indice

[I. First part of the course 3](#_Toc475567377)

[II. In the second part of the course 3](#_Toc475567378)

[III. Third and fourth parts of the course 3](#_Toc475567379)

[IV. Part four 3](#_Toc475567380)

[V. Part five 3](#_Toc475567381)

[1 Week I 3](#_Toc475567382)

[1.1 What databases have you experienced or interacted with today? 3](#_Toc475567383)

[1.1.1 Exemple: 4](#_Toc475567384)

[1.2 What organizational roles in interacting with databases do you want to play? 5](#_Toc475567385)

[1.2.1 Functional vs. information system professionals 5](#_Toc475567386)

[1.2.1.2 Information system professionals 5](#_Toc475567387)

[1.3 How does a DBMS differ from desktop software such as a spreadsheet or word processor? 6](#_Toc475567388)

[1.3.1 Desktop database 6](#_Toc475567389)

[1.3.2 DBMSs 6](#_Toc475567390)

[1.3.3 Desktop database vs. DBMS 6](#_Toc475567391)

[1.3.4 How much have software development costs been reduced by non-procedural database access? 6](#_Toc475567392)

[1.3.5 Query 6](#_Toc475567393)

[1.3.6 Non-procedural access 7](#_Toc475567394)

[1.4 What database transactions have you made today? 7](#_Toc475567395)

[1.4.1 Transactions Definition 7](#_Toc475567396)

[1.4.2 Exemple 1: 7](#_Toc475567397)

[1.4.3 Exemple 2 8](#_Toc475567398)

[1.4.3.3 internal features 8](#_Toc475567399)

[1.5 How do data warehouses affect your life? 8](#_Toc475567400)

[1.5.1 Management levels 9](#_Toc475567401)

[1.5.2 Data warehouse 9](#_Toc475567402)

[1.5.3 Operational databases vs. data warehousing 9](#_Toc475567403)

[2 Week II 10](#_Toc475567404)

[2.1 Relational Data Model and CREATE TABLE Statement 10](#_Toc475567405)

[2.1.1 A relational database 10](#_Toc475567406)

[2.1.2 Exemple 10](#_Toc475567407)

[2.2 Integrity rules 12](#_Toc475567408)

[2.2.1 Objectives 12](#_Toc475567409)

[2.2.2 Entity integrity: primary keys 13](#_Toc475567410)

[2.2.3 Referential integrity: foreign keys 13](#_Toc475567411)

[2.2.4 Examples 13](#_Toc475567412)

[2.2.5 Integrity Rule Violations 14](#_Toc475567413)

[2.2.6 Oracle Relational Diagram 14](#_Toc475567414)

[2.2.7 Summary 16](#_Toc475567415)

[2.3 Basics of the SQL CREATE TABLE Statement 16](#_Toc475567416)

[2.3.1 Lesson Objectives 17](#_Toc475567417)

[2.3.2 CREATE TABLE statement 17](#_Toc475567418)

[2.3.3 Common SQL Data Types 18](#_Toc475567419)

[2.3.4 Syntax Error 18](#_Toc475567420)

[2.3.5 Commun Data Type 18](#_Toc475567421)

[2.4 Integrity Constraint Syntax 19](#_Toc475567422)

[2.4.1 Objectives 20](#_Toc475567423)

[2.4.2 Unique constraints 20](#_Toc475567424)

[2.4.3 Not null 20](#_Toc475567425)

[2.4.4 Check constraints 20](#_Toc475567426)

[2.4.5 Constraint Overview 21](#_Toc475567427)

[2.4.6 Exemples 22](#_Toc475567428)

[2.4.7 Summary 24](#_Toc475567429)

[2.5 Practice Problems for Module 3 25](#_Toc475567430)

[2.6 Extra Problems for Module 3 29](#_Toc475567431)

[2.7 Assignment 1 Notes 34](#_Toc475567432)

[2.7.1 Assignment Details 35](#_Toc475567433)

[2.7.2 Basic CREATE TABLE Statement Requirements 35](#_Toc475567434)

[2.7.3 Constraints 36](#_Toc475567435)

