Contents

[General Questions 2](#_Toc21523282)

[1) What is Database? 2](#_Toc21523283)

[2) What is DBMS? 2](#_Toc21523284)

[3) What is RDBMS? How is it different from DBMS? 2](#_Toc21523285)

[4) What is SQL? 2](#_Toc21523286)

[5) What is the difference between SQL and MySQL? 3](#_Toc21523287)

[6) What are Tables and Fields? 3](#_Toc21523288)

[7) What are Constraints in SQL? 3](#_Toc21523289)

[8) What is a Primary Key? 4](#_Toc21523290)

[9) What is a UNIQUE constraint? 5](#_Toc21523291)

[10) Difference between primary key and unique? 6](#_Toc21523292)

[11) What is a Foreign Key? 6](#_Toc21523293)

[12) Foerign Key syantax 7](#_Toc21523294)

[13) What is a foreign key with Cascade DELETE in Oracle? 8](#_Toc21523295)

[14) Using a CREATE TABLE statement 9](#_Toc21523296)

[15) What is a foreign key with "Set NULL on Delete" in Oracle? 10](#_Toc21523297)

[16) Check constraint? 12](#_Toc21523298)

[17) What is index? 15](#_Toc21523299)

[18) Syntax of Index 15](#_Toc21523300)

[19) What are different type of Index – logically ? 16](#_Toc21523301)

[20) What is rowId? 17](#_Toc21523302)

[21) Type of index – technical? 17](#_Toc21523303)

[22) Drawback of index? 25](#_Toc21523304)

[23) Cluster and non cluster index? 26](#_Toc21523305)

[Query Questions 28](#_Toc21523306)

[24) What are different types of join? 29](#_Toc21523307)

[25) Difference between full outer join vs Cross join(cartesin join)? 38](#_Toc21523308)

[26) What is Self join? 38](#_Toc21523309)

[27) What is query and sub query? 38](#_Toc21523310)

[28) Difference between union, minus, intersect? 39](#_Toc21523311)

[29) What is cursor? 40](#_Toc21523312)

[30) Difference between truncate, drop and delete? 40](#_Toc21523313)

[31) Difference between group by, where and having clause? 40](#_Toc21523314)

[32) Different type of relationship with example? One to one, one to many. 41](#_Toc21523315)

[LINKS 41](#_Toc21523316)

# General Questions

## What is Database?

A database is an organized collection of data, stored and retrieved digitally from a remote or local computer system. Databases can be vast and complex, and such databases are developed using fixed design and modeling approaches.

## What is DBMS?

DBMS stands for Database Management System. DBMS is a system software responsible for the creation, retrieval, updation and management of the database. It ensures that our data is consistent, organized and is easily accessible by serving as an interface between the database and its end users or application softwares.

## What is RDBMS? How is it different from DBMS?

RDBMS stands for Relational Database Management System. The key difference here, compared to DBMS, is that RDBMS stores data in the form of a collection of tables and relations can be defined between the common fields of these tables. Most modern database management systems like MySQL, Microsoft SQL Server, Oracle, IBM DB2 and Amazon Redshift are based on RDBMS.

## What is SQL?

SQL stands for Structured Query Language. It is the standard language for relational database management systems. It is especially useful in handling organized data comprised of entities (variables) and relations between different entities of the data.

## What is the difference between SQL and MySQL?

SQL is a standard language for retrieving and manipulating structured databases. On the contrary, MySQL is a relational database management system, like SQL Server, Oracle or IBM DB2, that is used to manage SQL databases.

## What are Tables and Fields?

A table is an organized collection of data stored in the form of rows and columns. Columns can be categorized as vertical and rows as horizontal. The columns in a table are called fields while the rows can be referred to as records.

## What are Constraints in SQL?

Constraints are used to specify the rules concerning data in the table. It can be applied for single or multiple fields in an SQL table during creation of table or after creationg using the ALTER TABLE command. The constraints are:

**NOT NULL**- Restricts NULL value from being inserted into a column.

* + - **CHECK**- Verifies that all values in a field satisfy a condition.
    - **DEFAULT**- Automatically assigns a default value if no value has been specified for the field.
    - **UNIQUE**- Ensures unique values to be inserted into the field.
    - **INDEX**- Indexes a field providing faster retrieval of records.
    - **PRIMARY KEY**- Uniquely identifies each record in a table.
    - **FOREIGN KEY**- Ensures referential integrity for a record in another table.

## What is a Primary Key?

### The PRIMARY KEY constraint uniquely identifies each row in a table. It must contain UNIQUE values and has an implicit NOT NULL constraint.  A table in SQL is strictly restricted to have one and only one primary key, which is comprised of single or multiple fields (columns).

### **CREATE** **TABLE** Students ( */\* Create table with a single field as primary key \*/*

### ID INT **NOT** **NULL**

### Name VARCHAR(255)

### **PRIMARY** **KEY** (ID)

### );

**CREATE** **TABLE** Students ( */\* Create table with multiple fields as primary key \*/*

ID INT **NOT** **NULL**

LastName VARCHAR(255)

FirstName VARCHAR(255) **NOT** **NULL**,

**CONSTRAINT** PK\_Student

**PRIMARY** **KEY** (ID, FirstName)

);

**ALTER** **TABLE** Students */\* Set a column as primary key \*/*

**ADD** **PRIMARY** **KEY** (ID);

**ALTER** **TABLE** Students */\* Set multiple columns as primary key \*/*

**ADD** **CONSTRAINT** PK\_Student */\*Naming a Primary Key\*/*

**PRIMARY** **KEY** (ID, FirstName);

CREATE TABLE table\_name

(

column1 datatype null/not null,

column2 datatype null/not null,

...

CONSTRAINT constraint\_name PRIMARY KEY (column1, column2, ... column\_n)

);

### In Oracle, a **primary key** is a single field or combination of fields that uniquely defines a record. None of the fields that are part of the primary key can contain a null value. A table can have only one primary key.

### In Oracle, a primary key can not contain more than 32 columns.

### A primary key can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.

ALTER TABLE SYNTAX:

ALTER TABLE table\_name

ADD CONSTRAINT constraint\_name PRIMARY KEY (column1, column2, ... column\_n);

<https://www.techonthenet.com/oracle/primary_keys.php>

## What is a UNIQUE constraint?

A UNIQUE constraint ensures that all values in a column are different. This provides uniqueness for the column(s) and helps identify each row uniquely. Unlike primary key, **there can be multiple unique constraints defined per table.** The code syntax for UNIQUE is quite similar to that of PRIMARY KEY and can be used interchangeably.

**CREATE** **TABLE** Students ( */\* Create table with a single field as unique \*/*

ID INT **NOT** **NULL** **UNIQUE**

Name VARCHAR(255)

);

**CREATE** **TABLE** Students ( */\* Create table with multiple fields as unique \*/*

ID INT **NOT** **NULL**

LastName VARCHAR(255)

FirstName VARCHAR(255) **NOT** **NULL**

**CONSTRAINT** PK\_Student

**UNIQUE** (ID, FirstName)

);

**ALTER** **TABLE** Students */\* Set a column as unique \*/*

**ADD** **UNIQUE** (ID);

**ALTER** **TABLE** Students */\* Set multiple columns as unique \*/*

**ADD** **CONSTRAINT** PK\_Student */\* Naming a unique constraint \*/*

**UNIQUE** (ID, FirstName);

CREATE TABLE table\_name

(

column1 datatype [ NULL | NOT NULL ],

column2 datatype [ NULL | NOT NULL ],

...

