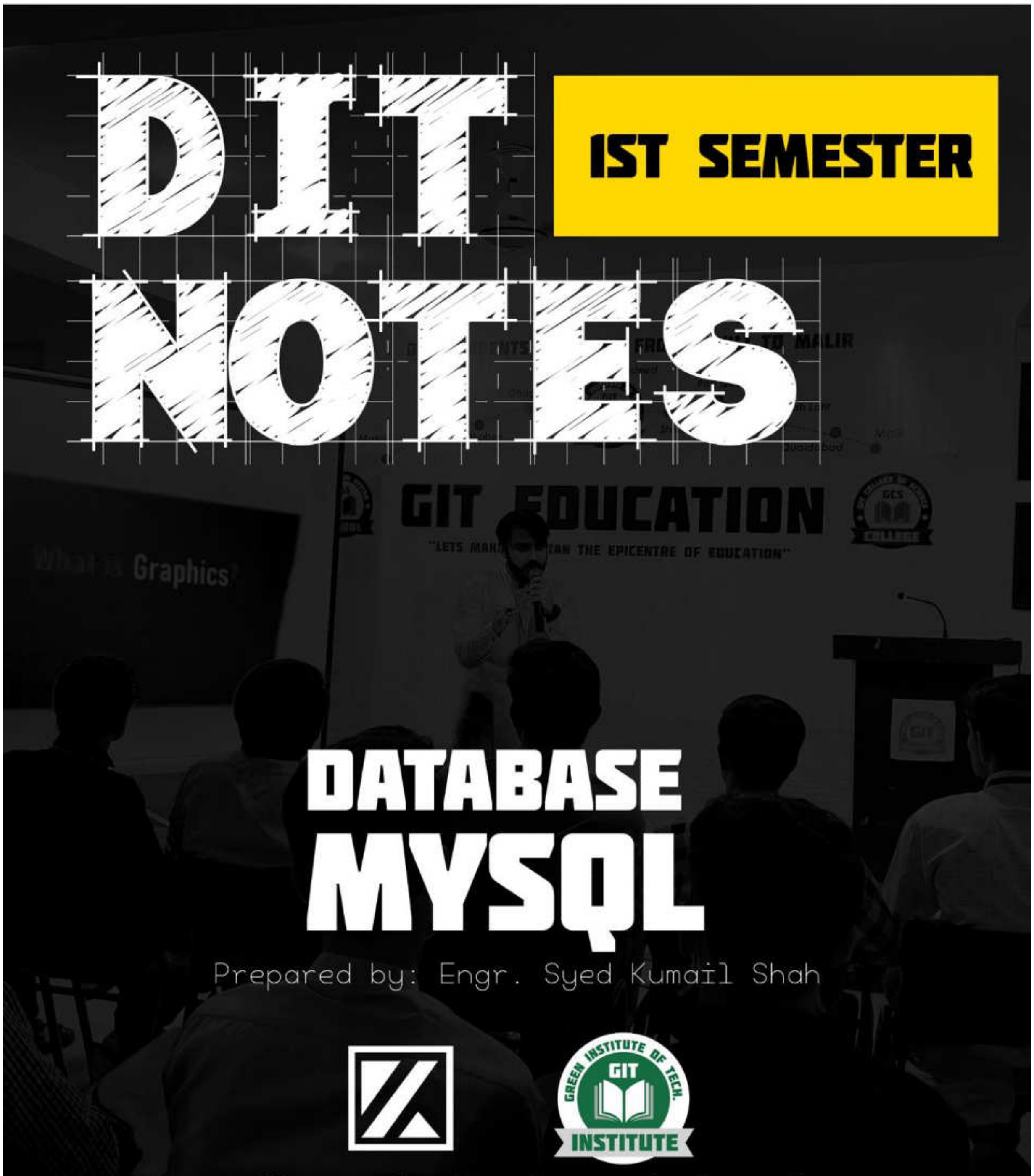
DIT NOTES

IST SEMESTER



DATABASE MYSQL

Prepared by: Engr. Syed Kumail Shah

New & Updated Book

DATABASE MYSQL Technical Board

Course Title: Database MYSQL

Objectives
To give the solid information about Database MYSQL as it is the basis of modern tools and technology. The course outline has been described in detail and in appropriate way covering different areas of Database MYSQL in the field of IT. Some topics have been restructured and added to the course outline which is needed for computer applications.

Course Duration	Theory	50 hours	2 hours per week
	Practical	50 hours	1 hours per week

Assessment Approach: Total assessment based on final examination and practical

Theory	100 Marks
Practical	100 Marks

Reference Books
Any reference book that covered the related topics

Introduction to Database Contents

(b) Group Function

- (xx) Min
- (xxi) Max
- (xxii) AVG
- (xxiii) SUM
- (xxiv) COUNT

(c) Conversion Function

- (xxv) TO-CHAR
- (xxvi) TO-NUMBER
- (xxvii) TO-DATE

(d Introduction Views and data dictionary

- (xxviii) Cerate simple / complex view
- (xxix) Importance of view in multi user environment.

(e UNDERSTANDING/ Managing User Privileges and Roles

- (i) Introduction to Multi user environment
- (ii) Definition of Role and privileges
- (xii) CREATE USER
- (xiii) CREATE ROLE
- (xiv) DROP USER
- (xv) CRANT PRIVILEGE (DBA,ALL,SELECT, UPDATE, DELETE, INSERT)
- (xvi) Revoking User Privileges and Roles.

Introduction to Data base

Data:

(1) Data is information that has been translated into a form that is more convenient to move or process. Relative to today's computers and transmission media, data is information converted into binary digital form.

Or Combination of raw facts and Figures about an entity is called data.

Types of data:

There are different types of data

i. Alphabetic data type:-It consist letter from A-Z, capital or from a-z small letter, e.g. Abid

Peshawar, Pakistan, Khan ii. Numeric data type;- It consist of digit from 0-9 e.g. 123, 567 etc

iii. Alphanumeric data type:- It consist alphabetic letter as well as numeric digit. Street no A/10 etc

iv. Graphic data: It consists tables, charts, graphics and statements v. Audio data: It consists only sounds. For example radio news.

vi. Video data: It consists photos, image .and moving picture. Such as TV news.

vii. Mixed data: It consists more than one type of data. Such as the combination of audio and video.

Information:

Meaningful, organized and processed form of data is called information**.**

To organize the Data in meaningful form upon which people can take necessary decision is called Information, e.g. 2, 1, 5.4 when sorted it become 1,2,4,5 which is information. Information is the meaningful, processed data, which is relevant and accurate and there by can be used in decision-making. Examples are voucher, bills, fee registration cards or library cards.

Metadata: Metadata describes other data. It provides information about a certain item's content. For example, an image may include metadata that describes how large the picture is, the color depth, the image resolution, when the image was created, and other data. A text document's metadata may contain information about how long the document is, who the author is, when the document was written, and a short summary of the document. Web pages often include metadata in the form of meta tags. Description and keywords meta tags are commonly used to describe the Web page's content. Most search engines use this data when adding pages to their search index.

Database:

A database is a collection of information that is organized so that it can easily be accessed, managed, and updated. In one view, databases can be classified according to types of content: bibliographic, full-text, numeric, and images.

In computing, databases are sometimes classified according to their organizational approach. The most prevalent approach is the relational database, a tabular database in which data is defined so that it can be reorganized and accessed in a number of different ways. A distributed database is one that can be dispersed or replicated among different points in a network. An object-oriented programming database is one that is congruent with the data defined in object classes and subclasses.

Advantages of Database System:

Database approach came into existence due to the bottlenecks of file processing system. In the database approach, the data is stored at a central location and is shared among multiple users. Thus, the main advantage of DBMS is centralized data management. The centralized nature of database system provides several advantages, which overcome the limitations of the conventional file processing system. These advantages are listed here.

- **Controlled data redundancy:** During database design, various files are integrated and each logical data item is stored at central location. This eliminates replicating the data item in different files, and ensures consistency and saves the storage space. Note that the redundancy in the database systems cannot be eliminated completely as there could be some performance and technical reasons for having some amount of redundancy. However, the DBMS should be capable of controlling this redundancy in order to avoid data inconsistencies.

- **Enforcing data integrity:** In database approach, enforcing data integrity is much easier. Various integrity constraints are identified by database designer during database design. Some of these data integrity constraints can be enforced automatically by the DBMS, and others may have to be checked by the application programs.

- **Data sharing:** The data stored in the database can be shared among multiple users or application programs. Moreover, new applications can be developed to use the same stored data. Due to shared data, it is possible to satisfy the data requirements

  of the new applications without having to create any additional data or with minimal modification.

- **Ease of application development:** The application programmer needs to develop the application programs according to the users" needs. The other issues like concurrent access, security, data integrity, etc., are handled by the DBMS itself. This makes the application development an easier task.

- **Data security:** Since the data is stored centrally, enforcing security constraints is much easier. The DBMS ensures that the only means of access to the database is through an authorized channel. Hence, data security checks can be carried out whenever access is attempted to sensitive data. To ensure security, a DBMS provides security tools such as user codes and passwords. Different checks can be established for each type of access (addition, modification, deletion, etc.) to each piece of information in the database.

- **Multiple user interfaces:** In order to meet the needs of various users having different technical knowledge, DBMS provides different types of interfaces such as query languages, application program interfaces, and graphical user interfaces (GUI) that include forms-style and menu-driven interfaces. A form-style interface displays a form to each user and user interacts using these forms. In menu-driven interface, the user interaction is through lists of options known as menus.

- **Backup and recovery:** The DBMS provides backup and recovery subsystem that is responsible for recovery from hardware and software failures. For example, if the failure occurs in between the transaction, the DBMS recovery subsystem either reverts back the database to the state which existed prior to the start of the transaction or resumes the transaction from the point it was interrupted so that its complete effect can be recorded in the database.

- **Program-data independence:** The independence between the programs and the data is known as program-data independence (or simply data independence). It is an important characteristic of DBMS as it allows changing the structure of the database without making any changes in the application programs that are using the database. To provide a high degree of data independence, the definition or the description of the database structure (structure of each file, the type and storage format of each data item) and various constraints on the data are stored separately in DBMS catalog

- **Data abstraction:** The property of DBMS that allows program-data independence is known as data abstraction. Data abstraction allows the database system to provide an abstract view of the data to its users without giving the physical storage and implementation details.

- **Supports multiple views of the data:** A database can be accessed by many users and each of them may have a different perspective or view of the data. A database system provides a facility to define different views of the data for different users. A view is a subset of the database that contains virtual data derived from the database files but it does not exist in physical form. That is, no physical file is created for storing the data values of the view; rather, only the definition of the view is stored.

Disadvantages of Database:

Although the database system yields considerable advantages over previous data management approaches, database systems do carry significant disadvantages. For example:

1. Increased costs.

Database systems require sophisticated hardware and software and highly skilled personnel. The cost of maintaining the hardware, software, and personnel required to operate and manage a database system can be substantial. Training, licensing, and regulation compliance costs are often overlooked when database systems are implemented.

2. Management complexity.

Database systems interface with many different technologies and have a significant impact on a company"s resources and culture. The changes introduced by the adoption of a database system must be properly managed to ensure that they help advance the company"s objectives. Given the fact that database systems hold crucial company data that are accessed from multiple sources, security issues must be assessed constantly.

3. Maintaining currency.

To maximize the efficiency of the database system, you must keep your system current. Therefore, you must perform frequent updates and apply the latest patches and security measures to all components. Because database technology advances rapidly, personnel training costs tend to be significant. Vendor dependence. Given the heavy investment in technology and personnel training, companies might be reluctant to change database vendors. As a consequence, vendors are less likely to offer pricing point advantages to existing customers, and those customers might be limited in their choice of database system components.

4. Frequent upgrade/replacement cycles.

DBMS vendors frequently upgrade their products by adding new functionality. Such new features often come bundled in new upgrade versions of the software. Some of these versions require hardware upgrades. Not only do the upgrades themselves cost money, but it also costs money to train database users and administrators to properly use and manage the new features.