[2.7.4 Populating Tables 36](#_Toc475567436)

[2.7.5 Initial CREATE TABLE Statements 37](#_Toc475567437)

[2.7.6 Submission 37](#_Toc475567438)

[3 WEEK III : Basic Query Formulation with SQL 37](#_Toc475567439)

[3.1 SQL Overview video lecture 41](#_Toc475567440)

[4 Week IV 43](#_Toc475567441)

[4.1.1 Lesson Objectives 43](#_Toc475567442)

[4.1.2 Information System 44](#_Toc475567443)

[4.1.3 Data Bases 44](#_Toc475567444)

[4.1.4 Importance of defining business rules 45](#_Toc475567445)

[4.1.5 Data base quality 45](#_Toc475567446)

[4.1.6 Broad Goals of Database Development 45](#_Toc475567447)

[4.1.7 Database Development Phases 46](#_Toc475567448)

[4.2 Basic ERD notation 46](#_Toc475567449)

[4.2.1 Lesson Objectives 46](#_Toc475567450)

[4.2.2 Basic Symbols 47](#_Toc475567451)

[4.2.3 the major differences between an ERD and a relational database diagram. 48](#_Toc475567452)

[4.2.4 Cardinalities 48](#_Toc475567453)

[4.2.5 Cardinality Notation 49](#_Toc475567454)

[4.2.6 Important Cardinalities 50](#_Toc475567455)

[4.2.7 The differences between ERD notation and relational database diagrams. 51](#_Toc475567456)

[4.3 Relationship variations I 52](#_Toc475567457)

[4.3.1 Lesson Objectives 52](#_Toc475567458)

[4.3.2 Identification Dependency 52](#_Toc475567459)

[4.3.3 M-N Relationship Equivalency Rule 54](#_Toc475567460)

[4.4 Relationship Variations II 55](#_Toc475567461)

[4.4.1 ERD Notation for Self-Referencing Relationships 55](#_Toc475567462)

[4.4.2 Instance Diagrams for Self-Referencing Relationships 56](#_Toc475567463)

[4.4.3 1.1.1 Associative Entity Types for M-way Relationships 57](#_Toc475567464)

[4.5 Exercise Problems for Module 6 58](#_Toc475567465)

# First part of the course

Create tables in a relational database and define integrity rules for the tables.

# In the second part of the course

Write statements in a database language to retrieve data from a relational database.

Apply the query formulation process to transform an unstructured problem statement into a database language statement.

# Third and fourth parts of the course

Data modeling, an important skill to develop a business database.

Understanding the notation of the entity relationship diagram.

# Part four

Create entity relationship diagrams to represent the data requirements of a business situation.

# Part five

Learn to refine an entity relationship diagram into a table design without excessive redundancy. This diagram indicates more topic detail in the flow of topics.

Week I

## What databases have you experienced or interacted with today?

Databases provide vital support for the daily operations and decision making in organizations.

* Daily operations such as: taking orders; making reservations and Paying an employee.
* Decision making tasks such as: resolving customer complaints; choosing suppliers and deciding on the location of a new store.

**Database Management Systems or DBMS**

***Database technology enables organizations to develop and deploy databases.***

Are the enabling tools to create and utilize database in organizations. Database management technology is a major part of the software industry with revolutionary evolvement over 40 years. This evolvement continues today.

Databases management technology provides the foundation to manage long-term memory of organizations. Memory about customers, employees, orders, products and other vital entities

***Data are raw facts coming in basically two varieties****.*

**Conventional facts** are names, addresses, dates of birth, interest rates, sales amounts and so on. Conventional raw facts usually involve numbers and texts that are encountered in everyday speech.

**Unconventional facts**, once thought as media, separate from conventional facts due to the advances in digital capture and manipulation of media, unconventional facts such as images shot on a cellphone, maps, engineering drawings, product videos, fingerprints and web pages are now considered data.

