers WHERE Country='Germany' ORDER BY City;

INTERSECT Operator

The SQL INTERSECT clause/operator is used to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement. This means INTERSECT returns only common rows returned by the two SELECT statements.

Just as with the UNION operator, the same rules apply when using the INTERSECT operator.

Syntax:

```
SELECT column1 [, column2 ] FROM table1 [, table2 ] [WHERE condition]
INTERSECT
SELECT column1 [, column2 ] FROM table1 [, table2 ] [WHERE condition]
```

MINUS Operator

The MINUS operator returns all the records in the first SELECT query that are not returned by the second SELECT query.

Syntax:

```
SELECT column1 [, column2 ] FROM table1 [, table2 ] [WHERE condition]
MINUS
SELECT column1 [, column2 ] FROM table1 [, table2 ] [WHERE condition]
```

MINUS Operator Example

Employee Table

EmpId	FirstName	LastName	Email	Salary	HireDate
1	'John'	'King'	'john.king@abc.com'	33000	2018-07-25
2	'James'	'Bond'			2018-07-29
3	'Neena'	'Kochhar'	'neena@test.com'	17000	2018-08-22
4	'Lex'	'De Haan'	'lex@test.com'	15000	2018-09-8
5	'Amit'	'Patel'		18000	2019-01-25
6	'Abdul'	'Kalam'	'abdul@test.com'	25000	2020-07-14

MINUS Operator Example

Employee_backup

EmpId	FirstName	LastName	Email	Salary	HireDate
1	'John'	'King'	'john.king@abc.com'	33000	2018-07-25
2	'James'	'Bond'			2018-07-29
3	'Neena'	'Kochhar'	'neena@test.com'	17000	2018-08-22
10	'Swati'	'Karia'	'swati@test.com'	16000	2018-08-22

```
SELECT * FROM Employee
MINUS
SELECT * from Employee_backup
```

MINUS Operator Example

The first query SELECT * FROM Employee will be executed first and then the second query SELECT * from Employee_backup will be executed.

The MINUS operator will return only those records from the first query result that does not exist in the second query result.

EmpId	FirstName	LastName	Email	Salary	HireDate
4	'Lex'	'De Haan'	'lex@test.com'	15000	2018-09-8
5	'Amit'	'Patel'		18000	2019-01-25
6	'Abdul'	'Kalam'	'abdul@test.com'	25000	2020-07-14

Views (Virtual tables)

- A view is a table whose contents are taken or derived from other tables.
- A view is a logical table based on a table or another view.
- A view contains no data of its own but is like a window through which data from tables can be viewed or changed.
- The tables on which a view is based are called base tables. The view is stored as a SELECT statement in the data dictionary.
- A view also has rows and columns as they are in a real table in the database.
- We can create a view by selecting fields from one or more tables present in the database.
- A View can either have all the rows of a table or specific rows based on certain condition.
- Once your view has been created, you can query the data dictionary view called USER_VIEWS to see the name of the view and the view definition.

Simple Views versus Complex Views

There are two classifications for views: simple and complex. The basic difference is related to the DML (INSERT, UPDATE, and DELETE) operations.

1. A simple view is one that:

- Derives data from only one table
- Contains no functions or groups of data
- Can perform DML operations through the view

2. A complex view is one that:

- Derives data from many tables
- Contains functions or groups of data
- Does not always allow DML operations through the view

Uses of a View

A good database should contain views due to the given reasons:

- **Restricting data access –**

  Views provide an additional level of table security by restricting access to a predetermined set of rows and columns of a table.

- **Hiding data complexity –**

  A view can hide the complexity that exists in a multiple table join.

- **Simplify commands for the user –**

  Views allows the user to select information from multiple tables without requiring the users to actually know how to perform a join.

Uses of a View

- **Store complex queries –**

  Views can be used to store complex queries.

- **Rename Columns –**

  Views can also be used to rename the columns without affecting the base tables provided the number of columns in view must match the number of columns specified in select statement.

- **Multiple view facility –**

  Different views can be created on the same table for different users.

Creating Views

CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view

[(alias[, alias]...)]

AS subquery

[WITH CHECK OPTION [CONSTRAINT constraint]]

[WITH READ ONLY [CONSTRAINT constraint]];

Symple Syntax:

CREATE VIEW view_name AS

SELECT column1, column2.....