Data base Model

Database Models:

A database model or simply a data model is an abstract model that describes how the data is represented and used. A data model consists of a set of data structures and conceptual tools that is used to describe the structure (data types, relationships, and constraints) of a database.A data model not only describes the structure of the data, it also defines a set of operations that can be performed on the data. A data model generally consists of **data model theory**, which is a formal description of how data may be structured and used, and **data model instance**, which is a practical data model designed for a particular application. The process of applying a data model theory to create a data model instance is known as **data modeling**.

Depending on the concept they use to model the structure of the database, the data models are categorized into three types, namely, *high-level* or conceptual data models, representational or *implementation data models* and *low-level* or physical data models.

Conceptual Data Model:

Conceptual data model describes the information used by an organization in a way that is independent of any implementation-level issues and details. The main advantage of conceptual data model is that it is independent of implementation details and hence, can be understood even by the end users having non-technical background. The most popular conceptual data model, that is, entity-relationship (E-R) model .

Representational Data Model:

The representational or implementation data models hide some data storage details from the users; however, can be implemented directly on a computer system. Representational data models are used most frequently in all traditional commercial DBMSs. The various representational data models are discussed here.

Hierarchical Data Model:

The hierarchical data model is the oldest type of data model, developed by IBM in 1968. This data model organizes the data in a tree-like structure, in which each **child node** (also known as **dependents**) can have only one **parent node**. The database based on the hierarchical data model comprises a set of records connected to one another through links. The link is an association between two or more records. The top of the tree structure consists of a single node that does not have any parent and is called the **root node**.

The root may have any number of dependents; each of these dependents may have any number of lower level dependents. Each child node can have only one parent node and a parent node can have any number of (many) child nodes. It, therefore, represents only one-to-one and one-to-many relationships. The collection of same type of records is known as a **record type.**

The main advantage of the hierarchical data model is that the data access is quite predictable in the structure and, therefore, both the retrieval and updates can be highly optimized by the DBMS. However, the main drawback of this model is that the links are „hard coded" into the data structure, that is, the link is permanently established and cannot be modified. The hard coding makes the hierarchical model rigid. In addition, the physical links make it difficult to expand or modify the database and the changes require substantial redesigning efforts.

Network Data Model:

The first specification of network data model was presented by Conference on Data Systems Languages (CODASYL) in 1969, followed by the second specification in 1971. It is powerful but complicated. In a network model the data is also represented by a collection of records, and relationships among data are represented by links. However, the link in a network data model represents an association between precisely two records. Like hierarchical data model, each record of a particular record type represents a node. However, unlike hierarchical data model, all the nodes are linked to each other without any hierarchy. The main difference between hierarchical and network data model is that in hierarchical data model, the data is organized in the form of trees and in network data model, the data is organized in the form of graphs.

The main advantage of network data model is that a parent node can have many child nodes and a child can also have many parent nodes. Thus, the network model permits the modeling of many-to-many relationships in data. The main limitation of the network data model is that it can be quite complicated to maintain all the links and a single broken link can lead to problems in the database.

Relational Data Model:

The relational data model was developed by E. F. Codd in 1970. In the relational data model, unlike the hierarchical and network models, there are no physical links. All data is maintained in the form of tables (generally, known as **relations**) consisting of rows and columns. Each row (record) represents an entity and a column (field) represents an attribute of the entity. The relationship between the two tables is implemented through a common attribute in the tables and not by physical links or pointers. This makes the querying much easier in a relational database system than in the hierarchical or network database systems. Thus, the relational model has become more programmer friendly and much more dominant and popular in both industrial and academic scenarios. Oracle, Sybase, DB2, Ingres, Informix, MS-SQL Server are few of the popular relational DBMS.

Object-Based Data Model:

In the recent years, the object-oriented paradigm has been applied to database technology, creating two new data models known as object-oriented data model and object-relational data model. The **object-oriented data model** extends the concepts of object-oriented programming language with persistence, versioning, concurrency control, data recovery, security, and other database capabilities. On the other hand, the **object-relational data model** is an extension of relational data model. It combines the features of both the relational data model and object-oriented data model.

Semi structured Data Model:

Unlike other data models, where every data item of a particular type must have the same set of attributes, the semi structured data model allows individual data items of the same type to have different set of attributes. In semi structured data model, the information about the description of the data (schema) is contained within the data itself, which is sometimes called *self-describing* data. In such databases there is no clear separation between the data and the schema and thus, allowing data of any type. Semi structured data model has recently emerged as an important topic of study for different reasons given here.

- There are data sources such as the Web, which is to be treated as databases; however, they cannot be constrained by a schema.
- The need of flexible format for data exchange between heterogeneous databases.
- To facilitate browsing of data.

Semi structured data model facilitates data exchange among heterogeneous data sources. It helps to discover new data easily and store it. It also facilitates querying the database without knowing the data types. However, it loses the data type information.

Physical Data Model:

Physical data model describes the data in terms of a collection of files, indices, and other storage structures such as record formats, record ordering, and access paths. This model specifies how the database will be executed in a particular DBMS software such as Oracle, Sybase, etc., by taking into account the facilities and constraints of a given database management system. It also describes how the data is stored on disk and what access methods are available to it.

Introduction to DBMS

A **database management system (DBMS)** is a software package with computer programs that controls the creation, maintenance, and use of a database. It allows organizations to conveniently develop databases for various applications. A database is an integrated collection of data records, files, and other objects. A DBMS allows different user application programs to concurrently access the same database. DBMS may use a variety of database models, such as the relational model or object model, to conveniently describe and support applications. It typically supports query languages, which are in fact high-level programming languages, dedicated database languages that considerably simplify writing database application programs. Database languages also simplify the database organization as well as retrieving and presenting information from it. A DBMS provides facilities for controlling data access, enforcing data integrity, managing concurrency control, and recovering the database after failures and restoring it from backup files, as well as maintaining database security.

Function of DBMS:

There are several functions that a DBMS performs to ensure data integrity and consistency of data in the database. The ten functions in the DBMS are: data dictionary management, data storage management, data transformation and presentation, security management, multiuser access control, backup and recovery management, data integrity management, database access languages and application programming interfaces, database communication interfaces, and transaction management.

1. Data Dictionary Management

Data Dictionary is where the DBMS stores definitions of the data elements and their relationships (metadata). The DBMS uses this function to look up the required data component structures and relationships. When programs access data in a database they are basically going through the DBMS. This function removes structural and data dependency and provides the user with data abstraction. In turn, this makes things a lot easier on the end user. The Data Dictionary is often hidden from the user and is used by Database Administrators and Programmers.

2. Data Storage Management

This particular function is used for the storage of data and any related data entry forms or screen definitions, report definitions, data validation rules, procedural code, and structures that can handle video and picture formats. Users do not need to know how data is stored or manipulated. Also involved with this structure is a term called performance tuning that relates to a database"s efficiency in relation to storage and access speed.

3. Data Transformation and Presentation

This function exists to transform any data entered into required data structures. By using the data transformation and presentation function the DBMS can determine the difference between logical and physical data formats.

4. Security Management

This is one of the most important functions in the DBMS. Security management sets rules that determine specific users that are allowed to access the database. Users are given a username and password or sometimes through biometric authentication (such as a fingerprint or retina scan) but these types of authentication tend to be more costly. This function also sets restraints on what specific data any user can see or manage.

5. Multiuser Access Control

Data integrity and data consistency are the basis of this function. Multiuser access control is a very useful tool in a DBMS, it enables multiple users to access the database simultaneously without affecting the integrity of the database.

6. Backup and Recovery Management

Backup and recovery is brought to mind whenever there is potential outside threats to a database. For example if there is a power outage, recovery management is how long it takes to recover the database after the outage. Backup management refers to the data safety and integrity; for example backing up all your mp3 files on a disk.

7. Data Integrity Management

The DBMS enforces these rules to reduce things such as data redundancy, which is when data is stored in more than one place unnecessarily, and maximizing data consistency, making sure database is returning correct/same answer each time for same question asked.

8. Database Access Languages and Application Programming Interfaces

A query language is a nonprocedural language. An example of this is SQL (structured query language). SQL is the most common query language supported by the majority of DBMS vendors. The use of this language makes it easy for user to specify what they want done without the headache of explaining how to specifically do it.

9. Database Communication Interfaces

This refers to how a DBMS can accept different end user requests through different network environments. An example of this can be easily related to the internet. A DBMS can provide access to the database using the Internet through Web Browsers (Mozilla Firefox, Internet Explorer, Netscape).

10. Transaction Management

This refers to how a DBMS must supply a method that will guarantee that all the updates in a given transaction are made or not made. All transactions must follow what is called the ACID properties.

A – Atomicity: states a transaction is an indivisible unit that is either performed as a whole and not by its parts, or not performed at all. It is the responsibility of recovery management to make sure this takes place.

C – Consistency: A transaction must alter the database from one constant state to another constant state.

I – Isolation: Transactions must be executed independently of one another.Part of a transaction in progress should not be able to be seen by another transaction.

D – Durability: A successfully completed transaction is recorded permanently in the database and must not be lost due to failures**.**

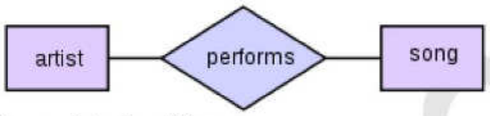
Entity Relationship Model

The entity-relationship (E-R) model is the most popular conceptual model used for designing a database. It was originally proposed by Dr. Peter Chen in 1976 as a way to unify the network and relational database views. The E-R model views the real world as a set of basic objects (known as **entities**), their characteristics (known as **attributes**), and associations among these objects (known as **relationships**). The *entities*, *attributes*, and *relationships* are the basic constructs of an E-R model.

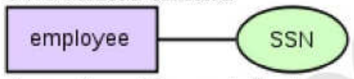
The information gathered from the user forms the basis for designing the E-R model. The *nouns* in the requirements specification document are represented as the entities, the *additional nouns* that describe the nouns corresponding to entities form the attributes, and the *verbs* become the relationships among various entities. The E-R model has several advantages listed as follows.

The Key Elements Of ER Model Are

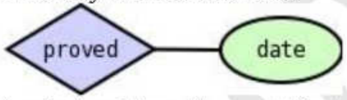
Entities, Relationships, and Attributes



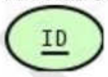
Two related entities



An entity with an attribute



A relationship with an attribute



Primary key

<u>Entity</u>

An entity may be defined as a thing which is recognized as being capable of an independent existence and which can be uniquely identified. An entity is an abstraction from the complexities of a domain. When we speak of an entity, we normally speak of some aspect of the real world which can be distinguished from other aspects of the real world.

An entity may be a physical object such as a house or a car, an event such as a house sale or a car service, or a concept such as a customer transaction or order. Although the term entity is the one most commonly used, following Chen we should really distinguish between an entity and an entity-type. An entity-type is a category. An entity, strictly speaking, is an instance of a given entity-type. There are usually many instances of an entity-type. Because the term entity-type is somewhat cumbersome, most people tend to use the term entity as a synonym for this term.

Entities can be thought of as nouns. Examples: a computer, an employee, a song, a mathematical theorem.

Relationship

A relationship captures how entities are related to one another. Relationships can be thought of as verbs, linking two or more nouns. Examples: an *owns* relationship between a company and a computer, a *supervises* relationship between an employee and a department, a *performs* relationship between an artist and a song, a *proved* relationship between a mathematician and a theorem.

Attribute

Entities and relationships can both have attributes. Examples: an *employee* entity might have a *Social Security Number* (SSN) attribute; the *proved* relationship may have a *date* attribute.

Every entity (unless it is a weak entity) must have a minimal set of uniquely identifying attributes, which is called the entity's primary key.

Entity–relationship diagrams don't show single entities or single instances of relations. Rather, they show entity sets and relationship sets. Example: a particular *song* is an entity. The collection of all songs in a database is an entity set. The *eaten* relationship between a child and her lunch is a single relationship. The set of all such child-lunch relationships in a database is a relationship set. In other words, a relationship set corresponds to a relation in mathematics, while a relationship corresponds to a member of the relation.