*Raw facts need interpretation, combination, formatting etc., to be used for decision-making. The transformation from data to information may be as simple as formatting a number with currency symbols and commas.*

**Three essential characteristics of Database.**

1. **Persistency**- denoted by the image of data storage devices, a magnetic disk and flash drive. Persistency means that data lasts longer than the state of a computer memory when a computer shuts down. Organizations need reliable long-term memory to operate successfully.
2. **inter-related-** Businesses must remember customers, employees, products, orders, courses, shipments and other vital entities. But businesses must also remember relationships among entities such as what orders have been placed by a customer, what products have been shipped, and what students have enrolled in a course;
3. **Databases are shared with many applications** - using the database, and many users simultaneously interacting with the database. For example, many customers may be simultaneously making airline reservations, ordering products and applying for a position. The database may support many applications such as orders, shipments, accounting and customer service.

### Exemple:

A database to support the operations and decision-making in a university. If you have taken university courses, you have interact with the university database. If not, you have been enrolled in K-12, you've interacted with a similar database. The entities in an organization should remember in this database are students, faculty, courses, offerings of courses and enrollments. Remembering these entities is not enough though ***(Persistency).*** The relationships among these entities are also vital to remember over time ***(inter-relation).*** Faculty teaching course offerings, students enrolling in course offerings, offerings schedule of courses are some of the important relationships that must be remembered over time. A university database supports a variety of applications such as registration, grade recording, assignment of faculty to course offerings and scheduling of courses. Many students, perspective students, faculty and administrators can be using the database simultaneously ***(shared by multiple users and usages)***.

## What organizational roles in interacting with databases do you want to play?

### Functional vs. information system professionals

#### Functional users

Who interact with databases as part of their work. Functional users can play a passive or active role when they're acting with databases. Indirect usage of a database is passive role. An indirect user is given a report or some data extracted from a database.

**A parametric user** is more active than an indirect user. A parametric user requests existing forms or reports using parameters, input values that change from usage to usage. For example, a parameter may indicate a date range, a sales territory, or department name.

**The power user** is the most active. Because decision making needs can be difficult to predict, ad hoc or unplanned usage of a database is important. A power user is skilled enough to build a formal report when needed. Power users should have a good understanding of non-procedural access, a skill described in the next lesson.

#### **Information system professionals**

Who participate in designing and implementing databases. They interact with databases as part of developing an information system.

**Developers** are responsible for collecting requirements, designing applications and implementing information systems. They create and use database views to develop forms, reports, and other parts of information system.

**Management** has an oversight roll in the development of databases in information systems.

***So information system professionals and developer's roles should have a good knowledge of database development and application development.***

One person may simultaneously play more than one role. For example, a functional user in a job such as financial analysis may play all three roles in different databases. In some organizations the distinction between functional users and information assistance professionals is blurred. In these organizations functional users may participate in designing and using databases.

#### Database Administrators and Data Administrator

**Database Administrators**  -Assist both information system professionals and functional users. Have a variety of both technical and nontechnical responsibilities. Technical responsibilities are more detail-oriented. Nontechnical responsibilities are more people-oriented.

**Data Administrator** -is a management role with the responsibilities to plan development of new databases and control usage of data throughout an organization.

|  |
| --- |
| The Data Administrator role typically has broader responsibilities than the Database Administrator role. |
| A Data Administrator primarily has planning and policy setting roles, while a Database Administrator has a more technical role focused on individual databases and DBMS's. |
| **A data administrator also use data resources in a broader context, considers all kinds of data, both traditional business data and non-traditional unstructured data, such as images, video, and social media.** |

## How does a DBMS differ from desktop software such as a spreadsheet or word processor?

A database measured system or DBMS is a collection of components that supports the creation, use and maintenance of databases.

Initially, DBMS is provided efficient storage and retrieval of data. Due to marketplace demands and product innovation, DBMSs have evolved to provide a broad range of features for data acquisition, storage, dissemination, maintenance, retrieval and formatting.

***“Because DBMS is continue to evolve, you must continually update your knowledge”.***

### Desktop database

This versions support small workgroups with smaller databases, tens of simultaneous users and modest reliability and performance requirements. When developing a document or spreadsheet, little or no planning is required. As a document or spreadsheet develops, styles and formulas can be added to provide dynamic planning for the entire document or spreadsheet.