CONSTRAINT constraint\_name UNIQUE (uc\_col1, uc\_col2, ... uc\_col\_n)

);

## Difference between primary key and unique?

### Primary Key - None of the fields that are part of the primary key can contain a null value.

### Unique Constraint: Some of the fields that are part of the unique constraint can contain null values as long as the combination of values is unique.

## What is a Foreign Key?

A FOREIGN KEY comprises of single or collection of fields in a table that essentially refer to the PRIMARY KEY in another table. Foreign key constraint ensures referential integrity in the relation between two tables.   
The table with the foreign key constraint is labelled as the child table, and the table containing the candidate key is labelled as the referenced or parent table.

**CREATE** **TABLE** Students ( */\* Create table with foreign key - Way 1 \*/*

ID INT **NOT** **NULL**

Name VARCHAR(255)

LibraryID INT

**PRIMARY** **KEY** (ID)

**FOREIGN KEY** (Library\_ID) **REFERENCES** Library(LibraryID)

);

**CREATE** **TABLE** Students ( */\* Create table with foreign key - Way 2 \*/*

ID INT **NOT NULL PRIMARY KEY**

Name VARCHAR(255)

LibraryID INT **FOREIGN KEY** (Library\_ID) **REFERENCES** Library(LibraryID)

);

**ALTER** **TABLE** Students */\* Add a new foreign key \*/*

**ADD** **FOREIGN** **KEY** (LibraryID)

**REFERENCES** Library (LibraryID);

## Foerign Key syantax

### A foreign key is a way to enforce referential integrity within your Oracle database. A foreign key means that values in one table must also appear in another table.

### The referenced table is called the parent table while the table with the foreign key is called the child table. The foreign key in the child table will generally reference a [primary key](https://www.techonthenet.com/oracle/primary_keys.php) in the parent table.

### A foreign key can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.

CREATE TABLE table\_name

(

column1 datatype null/not null,

column2 datatype null/not null,

...

CONSTRAINT fk\_column

FOREIGN KEY (column1, column2, ... column\_n)

REFERENCES parent\_table (column1, column2, ... column\_n)

);

### **Example**

CREATE TABLE supplier

( supplier\_id numeric(10) not null,

supplier\_name varchar2(50) not null,

contact\_name varchar2(50),

CONSTRAINT supplier\_pk PRIMARY KEY (supplier\_id)

);

CREATE TABLE products

( product\_id numeric(10) not null,

supplier\_id numeric(10) not null,

CONSTRAINT fk\_supplier

FOREIGN KEY (supplier\_id)

REFERENCES supplier(supplier\_id)

);

ALTER TABLE table\_name

ADD CONSTRAINT constraint\_name

FOREIGN KEY (column1, column2, ... column\_n)

REFERENCES parent\_table (column1, column2, ... column\_n);

### **Example**

ALTER TABLE products

ADD CONSTRAINT fk\_supplier

FOREIGN KEY (supplier\_id)

REFERENCES supplier(supplier\_id);

## What is a foreign key with Cascade DELETE in Oracle?

A foreign key with cascade delete means that if a record in the parent table is deleted, then the corresponding records in the child table will automatically be deleted. This is called a cascade delete in Oracle.

A foreign key with a cascade delete can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.

## Using a CREATE TABLE statement

### **Syntax**

The syntax for creating a foreign key with cascade delete using a CREATE TABLE statement in Oracle/PLSQL is:

CREATE TABLE table\_name

(

column1 datatype null/not null,

column2 datatype null/not null,

...

CONSTRAINT fk\_column

FOREIGN KEY (column1, column2, ... column\_n)

REFERENCES parent\_table (column1, column2, ... column\_n)

ON DELETE CASCADE

);

### **Example**

Let's look at an example of how to create a foreign key with cascade delete using the CREATE TABLE statement in Oracle/PLSQL.

For example:

CREATE TABLE supplier

( supplier\_id numeric(10) not null,

supplier\_name varchar2(50) not null,

contact\_name varchar2(50),

CONSTRAINT supplier\_pk PRIMARY KEY (supplier\_id)

);

CREATE TABLE products

( product\_id numeric(10) not null,

supplier\_id numeric(10) not null,

CONSTRAINT fk\_supplier

FOREIGN KEY (supplier\_id)

REFERENCES supplier(supplier\_id)

ON DELETE CASCADE

);

## What is a foreign key with "Set NULL on Delete" in Oracle?

### A foreign key with "set null on delete" means that if a record in the parent table is deleted, then the corresponding records in the child table will have the foreign key fields set to null. The records in the child table will **not** be deleted.

### A foreign key with a "set null on delete" can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.

### Using a CREATE TABLE statement

### **Syntax**

The syntax for creating a foreign key using a CREATE TABLE statement is:

CREATE TABLE table\_name

(

column1 datatype null/not null,

column2 datatype null/not null,

...

CONSTRAINT fk\_column

FOREIGN KEY (column1, column2, ... column\_n)

REFERENCES parent\_table (column1, column2, ... column\_n)

ON DELETE SET NULL

);

### **Example**

CREATE TABLE supplier

( supplier\_id numeric(10) not null,

supplier\_name varchar2(50) not null,

contact\_name varchar2(50),

CONSTRAINT supplier\_pk PRIMARY KEY (supplier\_id)

);

CREATE TABLE products

( product\_id numeric(10) not null,

supplier\_id numeric(10),

CONSTRAINT fk\_supplier

FOREIGN KEY (supplier\_id)

REFERENCES supplier(supplier\_id)

ON DELETE SET NULL

);

### In this example, we've created a primary key on the supplier table called supplier\_pk. It consists of only one field - the supplier\_id field. Then we've created a foreign key called fk\_supplier on the products table that references the supplier table based on the supplier\_id field.

### Because of the set null on delete, when a record in the supplier table is deleted, all corresponding records in the products table will have the supplier\_id values set to null.

## Check constraint?

### A **check constraint** allows you to specify a condition on each row in a table.

### A check constraint can NOT be defined on a [SQL View](https://www.techonthenet.com/oracle/views.php).

### The check constraint defined on a table must refer to only columns in that table. It can not refer to columns in other tables.

### A check constraint can NOT include a [SQL Subquery](https://www.techonthenet.com/oracle/subqueries.php).

### A check constraint can be defined in either a [SQL CREATE TABLE statement](https://www.techonthenet.com/oracle/tables/create_table.php) or a [SQL ALTER TABLE statement](https://www.techonthenet.com/oracle/tables/alter_table.php).

### Using a CREATE TABLE statement

The syntax for creating a check constraint using a CREATE TABLE statement in Oracle is:

CREATE TABLE table\_name

(

column1 datatype null/not null,

column2 datatype null/not null,

...

CONSTRAINT constraint\_name CHECK (column\_name condition) [DISABLE]

);

The DISABLE keyword is optional. If you create a check constraint using the DISABLE keyword, the constraint will be created, but the condition will not be enforced.

### **Example**

CREATE TABLE suppliers

(

supplier\_id numeric(4),

supplier\_name varchar2(50),

CONSTRAINT check\_supplier\_id

CHECK (supplier\_id BETWEEN 100 and 9999)

);

In this first example, we've created a check constraint on the suppliers table called check\_supplier\_id. This constraint ensures that the supplier\_id field contains values between 100 and 9999.

CREATE TABLE suppliers

(

supplier\_id numeric(4),

supplier\_name varchar2(50),

CONSTRAINT check\_supplier\_name

CHECK (supplier\_name = upper(supplier\_name))

);

In this second example, we've created a check constraint called check\_supplier\_name. This constraint ensures that the supplier\_name column always contains uppercase characters.