Certain cardinality constraints on relationship sets may be indicated as well.

Relationships, roles and cardinalities

In Chen's original paper he gives an example of a relationship and its roles. He describes a relationship "marriage" and its two roles "husband" and "wife".

A person plays the role of husband in a marriage (relationship) and another person plays the role of wife in the (same) marriage. These words are nouns. That is no surprise; naming things requires a noun.

Relationship names

A relationship expressed with a single verb implying direction, makes it impossible to discuss the model using the following proper English. For example:

- the song and the artist are related by a 'performs'
- the husband and wife are related by an 'is-married-to'.

Expressing the relationships with a noun resolves this:

- the song and the artist are related by a 'performance' □ the husband and wife are related by a 'marriage'".

Traditionally, the relationships are expressed twice, (using present continuous verb phrases), once in each direction. This gives two English statements per relationship. For example:

- The song **is performed by** the artist.
- The artist **performs** the song.

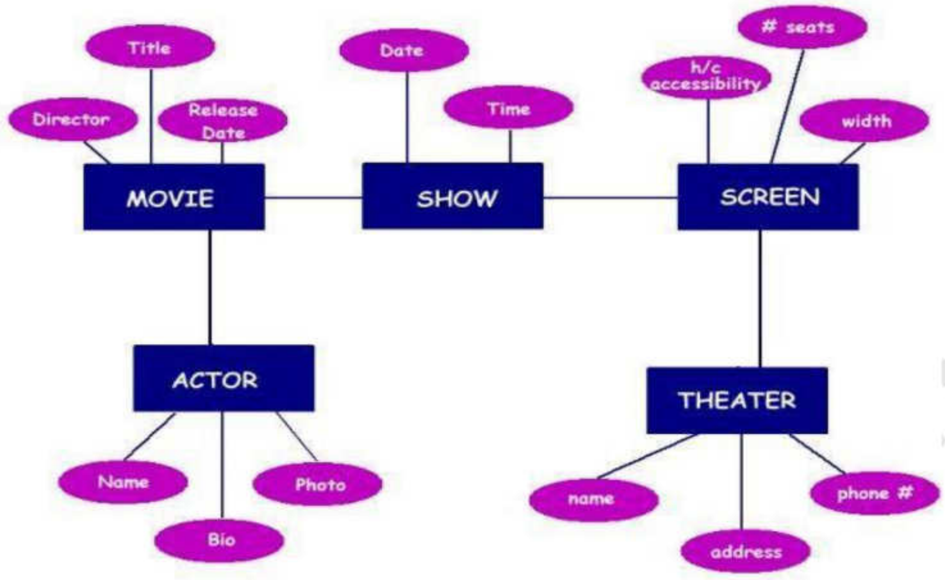
Role naming

It has also become prevalent to name roles with phrases e.g. is-the-owner-of and isowned-by etc. Correct nouns in this case are "owner" and "possession". Thus "person plays the role of owner" and "car plays the role of possession" rather than "person plays the role of is-the-owner-of" etc.

The use of nouns has direct benefit when generating physical implementations from semantic models. When a person has two relationships with car then it is possible to very simply generate names such as "owner_person" and "driver_person" which are immediately meaningful.

E-R Diagram:

Data models are tools used in analysis to describe the data requirements and assumptions in the system from a top-down perspective. They also set the stage for the design of databases later on in the SDLC.
There are three basic elements in ER models:
Entities are the "things" about which we seek information.
Attributes are the data we collect about the entities.
Relationships provide the structure needed to draw information from multiple entities.Generally, ERD's look like this:



De ve loping an ERD

Developing an ERD requires an understanding of the system and its components.

Consider a hospital:

Patients are treated in a single ward by the doctors assigned to them. Usually each patient will be assigned a single doctor, but in rare cases they will have two. Healthcare assistants also attend to the patients, a number of these are associated with each ward. Initially the system will be concerned solely with drug treatment. Each patient is required to take a variety of drugs a certain number of times per day and for varying lengths of time.
The system must record details concerning patient treatment and staff payment. Some staff are paid part time and doctors and care assistants work varying amounts of overtime at varying rates (subject to grade).
The system will also need to track what treatments are required for which patients and when and it should be capable of calculating the cost of treatment per week for each patient (though it is currently unclear to what use this information will be put).

How do we start an ERD?

1. Define Entities: these are usually nouns used in descriptions of the system, in the discussion of business rules, or in documentation; identified in the narrative (see highlighted items above).

2. Define Relationships: these are usually verbs used in descriptions of the system or in discussion of the business rules (entity ______ entity); identified in the narrative (see highlighted items above).

3. Add attributes to the relations; these are determined by the queries,and may also suggest new entities, e.g. grade; or they may suggest the need for keys or identifiers.

What questions can we ask?

a. Which doctors work in which wards?

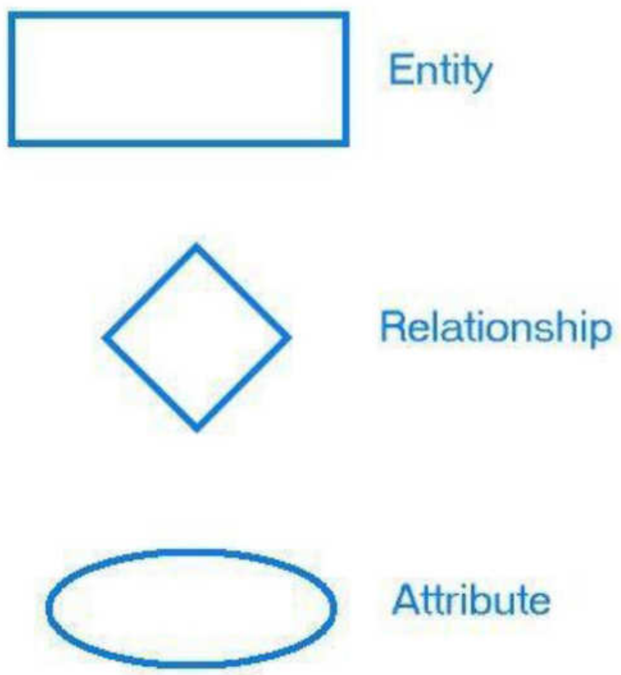
b. How much will be spent in a ward in a given week?

c. How much will a patient cost to treat?

d. How much does a doctor cost per week?

e. Which assistants can a patient expect to see?
f. Which drugs are being used? 4. Add cardinality to the relations
Many-to-Many must be resolved to two one-to-manys with an additional entity
Usually automatically happens
Sometimes involves introduction of a link entity (which will be all foreign key)
Examples: Patient-Drug
5. This flexibility allows us to consider a variety of questions such as:
a. Which beds are free?
b. Which assistants work for Dr. X?
c. What is the least expensive prescription?
d. How many doctors are there in the hospital?
e. Which patients are family related?

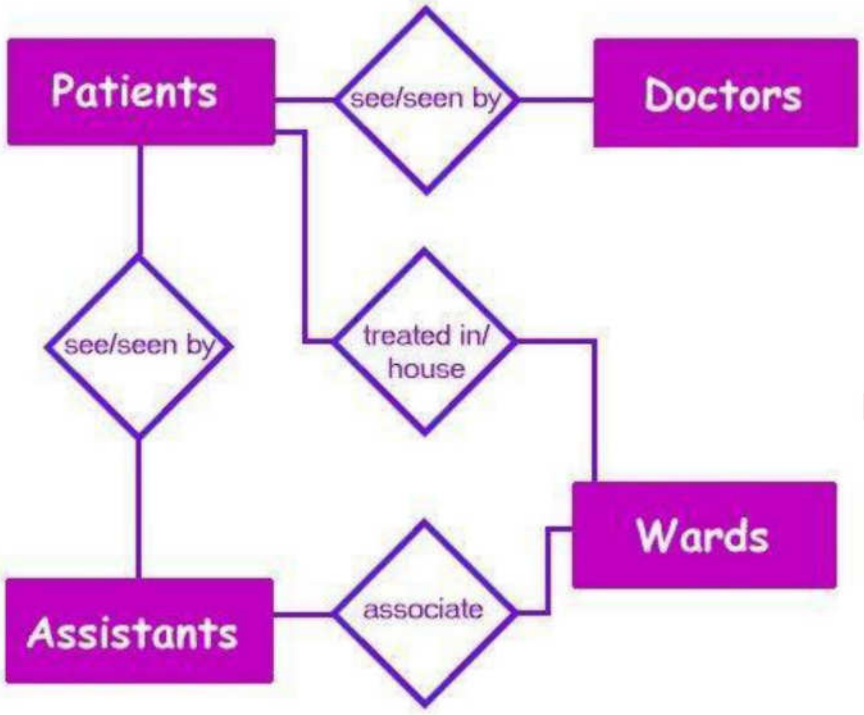
6. Represent that information with symbols. Generally E-R Diagrams require the use of the following symbols:



Reading an ERD

It takes some practice reading an ERD, but they can be used with clients to discuss business rules.

These allow us to represent the information from above such as the E-R Diagram below:



ERD brings out issues:

Many-to-Many

Ambiguities

Entities and their relationships

What data needs to be stored

The Degree of a relationship

SQL (Using MS SQL Server / Oracle/ My SQL)

What Is SQL?

SQL (pronounced "sequel") is an acronym for Structured Query Language; a standardized language used to access and manipulates data. The history of SQL corresponds closely with the development of a relational databases concept published in a paper by Dr. E. F. Codd at IBM in 1970. He applied mathematical concepts to the specification of a method for data storage and access; this specification, which became the basis for relational databases, was intended to overcome the physical dependencies of the then-available database systems. The SQL language (originally called "System R" in the prototype and later called "SEQUEL") was developed by the IBM Research Laboratory as a standard language to use with relational databases. In 1979 Oracle, then called Relational Software, Inc., introduced the first commercially available implementation of a relational database incorporating the SQL language.

The SQL language evolved with many additional syntax expansions incorporated into the American National Standards Institute (ANSI) SQL standards developed since. Individual database vendors continuously added extensions to the language, which eventually found their way into the latest ANSI standards used by relational databases today. Large-scale commercial implementations of relational database applications started to appear in the mid- to late 1980s, as early implementations were hampered by poor performance. Since then, relational databases and the SQL language have continuously evolved and improved.
Today, SQL is accepted as the universal standard database access language. Databases using the SQL language are entrusted with managing critical information affecting many aspects of our daily lives. Most applications developed today use a relational database, and Oracle continues to be one of the largest and most popular database vendors.

1. SQL is used by most commercial database applications.

2. Although the language has evolved over the years with a large array of syntax enhancements and additions, most of the basic functionality has remained essentially unchanged.

3. SQL knowledge will continue to be a fundamental skill because there is currently no mature and viable alternative language that accomplishes the same functionality.

4. Learning Oracle"s specific SQL implementation allows you great insight into the feature-rich functionality of one of the largest and most successful database vendors.

Overview of SQL Language Commands:

You work with tables, rows, and columns using the SQL language. SQL allows you to query data, create new data, modify existing data, and delete data. Within the SQL Language you can differentiate between individual sublanguages, which are a collection of individual commands.

SQL statements are classified into 6 categories:

1. Data Definition Language (DDL) Statements are:

- ALTER
- ANALYZE
- ASSOCIATE STATISTICS
- AUDIT
- COMMENT
- CREATE
- DISASSOCIATE STATISTICS
- DROP
- FLASHBACK
- GRANT
- NOAUDIT
- PURGE
- RENAME
- REVOKE
- TRUNCATE - UNDROP

2. Data Manipulation Language (DML) statements are:

- CALL
- DELETE
- EXPLAIN PLAN
- INSERT
- LOCK TABLE
- MERGE
- SELECT
- UPDATE

3. Transaction Control Language (TCL) statements are:

- COMMIT
- ROLLBACK
- SAVEPOINT
- SET TRANSACTION

4. Session Control Language (SSCL) statements are:

- ALTER SESSION - SET
ROLE

5. System Control Language (SCL) Statements are:

ALTER SYSTEM
EMBEDED SQL STATEMENTS

Data Types Used In SQL

DATE:

The DATE data type stores date and time information.