### DBMSs

In contrast, planning is essential for databases, even for small workgroup databases. A diagram showing the types of entities and relationships should be developed to support the intended usage of the database before a database can be implemented. The diagram provides a visual representation of the interrelated characteristic of databases, showing the types of entities, courses, faculty, offerings of a course, students enrollment in relationships among the types of entities.

These diagrams are not standard across database products like the SQL create table statement. For most commercial database products, an important feature will be a graphical tool to plan the type of information and the relationships in a database

The structured query language known as SQL provides the create table statement to define tables and relationships using the structured computer language. SQL provides a loose standard for the entire database software industry.

### Desktop database vs. DBMS

The distinguishing feature of database software from standard desktop software is the need to define and plan the details of the database before building and using the database. Although the structure of the database can be refined after deployment, the initial planning is essential and often laborious. The success of a database is contingent on the level of planning and design before the database is deployed.

### How much have software development costs been reduced by non-procedural database access?

### Query

A query is a request for data to answer a question. For example, the user may want to know the customers having large balances, or products with strong sales in a particular region.

### Non-procedural access

Non-procedural access allows users with limited computing skills to submit queries. The user specifies the part of a database to retrieve, not the implementation details of how retrieval occurs. Most importantly, users do not need to write complex procedures with loops.

Non-procedural database languages *do not have looping statements such as the statements for, while, and so on, because only the parts of the database to retrieve are specified*.

Non-procedural access can reduce the number of lines of code by a factor of 100 as compared to procedural access. Because a large part of business software development involves data access;

Non-procedural access can provide a dramatic improvement in software productivity, two orders of magnitude or a hundred times.

Non-procedural access makes form and report creation possible without extensive coding.

Non-procedural access reduces development time substantially because the detailed coding for database access is not necessary to write.

Non-procedural access in application development tools, though convenient and powerful, are sometimes not efficient enough or do not provide really the level of control necessary for application development. When these tools are not adequate, DBMSs provide an interface to use non-procedural database language statements inside a computer programming language. Most commercial DBMSs have a procedural language interface. For example, Oracle has the language PL/SQL and Microsoft SQL server has a language Transact-SQL.

## What database transactions have you made today?

Transaction processing has strong analogy to production management. Whereas production management involves the control of physical goods, transaction management involves the control of information goods. Transaction management, like management of physical goods is enormously important to modern organizations. Organization such as banks with automatic tellers, airlines with online reservations systems, and universities with online registrations could not function without reliable and efficient transactions processing.

### Transactions Definition

Transactions are units of information work that must be processed together. DBMS's provide services that ensure reliable transaction processing with no data losses from concurrent users and failures after completion.

A database transaction involves a collection of operations that must be reliably processed is one unit of work. So transactions are all or nothing meaning that all operations of a transaction must either succeed or the entire transaction is cancelled.

Transactions should be processed reliably so that there's no loss of data due to interference among concurrent users, users operating at the same time, and failures such as operating system crashes. The requirement for no loss of data adds overhead, but this requirement is crucial to the success of organizations.

### Exemple 1:

#### Pseudo Code

#### Explanation

Now let's look at some pseudocode for an airline transaction. This code indicates the data request from a user, such as flight preferences, in the interaction with the database such as updating a flight record. Making an airline reservation typically involve reservations for the departure and return. To a traveler, the combination of the departure and the return is a transaction, not the departure and the return separately. If a departure and return were considered separately, a traveler may make a departure without obtaining a desired return flight. In the pseudo code example, you should note that the multiple database operations are necessary. Two update operations for the departure and return records and one INSERT operation for the reservation record are necessary. You should also note the exception handling indicted by the line with On Error. If any kind of error occurs, such as loss of connection, any partial actions are removed with a ROLLBACK statement. After the COMMIT executes, all actions are recorded in a database without interference from multiple users and loss of data from failures.

### Exemple 2

#### Pseudo Code

#### Explanation

For a second example, let's consider ATM transaction. These transactions have been common since the early 1970s during the early days of database management software and database development. The structure of the ATM transaction is similar to the airline reservation example. The ATM transaction example shows data requested from the user such as account number and transaction type. Multiple database actions occur such as posted to debt and corresponding credit to the bank's accounts. If failure occurs before the transaction terminates all actions are removed or undone with the on error and rollback line.