### Using an ALTER TABLE statement

The syntax for creating a check constraint in an ALTER TABLE statement in Oracle is:

ALTER TABLE table\_name

ADD CONSTRAINT constraint\_name CHECK (column\_name condition) [DISABLE];

The DISABLE keyword is optional. If you create a check constraint using the DISABLE keyword, the constraint will be created, but the condition will not be enforced.

### **Example**

ALTER TABLE suppliers

ADD CONSTRAINT check\_supplier\_name

CHECK (supplier\_name IN ('IBM', 'Microsoft', 'NVIDIA'));

In this example, we've created a check constraint on the existing suppliers table called check\_supplier\_name. It ensures that the supplier\_name field only contains the following values: IBM, Microsoft, or NVIDIA.

### Drop a Check Constraint

The syntax for dropping a check constraint is:

ALTER TABLE table\_name

DROP CONSTRAINT constraint\_name;

### **Example**

ALTER TABLE suppliers

DROP CONSTRAINT check\_supplier\_id;

In this example, we're dropping a check constraint on the suppliers table called check\_supplier\_id.

### Enable a Check Constraint

The syntax for enabling a check constraint in Oracle is:

ALTER TABLE table\_name

ENABLE CONSTRAINT constraint\_name;

### **Example**

ALTER TABLE suppliers

ENABLE CONSTRAINT check\_supplier\_id;

In this example, we're enabling a check constraint on the suppliers table called check\_supplier\_id.

### Disable a Check Constraint

The syntax for disabling a check constraint in Oracle is:

ALTER TABLE table\_name

DISABLE CONSTRAINT constraint\_name;

### **Example**

ALTER TABLE suppliers

DISABLE CONSTRAINT check\_supplier\_id;

In this example, we're disabling a check constraint on the suppliers table called check\_supplier\_id.

## What is index?

### Just like we have index present in the textbooks to help us find the particular topic in the book, Oracle index behaves the same way.we have different types of indexes in oracle. Index are different structure than table, a different space where indexed column and corresponding row address is stored.

### Indexes are used to search the rows in the [oracle table](https://techgoeasy.com/oracle-create-table-statement/) quickly.  If the index is not present the select query has to read the whole table and returns the rows. With Index, the rows can be retrieved quickly

### We should create Indexes when selecting a small percentage of rows from a table (less than 2-4%). If the % of rows returned is high then index scan will be slow. It also depends on the data distribution i.e [clustering factor](https://techgoeasy.com/clustering-factor/)

### Indexes are logically and physically independent of the data in the associate table.

### Indexes are optional structures associated with tables and clusters. You can create indexes on one or more columns of a table to speed [SQL statement](https://techgoeasy.com/oracle-sql-tutorial-basic-sql-statement/) execution on that table.

### Indexes are the primary means of reducing disk I/O when properly used.

### The query decides at the beginning whether to use index or no

### The best thing with indexes is that retrieval performance of indexed data remains almost constant, even as new rows are inserted. However, the presence of many indexes on a table decreases the performance of updates, deletes, and inserts because Oracle must also update the indexes associated with the table.

### If you are  owner of table, you can create index  or if you want to create index for table in another schema then you should either have  CREATE ANY INDEX [system privilege](https://techgoeasy.com/create-user-system-privileges-object-privileges/)  or index privilege on that table

## Syntax of Index

This Oracle tutorial explains how to **create, rename and drop indexes** in Oracle with syntax and examples.

he syntax for creating an index in Oracle/PLSQL is:

CREATE [UNIQUE] INDEX index\_name

ON table\_name (column1, column2, ... column\_n)

[ COMPUTE STATISTICS ];

**COMPUTE STATISTICS**

It tells Oracle to collect statistics during the creation of the index. The statistics are then used by the optimizer to choose a "plan of execution" when SQL statements are executed.

CREATE INDEX supplier\_idx

ON supplier (supplier\_name, city);

## What are different type of Index – logically ?

### It defines the application characteristics of the Index

### **Unique or Non Unique:** Index can be Unique or non Unique. Oracle create unique index for Primary key and unique key constraints

### **Composite:** The index can be comprised of single of multiple columns. Composite indexes can speed retrieval of data for SELECT statement in which the WHERE clause references all or the leading portion of the columns in the composite index.

### **Function Based indexes:** The indexed column’s data is based on a calculation

## What is rowId?

### ROWID returns the address of each row in the table. Oracle assigns a ROWID to each row.ROWID consists of following

### The data object number of the object

### The data block in the datafile in which the row resides

### The position of the row in the data block (first row is 0)

### The datafile in which the row resides (first file is 1). The file number is relative to the [tablespace](https://techgoeasy.com/oracle-create-tablespace/).

### 

### Oracle uses ROWID internally to access rows. For instance, Oracle stores ROWID in index and uses it to access the row in the table.

### You can display ROWID of rows using SELECT command as follows:

### select rowid, emp\_name from emp;

### ROWID EMP\_NAME

### AAADC576474722aSAAA   John

### Oracle provides a package called DBMS\_ROWID to decode ROWID.

### Once a row is assigned a ROWID Oracle does not change ROWID during the lifetime of the row. But it changes when the table is rebuild.

## Type of index – technical?

### 1) B-Tree

### 2) Compressed B-Tree

### 3)  Bitmap

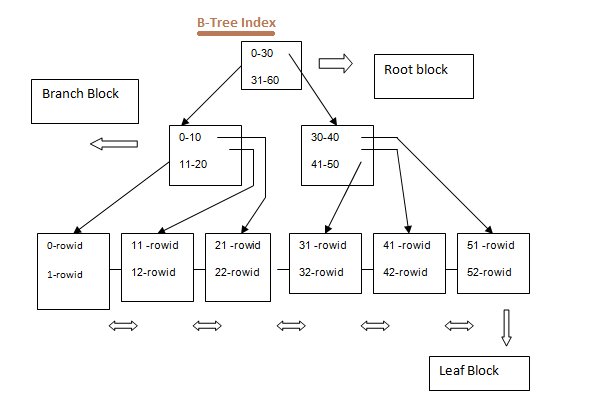
### 4) Function-Based

### 5) Reverse Key (RKI)

### 6) Index organized table (IOT).

### **B – Tree Index:**

* B-Tree Indexes (balanced tree) are the most common type of index.
* B-Tree index stored the ROWID and the index key value in a tree structure.
* When creating an index, a ROOT block is created, then BRANCH blocks are created and finally LEAF blocks.
* Each branch holds the range of data its leaf blocks hold, and each root holds the range of data its branches hold:
* B-Tree indexes are most useful on columns that appear in the where clause (SELECT … WHERE EMPNO=1).
* The Oracle server, keeps the tree balanced by splitting index blocks, when new data is inserted to the table.
* Whenever a DML statement is performed on the index’s table, index activity occurs, making the index to grow (add leaf and branches).

[](https://techgoeasy.com/wp-content/uploads/2016/03/b-tree_index.png)

**Advantages**

* All leaf blocks of the tree are at the same depth.
* B-tree indexes automatically stay balanced.
* All blocks of the B-tree are three-quarters full on the average.
* B-trees provide excellent retrieval performance for a wide range of queries, including exact match and range searches.
* Inserts, updates, and deletes are efficient, maintaining key order for fast retrieval.
* B-tree performance is good for both small and large tables, and does not degrade as the [size of a table](https://techgoeasy.com/query-to-check-table-size-in-oracle/) grows.

