Data Modeling:

- **Conceptual Data Modeling:** High level over view (related to business). Collecting requirement from the clients. Which are all the tables has to involve and all.
- **Logical Data Modeling:** Entity relationship between the dim/facts with PK/FK.
- **Physical Data Modeling:** Data types and constraints…..

Dimensions

- **Slowly Changing Dimension:** Slowly change (data) over a period of time

  SCD Type1: Override the old value - Space saving / No History
  SCD Type2: Create a new Row - Complete history/ space allocation is more
  SCD Type3: Create a New Column - Table size increase.
- **Conformed Dimension:** Dimension can shares across the fact tables.
- **Degenerated Dimension:** Dimension which has stored in fact table and has no associated dimension table is called as degenerate dimension.
- **Junk Dimension:** Those are not used regularly and placed in single in new dimension.(Low Cardinality attribute)
- **Role Playing Dimension:** A dimension which has multiple valid relationship with fact/dimension table is called as Roleplaying dimension.
- **Mini Dimension:** Way to manage the growth of a wide dimensions (many attributes) where some of the dimension will track the history. By create mini dimensions we can limit the growth of a dimension.

Facts

- **Additive Fact:** Columns/rows can combine to get the final output.
- **Semi Additive Fact:** Some of the columns and not for all the column. Ex: Balance of a company in 1st day and 2nd day is 5000 and 700. Based on that we cannot say that company got 5700 balance.
- **Non-Additive Fact:** No meaning full results **Ex:** Profit of a company in 1st day and 2nd day is 30% and 70%. Based on that we cannot say that company got 100% profit.
- **Fact Less Fact Table:** In a fact table only dimension attribute (id's,key's) values and there is no measurable attribute then it's called as fact less fact table.

Star vs Snow Flake Schema:

Star	Snow
De-Normalizes	Normalized
Less Redundancy	More Redundancy
Simple Query	Complex Query
Less Joins	More Joins
Performance is better	Performance is poor

OLTP VS OLAP

OLTP (Online Transactional Processing	OLAP (Online Analytical Processing)
Day to Day	Data warehousing
Inset, update and delete Commands will use	Select Statement will use
Normalized	De- Normalized
No.Of tables is more	No.Of tables is Less
Limited number of Indexes	More number of Indexes
Source	Target
End-users	BA's

Types of OLAP – Strategic business decisions

- **ROLAP (Relational OLAP):** Use relations and query to retrieve the data in transactional data to generate report.
- **MOLAP (Multi-dimensional OLAP) :** model data on Multidimensional cubes.
- **HOLPA (Hybrid OLAP) :** ROLAP+MOLAP
- **DOLAP :** Desktop OLAP
- **Mobile OLAP**
- **WOLAP –** Web OLAP
- **SOLAP –** Spatial OLAP
- **HTAP –** Hybrid Transaction OLAP

Constraints: While creating the table.

- **Not Null Constraint:** Each and every row has some values in the table
- **Default Constraint:** miss giving some values and provide the default values.
- **Unique Constraint:** All the values entered in the table are different.
- **Primary Key Constraint:** Not Null + Unique
- **Foreign Key Constraint:** Which will act as alternate key in a table.
- **Check Constraint:** Based on condition will create/restrict the columns.

ODS (Operational Data Source)

- Contains minimal period of data, Operational Monitoring and Processing

Types of keys in DBMS

- **Primary Key** – A primary is a column or set of columns in a table that uniquely identifies tuples (rows) in that table.
- **Super Key** – A super key is a set of one of more columns (attributes) to uniquely identify rows in a table. Super Key is a superset of Candidate key.
- **Candidate Key** – A super key with no redundant attribute is known as candidate key
- **Alternate Key** – Out of all candidate keys, only one gets selected as primary key, remaining keys are known as alternate or secondary keys.
- **Composite Key** – A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called composite key.
- **Foreign Key** – Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.
- **Surrogate Key:** System generated unique key.

Functional dependency

- The attributes of a table is said to be dependent on each other when an attribute of a table uniquely identifies another attribute of the same table.

Trivial Functional Dependencies

- A **trivial functional dependency** occurs when you describe a functional dependency of an attribute on a collection of attributes that includes the original attribute. For example, "{A, B} -> B" is a trivial functional dependency, as is "{name, SSN} -> SSN". This type of functional dependency is called trivial because it can be derived from common sense. It is obvious that if you already know the value of B, then the value of B can be uniquely determined by that knowledge.