#### internal features of a DBMS

Database management systems provide two services to ensure reliable transaction processing. DBMS's ensure that concurrent users, that is users operating at the same time, do not overwrite each other or cancel each others actions. Without proper control, work can be lost as one concurrent user overrides another concurrent user. DBMS's also ensure that a failure after a transaction finishes will not result in loss of data.

Concurrency for controlling actions of concurrent users and the recovery manager for dealing with failures, such as communication errors and software crashes.

It is important to note that these services are internal and transparent to the database developers. In common usage, transparency means that you can see through an object, rendering its inner details invisible. For DB messes, transparency means that the inner details of transaction services are invisible. Transparency is very important because services that ensure reliable processing are really difficult to implement. By providing these services, DBMS has improved the productivity of database developers because database developers do not need to write any code to use these services. However these services have overhead that may require more resources such as servers, disks, and memory. This lesson has introduced the ideas of transactions and transaction processing.

## How do data warehouses affect your life?

The traditional decision-making hierarchy depicts management levels in volume of decisions at each level.

### Management levels

#### Lower level -short-term problems

Lower level management deals with the short-term problems related to individual transactions and daily operations. Typical operational decisions involve resolution of shipment delays, scheduling employees and restocking products.

#### Middle management -Typical tactical decisions

Middle management makes decisions to implement organizational strategies, typically on an annual basis. Typical tactical decisions involve forecasting annual sales, choosing suppliers and related contract terms, and determining the annual staffing levels.

#### Top-level management - strategic decisions

Top-level management make strategic decisions that guide the direction of an organization in the long run. Typical strategic decisions involve identification of new markets, determining pricing levels and choosing new locations for plants and stores.

Operational databases directly support major functions such as order processing, manufacturing, accounts payable and product distribution. The reasons for investing in an operational database are typically:

* faster processing;
* larger volumes of business and reduced personnel costs;
* Operational databases provide the raw materials for management decision-making;
* Lower level management can obtain exception and problem with reports directly from operational databases.

However, much value must be added to leverage the operational databases for middle and upper management. The operational databases must be summarized and integrated to provide value for tactical and strategic decision-making. Integration is necessary because operational database often are developed in isolation without regard for the information needs of tactical and strategic decision-making.

Strategic decision-making also requires data sources external to an organization such as industry and government data sets.

### Data warehouse

Data warehouse, a term coined by William Inmon in 1990, refers to a logically centralized data repository where data from operational databases and other sources are integrated, cleaned and standardized to support business intelligence.

Tangible benefits from a data warehouse can include increased revenue and reduce expenses enabled by business analysis that was not possible before the data warehouse was deployed.

For example, a data warehouse may enable reduced losses due to improved fraud detection, improved customer attention to target marketing and reduction in inventory carrying cost through improved demand forecasting.

### Operational databases vs. data warehousing

Transaction processing relies on operational databases with current data at the individual level. While business intelligence processing utilizes data warehouses with historical data at both the individual and summarized levels.

Individual-level data provides flexibility for responding to a wide range of business intelligence needs. While summarized data provides fast response to repetitive queries. For example, an order entry transaction requires data about individual customers, orders and inventory items, while business intelligence applications may use monthly sales over a period of several years.

Operational databases therefore have a process orientation. For example, all data relevant to a particular business process such as order entry compare to a subject orientation for data warehouses, for example, all customer data or all order data.

A transaction typically updates only a few records, whereas a business intelligence application may query thousands to millions of records.

Business intelligence processing requires substantial daily processing using non-peak hours for transformations and integration.

Data warehouses are an essential infrastructure to support tactical decision-making with a medium-term impact, and strategic decision-making with a long-term impact. Business intelligence processing involves substantial processing of data from operational databases and external data sources. For transformations and integration, as well as large amounts of data for reporting.

# Week II

## Relational Data Model and CREATE TABLE Statement

What I want you to think about throughout this lesson, why is the relational data model commercially dominant?