CREATE <UNIQUE|NON UNIQUE>  INDEX <index\_name>

ON <table\_name> (<column\_name>,<column\_name>…)

TABLESPACE <tablespace\_name>;

Example

Create index   scott.exp\_idx   on table scott.example(  name)

Tablespace TOOLS;

### **What is compressed B-tree Indexes**

Compressed B-Tree Indexes are built on large tables, in a data warehouse environment. In this type of index, duplicate occurrences of the same value are eliminated, thus reducing the amount of storage space, the index requires. In a compressed B-Tree index, for each key value, a list of ROWIDs are kept:

Specifying the COMPRESS keyword when creating an index (CREATE INDEX … COMPRESS) will create a compressed B-Tree index. A regular B-Tree index can be rebuilt using the COMPRESS keyword to compress it.

CREATE <UNIQUE|NON UNIQUE>  INDEX <index\_name>

ON <table\_name> (<column\_name>,<column\_name>…)

PCTFREE <integer>

TABLESPACE <tablespace\_name>

Compress  <column number>

### **What is Bitmap Indexes**

Bitmap Indexes are most appropriate on low distinct cardinality data (as opposed to B-Tree indexes).

This type of index, creates a binary map of all index values, and store that map in the index blocks, this means that the index will require less space than B-Tree index.

Each bit in the bitmap corresponds to a possible rowid. If the bit is set, then it means that the row with the corresponding rowid contains the key value. A mapping function converts the bit position to an actual rowid, so the bitmap index provides the same functionality as a regular index even though it uses a different representation internally. If the number of different key values is small, then bitmap indexes are very space efficient

The Bitmap index is useful on large columns with low-DML activity like marital status (M/S) or gender (M/F).

Bitmap Index structure contain a map of bits which indicate the value in the column, for example, for the GENDER column, the index block will hold the starting ROWID, the ending ROWID and the bit map:

Bitmap indexes are very useful when created on columns with low cardinality, used with the AND & OR operator in the query condition:

CREATE BITMAP  INDEX <index\_name>

ON <table\_name> (<column\_name>,<column\_name>…)

PCTFREE <integer>

TABLESPACE <tablespace\_name>

Example

CREATE BITMAP INDEX ON emp\_data(gender);

SELECT COUNT(\*) FROM emp\_data

WHERE GENDER=’M”;

**Advantages Of Bitmap Indexes**

* Reduced response time for large classes of queries
* A substantial reduction of space usage compared to other indexing techniques
* Dramatic performance gains even on very low end hardware
* Very efficient parallel DML and loads

### **Function Based Indexes**

Function-Based Indexes are indexes created on columns that a function is usually applied on.

When using a function on an indexed column, the index is ignored, therefore a function-based index is very useful for these operations.

CREATE INDEX <index\_name>

ON <table\_name> [ Function(<column\_name>,<column\_name.)]

TABLESPACE <tablespace\_name>;

Example

CREATE INDEX EMP\_IDX on EMP(UPPER(ENAME));

SELECT \*

FROM Emp

WHERE UPPER(Ename) like ‘JOHN`;

### **What is Reverse-Key Indexes**

They  are special types of B-Tree indexes and are very useful when created on columns contain sequential numbers.

When using a regular B-Tree, the index will grow to have many branches and perhaps several levels, thus causing performance degradation, the RKI solve the problem by reversing the bytes of each column key and indexing the new data.

This method distributes the data evenly in the index. Creating a RKI is done using the REVERSE keyword: CREATE INDEX … ON … REVERSE;

CREATE INDEX <index\_name>

ON <table\_name> (<column\_name>)

TABLESPACE <tablespace\_name>

REVERSE;

Example

CREATE INDEX emp\_idx i ON emp\_table (firstname,lastname) REVERSE;

### **What is Index Organized Tables (IOT) –**

When we are  using B-Tree, Bitmap and Reverse key  indexes are used for tables that store data in an unordered fashion (Heap Tables).

These indexes contain the location of the ROWID of required table row, thus allowing direct access to row data

An index-organized table differs from an ordinary table because the data for the table is held in its associated index. Changes to the table data, such as adding new rows, updating rows, or deleting rows, result in updating the index.

The index-organized table is like an ordinary table with an index on one or more of its columns, but instead of maintaining two separate storage for the table and the B-tree index, the database system maintains only a single B-tree index which contains both the encoded key value and the associated column values for the corresponding row. Rather than having a row’s rowid as the second element of the index entry, the actual data row is stored in the B-tree index. The data rows are built on the primary key for the table, and each B-tree index entry contains <primary\_key\_value, non\_primary\_key\_column\_values>. Index-organized tables are suitable for accessing data by the primary key or any key that is a valid prefix of the primary key.

There is no duplication of key values because only non-key column values are stored with the key. You can build secondary indexes to provide efficient access by other columns. Applications manipulate the index-organized table just like an ordinary table, using SQL statements. However, the database system performs all operations by manipulating the corresponding B-tree index.

Features of Index organized  table

Primary key uniquely identifies a row; primary key must be specified

Primary key based access

Logical rowid in ROWID pseudocolumn allows building secondary indexes

UNIQUE constraint not allowed but triggers are allowed

Cannot be stored in a cluster

Can contain LOB columns but not LONG columns

Distribution and replication not supported

There are 2 benefits of using IOT: 1. table rows are indexes, access to table is done using its primary key, the row is returned quickly from IOT than heap tables. 2.

CREATE TABLE command:

CREATE TABLE …

ORGANIZATION INDEX TABLESPACE … (specify this is an IOT)

PCTTHRESHOLD … (specify % of block to hold in order to store row data, valid 0-50 (default 50))

INCLUDING … (specify which column to break a row when row length exceeds PCTTHRESHOLD) OVERFLOW TABLESPACE … (specify the tablespace where the second part of the row will be stored) MAPPING TABLE; (cause creation of a mapping table, needed when creating Bitmap index on IOT)

The Mapping Table maps the index’s physical ROWIDs to logical ROWIDs in the IOT. IOT use logical ROWIDs to manage table access by index because physical ROWIDs are changed whenever data is added to or removed from the table. In order to distinct the IOT from other indexes, query the USER\_INDEXES view using the pct\_direct\_access column. Only IOT will have a non-NULL value for this column.

### **Application Domain Indexes**

Oracle provides extensible indexing to accommodate indexes on complex data types such as documents, spatial data, images, and video clips and to make use of specialized indexing techniques.

With extensible indexing, you can encapsulate application-specific index management routines as an indextype schema object and define a domain index (an application-specific index) on table columns or attributes of an object type. Extensible indexing also provides efficient processing of application-specific operators.

The application software, called the cartridge, controls the structure and content of a domain index. The Oracle server interacts with the application to build, maintain, and search the domain index. The index structure itself can be stored in the Oracle database as an index-organized table or externally as a file.

**Using Domain Indexes**

Domain indexes are built using the indexing logic supplied by a user-defined indextype. An indextype provides an efficient mechanism to access data that satisfy certain operator predicates. Typically, the user-defined indextype is part of an Oracle option, like the Spatial option.

For example, the SpatialIndextype allows efficient search and retrieval of spatial data that overlap a given bounding box.

The cartridge determines the parameters you can specify in creating and maintaining the domain index. Similarly, the [performance](https://techgoeasy.com/oracle-tutorials/oracle-performance-tuning/) and storage characteristics of the domain index are presented in the specific cartridge documentation.