NUMBER:

Columns with the data type NUMBER allow only numeric data; no text, hyphens, or dashes are permitted. A column defined as NUMBER (5,2) can have a maximum of three digits before the decimal point and two digits after the decimal point. The first digit (5) is called the precision; the second digit (2) is referred to as the scale. The smallest allowed number is – 999.99, and the largest is 999.99. A column definition with a zero scale, such as NUMBER (5) or NUMBER (5,0), allows integers in the range from – 99,999 to 99,999.

VARCHAR2 and CHAR:

The VARCHAR2 and CHAR data types store alphanumeric data (for example, text, numbers, special characters). VARCHAR2 is the variable-length data type and the most commonly used alphanumeric data type; its maximum size is 4,000 characters. The main difference between VARCHAR2 and CHAR is that the CHAR data type is a fixed- length data type, and any unused room is blank padded with spaces.

For example, a column defined as CHAR(10) and containing the four-character-length value JOHN in a row will have six blank characters padded at the end to make the total length 10 spaces. (If the column is stored in a VARCHAR2(10) column instead, it stores four characters only.) A CHAR column can store up to 2,000 characters.

Other Data Types:

The data types BLOB and BFILE are binary data types that deal with access to multimedia content such as movies, images, or music. The main difference between these two data types is how the data is stored within the Oracle database. The BLOB data type stores the content inside the Oracle database, whereas the BFILE data type stores only a reference to the file location directory and the file name.

Data Retrieval Language(DRL)

DRL:

DRL means Data Retrieval Language. This will be used for the retrieval of the data from the database.
In order to see the data present in the database, we will use DRL statement. We have only one DRL
Statement.

SELECT is the only DRL statement in SQL.

SELECT:

Select statement is used to see the data present in the database. Syntax for writing this statement is,

Syntax:

SQL>SELECT col_list FROM table name;
Or
SQL>SELECT column_name(s) FROM table_name;
Or
SQL>SELECT * FROM table_name;

An SQL SELECT Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to select the content of the columns named "LastName" and "FirstName" from the table above.

We use the following SELECT statement:

SQL>SELECT LastName,FirstName FROM Persons;

The result-set will look like this:

LastName	FirstName
Hansen	Ola
Svendson	Tove
Pettersen	Kari

SELECT * Example

Now we want to select all the columns from the "Persons" table.

We use the following SELECT statement:

SQL>SELECT * FROM Persons

Tip: The asterisk (*) is a quick way of selecting all columns!

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The SQL SELECT DISTINCT Statement

In a table, some of the columns may contain duplicate values. This is not a problem, however, sometimes you will want to list only the different (distinct) values in a table.

The DISTINCT keyword can be used to return only distinct (different) values.

Syntax:

SQL>SELECT DISTINCT column_name(s) FROM table_name;

SQL>SELECT DISTINCT Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to select only the distinct values from the column named "City" from the table above.

We use the following SELECT statement: SQL>**SELECT**

DISTINCT City FROM Persons;

The result-set will look like this:

City
Sandnes
Stavanger

Clauses

Clauses are those statements that allows a user to add logical condition to the queries and makes results of more accurate and precise. The sql clauses are

- WHERE Clause
- GROUP BY
- ORDER BY
- HAVIING Clause

Where Clause:

The **WHERE** clause is used to extract only those records that fulfill a specified criterion.

Syntax:

```
SQL>SELECT column_name(s)
FROM table_name
WHERE column_name operator value;
```

WHERE Clause Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to select only the persons living in the city "Sandnes" from the table above.
We use the following SELECT statement:

SQL>SELECT * FROM Persons
WHERE City='Sandnes';

The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Quotes Around Text Fields;

SQL uses single quotes around text values (most database systems will also accept double quotes).
However, numeric values should not be enclosed in quotes.
For text values:

This is correct:
SQL>SELECT * FROM Persons WHERE FirstName='Tove';
This is wrong:
SQL>SELECT * FROM Persons WHERE FirstName=Tove;
For numeric values:
This is correct:
SQL>SELECT * FROM Persons WHERE Year=1965;
This is wrong:
SQL>SELECT * FROM Persons WHERE Year='1965';

The ORDER BY Clause:

The ORDER BY keyword is used to sort the result-set by a specified column.
The ORDER BY keyword sort the records in ascending order by default.
If you want to sort the records in a descending order, you can use the DESC keyword.
SQL ORDER BY Syntax
SQL >SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC|DESC;

ORDER BY Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger

Now we want to select all the persons from the table above, however, we want to sort the persons by their last name.
We use the following SELECT statement:
SQL >SELECT * FROM Persons ORDER BY
LastName;
The result-set will look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
4	Nilsen	Tom	Vingvn 23	Stavanger
3	Pettersen	Kari	Storgt 20	Stavanger
2	Svendson	Tove	Borgvn 23	Sandnes

ORDER BY DESC Example

Now we want to select all the persons from the table above, however, we want to sort the persons descending by their last name. We use the following SELECT statement:
SQL >SELECT * FROM Persons
ORDER BY LastName DESC;
The result-set will look like this:

P_Id	LastName	FirstName	Address	City
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger
1	Hansen	Ola	Timoteivn 10	Sandnes

The HAVING Clause:

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

SQL HAVING Syntax

```
SQL>SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value;
```

SQL HAVING Example

We have the following "Orders" table:

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Now we want to find if any of the customers have a total order of less than 2000.

We use the following SQL statement:

```
SQL>SELECT Customer,SUM(OrderPrice) FROM Orders
GROUP BY Customer
HAVING SUM(OrderPrice)<2000;
```

The result-set will look like this:

Customer	SUM(OrderPrice)
Nilsen	1700

Now we want to find if the customers "Hansen" or "Jensen" have a total order of more than 1500.

We add an ordinary WHERE clause to the SQL statement:

```
SQL>SELECT Customer, SUM(Order Price) FROM Orders
WHERE Customer='Hansen' OR Customer='Jensen'
GROUP BY Customer HAVING
SUM(Order Price)>1500;
```

The result-set will look like this:

Customer	SUM(OrderPrice)
Hansen	2000
Jensen	2000

The GROUP BY Statement:

The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

SQL GROUP BY Syntax

```
SQL>SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name;
```

SQL GROUP BY Example

We have the following "Orders" table:

O_Id	OrderDate	OrderPrice	Customer
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Now we want to find the total sum (total order) of each customer.

We will have to use the GROUP BY statement to group the customers.

We use the following SQL statement:

SQL>SELECT Customer,SUM(OrderPrice) FROM Orders GROUP BY Customer;

The result-set will look like this:

Customer	SUM(OrderPrice)
Hansen	2000
Nilsen	1700
Jensen	2000

GROUP BY statement:

SQL>SELECT Customer,SUM(OrderPrice) FROM Orders;

The result-set will look like this:

Customer	SUM(OrderPrice)
Hansen	5700
Nilsen	5700
Hansen	5700
Hansen	5700
Jensen	5700
Nilsen	5700

Explanation of why the above SELECT statement cannot be used:

The SELECT statement above has two columns specified (Customer and SUM(OrderPrice). The "SUM(OrderPrice)" returns a single value (that is the total sum of the "OrderPrice" column), while "Customer" returns 6 values (one value for each row in the "Orders" table). This will therefore not give us the correct result. However, you have seen that the GROUP BY statement solves this problem.

GROUP BY More Than One Column

We can also use the GROUP BY statement on more than one column, like this:

SQL>SELECT Customer,OrderDate,SUM(OrderPrice) FROM Orders
GROUP BY Customer,OrderDate;

SQL Operators

To filter a data set, you need to specify a WHERE clause condition, which results in true, false, or unknown. The condition consists of an expression that can be a column of any data type, a string or text literal (sometimes referred to as a text constant or character literal), a number, a function, a mathematical computation, or any combination of these. The comparison operators evaluate the expressions for the selection of the appropriate data.

Comparison Operator	Definition
=	Equal
!=, <>	Not equal
>, >=	Greater than, greater than or equal to
<, <=	Less than, less than or equal to
BETWEEN ... AND ...	Inclusive of two values

LIKE	Pattern matching with wildcard characters % and
	_ IN (...)
	List of values
IS NULL	Test for null values

The Equality and Inequality Operators

One of the most commonly used comparison operators is the equality operator, denoted by the = symbol. For example, if you are asked to provide the first name, last name, and phone number of a teacher with the last name bano, you write the following SQL statement.

```
SQL>SELECT first_name, last_name, phone
FROM instructor
```

WHERE last_name = 'bano';

```
FIRST_NAME      LAST_NAME       PHONE
NADIA            Bano          2125551212 1 row
selected.
```

Just as equality is useful, so is inequality.

```
SQL>SELECT first_name, last_name, phone
FROM instructor
WHERE last_name <> 'Schorin';
```

FIRST_NAME	LAST_NAME	PHONE
Fernand	Hanks	2125551212
Tom	Wojick	2125551212
Marilyn	Frantzen	2125551212
Irene	Willig	2125551212

9 rows selected.

The Greater Than and Less Than Operators

The comparison operators >, <, >=, and <= can all be used to compare values in columns. In the following example, the >=, or greater than or equal to, operator is used to retrieve a list of course descriptions for which the course cost is greater than or equal to 1195.

```
SQL>SELECT description, cost
FROM course
WHERE cost >= 1195;
```

DESCRIPTION	COST
Technology Concepts	1195
Intro to Information Systems	1195
Database System Principles	1195
Java Developer III	1195

26 rows selected.

In this example, the value 1195 is not enclosed in single quotation marks because it is a number literal.

The BETWEEN Operator

The BETWEEN operator tests for a range of values.

SQL>**SELECT description, cost**
FROM course
WHERE cost BETWEEN 1000 AND 1100;

DESCRIPTION	COST
Unix Tips and Techniques	1095
Intro to the Internet	1095
Intro to the BASIC Language	1095

3 rows selected.
BETWEEN is inclusive of both values defining the range; the result set includes courses that cost 1000 and 1100 and everything in between. In this example, the lower end of the range must be listed first.
BETWEEN is most useful for number and date comparisons, but it can also be used for comparing text strings in alphabetical order. Date comparisons are discussed in Chapter 5.

The IN Operator

The IN operator works with a list of values, separated by commas, contained within a set of parentheses. The following query looks for courses for which the cost is either 1095 or 1595.
SQL>**SELECT description, cost**
FROM course
WHERE cost IN (1095, 1595);

DESCRIPTION	COST
Programming Techniques	1595
Unix Tips and Techniques	1095
Intro to the Internet	1095
Intro to the BASIC Language	1095

4 rows selected.

The LIKE Operator

A very useful comparison operator is LIKE, which performs pattern matching, using the percent (%) and underscore (_) characters as wildcards. The percent wildcard is used to denote multiple characters, while the underscore wildcard is used to denote a single character. The following query retrieves rows where the last name begins with the uppercase letter S and ends in anything else.
SQL>**SELECT first_name, last_name, phone**
FROM instructor
WHERE last_name LIKE 'S%';

FIRST_NAME	LAST_NAME	PHONE
Nina	Schorin	2125551212
Todd	Smythe	2125551212

2 rows selected.

The % character may be placed at the beginning, end, or anywhere within the literal text, but it must always be within the single quotation marks. This is also true of the underscore wildcard character, as in the following statement.

SQL>SELECT first_name, last_name
FROM instructor
WHERE last_name LIKE '_o%';

FIRST_NAME	LAST_NAME
Tom	Wojick
Anita	Morris
Charles	Lowry

3 rows selected.

The WHERE clause returns only rows where the last name begins with any character but the second letter must be a lowercase o. The rest of the last name is irrelevant. The LIKE operator works well for simple pattern matching. For a more complex pattern, you might want to consider using Oracle"s regular expression functionality, discussed in Chapter 16, "Regular Expressions and Hierarchical Queries."

The NOT Operator

All the previously mentioned operators can be negated with the NOT comparison operator (for example, NOT BETWEEN, NOT IN, NOT LIKE).