Identify components in sample tables using additional rows that you added.

Because tables are used to communicate ideas in many fields, the terminology of tables, rows, and columns is familiar to most users.

### A relational database

A relational database consists of a collection of tables. Each table has a heading, or definition part, and a body, or content part.

The heading part consists the table name and column names. For example, a student table may have columns for student number, student address, city, state, zip, class, major, and cumulative grade point average.

The body shows the rows of the table. Each row in a student table represents a student enrolled at a university.

### Exemple

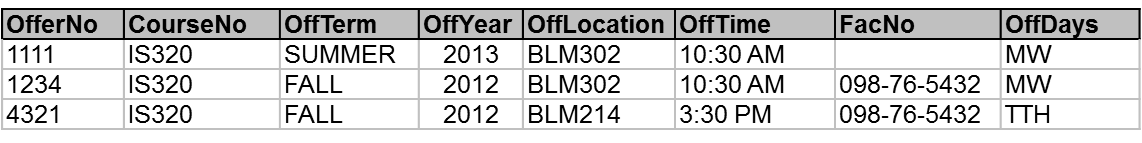
A student table for major university may have, say, 30 to 50 columns, and more than 30,000 rows, too many to view at one time.

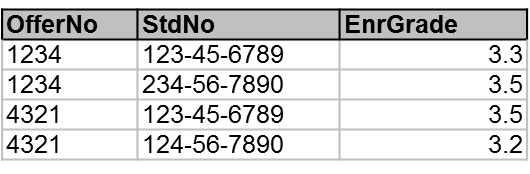
To understand the table, it's also useful to view some of its rows.

A table listing or data sheet shows the column names in the first row in the body in the other rows. The sample student table contains three sample rows, representing university students.

To facilitate communication, the naming convention for column names uses a table abbreviation std followed by a descriptive name.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **StdNo** | **StdFirstName** | **StdLastName** | **StdCity** | **StdState** | **StdZip** | **StdMajor** | **StdClass** | **StdGPA** |
| 123-45-6789 | HOMER | WELLS | SEATTLE | WA | 98121-1111 | IS | FR | 3.00 |
| 124-56-7890 | BOB | NORBERT | BOTHELL | WA | 98011-2121 | FIN | JR | 2.70 |
| 234-56-7890 | CANDY | KENDALL | TACOMA | WA | 99042-3321 | ACCT | JR | 3.50 |





**Because column names are often used without identifying the associated tables, the abbreviation supports easy table association. Mixed case in a name, highlights the different parts of a column name. It is not enough to understand each table individually. To understand a relational database, connections or relationships among tables must also be understood. The rows in a table are usually related to rows in other tables matching or identical values indicate relationships between tables.**

So consider the sample enrollment table, in which each row represents a student enrolled in an offering of a course. The values in the std number column of the enrollment table match the std number values in the sample student table. For example, the first and third rows, the enrollment table had the same std number of value, 123-45-6789, as the first row of the student table. Likewise, the values in the offer number column of the enrollment table match the offer number column in the offering table.



This graphical depiction of matching values, can clarify the concept of relationships among tables. In the table listings, the matching values are difficult to identify.

This graphical depiction only works on small sample tables as arrows cross and become difficult to follow with more rows. In the graphical depiction, you should see that the first student row 123-45-6789 is related to the first and third enrollment rows. Likewise, the first offering row, 1234, matches to the first two enrollment rows.

The concept of matching values is crucial in relational databases. To extract meaningful information, it is often necessary to combine multiple tables using matching values. For example, if I'm matching on student dot student number, enrollment dot student number, you could combine the student and enrollment tables. Similarly, by matching an enrollment dot offer number and offering dot offer number, you could combine the enrollment and offering tables. Understanding the connections between tables or columns in which tables can be combined is crucial for extracting useful information.

|  |  |  |
| --- | --- | --- |
| **Table-Oriented** | **Set-Oriented** | **Record-Oriented** |
| Table | Relation | Record Type, File |
| Row | Tuple | Record |
| Column | Attribute | Field |

You should be aware that other terminology is used besides table, row, and column. The diversions in terminology is due to the different groups that use databases.