So far we have covered different types of indexes in oracle with example,lets now check how to alter/drop/recreate them

## Drawback of index?

### Indexes increase performance of  select query, they can also decrease performance of data manipulation.

### Many indexes on a table can slow down INSERTS and DELETES drastically

### The more the indexes on the table, the more time inserts and delete will take.

### Similarly every change to an indexed column will need a change to index.

### So we need to choose the index very carefully and drop which are not in use.

### Though the extra space occupied by indexes is also a consideration, it may not matter much since the cost of data storage has declined substantially.

## Cluster and non cluster index?

### **1. Clustered –** Clustered index is the type of indexing that established a physical shorting order of rows.Suppose you have a table *Student\_info* which contains *ROLL\_NO* as a primary key than Clustered index which is self created on that primary key will short the *Student\_info* table as per *ROLL\_NO*. Clustered index is like Dictionary, in the dictionary shorting order is alphabetical there is no separate index page.

### **Examples:**

### **Input:**

### CREATE TABLE Student\_info

### (

### ROLL\_NO int(10) primary key,

### NAME varchar(20),

### DEPARTMENT varchar(20),

### );

### insert into Student\_info values(1410110405, 'H Agarwal', 'CSE')

### insert into Student\_info values(1410110404, 'S Samadder', 'CSE')

### insert into Student\_info values(1410110403, 'MD Irfan', 'CSE')

### SELECT \* FROM Student\_info

### **Output:**

|  |  |  |
| --- | --- | --- |
| **ROLL\_NO** | **NAME** | **DEPARTMENT** |
| 1410110403 | MD Irfan | CSE |
| 1410110404 | S Samadder | CSE |
| 1410110405 | H Agarwal | CSE |

### If we want to create Clustered index on other column then first we have to remove the primary key after that we can remove the previous index.

### Note that defining a column as a primary key makes that column the Clustered Index of that table. To make any other column, clustered index first we have to remove the previous one as follows below procedure.

### **Syntax:**

### //Drop index

### drop index table\_name.index\_name

### //Create Clustered index index

### create Clustered index IX\_table\_name\_column\_name

### on table\_name (column\_name ASC)

### **Note:** We can create only one clustered index in a table.

### **2. Non-clustered:** The Non-Clustered index is an index structure separate from the data stored in a table that reorders one or more selected columns. The non-clustered index is created to improve the performance of frequently used queries not covered by clustered index. It’s like a textbook, the index page is created separately at the beginning of that book.

### **Examples:**

### **Input:**

### CREATE TABLE Student\_info

### (

### ROLL\_NO int(10),

### NAME varcha r(20),

### DEPARTMENT varchar(20),

### );

### insert into Student\_info values(1410110405, 'H Agarwal', 'CSE')

### insert into Student\_info values(1410110404, 'S Samadder', 'CSE')

### insert into Student\_info values(1410110403, 'MD Irfan', 'CSE')

### SELECT \* FROM Student\_info

### **Output:**

|  |  |  |
| --- | --- | --- |
| **ROLL\_NO** | **NAME** | **DEPARTMENT** |
| 1410110405 | H Agarwal | CSE |
| 1410110404 | S Samadder | CSE |
| 1410110403 | MD Irfan | CSE |

### **Note:** We can create one or more Non\_Clustered index in a table.

### **Syntax:**

### //Create Non-Clustered index

### create NonClustered index IX\_table\_name\_column\_name

### on table\_name (column\_name ASC)

### Table:Student\_info

|  |
| --- |
|  |
| **ROLL\_NO** | **NAME** | **DEPARTMENT** |
| 1410110405 | H Agarwal | CSE |
| 1410110404 | S Samadder | CSE |
| 1410110403 | MD Irfan | CSE |

### **Input:** create NonClustered index IX\_Student\_info\_NAME on Student\_info (NAME ASC) **Output:**

### Index

|  |
| --- |
|  |
| **NAME** | **ROW\_ADDRESS** |
| H Agarwal | 1 |
| MD Irfan | 3 |
| S Samadder | 2 |

### **Clustered vs Non-Clustered index:**

### In a table there can be only one clustered index or one or more tha one non\_clustered index.

### In Clustered index there is no separate index storage but in Non\_Clustered index there is separate index storage for the index.

### Clustered index is slower than Non\_Clustered index.

# Query Questions

## What are different types of join?

### Oracle JOINS are used to retrieve data from multiple tables. An Oracle JOIN is performed whenever two or more tables are joined in a SQL statement.

### There are 4 different types of Oracle joins:

### Oracle INNER JOIN (or sometimes called simple join)

### Oracle LEFT OUTER JOIN (or sometimes called LEFT JOIN)

### Oracle RIGHT OUTER JOIN (or sometimes called RIGHT JOIN)

### Oracle FULL OUTER JOIN (or sometimes called FULL JOIN)

So let's discuss Oracle JOIN syntax, look at visual illustrations of Oracle JOINS, and explore Oracle JOIN examples.

### **INNER JOIN (simple join)**

### Chances are, you've already written a statement that uses an Oracle INNER JOIN. It is the most common type of join. Oracle INNER JOINS return all rows from multiple tables where the join condition is met.

### **Syntax**

### The syntax for the INNER JOIN in Oracle/PLSQL is:

### SELECT columns

### FROM table1

### INNER JOIN table2

### ON table1.column = table2.column;

### **Visual Illustration**

### In this visual diagram, the Oracle INNER JOIN returns the shaded area:

### Oracle

### The Oracle INNER JOIN would return the records where table1 and table2 intersect.

### **Example**

### Here is an example of an Oracle INNER JOIN:

### SELECT suppliers.supplier\_id, suppliers.supplier\_name, orders.order\_date

### FROM suppliers

### INNER JOIN orders

### ON suppliers.supplier\_id = orders.supplier\_id;

### This Oracle INNER JOIN example would return all rows from the suppliers and orders tables where there is a matching supplier\_id value in both the suppliers and orders tables.

### Let's look at some data to explain how the INNER JOINS work:

### We have a table called suppliers with two fields (supplier\_id and supplier\_name). It contains the following data:

| supplier\_id | supplier\_name |
| --- | --- |
| 10000 | IBM |
| 10001 | Hewlett Packard |
| 10002 | Microsoft |
| 10003 | NVIDIA |

### We have another table called orders with three fields (order\_id, supplier\_id, and order\_date). It contains the following data:

| order\_id | supplier\_id | order\_date |
| --- | --- | --- |
| 500125 | 10000 | 2003/05/12 |
| 500126 | 10001 | 2003/05/13 |
| 500127 | 10004 | 2003/05/14 |

### If we run the Oracle SELECT statement (that contains an INNER JOIN) below:

### SELECT suppliers.supplier\_id, suppliers.supplier\_name, orders.order\_date

### FROM suppliers

### INNER JOIN orders

### ON suppliers.supplier\_id = orders.supplier\_id;

### Our result set would look like this:

| supplier\_id | Name | order\_date |
| --- | --- | --- |
| 10000 | IBM | 2003/05/12 |
| 10001 | Hewlett Packard | 2003/05/13 |

### The rows for Microsoft and NVIDIA from the supplier table would be omitted, since the supplier\_id's 10002 and 10003 do not exist in both tables. The row for 500127 (order\_id) from the orders table would be omitted, since the supplier\_id 10004 does not exist in the suppliers table.

### **Old Syntax**

### As a final note, it is worth mentioning that the Oracle INNER JOIN example above could be rewritten using the older implicit syntax as follows (but we still recommend using the INNER JOIN keyword syntax):

### SELECT suppliers.supplier\_id, suppliers.supplier\_name, orders.order\_date

### FROM suppliers, orders

### WHERE suppliers.supplier\_id = orders.supplier\_id;

### LEFT JOIN

### Another type of join is called an Oracle LEFT OUTER JOIN. This type of join returns all rows from the LEFT-hand table specified in the ON condition and **only** those rows from the other table where the joined fields are equal (join condition is met).