SQL>SELECT phone
FROM instructor
WHERE last_name NOT LIKE 'S%';

This query returns all the phone numbers of instructors with a last name that does not begin with the uppercase letter S. This SQL statement does not list LAST_NAME in the SELECT list. There is no rule about columns in the WHERE clause having to exist in the SELECT list.

The IS NULL and IS NOT NULL Operators

Recall that NULL means an unknown value. The IS NULL and IS NOT NULL operators evaluate whether a data value is NULL or not. The following SQL statement returns courses that do not have a prerequisite.

SQL>SELECT description, prerequisite
FROM course
WHERE prerequisite IS NULL;

Null values represent the unknown; a null cannot be equal or unequal to any value or to another null. Therefore, you should always use the IS NULL or IS NOT NULL operator when testing for nulls. There are a few exceptions when nulls are treated differently and a null can be equal to another null. One such example is the use of DISTINCT

Logical Operators

To harness the ultimate power of the WHERE clause, comparison operators can be combined with the help of the logical operators AND and OR. These logical operators are also referred to as Boolean operators. They group expressions, all within the same WHERE clause of a single SQL statement.
For example, the following SQL query combines two comparison operators with the help of the AND Boolean operator. The result shows rows where a course costs 1095 and the course description starts with the letter I.

```
SQL>SELECT description, cost
FROM course
WHERE cost = 1095
AND description LIKE 'I%';
```

DESCRIPTION	COST
Intro to the Internet	1095
Intro to the BASIC Language	1095
Precedence of Logical Operators	

When AND and OR are used together in a WHERE clause, the AND operator always takes precedence over the OR operator, meaning that any AND conditions are evaluated first. If there are multiple operators of the same precedence, the left operator is executed before the right. You can manipulate the precedence in the WHERE clause with the use of parentheses. In the following SQL statement, the AND and OR logical operators are combined.

```
SQL>SELECT description, cost, prerequisite
FROM course
WHERE cost = 1195
AND prerequisite = 20
OR prerequisite = 25;
```

DESCRIPTION	COST	PREREQUISITE
Hands-On Windows	1195	20
Systems Analysis	1195	20
Project Management	1195	20
GUI Design Lab	1195	20

Intro to SQL	1195	20

The preceding SQL statement selects any record that has either a cost of 1195 and a prerequisite of 20 or just a prerequisite of 25, no matter what the cost. The sixth row, Intro to the BASIC Language, is selected because it satisfies the OR expression prerequisite = 25. The seventh row, Database System Principles, satisfies only one of the AND conditions, not both. However, the row is part of the result set because it satisfies the OR condition.

Here is the same SQL statement, but with parentheses to group the expressions in the WHERE clause.

```
SQL>SELECT description, cost, prerequisite
FROM course
WHERE cost = 1195
AND (prerequisite = 20
OR prerequisite = 25);
```

DESCRIPTION	COST	PREREQUISITE
Hands-On Windows	1195	20
Systems Analysis	1195	20
Project Management	1195	20
GUI Design Lab	1195	20
Intro to SQL	1195	20
Database System Principles	1195	25

6 rows selected.

The first expression selects only courses where the cost is equal to 1195. If the prerequisite is either 25 or 20, then the second condition is also true. Both expressions need to be true for the row to be displayed. These are the basic rules of logical operators. If two conditions are combined with the AND operator, both conditions must be true; if two conditions are connected by the OR operator, only one of the conditions needs to be true for the record to be selected.

The result set returns six rows instead of seven. The order in which items in the WHERE clause are evaluated is changed by the use of parentheses and results in different output.

To ensure that your SQL statements are clearly understood, it is always best to use parentheses.

NULLs and Logical Operators

SQL uses tri-value logic; this means a condition can evaluate to true, false, or unknown. (This is in contrast to Boolean logic, where a condition must be either true or false.) A row is returned when the condition evaluates to true. The following query returns rows from the COURSE table, starting with the words "Intro to" as the description and a value equal or larger than 140 in the PREREQUISITE column.

```
SQL>SELECT description, prerequisite
FROM course
WHERE description LIKE 'Intro to%' AND prerequisite >= 140;
```

DESCRIPTION	PREREQUISITE
Intro to Programming	140
Intro to Unix	310

2 rows selected.

Rows with a null value in the PREREQUISITE column are not included because null is an unknown value. This null value in the column is not greater than or equal to 140. Therefore, the row Intro to Information Systems does not satisfy both conditions and is excluded from the result set. Following is the list of course descriptions with null values in the PREREQUISITE column. It shows the row Intro to Information Systems and the null value in the PREREQUISITE column.

```
SQL>SELECT description, prerequisite, cost
FROM course
WHERE prerequisite IS NULL;
```

DESCRIPTION	PREREQUISITE	COST
Technology Concepts		1195
Intro to Information Systems		1195
Java for C/C++ Programmers		1195

Operating Systems	1195

4 rows selected.

The AND truth table in Table 3.2 illustrates the combination of two conditions with the AND operator. Only if both conditions are true is a row returned for output. In this example, with the prerequisite being null, the condition is unknown, and therefore the row is not included in the result. The comparison against a null value yields unknown unless you specifically test for it with the IS NULL or IS NOT operators.

For the OR condition, just one of the conditions needs to be true. Again, let"s examine how nulls behave under this scenario, using the same query, but this time with the OR operator. The Intro to Information Systems course is now listed because it satisfies the „Intro to%" condition only. In addition, rows such as DB Programming with Java do not start with "Intro to" as the description but satisfy the second condition, which is a prerequisite of greater than or equal to 140.

SQL>SELECT description, prerequisite
FROM course
WHERE description LIKE 'Intro to%' OR prerequisite >= 140;

DESCRIPTION	**PREREQUISITE**
Intro to Information Systems	
Intro to Programming	140
Programming Techniques	204
Intro to Java Programming	80
Intro to Unix	310
Database Design	420
Internet Protocols	310
Intro to SQL	20
Oracle Tools	220
Intro to the Internet	10
Intro to the BASIC Language	25
Java Developer III	350
DB Programming with Java	350

13 rows selected

DDL (Data Definition Language)

Data definition language consist of 5 types of statements which are

1. CREATE Statement

   Used to create a table in sql.

2. ALTER Statement

   Used to modify a table in sql.

3. DROP Statement

   Used to Drop table from data base.

4. RENAME Statement

   Used To rename a Table

5. TRUNCATE Statement

   Used To Delete rows of a table.

The CREATE DATABASE Statement:

The CREATE DATABASE statement is used to create a database.

SQL CREATE DATABASE Syntax

SQL>CREATE DATABASE database_name; **CREATE**

DATABASE Example

Now we want to create a database called "my_db".

We use the following CREATE DATABASE statement:

SQL>CREATE DATABASE my_db;

The CREATE TABLE Statement

The CREATE TABLE statement is used to create a table in a database.

SQL CREATE TABLE Syntax

```
SQL>CREATE TABLE table_name
(
column_name1 data_type, column_name2
data_type, column_name3 data_type,
.... );
```

The data type specifies what type of data the column can hold.

CREATE TABLE Example

Now we want to create a table called "Persons" that contains five columns: P_Id, LastName, FirstName, Address, and City.

We use the following CREATE TABLE statement:

```
SQL>CREATE TABLE Persons
(
P_Id int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255)
);
```

The empty "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City

The empty table can be filled with data with the INSERT INTO statement.

The ALTER TABLE Statement:

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

SQL ALTER TABLE Syntax

To add a column in a table, use the following syntax:

```
SQL>ALTER TABLE table_name
ADD column_name datatype;
```

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

SQL>ALTER TABLE table_name DROP COLUMN column_name;

To change the data type of a column in a table, use the following syntax:

My SQL / SQL Server / MS Access:

SQL>ALTER TABLE table_name ALTER COLUMN column_name datatype

Oracle:

SQL>ALTER TABLE table_name MODIFY column_name datatypel;

SQL ALTER TABLE Example

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to add a column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

```
ALTER TABLE Persons
ADD DateOfBirth date
```

Notice that the new column, "DateOfBirth", is of type date and is going to hold a date.

The "Persons" table will now like this:

P_Id	LastName	FirstName	Address	City	DateOfBirth
1	Hansen	Ola	Timoteivn 10	Sandnes	
2	Svendson	Tove	Borgvn 23	Sandnes	
3	Pettersen	Kari	Storgt 20	Stavanger	

Change Data Type Example

Now we want to change the data type of the column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

```
SQL>ALTER TABLE Persons
ALTER COLUMN DateOfBirth year;
```

Notice that the "DateOfBirth" column is now of type year and is going to hold a year in a two-digit or four-digit format.

DROP COLUMN Example

Next, we want to delete the column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

```
SQL>ALTER TABLE Persons
DROP COLUMN DateOfBirth;
```

The "Persons" table will now like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

AUTO INCREMENT a Field

Very often we would like the value of the primary key field to be created automatically every time a new record is inserted.

We would like to create an auto-increment field in a table.

You will have to create an auto-increment field with the sequence object (this object generates a number sequence).

Use the following CREATE SEQUENCE syntax:

```
SQL>CREATE SEQUENCE seq_person
MINVALUE 1
START WITH 1
INCREMENT BY 1
CACHE 10;
```

The code above creates a sequence object called seq_person, that starts with 1 and will increment by 1. It will also cache up to 10 values for performance. The cache option specifies how many sequence values will be stored in memory for faster access.

To insert a new record into the "Persons" table, we will have to use the nextval function (this function retrieves the next value from seq_person sequence):

```
SQL>INSERT INTO Persons (P_Id,FirstName,LastName) VALUES
(seq_person.nextval,'Lars','Monsen');
```

The SQL statement above would insert a new record into the "Persons" table. The "P_Id" column would be assigned the next number from the seq_person sequence. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen

The DROP INDEX Statement:

The DROP INDEX statement is used to delete an index in a table.

```
SQL>ALTER TABLE table_name DROP INDEX index_name;
```

The DROP TABLE Statement;

The DROP TABLE statement is used to delete a table.

SYNTAX:

```
SQL>DROP TABLE table_name;
```

The DROP DATABASE Statement:

The DROP DATABASE statement is used to delete a database.

SQL>DROP DATABASE database_name;

The TRUNCATE TABLE Statement:

What if we only want to delete the data inside the table, and not the table itself?

Then, use the TRUNCATE TABLE statement:

SQL>TRUNCATE TABLE table_name;

The RENAME Statement:

Used to rename a table

Syntax:
SQL>RENAME TABLE Old_table_name TO New_table_name;

DML (Data Manipulation Language)

DML stands for data manipulation language. Manipulation means adding new records, deleting records and updating data base.
It consist of three types of statements INSERT Statement

Used To insert New Records to existing tables.

UPDATE Statement

Used to update inserted record deleted record

DELETE Statement

Used to delete records.

The INSERT INTO Statement:

The INSERT INTO statement is used to insert a new row in a table.

SQL INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two forms.

The first form doesn't specify the column names where the data will be inserted, only their values:

```
SQL>INSERT INTO table_name
VALUES (value1, value2, value3,...);
```

The second form specifies both the column names and the values to be inserted:

```
SQL>INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...);
```

SQL INSERT INTO Example

We have the following "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to insert a new row in the "Persons" table.

We use the following SQL statement:

**SQL>INSERT INTO Persons
VALUES (4,'Nilsen', 'Johan', 'Bakken 2', 'Stavanger');**

The "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger

Insert Data Only in Specified Columns

It is also possible to only add data in specific columns.

The following SQL statement will add a new row, but only add data in the "P_Id", "LastName" and the "FirstName" columns:

SQL>INSERT INTO Persons (P_Id, LastName, FirstName) VALUES (5, 'Tjessem', 'Jakob');

The "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob		

The UPDATE Statement:

The UPDATE statement is used to update existing records in a table.

SQL UPDATE Syntax

```
SQL>UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value;
```

SQL UPDATE Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob		

Now we want to update the person "Tjessem, Jakob" in the "Persons" table.