**The table-oriented** terminology originally appealed to end users.

**The set-oriented** terminology appeals to academic researchers.

**The record-oriented** terminology really was originally appealed to information systems professionals.

You should expect to see a mix of terminology in your career.

The relational data model is commercially dominate for several reasons:

* To organize data is simple and familiar.
* The relational data model has strong theoretical foundation from academic and commercial research and many commercial university and open source products have implemented the relational model over more than 40 years.
* In addition, there's an important commercial standard.

## Integrity rules

First question, what is the consequence of two taxpayers or customer with the same government identifier or customer identifier?

Second question, what is the consequence of a shipment associated with the wrong order?

### Objectives

* Identify 1-M relationships and associated PKs and FKs
* Find errors in rows with either orphan FKs or missing FKs
* Identify situations for FK requirements
  + FK is necessary
  + FK can have the null value

### Entity integrity: primary keys

Entity integrity means that each table must have a column or combination of columns, that is two or more columns known as the primary key.

* Each table has column(s) with unique values
* No missing values for primary keys
* Ensures traceable entities

The primary key of each table must have unique values with no rows having a missing value. Unique means that no two rows of the table have the same value. In the university database example from lesson one, student number and student is unique in the combination of student number and offer number is unique in the enrollment table.

Entity integrity ensures entities, people, things, places and events are uniquely identified in a database. For auditing, security and communication reasons, it is important that business entities are easily traceable and unique.

### Referential integrity: foreign keys

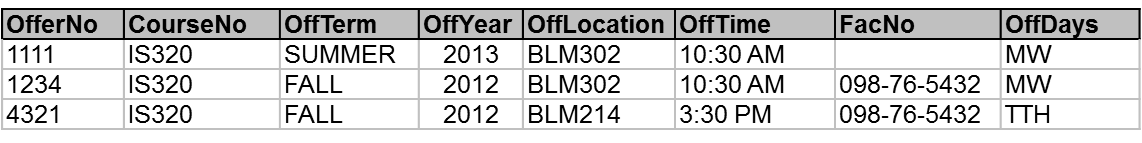
Referential integrity means that the column values in one table must match column values in a related table. For example, the value of student number in each row of the enrollment table must match the value of student number in some row of the student table.

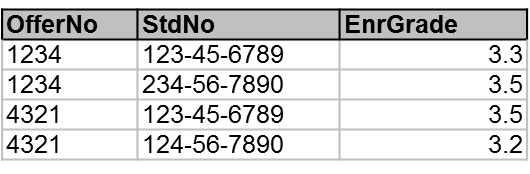
* Values of a column in one table match values from a source table
* Ensures valid references among tables

Referential integrity ensures that a database contains valid connections. For example, it is critical that each row of the enrollment table contains a student number of a valid student. Otherwise, some enrollments can be meaningless, possibly resulting in a student's denied enrollment, because non-existing students took their places.

### Examples

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **StdNo** | **StdFirstName** | **StdLastName** | **StdCity** | **StdState** | **StdZip** | **StdMajor** | **StdClass** | **StdGPA** |
| 123-45-6789 | HOMER | WELLS | SEATTLE | WA | 98121-1111 | IS | FR | 3.00 |
| 124-56-7890 | BOB | NORBERT | BOTHELL | WA | 98011-2121 | FIN | JR | 2.70 |
| 234-56-7890 | CANDY | KENDALL | TACOMA | WA | 99042-3321 | ACCT | JR | 3.50 |





- Student rows are uniquely identified by StdSSN

- Offering rows are uniquely identified by OfferNo

- Enrollment rows are uniquely identified by the combination of StdSSN and OfferNo

- Enrollment.StdSSN refers to a valid StdSSN value in the Student table

- Enrollment.OfferNo refers to a valid OfferNo in the Offering table

### Integrity Rule Violations

***A database diagram is an important tool for understanding relationships and identifying integrity rules***

The fourth row of the student table has a missing value. The missing value is known as a null value. A null value means that the value has not been entered by a user. So the value is unknown or in some cases, does not apply to the row.



missing values are shown by hyphens(…)

There are two violations of the entity integrity rule and one violation of the referential integrity rule:

* Missing value for PK of Student table (4th row)
* Orphan row in Enrollment: 5th row with OfferNo 6789
* Missing value of part of a PK for Enrollment (6th row)
* Add invideo quiz questions for the violations.