### **Syntax**

### The syntax for the Oracle **LEFT OUTER JOIN** is:

### SELECT columns

### FROM table1

### LEFT [OUTER] JOIN table2

### ON table1.column = table2.column;

### In some databases, the LEFT OUTER JOIN keywords are replaced with LEFT JOIN.

### **Visual Illustration**

### In this visual diagram, the Oracle LEFT OUTER JOIN returns the shaded area:

### Oracle

### The Oracle LEFT OUTER JOIN would return the all records from table1 and only those records from table2 that intersect with table1.

### **Example**

### Here is an example of an Oracle LEFT OUTER JOIN:

### SELECT suppliers.supplier\_id, suppliers.supplier\_name, orders.order\_date

### FROM suppliers

### LEFT OUTER JOIN orders

### ON suppliers.supplier\_id = orders.supplier\_id;

### This LEFT OUTER JOIN example would return all rows from the suppliers table and only those rows from the orders table where the joined fields are equal.

### If a supplier\_id value in the suppliers table does not exist in the orders table, all fields in the orders table will display as <null> in the result set.

### Let's look at some data to explain how LEFT OUTER JOINS work:

### We have a table called suppliers with two fields (supplier\_id and supplier\_name). It contains the following data:

| supplier\_id | supplier\_name |
| --- | --- |
| 10000 | IBM |
| 10001 | Hewlett Packard |
| 10002 | Microsoft |
| 10003 | NVIDIA |

### We have a second table called orders with three fields (order\_id, supplier\_id, and order\_date). It contains the following data:

| order\_id | supplier\_id | order\_date |
| --- | --- | --- |
| 500125 | 10000 | 2003/05/12 |
| 500126 | 10001 | 2003/05/13 |

### If we run the SELECT statement (that contains a LEFT OUTER JOIN) below:

### SELECT suppliers.supplier\_id, suppliers.supplier\_name, orders.order\_date

### FROM suppliers

### LEFT OUTER JOIN orders

### ON suppliers.supplier\_id = orders.supplier\_id;

### Our result set would look like this:

| supplier\_id | supplier\_name | order\_date |
| --- | --- | --- |
| 10000 | IBM | 2003/05/12 |
| 10001 | Hewlett Packard | 2003/05/13 |
| 10002 | Microsoft | <null> |
| 10003 | NVIDIA | <null> |

### The rows for Microsoft and NVIDIA would be included because a LEFT OUTER JOIN was used. However, you will notice that the order\_date field for those records contains a <null> value.

### **Old Syntax**

### As a final note, it is worth mentioning that the LEFT OUTER JOIN example above could be rewritten using the older implicit syntax that utilizes the outer join operator (+) as follows (but we still recommend using the LEFT OUTER JOIN keyword syntax):

### SELECT suppliers.supplier\_id, suppliers.supplier\_name, orders.order\_date

### FROM suppliers, orders

### WHERE suppliers.supplier\_id = orders.supplier\_id(+);

### RIGHT OUTER JOIN

### Another type of join is called an Oracle RIGHT OUTER JOIN. This type of join returns all rows from the RIGHT-hand table specified in the ON condition and **only** those rows from the other table where the joined fields are equal (join condition is met).

### **Syntax**

### The syntax for the Oracle **RIGHT OUTER JOIN** is:

### SELECT columns

### FROM table1

### RIGHT [OUTER] JOIN table2

### ON table1.column = table2.column;

### In some databases, the RIGHT OUTER JOIN keywords are replaced with RIGHT JOIN.

### **Visual Illustration**

### In this visual diagram, the Oracle RIGHT OUTER JOIN returns the shaded area:

### Oracle

### The Oracle RIGHT OUTER JOIN would return the all records from table2 and only those records from table1 that intersect with table2.

### **Example**

### Here is an example of an Oracle RIGHT OUTER JOIN:

### SELECT orders.order\_id, orders.order\_date, suppliers.supplier\_name

### FROM suppliers

### RIGHT OUTER JOIN orders

### ON suppliers.supplier\_id = orders.supplier\_id;

### This RIGHT OUTER JOIN example would return all rows from the orders table and only those rows from the suppliers table where the joined fields are equal.

### If a supplier\_id value in the orders table does not exist in the suppliers table, all fields in the suppliers table will display as <null> in the result set.

### Let's look at some data to explain how RIGHT OUTER JOINS work:

### We have a table called suppliers with two fields (supplier\_id and supplier\_name). It contains the following data:

| supplier\_id | supplier\_name |
| --- | --- |
| 10000 | Apple |
| 10001 | Google |

### We have a second table called orders with three fields (order\_id, supplier\_id, and order\_date). It contains the following data:

| order\_id | supplier\_id | order\_date |
| --- | --- | --- |
| 500125 | 10000 | 2013/08/12 |
| 500126 | 10001 | 2013/08/13 |
| 500127 | 10002 | 2013/08/14 |

### If we run the SELECT statement (that contains a RIGHT OUTER JOIN) below:

### SELECT orders.order\_id, orders.order\_date, suppliers.supplier\_name

### FROM suppliers

### RIGHT OUTER JOIN orders

### ON suppliers.supplier\_id = orders.supplier\_id;

### Our result set would look like this:

| order\_id | order\_date | supplier\_name |
| --- | --- | --- |
| 500125 | 2013/08/12 | Apple |
| 500126 | 2013/08/13 | Google |
| 500127 | 2013/08/14 | <null> |

### The row for 500127 (order\_id) would be included because a RIGHT OUTER JOIN was used. However, you will notice that the supplier\_name field for that record contains a <null> value.

### **Old Syntax**

### As a final note, it is worth mentioning that the RIGHT OUTER JOIN example above could be rewritten using the older implicit syntax that utilizes the outer join operator (+) as follows (but we still recommend using the RIGHT OUTER JOIN keyword syntax):

### SELECT orders.order\_id, orders.order\_date, suppliers.supplier\_name

### FROM suppliers, orders

### WHERE suppliers.supplier\_id(+) = orders.supplier\_id;

### FULL OUTER JOIN

### Another type of join is called an Oracle FULL OUTER JOIN. This type of join returns all rows from the LEFT-hand table and RIGHT-hand table with nulls in place where the join condition is not met.

### **Syntax**

### The syntax for the Oracle **FULL OUTER JOIN** is:

### SELECT columns

### FROM table1

### FULL [OUTER] JOIN table2

### ON table1.column = table2.column;

### In some databases, the FULL OUTER JOIN keywords are replaced with FULL JOIN.

### **Visual Illustration**

### In this visual diagram, the Oracle FULL OUTER JOIN returns the shaded area:

### Oracle

### The Oracle FULL OUTER JOIN would return the all records from both table1 and table2.

### **Example**

### Here is an example of an Oracle FULL OUTER JOIN:

### SELECT suppliers.supplier\_id, suppliers.supplier\_name, orders.order\_date

### FROM suppliers

### FULL OUTER JOIN orders

### ON suppliers.supplier\_id = orders.supplier\_id;

### This FULL OUTER JOIN example would return all rows from the suppliers table and all rows from the orders table and whenever the join condition is not met, <nulls> would be extended to those fields in the result set.