Full Functional Dependencies

- A **full functional dependency** occurs when you already meet the requirements for a functional dependency and the set of attributes on the left side of the functional dependency statement cannot be reduced any further. For example, "{SSN, age} -> name" is a functional dependency, but it is not a full functional dependency because you can remove age from the left side of the statement without impacting the dependency relationship.

Transitive Dependencies

- **Transitive dependencies** occur when there is an indirect relationship that causes a functional dependency. For example, "A -> C" is a transitive dependency when it is true only because both "A -> B" and "B -> C" are true.

Multivalued Dependencies

Multivalued dependencies occur when the presence of one or more rows in a table implies the presence of one or more other rows in that same table. For example, imagine a car company that manufactures many models of car, but always makes both red and blue colors of each model. If you have a table that contains the model name, color and year of each car the company manufactures, there is a multivalued dependency in that table. If there is a row for a certain model name and year in blue, there must also be a similar row corresponding to the red version of that same car.

Normalization of Database

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anamolies. It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables.

Normalization is used for mainly two purpose,

- Eliminating reduntant(useless) data.
- Ensuring data dependencies make sense i.e data is logically stored.

Problem Without Normalization

Without Normalization, it becomes difficult to handle and update the database, without facing data loss. Insertion, Updation and Deletion Anamolies are very frequent if Database is not Normalized. To understand these anomalies let us take an example of **Student** table.

S_id	**S_Name**	**S_Address**	**Subject_opted**
401	Adam	Noida	Bio
402	Alex	Panipat	Maths
403	Stuart	Jammu	Maths
404	Adam	Noida	Physics

- **Updation Anamoly :** To update address of a student who occurs twice or more than twice in a table, we will have to update **S_Address** column in all the rows, else data will become inconsistent.
- **Insertion Anamoly :** Suppose for a new admission, we have a Student id(S_id), name and address of a student but if student has not opted for any subjects yet then we have to insert **NULL** there, leading to Insertion Anamoly.
- **Deletion Anamoly :** If (S_id) 401 has only one subject and temporarily he drops it, when we delete that row, entire student record will be deleted along with it.

Normalization Rule

Normalization rule are divided into following normal form.

1. First Normal Form
2. Second Normal Form
3. Third Normal Form
4. BCNF

First Normal Form (1NF)

As per First Normal Form, no two Rows of data must contain repeating group of information i.e each set of column must have a unique value, such that multiple columns cannot be used to fetch the same row. Each table should be organized into rows, and each row should have a primary key that distinguishes it as unique.

The **Primary key** is usually a single column, but sometimes more than one column can be combined to create a single primary key. For example consider a table which is not in First normal form

Student Table :

Student	**Age**	**Subject**
Adam	15	Biology, Maths
Alex	14	Maths
Stuart	17	Maths

In First Normal Form, any row must not have a column in which more than one value is saved, like separated with commas. Rather than that, we must separate such data into multiple rows.

Student Table following 1NF will be :

Student	**Age**	**Subject**
Adam	15	Biology
Adam	15	Maths
Alex	14	Maths
Stuart	17	Maths

Using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

Second Normal Form (2NF)

As per the Second Normal Form there must not be any partial dependency of any column on primary key. It means that for a table that has concatenated primary key, each column in the table that is not part of the primary key must depend upon the entire concatenated key for its existence. If any column depends only on one part of the concatenated key, then the table fails **Second normal form**.

In example of First Normal Form there are two rows for Adam, to include multiple subjects that he has opted for. While this is searchable, and follows First normal form, it is an inefficient use of space. Also in the above Table in First Normal Form, while the candidate key is {**Student**, **Subject**}, **Age** of Student only depends on Student column, which is incorrect as per Second Normal Form. To achieve second normal form, it would be helpful to split out the subjects into an independent table, and match them up using the student names as foreign keys.

New Student Table following 2NF will be :

Student	**Age**
Adam	15
Alex	14
Stuart	17

In Student Table the candidate key will be **Student** column, because all other column i.e **Age** is dependent on it.

New Subject Table introduced for 2NF will be :

Student	**Subject**
Adam	Biology
Adam	Maths
Alex	Maths
Stuart	Maths

In Subject Table the candidate key will be {**Student**, **Subject**} column. Now, both the above tables qualifies for Second Normal Form and will never suffer from Update Anomalies. Although there are a few complex cases in which table in Second Normal Form suffers Update Anomalies, and to handle those scenarios Third Normal Form is there.