We use the following SQL statement:

```
SQL>UPDATE Persons
SET Address='Nissestien 67', City='Sandnes' WHERE
LastName='Tjessem' AND FirstName='Jakob';
```

The "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob	Nissestien 67	Sandnes

SQL UPDATE Warning

Be careful when updating records. If we had omitted the WHERE clause in the example above, like this:

```
SQL>UPDATE Persons
SET Address='Nissestien 67', City='Sandnes';
```

The "Persons" table would have looked like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Nissestien 67	Sandnes
2	Svendson	Tove	Nissestien 67	Sandnes
3	Pettersen	Kari	Nissestien 67	Sandnes
4	Nilsen	Johan	Nissestien 67	Sandnes
5	Tjessem	Jakob	Nissestien 67	Sandnes

The DELETE Statement:

The DELETE statement is used to delete rows in a table.

SQL DELETE Syntax

```
SQL>DELETE FROM table_name
WHERE some_column=some_value;
```

SQL DELETE Example

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob	Nissestien 67	Sandnes

Now we want to delete the person "Tjessem, Jakob" in the "Persons" table.

We use the following SQL statement:

```
SQL>DELETE FROM Persons
```

WHERE LastName='Tjessem' AND FirstName='Jakob';

The "Persons" table will now look like this:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger

Delete All Rows

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

SQL>DELETE FROM table_name;

or

SQL>DELETE * FROM table_name;

DCL (Data Control Language)

Data control language is used to control database operations. It consist of the following types of ststements.

- **COMMIT Statement**
- **ROLLBACK Statement**

COMMIT Statement

The COMMIT Statement terminates the current transaction and makes all changes under the transaction persistent. It *commits* the changes to the database. The COMMIT statement has the following general format:

```
    COMMIT [WORK]
```

WORK is an optional keyword that does not change the semantics of COMMIT.

ROLLBACK Statement

The ROLLBACK Statement terminates the current transaction and rescinds all changes made under the transaction. It *rolls back* the changes to the database. The ROLLBACK statement has the following general format:

```
    ROLLBACK [WORK]
```

WORK is an optional keyword that does not change the semantics of ROLLBACK.

SQL Constraint

Constraints are used to limit the type of data that can go into a table.

Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement).

We will focus on the following constraints:

- **NOT NULL**
- **UNIQUE**
- **PRIMARY KEY**
- **FOREIGN KEY**
- **CHECK**
- **DEFAULT**

SQL NOT NULL Constraint

The NOT NULL constraint enforces a column to NOT accept NULL values.

The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field.

The following SQL enforces the "P_Id" column and the "LastName" column to not accept NULL values:

```
SQL>CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255) ) ;
```

SQL UNIQUE Constraint

The UNIQUE constraint uniquely identifies each record in a database table.

The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.

Note that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

SQL UNIQUE Constraint on CREATE TABLE

The following SQL creates a UNIQUE constraint on the "P_Id" column when the "Persons" table is created:

```
SQL>CREATE TABLE Persons
(
P_Id int NOT NULL UNIQUE,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
);
```

SQL UNIQUE Constraint on ALTER TABLE

To create a UNIQUE constraint on the "P_Id" column when the table is already created, use the following SQL:

```
SQL>ALTER TABLE Persons
ADD UNIQUE (P_Id);
```

To allow naming of a UNIQUE constraint, and for defining a UNIQUE constraint on multiple columns, use the following SQL syntax:

```
SQL>ALTER TABLE Persons
ADD CONSTRAINT uc_PersonID UNIQUE (P_Id,LastName);
```

To DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL:

```
SQL>ALTER TABLE Persons
DROP CONSTRAINT uc_PersonID;
```

SQL PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a database table.

Primary keys must contain unique values.

A primary key column cannot contain NULL values.

Each table should have a primary key, and each table can have only ONE primary key.

SQL PRIMARY KEY Constraint on CREATE TABLE

The following SQL creates a PRIMARY KEY on the "P_Id" column when the "Persons" table is created:

```
SQL>CREATE TABLE Persons
(
P_Id int NOT NULL PRIMARY KEY,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
) ;
```

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

```
SQL>CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,LastName)
) ;
```

SQL PRIMARY KEY Constraint on ALTER TABLE

To create a PRIMARY KEY constraint on the "P_Id" column when the table is already created, use the following SQL:

```
SQL>ALTER TABLE Persons
ADD PRIMARY KEY (P_Id) ;
```

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

```
SQL>ALTER TABLE Persons
ADD CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,LastName) ;
```

To DROP a PRIMARY KEY Constraint

To drop a PRIMARY KEY constraint, use the following SQL:

```
SQL>ALTER TABLE Persons
DROP CONSTRAINT pk_PersonID;
```

SQL FOREIGN KEY Constraint

A FOREIGN KEY in one table points to a PRIMARY KEY in another table.

Let's illustrate the foreign key with an example. Look at the following two tables:

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Note that the "P_Id" column in the "Orders" table points to the "P_Id" column in the "Persons" table.

The "P_Id" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "P_Id" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

The FOREIGN KEY constraint also prevents that invalid data form being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

SQL FOREIGN KEY Constraint on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "P_Id" column when the "Orders" table is created:

```
SQL>CREATE TABLE Orders
(
O_Id int NOT NULL PRIMARY KEY,
OrderNo int NOT NULL,
P_Id int FOREIGN KEY REFERENCES Persons(P_Id)
) ;
```

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

```
SQL>CREATE TABLE Orders
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
PRIMARY KEY (O_Id),
CONSTRAINT fk_PerOrders FOREIGN KEY (P_Id)
REFERENCES Persons(P_Id)
) ;
```

SQL FOREIGN KEY Constraint on ALTER TABLE

To create a FOREIGN KEY constraint on the "P_Id" column when the "Orders" table is already created, use the following SQL:

```
SQL>ALTER TABLE Orders
ADD FOREIGN KEY (P_Id)
REFERENCES Persons(P_Id) ;
```

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

```
SQL>ALTER TABLE Orders
ADD CONSTRAINT fk_PerOrders
FOREIGN KEY (P_Id)
REFERENCES Persons(P_Id) ;
```

To DROP a FOREIGN KEY Constraint

To drop a FOREIGN KEY constraint, use the following SQL:

```
SQL>ALTER TABLE Orders
DROP CONSTRAINT fk_PerOrders;
```

SQL CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a single column it allows only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

SQL CHECK Constraint on CREATE TABLE

The following SQL creates a CHECK constraint on the "P_Id" column when the "Persons" table is created. The CHECK constraint specifies that the column "P_Id" must only include integers greater than 0.

```
SQL>CREATE TABLE Persons
(
P_Id int NOT NULL CHECK (P_Id>0),
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
) ;
```

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

```
SQL>CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT chk_Person CHECK (P_Id>0 AND City='Sandnes')
) ;
```

SQL CHECK Constraint on ALTER TABLE

To create a CHECK constraint on the "P_Id" column when the table is already created, use the following SQL:

```
SQL>ALTER   TABLE   Persons   ADD
CHECK (P_Id>0) ;
```

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

```
SQL>ALTER TABLE Persons
ADD CONSTRAINT chk_Person CHECK (P_Id>0 AND
City='Sandnes') ;
```

To DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL:

```
SQL>ALTER TABLE Persons DROP
CONSTRAINT chk_Person;
```

SQL DEFAULT Constraint

The DEFAULT constraint is used to insert a default value into a column.

The default value will be added to all new records, if no other value is specified.

SQL DEFAULT Constraint on CREATE TABLE

The following SQL creates a DEFAULT constraint on the "City" column when the "Persons" table is created:

```
SQL>CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255) DEFAULT 'Sandnes'
) ;
```

The DEFAULT constraint can also be used to insert system values, by using functions like GETDATE():

```
SQL>CREATE TABLE Orders
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
OrderDate date DEFAULT GETDATE()
) ;
```

SQL DEFAULT Constraint on ALTER TABLE

To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

```
SQL>ALTER TABLE Persons
```

ALTER COLUMN City SET DEFAULT 'SANDNES';

Or

SQL>**ALTER TABLE Persons
MODIFY City DEFAULT 'SANDNES';**

To DROP a DEFAULT Constraint

To drop a DEFAULT constraint, use the following SQL:

SQL>**ALTER TABLE Persons
ALTER COLUMN City DROP DEFAULT;**

SQL FUNCTIONS

Character Functions

All character functions require alphanumeric input parameters. The input can be a text literal or character literal, sometimes referred to as a string, or text, constant, or a column of data type VARCHAR2, CHAR, or CLOB. Text literals are always surrounded by single quotation marks. This lab discusses the most frequently used character functions.

The LOWER Function

The LOWER function transforms data into lowercase. The following query shows how both a column and a text constant serve as individual parameters for the LOWERfunction.

SQL>SELECT state, LOWER(state), LOWER('State') FROM zipcode;

The UPPER and INITCAP Functions

The UPPER function is the exact opposite of the LOWER function: It transforms data into uppercase. The INITCAP function capitalizes the first letter of a word and lowercases the rest of the word.

SQL>SELECT UPPER(city) as "Upper Case City", state, INITCAP(state)

FROM zipcode

WHERE zip = '10035'

The LPAD and RPAD Functions

The LPAD and RPAD functions also transform data: They left pad and right pad strings, respectively. When you pad a string, you add to it. These functions can add characters, symbols, or even spaces to your result set. Unlike the LOWER, UPPER, or INITCAP functions, these functions take more than one parameter as their input.

SQL>SELECT RPAD(city, 20, '*') "City Name", LPAD(state, 10, '-') "State Name"

FROM zipcode

The LTRIM, RTRIM, and TRIM Functions

LTRIM and RTRIM are the opposite of LPAD and RPAD because they trim, or remove, unwanted characters, symbols, or spaces in strings. In this example, you see the use of the DUAL table to trim the zero (0) from the left, the right, and both sides. If both the left and right sides of the string are trimmed, you need to nest the function. The result of one function provides the input for the other function.

SQL>SELECT LTRIM('0001234500', '0') left, RTRIM('0001234500', '0') right, LTRIM(RTRIM('0001234500', '0'), '0') both FROM dual

The TRIM function removes leading characters, trailing characters, or both, effectively doing the job of LTRIM and RTRIM in one function. If you want the function to act like LTRIM, specify LEADING as the first parameter; for RTRIM, use the TRAILING option; for both, either specify the BOTH keyword or omit it altogether.

The syntax for TRIM is as follows. The parameter named char1 indicates the single character to be removed; char2 is the string to be trimmed. If you don"t specify char1, blank spaces are assumed.

SQL>TRIM([LEADING|TRAILING|BOTH] char1 FROM char2)

The SUBSTR Function

SUBSTR transforms a string, returning a substring or subset of a string, based on its input parameters. The following query displays student last names, the first five characters of those last names in the second column, and the remaining characters of those last names in the third column.

SUBSTR(char1, starting_position [, substring_length])

SQL>SELECT last_name, SUBSTR(last_name, 1, 5), SUBSTR(last_name, 6)

FROM student

The INSTR Function

INSTR, meaning in string, looks for the occurrence of a string inside another string, returning the starting position of the search string within the target string. Unlike the other string functions, INSTR does not return another string; rather, it returns a number. The following query displays course descriptions and the position in which the first occurrence of the string „er", if any, in the DESCRIPTION column appears. **INSTR(char1, char2 [,starting_position [, occurrence]])**

SQL>SELECT description, INSTR(description, 'er') FROM course

The LENGTH Function

The LENGTH function determines the length of a string, expressed as a number. The following SQL statement selects a text literal from the DUAL table in conjunction with the LENGTH function.

SQL>SELECT LENGTH('Hello there') FROM dual LENGTH('HELLOTHERE')

Concatenation

Concatenation connects strings together to become one. Strings can be concatenated to produce a single column in the result set. There are two methods of concatenation in Oracle: One is with the CONCAT function, the other is the concatenation operator (||), which is two vertical bars or pipe symbols. The syntax of the CONCAT function is as follows.

CONCAT(char1, char2)

When you want to concatenate cities and states together using the CONCAT function, you can use the function as follows.