### If a supplier\_id value in the suppliers table does not exist in the orders table, all fields in the orders table will display as <null> in the result set. If a supplier\_id value in the orders table does not exist in the suppliers table, all fields in the suppliers table will display as <null> in the result set.

### Let's look at some data to explain how FULL OUTER JOINS work:

### We have a table called suppliers with two fields (supplier\_id and supplier\_name). It contains the following data:

| supplier\_id | supplier\_name |
| --- | --- |
| 10000 | IBM |
| 10001 | Hewlett Packard |
| 10002 | Microsoft |
| 10003 | NVIDIA |

### We have a second table called orders with three fields (order\_id, supplier\_id, and order\_date). It contains the following data:

| order\_id | supplier\_id | order\_date |
| --- | --- | --- |
| 500125 | 10000 | 2013/08/12 |
| 500126 | 10001 | 2013/08/13 |
| 500127 | 10004 | 2013/08/14 |

### If we run the SELECT statement (that contains a FULL OUTER JOIN) below:

### SELECT suppliers.supplier\_id, suppliers.supplier\_name, orders.order\_date

### FROM suppliers

### FULL OUTER JOIN orders

### ON suppliers.supplier\_id = orders.supplier\_id;

### Our result set would look like this:

| supplier\_id | supplier\_name | order\_date |
| --- | --- | --- |
| 10000 | IBM | 2013/08/12 |
| 10001 | Hewlett Packard | 2013/08/13 |
| 10002 | Microsoft | <null> |
| 10003 | NVIDIA | <null> |
| <null> | <null> | 2013/08/14 |

### The rows for Microsoft and NVIDIA would be included because a FULL OUTER JOIN was used. However, you will notice that the order\_date field for those records contains a <null> value.

### The row for supplier\_id 10004 would be also included because a FULL OUTER JOIN was used. However, you will notice that the supplier\_id and supplier\_name field for those records contain a <null> value.

### **Old Syntax**

### As a final note, it is worth mentioning that the FULL OUTER JOIN example above could not have been written in the old syntax without using a [UNION query](https://www.techonthenet.com/oracle/union.php).

## Difference between full outer join vs Cross join(cartesin join)?

### A cross join produces a cartesian product between the two tables, returning all possible combinations of all rows. It has no on clause because you're just joining everything to everything.

### A full outer join is a combination of a left outer and right outer join. It returns all rows in both tables that match the query's where clause, and in cases where the on condition can't be satisfied for those rows it puts null values in for the unpopulated fields.

## What is Self join?

### A **self JOIN**is a case of regular join where a table is joined to itself based on some relation between its own column(s). Self-join uses the INNER JOIN or LEFT JOIN clause and a table alias is used to assign different names to the table within the query.

### **SELECT** A.emp\_id **AS** "Emp\_ID",A.emp\_name **AS** "Employee",

### B.emp\_id **AS** "Sup\_ID",B.emp\_name **AS** "Supervisor"

### **FROM** employee A, employee B

### **WHERE** A.emp\_sup = B.emp\_id;

## What is query and sub query?

### A query is a request for data or information from a database table or combination of tables. A database query can be either a select query or an action query.

### **SELECT** fname, lname */\* select query \*/*

### **FROM** myDb.students

### **WHERE** student\_id = 1;

### **UPDATE** myDB.students */\* action query \*/*

### **SET** fname = 'Captain', lname = 'America'

### **WHERE** student\_id = 1;

### A subquery is a query within another query, also known as **nested query**or **inner query**. It is used to restrict or enhance the data to be queried by the main query, thus restricting or enhancing the output of the main query respectively. For example, here we fetch the contact information for students who have enrolled for the maths subject:

### **SELECT** name, email, mob, address

### **FROM** myDb.contacts

### **WHERE** roll\_no **IN** (

### **SELECT** roll\_no

### **FROM** myDb.students

### **WHERE** subject = 'Maths');

### There are two types of subquery - **Correlated** and **Non-Correlated**.

### A **correlated**subquery cannot be considered as an independent query, but it can refer the column in a table listed in the FROM of the main query.

### A **non-correlated**subquery can be considered as an independent query and the output of subquery is substituted in the main query.

## Difference between union, minus, intersect?

### The **UNION**operator combines and returns the result-set retrieved by two or more SELECT statements. The **MINUS**operator in SQL is used to remove duplicates from the result-set obtained by the second SELECT query from the result-set obtained by the first SELECT query and then return the filtered results from the first. The **INTERSECT**clause in SQL combines the result-set fetched by the two SELECT statements where records from one match the other and then returns this intersection of result-sets.

### Certain conditions need to be met before executing either of the above statements in SQL -

### Each SELECT statement within the clause must have the same number of columns

### The columns must also have similar data types

### The columns in each SELECT statement should necessarily have the same order

### **SELECT** name **FROM** Students */\* Fetch the union of queries \*/*

### **UNION**

### **SELECT** name **FROM** Contacts;

### **SELECT** name **FROM** Students */\* Fetch the union of queries with duplicates\*/*

### **UNION ALL**

### **SELECT** name **FROM** Contacts;

### **SELECT** name **FROM** Students */\* Fetch names from students \*/*

### **MINUS** */\* that aren't present in contacts \*/*

### **SELECT** name **FROM** Contacts;

### **SELECT** name **FROM** Students */\* Fetch names from students \*/*

### **INTERSECT** */\* that are present in contacts as well \*/*

### **SELECT** name **FROM** Contacts;

## What is cursor?

### A database cursor is a control structure that allows for traversal of records in a database. Cursors, in addition, facilitates processing after traversal, such as retrieval, addition and deletion of database records. They can be viewed as a pointer to one row in a set of rows.

### Working with SQL Cursor

### **DECLARE** a cursor after any variable declaration. The cursor declaration must always be associated with a SELECT Statement.

### Open cursor to initialize the result set. The **OPEN** statement must be called before fetching rows from the result set.

### **FETCH** statement to retrieve and move to the next row in the result set.

### Call the **CLOSE** statement to deactivate the cursor.

### Finally use the **DEALLOCATE** statement to delete the cursor definition and release the associated resources.

### **DECLARE** @**name** **VARCHAR**(50) */\* Declare All Required Variables \*/*

### **DECLARE** db\_cursor **CURSOR** **FOR** */\* Declare Cursor Name\*/*

### **SELECT** name

### **FROM** myDB.students

### **WHERE** parent\_name **IN** ('Sara', 'Ansh')

### **OPEN** db\_cursor */\* Open cursor and Fetch data into @name \*/*

### **FETCH** next

### **FROM** db\_cursor

### **INTO** @**name**

### **CLOSE** db\_cursor */\* Close the cursor and deallocate the resources \*/*

### **DEALLOCATE** db\_cursor

## Difference between truncate, drop and delete?

## Difference between group by, where and having clause?

### A WHERE clause is used is filter records from a result.  The filter occurs before any groupings are made.

### A HAVING clause is used to filter values from a group. Only columns or expression in the group can be included in the HAVING clause’s conditions. But their true power lies in their ability to compare and filter based on aggregate function results.  For instance, you can select all orders totaling more than $10,000

### **GROUP BY clause** summaries identical rows into a single/distinct group and returns a single row with the summary for each group, by using appropriate [Aggregate function](https://www.geeksforgeeks.org/database-management-system-aggregate-functions/) in the SELECT list, like COUNT(), SUM(), MIN(), MAX(), AVG(), etc.

<https://oracle-base.com/articles/misc/sql-for-beginners-the-group-by-clause-and-having-clause>

## Different type of relationship with example? One to one, one to many.

### **One-to-One** - This can be defined as the relationship between two tables where each record in one table is associated with the maximum of one record in the other table.