Third Normal Form (3NF)

Third Normal form applies that every non-prime attribute of table must be dependent on primary key, or we can say that, there should not be the case that a non-prime attribute is determined by another non-prime attribute. So this *transitive functional dependency* should be removed from the table and also the table must be in **Second Normal form**. For example, consider a table with following fields.

Student_Detail Table :

Student_id Student_name DOB Street city State Zip

In this table Student_id is Primary key, but street, city and state depends upon Zip. The dependency between zip and other fields is called **transitive dependency**. Hence to apply **3NF**, we need to move the street, city and state to new table, with **Zip** as primary key.

New Student_Detail Table :

Student_id Student_name DOB Zip

Address Table :

Zip Street city state

The advantage of removing transtive dependency is,

- Amount of data duplication is reduced.
- Data integrity achieved.

Boyce and Codd Normal Form (BCNF)

Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with certain type of anamoly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, following conditions must be satisfied:

- R must be in 3rd Normal Form
- and, for each functional dependency (X -> Y), X should be a super Key.

Consider the following relationship : **R (A,B,C,D)**

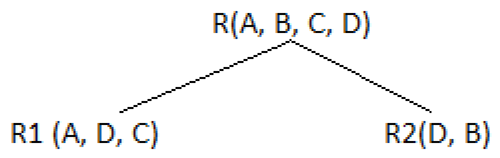
and following dependencies :

A -> BCD
BC -> AD
D -> B

Above relationship is already in 3rd NF. Keys are **A** and **BC.**

Hence, in the functional dependency, **A -> BCD**, A is the super key.
in second relation, **BC -> AD**, BC is also a key.
but in, **D -> B**, D is not a key.

Hence we can break our relationship R into two relationships **R1** and **R2**.



Breaking, table into two tables, one with A, D and C while the other with D and B.

DDL, DML, and DCL

- **DDL (Data Definition Language):** Create, Alter, Drop (remove the existing entries) and Truncate (remove all the rows in a table)
- **DML (Data Manipulation Language):** Select, Insert, Update, Delete (remove selected rows), Bulk Insert (Import a data file into a database table/view) and Merge (join with DML commands).
- **DCL (Data Control Language):** Grant, Revoke, Execute as statement and Execute as Clause and Revert.
- **TCL (Transaction Control Language):** Begin, Commit and Roll Back

Rank, Dence_rank and Row_number

- **Row_number()** : Used for generating serial number
- **Dense_rank ()** will give continuous rank.
- **Rank ():** Will skip rank in case of clash of rank.

Views vs Materialized views

Views	**Materialized Views**
Query result is not stored in the disk or database/virtual table	Materialized view allow to store query result in disk or table.
when we create view using any table, rowid of view is same as original table	Materialized view rowid is different.
Latest data	Refresh the views to get the latest data
Performance is less	Performance is more
Logical view or Table, no separate copy	we get physically separate copy of table
Trigger is not required to refresh	Trigger is required to refresh

Indexes

An index is an optional structure, associated with a table or table cluster, that can sometimes speed data access. By creating an index on one or more columns of a table, you gain the ability in some cases to retrieve a small set of randomly distributed rows from the table. Indexes are one of many means of reducing disk I/O.

If a heap-organized table has no indexes, then the database must perform a full table scan to find a value.

Indexes have the following properties:

- Usability
- Visibility

Composite Indexes

A composite index, also called a concatenated index, is an index on multiple columns in a table. Columns in a composite index should appear in the order that makes the most sense for the queries that will retrieve data and need not be adjacent in the table.

Composite indexes can speed retrieval of data for SELECT statements in which the WHERE clause references all or the leading portion of the columns in the composite index. Therefore, the order of the columns used in the definition is important. In general, the most commonly accessed columns go first.

Unique and Nonunique Indexes

Indexes can be unique or nonunique. Unique indexes guarantee that no two rows of a table have duplicate values in the key column or columns. For example, no two employees can have the same employee ID. Thus, in a unique index, one rowid exists for each data value. The data in the leaf blocks is sorted only by key.

Nonunique indexes permit duplicates values in the indexed column or columns. For example, the first_name column of the employees table may contain multiple Mike values. For a nonunique index, the rowid is included in the key in sorted order, so nonunique indexes are sorted by the index key and rowid (ascending).