SQL>SELECT CONCAT(city, state) FROM zipcode CONCAT(CITY,STATE)

Number Functions

Number functions are valuable tools for operations such as rounding numbers or computing the absolute value of a number. There are several single-row number functions in

The ABS Function

The ABS function computes the absolute value of a number, measuring its magnitude.

```
SQL>SELECT 'The absolute value of -29 is '||ABS(-29) FROM dual

The absolute value of -29 is 29

1 row selected.
```

ABS takes only a single input parameter, and its syntax is as follows.

ABS(value)

The SIGN Function

The SIGN function tells you the sign of a value, returning a number 1 for a positive number, –1 for a negative number, or 0 for zero. The following example compares SIGN with the ABS function.

SQL>SELECT -14, SIGN(-14), SIGN(14), SIGN(0), ABS(-14) FROM dual

```
-14    -1     1      0      14

1 row selected.
```

SIGN also takes only a single input parameter, and its syntax is as follows.

SIGN(value)

Most single-row functions return NULL when a NULL is the input parameter.

ROUND and TRUNC Functions

ROUND and TRUNC are two useful functions that round and truncate (or cut off) values, respectively, based on a given number of digits of precision. The next SELECTstatement illustrates the use of ROUND and TRUNC, which both take two input parameters. Observe the differences in the result.

SQL>SELECT 222.34501, ROUND(222.34501, 2), TRUNC(222.34501, 2)

FROM dual

```
222.34501     222.35        222.34
```

1 row selected.

Here, ROUND (222.34501, 2) rounds the number 222.34501 to two digits to the right of the decimal, rounding the result up to 222.35, following the normal convention for rounding. In contrast, TRUNC cuts off all digits beyond two digits to the right of the decimal, resulting in 222.34. ROUND and TRUNC can be used to affect the left side of the decimal as well by passing a negative number as a parameter.

SQL>SELECT 222.34501, ROUND(222.34501, -2), TRUNC(222.34501, -2)

FROM dual

The following is the syntax for ROUND and TRUNC.

ROUND(value [, precision])

TRUNC(value [, precision])

The FLOOR and CEIL Functions

The CEIL function returns the smallest integer greater than or equal to a value; the FLOOR function returns the largest integer equal to or less than a value. These functions perform much like the ROUND and TRUNC functions, without the optional precision parameter.

SQL>SELECT FLOOR(22.5), CEIL(22.5), TRUNC(22.5), ROUND(22.5) FROM dual

FLOOR(22.5) CEIL(22.5) TRUNC(22.5) ROUND(22.5);

```
22      23      22      23
```

1 row selected.

The syntax for the FLOOR and CEIL functions is as follows.

FLOOR(value)

CEIL(value)

The MOD Function

MOD is a function that returns the modulus, or the remainder of a value divided by another value. It takes two input parameters, as in the following SELECT statement.

SQL>SELECT MOD(23, 8) FROM dual MOD(23,8)

The MOD function divides 23 by 8 and returns a remainder of 7. The following is the syntax for MOD.

MOD(value, divisor);

The MOD function is particularly useful if you want to determine whether a value is odd or even. If you divide by 2 and the remainder is a zero, this indicates that the value is even; if the remainder is 1, it means that the value is odd.

The REMAINDER Function

The REMAINDER function calculates the remainder, according to the IEEE specification. The syntax is as follows.

REMAINDER(value, divisor)

The difference between REMAINDER and the MOD function is that MOD uses FLOOR in its computations, whereas REMAINDER uses ROUND. The next example shows that the results between the MOD and REMAINDER functions can be different.

SQL>SELECT MOD(23,8), REMAINDER(23,8) FROM DUAL

MOD(23,8) REMAINDER(23,8);

The COALESCE Function

The COALESCE function is similar to the NVL function, but with an additional twist. Instead of specifying one substitution expression for a null value, you can optionally evaluate multiple substitution columns or substitution expressions. The syntax is as follows.

COALESCE(input_expression, substitution_expression_1, [, substitution_expression_n]);

SQL>SELECT student_id, midterm_grade, finalexam_grade, quiz_grade, COALESCE(midterm_grade, finalexam_grade, quiz_grade) "Coalesce"

FROM grade_summary;

The NVL Function

The NVL function replaces a NULL value with a default value. NULLs represent a special challenge when used in calculations. A computation with an unknown value yields another unknown value, as shown in the following example

SQL>SELECT 60+60+NULL FROM dual;

60+60+NULL

To avoid this problem, you can use the NVL function to substitute the NULL for another value.

NVL(input_expression, substitution_expression)

The NVL function requires two parameters: an input expression (for example, a column, literal, or computation) and a substitution expression. If the input expression does not contain a NULL value, the input parameter is returned. If the input parameter does contain a NULL value, the substitution parameter is returned. In the following example, the substitution value is the number literal 1000. The NULL is substituted with 1000, resulting in the output 1120.

SQL>SELECT 60+60+NVL(NULL, 1000) FROM dual

60+60+NVL(NULL,1000);

Conversion Function

Changing the Date Display Format

When you query a DATE data type column, Oracle displays it in the default format determined by the database NLS_DATE_FORMAT parameter. The most frequent setup values you will see are DD-MON-YYYY and DD-MON-RR. The RR represents a twodigit year based on the century; if the two-digit year is between 50 and 99, then it"s the previous century; if the two-digit year is between 00 and 49, it"s the current century.

SQL>SELECT last_name, registration_date

FROM student

WHERE student_id IN (123, 161, 190);

```
LAST_NAME         REGISTRATION

Affinito           03-FEB-07

Grant             02-FEB-07

Radicola          27-JAN-07
3 rows  selected.
```

To change the display format of the column REGISTRATION_DATE, you use the TO_CHAR function together with a format model, also referred to as a format mask. The result shows the registration date in both the default date format and the MM/DD/YYYY format.

SQL>SELECT last_name, registration_date, TO_CHAR(registration_date, 'MM/DD/YYYY') AS "Formatted";

TO_CHAR(date [,format_mask])

Converts datetime data types into VARCHAR2 to use a different display format than the default date format. (The TO_CHAR function can be used with other data types besides DATE.

TO_DATE(char [,format_mask])

Converts a text literal to a DATE data type. As with all other date-related conversion functions, the format_mask is optional if the literal is in the default format; otherwise, a format mask must be specified.

Performing a Date Search

You"ll find that you often need to query data based on certain date criteria. For example, if you need to look for all students with a registration date of January 22, 2007, you write a SQL statement similar to the following.

SQL>SELECT last_name, registration_date

FROM student

WHERE registration_date = TO_DATE('22-JAN-2007', 'DD-MON-YYYY');

```
LAST_NAME       REGISTRAT

Crocitto        22-JAN-07

Landry          22-JAN-07

Sethi           22-JAN-07

Walter          22-JAN-07
```

In the WHERE clause, the text literal „22-JAN-2007" is converted to a DATE data type using the TO_DATE function and the format model. The TO_DATE function helps Oracle understand the text literal, based on the supplied format mask. The text literal is converted to the DATE data type, which is then compared to the REGISTRATION_DATE column, also of data type DATE. Now you are comparing identical data types.

The format mask needs to agree with your text literal; otherwise, Oracle will not be able to interpret the text literal correctly and will return the following error message.

SQL>SELECT last_name, registration_date

FROM student

WHERE registration_date = TO_DATE('22/01/2007', 'DD-MON-YYYY') WHERE

registration_date = TO_DATE('22/01/2007', 'DD-MON-YYYY');

Introduction Views and data dictionary

Create simple / complex view :

A view is a virtual table that consists of columns and rows, but it is only the SELECT statement that is stored, not a physical table with data. A view"s SELECT query may reference one or multiple tables, called base tables. The base tables are typically actual tables or other views.

Advantages of Views

Views simplify the writing of queries. You can query a single view instead of writing a complicated SQL statement that joins many tables. The complexity of the underlying

SQL statement is hidden from the user and contained only in the view.

Views are useful for security reasons because they can hide data. The data retrieved from a view can show only certain columns if you list those columns in the SELECT list of the query. You can also restrict the view to display specific rows with the WHERE clause of the query.

In a view, you can give a column a different name from the one in the base table. Views may be used to isolate an application from a change in the definition of the base tables. Rather than change the program, you can make changes to the view.

A view looks just like any other table. You can describe and query the view and also issue INSERT, UPDATE, and DELETE statements to a certain extent, as you will see when performing the exercises in this lab.

Creating a View

The simplified syntax for creating a view is as follows.

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

[(column_alias[, column_alias]...)] AS query

[WITH CHECK OPTION|WITH READ ONLY [CONSTRAINT constraintname]] The following statements create a view called COURSE_NO_COST and describe the new view.

SQL:>CREATE OR REPLACE VIEW course_no_cost AS SELECT course_no, description, prerequisite FROM course;

View created.

SQL> DESC course_no_cost;

The COURSE_NO_COST view hides a number of columns that exist in the COURSE table. You do not see the COST column or the CREATED_DATE, CREATED_BY, MODIFIED_DATE, and MODIFIED_BY columns. The main purpose of this view is security. You can grant access just to the view COURSE_NO_COST instead of to the COURSE table itself. For more information on granting access privileges to database objects, see Chapter 15.

Using Column Aliases

The following statement demonstrates a view with column names different from the column names in the base tables. Here the view named STUD_ENROLL shows a list of the STUDENT_ID, the last name of the student in capital letters, and the number of classes the student is enrolled in. The column STUDENT_ID from the STUDENT table is renamed in the view to STUD_ID, using a column alias. When a column contains an expression such as a function, a column alias is required. The two expressions in the STUD_ENROLL view, namely the student last name in caps and the count of classes enrolled, are therefore aliased.

SQL>CREATE OR REPLACE VIEW stud_enroll AS SELECT s.student_id stud_id,

UPPER(s.last_name) last_name, COUNT(*) num_enrolled

FROM student s, enrollment e

WHERE s.student_id = e.student_id

GROUP BY s.student_id, UPPER(s.last_name);

The OR REPLACE keyword is useful if the view already exists. It allows you to replace the view with a different SELECT statement without having to drop the view first. This also means you do not have to re-grant privileges to the view; the rights to the view are retained by those who have already been granted access privileges.

The following example shows an alternate SQL statement for naming columns in a view, whereby the view‟s columns are listed in parentheses after the view name.

SQL>CREATE OR REPLACE VIEW stud_enroll

(stud_id, last_name, num_enrolled) AS SELECT s.student_id,

UPPER(s.last_name), COUNT(*)

FROM student s, enrollment e

WHERE s.student_id = e.student_id

GROUP BY s.student_id, UPPER(s.last_name);

Altering a View

You use the ALTER VIEW command to recompile a view if it becomes invalid. This can occur after you alter one of the base tables. The syntax of the ALTER VIEW statement is as follows.

ALTER VIEW viewname COMPILE

The ALTER VIEW command allows for additional syntax options not mentioned. These options let you create primary or unique constraints on views. However, these constraints are not enforced and do not maintain data integrity, and an index is never built because they can only be created in DISABLE NOVALIDATE mode. These constraint types are primarily useful with materialized views, a popular data warehousing feature that allows you to physically store pre-aggregated results and/or joins for speedy access. Unlike the views discussed in this chapter, materialized views result in physical data stored in tables.

Renaming a View

The RENAME command allows you to change the name of a view.

RENAME stud_enroll TO stud_enroll2

All underlying constraints and granted privileges remain intact. However, any objects that use this view (perhaps another view or a PL/SQL procedure, package, or function) become invalid and need to be compiled.

Dropping a View

To drop a view, you use the DROP VIEW command. The following statement drops the STUD_ENROLL2 view. **DROP VIEW stud_enroll2**

View dropped.

UNDERSTANDING/ Managing User Privileges a Roles

Oracle protects the data in the database by implementing security via users, roles, and privileges. The SQL language commands used to accomplish these security tasks are known as Data Control Language (DCL) commands.