### **One-to-Many** & **Many-to-One** - This is the most commonly used relationship where a record in a table is associated with multiple records in the other table.

### **Many-to-Many** - This is used in cases when multiple instances on both sides are needed for defining a relationship.

### **Self Referencing Relationships** - This is used when a table needs to define a relationship with itself.

### There are 3 types of [relationships](https://database.guide/what-is-a-relationship/) in relational database design. They are:

### One-to-One

### One-to-Many (or Many-to-One)

### Many-to-Many

### These are explained below.

### One-to-One

### A [row](https://database.guide/what-is-a-row/) in [table](https://database.guide/what-is-a-table/) A can have only one matching row in table B, and vice versa.

### [Diagram of a one-to-one relationship](https://database.guide/wp-content/uploads/2016/05/relationship-diagram-one-to-one.png)

### *Example of a one-to-one relationship*

### This is not a common relationship type, as the data stored in table B could just have easily been stored in table A. However, there are some valid reasons for using this relationship type. A one-to-one relationship  can be used for security purposes, to divide a large table, and various other specific purposes.

### In the above example, we could just as easily have put an HourlyRate [field](https://database.guide/what-is-a-field/) straight into the Employee table and not bothered with the Pay table. However, hourly rate could be sensitive data that only certain database users should see. So, by putting the hourly rate into a separate table, we can provide extra security around the Pay table so that only certain users can access the data in that table.

### One-to-Many (or Many-to-One)

### This is the most common relationship type. In this type of relationship, a row in table A can have many matching rows in table B, but a row in table B can have only one matching row in table A.

### [Diagram of one-to-many relationship](https://database.guide/wp-content/uploads/2016/05/relationship-diagram-one-to-many.png)

### *Example of one-to-many relationship.*

### One-to-Many relationships can also be viewed as Many-to-One relationships, depending on which way you look at it.

### In the above example, the Customer table is the “many” and the City table is the “one”. Each customer can only be assigned one city,. One city can be assigned to many customers.

### Many-to-Many

### In a many-to-many relationship, a row in table A can have many matching rows in table B, and vice versa.

### A many-to-many relationship could be thought of as two one-to-many relationships, linked by an intermediary table.

### The intermediary table is typically referred to as a “junction table” (also as a “cross-reference table”). This table is used to link the other two tables together. It does this by having two fields that reference the [primary key](https://database.guide/what-is-a-primary-key/) of each of the other two tables.

### The following is an example of a many-to-many relationship:

### [Screenshot of a many-to-many relationship in the Relationships tab.](https://database.guide/wp-content/uploads/2016/05/create_a_relationship_in_access_2013_5.png)

### *This is the Relationships tab that is displayed when you create a relationship Microsoft Access. In this case, a many-to-many relationship has just been created. The Orders table is a junction table that cross-references the Customers table with the Products table.*

### So in order to create a many-to-many relationship between the Customers table and the Products table, we created a new table called Orders.

### In the Orders table, we have a field called CustomerId and another called ProductId. The values that these fields contain should correspond with a value in the corresponding field in the referenced table. So any given value in Orders.CustomerId should also exist in the Customer.CustomerId field. If this wasn’t the case then we could have orders for customers that don’t actually exist. We could also have orders for products that don’t exist. Not good [referential integrity](https://database.guide/what-is-referential-integrity/).

### Most database systems allow you to specify whether the database should enforce referential integrity. So, when a user (or a process) attempts to insert a [foreign key](https://database.guide/what-is-a-foreign-key/) value that  doesn’t exist in the primary key field, an error will occur.

### In our example, Orders.CustomerId field is a foreign key to the Customers.CustomerId (which is the primary key of that table). And the Orders.ProductId field is a foreign key to the Products.ProductId field (which is the primary key of that table).

## What is alias?

### An alias is a feature of SQL that is supported by most, if not all, RDBMSs. It is a temporary name assigned to the table or table column for the purpose of a particular SQL query. In addition, aliasing can be employed as an obfuscation technique to secure the real names of database fields. A table alias is also called a **correlation name**.

### An alias is represented explicitly by the **AS** keyword but in some cases the same can be performed without it as well. Nevertheless, using the AS keyword is always a good practice.

### **SELECT** A.emp\_name **AS** "Employee" */\* Alias using AS keyword \*/*

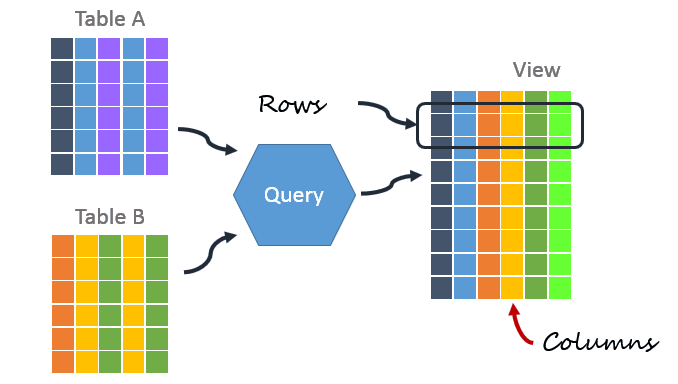
### B.emp\_name **AS** "Supervisor"

### **FROM** employee A, employee B */\* Alias without AS keyword \*/*

### **WHERE** A.emp\_sup = B.emp\_id;

## What is View?

### A view in SQL is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.



### **\*\*** Can you update the data in an Oracle VIEW?

### **Answer:** A VIEW in Oracle is created by joining one or more tables. When you update record(s) in a VIEW, it updates the records in the underlying tables that make up the View.

### So, yes, you can update the data in an Oracle VIEW providing you have the proper privileges to the underlying Oracle tables.

### **\*\*** Does the Oracle View exist if the table is dropped from the database?

### **Answer:** Yes, in Oracle, the VIEW continues to exist even after one of the tables (that the Oracle VIEW is based on) is dropped from the database. However, if you try to query the Oracle VIEW after the [table has been dropped](https://www.techonthenet.com/oracle/tables/drop_table.php), you will receive a message indicating that the Oracle VIEW has errors.

### If you [recreate the table](https://www.techonthenet.com/oracle/tables/create_table.php) (the table that you had dropped), the Oracle VIEW will again be fine.

## What is normalization and its type?

### Normalization represents the way of organizing structured data in the database efficiently. It includes creation of tables, establishing relationships between them, and defining rules for those relationships. Inconsistency and redundancy can be kept in check based on these rules, hence, adding flexibility to the database.

## What are the various form of normalization?

**First Normal Form**

A relation is in first normal form if every attribute in that relation is a **single-valued attribute**. If a relation contains composite or multi-valued attribute, it violates the first normal form. Let's consider the following **students** table. Each student in the table, has a name, his/her address and the books they issued from the public library -

### 

<https://www.youtube.com/watch?v=mUtAPbb1ECM&list=PLLGlmW7jT-nTr1ory9o2MgsOmmx2w8FB3&index=2>

## What is denormalization?

### Denormalization is the inverse process of normalization, where the normalized schema is converted into a schema which has redundant information. The performance is improved by using redundancy and keeping the redundant data consistent. The reason for performing denormalization is the overheads produced in query processor by an over-normalized structure.

# LINKS

<https://www.techonthenet.com/oracle/index.php>

<https://techgoeasy.com/oracle-indexes/>

<https://www.interviewbit.com/sql-interview-questions/>

<https://database.guide/the-3-types-of-relationships-in-database-design/>