Types of Indexes

Oracle Database provides several indexing schemes, which provide complementary performance functionality. The indexes can be categorized as follows:

- B-tree indexes

  These indexes are the standard index type. They are excellent for primary key and highly-selective indexes. Used as concatenated indexes, B-tree indexes can retrieve data sorted by the indexed columns. B-tree indexes have the following subtypes:

  - Index-organized tables

    An index-organized table differs from a heap-organized because the data is itself the index. See "Overview of Index-Organized Tables".

  - Reverse key indexes

    In this type of index, the bytes of the index key are reversed, for example, 103 is stored as 301. The reversal of bytes spreads out inserts into the index over many blocks. See "Reverse Key Indexes".

  - Descending indexes

This type of index stores data on a particular column or columns in descending order. See "Ascending and Descending Indexes".

- B-tree cluster indexes

  This type of index is used to index a table cluster key. Instead of pointing to a row, the key points to the block that contains rows related to the cluster key. See "Overview of Indexed Clusters".

- Bitmap and bitmap join indexes

  In a bitmap index, an index entry uses a bitmap to point to multiple rows. In contrast, a B-tree index entry points to a single row. A bitmap join index is a bitmap index for the join of two or more tables. See "Bitmap Indexes".

- Function-based indexes

  This type of index includes columns that are either transformed by a function, such as the UPPER function, or included in an expression. B-tree or bitmap indexes can be function-based. See "Function-Based Indexes".

- Application domain indexes

  This type of index is created by a user for data in an application-specific domain. The physical index need not use a traditional index structure and can be stored either in the Oracle database as tables or externally as a file. See "Application Domain Indexes".

See Also:

Oracle Database Performance Tuning Guide to learn about different index types

SOME IMPORTANT QUERYS IN ORACLE

-- -- to display all records with row number

select rownum,typ,werk,werk_ressort,jahr,monat,atnr,kennzahl,gf from cog_contrib_common.ol_test

--to display the nth row in a table

select * from cog_contrib_common.ol_test a where &n=(select count(rowid) from Cog_contrib_common.ol_test b where a.rowid>=b.rowid)

-- to display the records with alternate rows or(even/odd) rows

select sname,sno,city from (select rownum r, sname,sno,city from student2) where mod(r,2) = 1

--select first n rows from the tabel

```
select * from

(select rownum a,typ,werk,werk_ressort,jahr,monat,atnr,kennzahl,gf from
cog_contrib_common.ol_test)

where a<11
```

--select last n rows from the tabel

```
select * from

(select rownum a,typ,werk,werk_ressort,jahr,monat,atnr,kennzahl,gf from
cog_contrib_common.ol_test)

where a>(select max(rownum-10) from cog_contrib_common.ol_test)
```

-- to display particular record

```
select level,max(sno) from cog_contrib_common.student2

where level=&level connect by prior sno>sno group by sno;
```

-- to display the record which have duplicate only

```
select count(*),typ,werk,werk_ressort,jahr,monat,atnr,kennzahl,gf from
cog_contrib_common.ol_test group by typ,werk,werk_ressort,jahr,monat,atnr,kennzahl,gf
having count(*)>1;
```

BLOB

A BLOB data type is a binary string with a varying length which is used in storing two gigabytes memory. Length should be stated in Bytes for BLOB

NVL

The NVL function is used for replacing NULL values with given or another value. E.g. NVL(Value, replace value)

COALESCE

COALESCE function is used to return the value which is set to be not null in the list. If all values in the list are null, then the coalesce function will return NULL.

Where vs Having

Where: will restrict the rows

Having: Will restrict the groups of returned rows

DECODE (Alternate of Case)

`DECODE` compares `expr` to each `search` value one by one. If `expr` is equal to a `search`, then Oracle Database returns the corresponding `result`. If no match is found, then Oracle returns `default`. If `default` is omitted, then Oracle returns null.

```
Ex: SELECT product_id,
       DECODE (warehouse_id, 1, 'Southlake',
                             2, 'San Francisco',
                             3, 'New Jersey',
                             4, 'Seattle',
                                'Non domestic')
       "Location of inventory" FROM inventories
       WHERE product_id < 1775;
```

SUBSTR VS INSTR

SUBSTR returns specific portion of a string and INSTR provides character position in which a pattern is found in a string.

SUBSTR returns string whereas INSTR returns numeric.

Union VS Union All

Union: Will eliminates duplicates

Union All : It will retrieve all the rows from two tables.