FROM table_name

WHERE condition;

Creating Views

`OR REPLACE`	re-creates the view if it already exists
`FORCE`	creates the view regardless of whether or not the base tables exist
`NOFORCE`	creates the view only if the base tables exist (This is the default.)
`view`	is the name of the view
`alias`	specifies names for the expressions selected by the view's query (The number of aliases must match the number of expressions selected by the view.)
`subquery`	is a complete `SELECT` statement (You can use aliases for the columns in the `SELECT` list.)
`WITH CHECK OPTION`	specifies that only rows accessible to the view can be inserted or updated
`constraint`	is the name assigned to the `CHECK OPTION` constraint
`WITH READ ONLY`	ensures that no DML operations can be performed on this view

Guidelines for creating a view

- The subquery that defines a view can contain complex SELECT syntax, including joins, groups, and subqueries.
- The subquery that defines the view cannot contain an ORDER BY clause. The ORDER BY clause is specified when you retrieve data from the view.
- If you do not specify a constraint name for a view created with the WITH CHECK OPTION, the system assigns a default name in the format SYS_Cn.
- You can use the OR REPLACE option to change the definition of the view without dropping and re-creating it or regranting object privileges previously granted on it.

Creating Views

Examples:

Creating View from a single table:

```
CREATE VIEW DetailsView AS SELECT NAME, ADDRESS
FROM StudentDetails WHERE S_ID < 5;
```

*SELECT * FROM DetailsView;*

Creating View from multiple tables:

```
CREATE VIEW MarksView AS
SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS
FROM StudentDetails, StudentMarks
WHERE StudentDetails.NAME = StudentMarks.NAME;
```

*SELECT * FROM MarksView;*

Creating Views (Column aliases)

- CREATE VIEW sal

  AS SELECT employee_id ID_NUMBER, last_name NAME, salary*12 ANN_SALARY

  FROM employees WHERE department_id = 50;

- CREATE VIEW salv (ID_NUMBER, NAME, ANN_SALARY)

  AS SELECT employee_id, last_name, salary*12

  FROM employees WHERE department_id = 50;

The WITH CHECK OPTION

- It is possible to perform referential integrity checks through views. You can also enforce constraints at the database level. The view can be used to protect data integrity, but the use is very limited.
- The WITH CHECK OPTION clause specifies that INSERTs and UPDATEs performed through the view cannot create rows which the view cannot select, and therefore it allows integrity constraints and data validation checks to be enforced on data being inserted or updated.
- If there is an attempt to perform DML operations on rows that the view has not selected, an error is displayed, with the constraint name if that has been specified.

The WITH CHECK OPTION

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

UPDATE empvu20  SET  department_id = 10
WHERE employee_id = 201;
```

ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation

Note:
No rows are updated because if the department number were to change to 10, the view would no longer be able to see that employee. Therefore, with the WITH CHECK OPTION clause, the view can see only employees in department 20 and does not allow the department number for those employees to be changed through the view.

Denying DML Operations

You can ensure that no DML operations occur on your view by creating it with the WITH READ ONLY option.

Example:

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

Any attempts to remove a row from a view with a read-only constraint results in an error. ORA-01752: cannot delete from view without exactly one key-preserved table

```
DELETE FROM empvu
WHERE employee_number = 200;
```

Any attempt to insert a row or modify a row using the view with a read-only constraint results in Oracle server error: 01733: virtual column not allowed here.

Updatable view

All views are not updatable. So, UPDATE command is not applicable to all views. An updatable view is one which allows performing a UPDATE command on itself without affecting any other table.

When can a view be updated?

1. The view is defined based on one and only one table.
2. The view must include the PRIMARY KEY of the table based upon which the view has been created.
3. The view should not have any field made out of aggregate functions.
4. The view must not have any DISTINCT clause in its definition.

Updatable view

When can a view be updated?

5. The view must not have any GROUP BY or HAVING clause in its definition.
6. The view must not have any SUBQUERIES in its definitions.
7. If the view you want to update is based upon another view, the later should be updatable.
8. Any of the selected output fields (of the view) must not use constants, strings or value expressions.

Key differences between Table and View

- A table is a database object that holds information used in applications and reports. On the other hand, a view is also a database object utilized as a table and can also link to other tables.
- A table consists of rows and columns to store and organized data in a structured format, while the view is a result set of SQL statements.
- A table is structured with columns and rows, while a view is a virtual table extracted from a database.
- The table is an independent data object while views are usually depending on the table.

Key differences between Table and View

- The table is an actual or real table that exists in physical locations. On the other hand, views are the virtual or logical table that does not exist in any physical location.
- A table allows to performs add, update or delete operations on the stored data. On the other hand, we cannot perform add, update, or delete operations on any data from a view. If we want to make any changes in a view, we need to update the data in the source tables.
- We cannot replace the table object directly because it is stored as a physical entry. In contrast, we can easily use the replace option to recreate the view because it is a pseudo name to the SQL statement running behind on the database server.