Oracle provides several different ways to enforce access control to ensure that only authorized users can log in and access the appropriate data. You want to avoid situations in which a user can accidentally drop an important table or sidestep security rules. Every database user has certain system privileges that determine the type of actions the user can perform, such as create tables, drop views, or create other users.

Object privileges avoid any wrongful data modifications to individual tables or columns. The owner of database objects can assign these object privileges that control exactly who can access what objects and to what extent.

System and object privileges can be grouped together into a role. Part of a database administrator‟s (DBA‟s) job is to set up the correct security for database users. It is vital that the database be properly protected against any wrongful actions and unauthorized access.

What Is a Schema?

A schema is a collection of objects (for example, tables, views, indexes, sequences, triggers, synonyms). Each schema is owned by a single user account with the same name; in fact, the two terms schema and user account are often used interchangeably.

You can list the types of objects in the STUDENT schema by querying the USER_OBJECTS data dictionary view.

SQL>SELECT DISTINCT object_type FROM user_objects OBJECT_TYPE;

To see all the different types of objects available for the user accessing the database, query the ALL_OBJECTS view. The result set on your database may vary from this result, as various users may have different object types and different privileges.

SQL>SELECT DISTINCT object_type FROM all_objects OBJECT_TYPE; **Special**

Users: SYSTEM and SYS

When an Oracle database is created, it comes with a number of default accounts. Two extremely important accounts are SYS and SYSTEM.

The SYS account is the most privileged user. It owns the data dictionary. If you drop any of the objects of the SYS schema, you endanger the critical operation of the

Oracle database.

The SYSTEM account is automatically granted the DBA role for database administration tasks. This role is typically used to create regular user accounts or accounts with the DBA role. Oracle suggests that you create an administrative type of account after you create the database. You can then use this DBA account to perform daily administrative tasks, without having to use the SYS and SYSTEM accounts.

Creating Users

To log in to the Oracle database, a user must have a username, a password, and certain system privileges. A username is created with the CREATE USER command, which uses the following syntax.

SQL>CREATE USER user IDENTIFIED

{BY password [REPLACE oldPassword]

|EXTERNALLY|GLOBALLY AS 'external name'} [{DEFAULT TABLESPACE tablespace |

TEMPORARY TABLESPACE {tablespace|tablespace_group}| QUOTA {integer[K|M] | UNLIMITED} ON tablespace [[QUOTA {integer[K|M] | UNLIMITED} ON tablespace]...]| PROFILE profile |

PASSWORD EXPIRE | ACCOUNT {LOCK|UNLOCK}

}];

To create a new user, first log in as a user with the rights to create other users (for example, an account with DBA privileges or the user SYSTEM).

The following statement creates a new user called MUSIC with the password listen.

SQL>CREATE USER music IDENTIFIED BY listen

DEFAULT TABLESPACE users

TEMPORARY TABLESPACE temp

QUOTA 15 M ON users;

Changing the Password and Altering the User Settings

When an individual user"s account settings need to change, such as the password or the default tablespace, the user can be altered. The syntax of the ALTER USER command is as follows.

SQL>ALTER USER {user {IDENTIFIED

{BY password [REPLACE oldPassword]| EXTERNALLY|GLOBALLY AS 'external name'} DEFAULT TABLESPACE tablespace |

TEMPORARY TABLESPACE {tablespace|tablespace_group} | QUOTA {integer [K|M] | UNLIMITED} ON tablespace [[QUOTA {integer [K|M] | UNLIMITED} ON tablespace]...]| PROFILE profile |

DEFAULT ROLE

{role [,role]...|ALL[EXCEPT role [,role]...]|NONE}| PASSWORD EXPIRE | ACCOUNT {LOCK|UNLOCK}

}};

The following statement changes MUSIC"s password from listen to tone and changes the default tablespace from USERS to USER_DATA.

SQL>ALTER USER music IDENTIFIED BY tone

DEFAULT TABLESPACE USER_DATA;

User altered.

Dropping Users

You use the following command to drop a user.

SQL>DROP USER music;

User dropped.

The DROP USER command drops the user if the user does not own any objects. The syntax for the DROP USER command is a follows.

DROP USER user [CASCADE]

If you want to also drop the objects owned by the user, execute the DROP USER command with the CASCADE keyword.

SQL>DROP USER music CASCADE;

If the objects and their data need to be preserved, be sure to first back up the data by using the Oracle Data Pump utility or any other reliable method.

What Are Privileges?

A privilege is a right to execute a particular type of SQL statement. There are two types of privileges: system privileges and object privileges. An example of a system privilege is the right to create a table or an index. A particular object privilege allows you to access an individual object, such as the privilege to select from the INSTRUCTOR table, to delete from the ZIPCODE table, or to SELECT a number from a specific sequence.

System Privileges

To establish a connection to the database, a user must be granted certain system privileges. These privileges are granted either individually or in the form of roles. A role is a collection of privileges.

Although the user MUSIC has been created, the user cannot start a session, as you see from the following error message. The user lacks the CREATE SESSION system privilege to log in to the database.

CONN music/tone

ERROR: ORA-01045: user MUSIC lacks CREATE SESSION

privilege; logon denied

For example, if you have the CREATE TABLE privilege, you can create tables in your schema; if you have the CREATE ANY TABLE privilege, you can create tables in another user"s schema. The CREATE TABLE privilege includes the CREATE INDEX privilege, but before you are allowed to create these objects, you must make sure you have either been granted a quota on the individual tablespace on which you would like to place the object or the RESOURCE role, which provides you with unlimited space on all tablespaces.

Examples of System Privileges

System Privilege Name

Session

CREATE SESSION ALTER SESSION

Table

CREATE TABLE

CREATE ANY TABLE ALTER ANY TABLE

DROP ANY TABLE SELECT ANY TABLE

UPDATE ANY TABLE DELETE ANY TABLE

FLASHBACK ANY TABLE

Index

CREATE ANY INDEX

ALTER ANY INDEX

DROP ANY INDEX

Sequence

CREATE SEQUENCE

CREATE ANY SEQUENCE

ALTER ANY SEQUENCE

DROP ANY SEQUENCE

View

CREATE ANY VIEW

DROP ANY VIEW

The GRANT Command

You give a system privilege or an object privilege to a user by using the GRANT command. You can grant privileges individually or through a role. The syntax for granting system privileges is as follows.

GRANT {system_privilege|role|ALL PRIVILEGES} [,{system_privilege|role|ALL PRIVILEGES}]...

TO {user|role|PUBLIC}[,{user|role|PUBLIC}]... [WITH ADMIN OPTION]

The following statement grants the CREATE SESSION system privilege to the MUSIC user. This allows the MUSIC user to establish a session to the database.

SQL>GRANT CREATE SESSION TO music;

Object privileges grant certain privileges on specific objects, such as tables, views, or sequences. You can grant object privileges to other users when you want them to have access to objects you created. You can also grant the user access to objects you do not own if the object‟s owner gives you permission to extend rights to others.

The following is the general syntax for granting object privileges. GRANT

{object_privilege|ALL [PRIVILEGES]} [(column[,column]...)]

[,{object_privilege|ALL [PRIVILEGES]} [(column[,column]...)]]...

ON objectname

TO {user|role|PUBLIC}[,{user|role|PUBLIC}]... [WITH GRANT OPTION]

For example, the following statement connects as the STUDENT user account within SQL*Plus and grants the SELECT privilege on the COURSE table to the new user MUSIC.

CONN student/learn Connected.

SQL>GRANT SELECT ON course TO music;

Grant succeeded.

In this case, the STUDENT user is the grantor, and MUSIC is the grantee, the recipient of the privileges. Now the MUSIC user can query the COURSE table.

In addition to SELECT, other object privileges can be granted on a table, such as

INSERT, UPDATE, DELETE, ALTER, INDEX, and REFERENCES (refer to Table 15.2). The ALTER privilege allows another user to change table definitions with the ALTER table command, the INDEX privilege allows the creation of indexes on the table, and the REFERENCES privilege allows the table to be referenced with a foreign key constraint. You can also grant all object privileges at once by using the GRANT ALL command.

Object privileges can be assigned to other database objects, such as sequences, packages, procedures, and functions. SELECT and ALTER privileges can be granted on sequences. Packages, procedures, and functions require the EXECUTE privilege if other users want to run these stored programs.

If an object, such as a table, is dropped and then re-created, the grants need to be reissued. This is not the case if the object is replaced with the CREATE OR REPLACE keywords available for views and stored programs.

You can grant UPDATE and REFERENCES privileges on individual columns on a table. For example, to grant update on the columns COST and DESCRIPTION of the COURSE table, execute the following command.

SQL>GRANT UPDATE (cost, description) ON course TO music;

Grant succeeded.

Roles

A role is several privileges collected under one role name. Using roles aids in administration of multiple privileges to users. Oracle includes predefined roles; three popular ones that contain a number of different system privileges are CONNECT, RESOURCE, and DBA.

The CONNECT role contains the CREATE SESSION system privilege that allows a user to start a session. (In prior Oracle versions, this role could also create views, tables, and sequences, among other operations.)

The RESOURCE role allows a user to create tables and indexes on any tablespace and to create PL/SQL stored objects (for example, packages, procedures, functions). The DBA role includes all system privileges. This role is usually granted to a user who performs database administration tasks. Table 15.3 lists the system privileges associated with each role.

Role	**Purpose**
CONNECT	Contains CREATE SESSION system privilege.

RESOURCE	Includes these system privileges: CREATE TABLE, CREATE SEQUENCE, CREATE TRIGGER, CREATE PROCEDURE, CREATE CLUSTER, CREATE INDEXTYPE, CREATE OPERATOR, and CREATE TYPE.
DBA	Includes all system privileges and allows them to be granted WITH ADMIN OPTION.

This does not include the privilege to start up and shut down the database.

SQL>GRANT CREATE VIEW TO music
SQL>GRANT CONNECT, RESOURCE TO music

Extending Privileges to Others

To extend an object privilege to another user, you must be the owner of the object or have received the privilege to pass it on to others through WITH GRANT OPTION. You may also pass on the privilege if you have been granted the GRANT ANY OBJECT system privilege.

The following SQL statement grants all object privileges on the COURSE table to the MUSIC user. It also passes to the MUSIC user the ability to grant these privileges to yet other users, using the WITH GRANT OPTION. Here, MUSIC is the grantee but can become a grantor if the privilege is passed on to another user.

SQL>GRANT ALL ON course TO music WITH GRANT OPTION;

To allow users to pass on system privileges to other users, you must have been granted the system privilege with WITH ADMIN OPTION or have been granted the GRANT ANY PRIVILEGE system privilege. For example, after execution of the following statement, the user MUSIC can grant the CREATE SESSION system privilege to other users.

SQL>GRANT CREATE SESSION TO music WITH ADMIN OPTION;

If you want other users to pass on a role to others, you must have either created the role, have been granted the role through the WITH ADMIN OPTION, or been granted the GRANT ANY ROLE system privilege.

You can see which system privileges you received through a role by querying the Oracle data dictionary view ROLE_SYS_PRIVS. For granted system privileges, query the data dictionary views USER_SYS_PRIVS or DBA_SYS_PRIVS. Table 15.4 lists a number of data dictionary views you may find useful when trying to determine individual object privileges, system privileges, and roles.

The REVOKE Command

Privileges can be taken away with the REVOKE command. Use the following syntax to revoke system privileges.

REVOKE {system_privilege|role|ALL PRIVILEGES} [,{system_privilege|role|ALL PRIVILEGES}]...FROM {user|role|PUBLIC}[,{user|role|PUBLIC}]...

The next example shows how the RESOURCE role is revoked from the user MUSIC.

SQL>REVOKE RESOURCE FROM music

Revoke succeeded.

Object privileges can also be revoked as in the following statement.

SQL>REVOKE UPDATE ON course FROM music

Revoke succeeded.

The syntax for revoking object privileges is listed here.

REVOKE {object_privilege|ALL [PRIVILEGES]} [,{object_privilege|ALL [PRIVILEGES]}

ON objectname

FROM {user|role|PUBLIC}[,{user|role|PUBLIC}]... [CASCADE CONSTRAINTS]