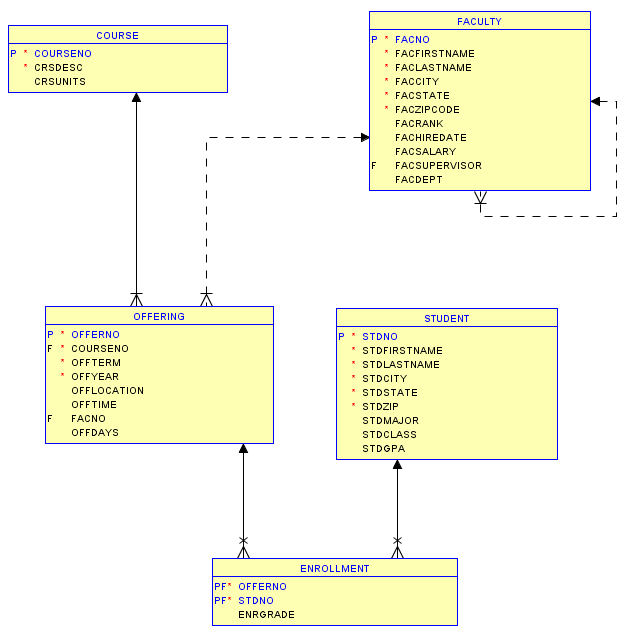
### Oracle Relational Diagram

The Oracle relational diagram shows the tables in relationships of the university database.

**Primary key** of the table is marked with a P. For example, course number is the primary key of the course table.

**Foreign keys** are marked with the FK abbreviation. For example, the offering table has two foreign keys. Course number referencing a course table and fact number referencing a faculty table.

**Lines represent relationships with referential integrity constraints**. Each relationship has a parent table appearing next to the arrow in the line and a child table appearing next to the foot symbol.



**PF-**both part of the primary key and the foreign key

**dashed line** - the forward key can accept missing or null values

**One to many relationship**-A course can be related to many offerings.

**Solid line** means that the foreign key cannot have missing values

Parent

child

For example, the line between course and offering means that a relationship exists from course to offering with *course as the parent table* and *offering as the child table*.

Therefore, you can say that there is a **one to many relationship**- This relationship means that a course can be related to many offerings.

A **solid line** means that the foreign key cannot have missing values. For example, the values in course number column of the offering table cannot have missing or null values, because the relationship line is solid in the diagram.

A **dashed line** in the diagram means that the forward key can accept missing or null values. For example, fact number four in key in the offering table can accept null values is the faculty for an offering may not be assigned until after the offering is initially entered into the database.

The **PF abbreviation**, indicates that the column is both part of the primary key and the foreign key. For example, offer number and student number in the enrollment table are both components of the primary key. The primary key is a combination of both columns. In addition, they are both foreign keys to the offering and student tables respectively.

Most tables are combined using a relationship with a primary key from a parent table matching to a foreign key in a child table. For example, you would typically combine the course and offering tables on course number in the course table matching the course number in the offering table.

**Oracle Relational Diagram**

* Created in Oracle SQL Developer
* Select New Design in Data Modeler -> Browser
* Drag tables into design window
* View Details: show only columns in this diagram
* Can also show other details such as data types

Notation

* Solid line: mandatory relationship (NOT NULL constraint for FK)
* Dashed line: optional relationship (NULL values allowed)
* Cross: FK is part of PK

### Summary

A database diagram is essential for understanding the major components, primary keys, foreign keys and relationships of a database and formulating queries that combine tables.

* ***To understand a relational database, you need to identify primary keys and foreign keys.*** 
  + Primary keys should have unique values for each row.
  + No part of a primary key can have a missing or null value.
  + Understanding existing databases is crucial to query formulation
  + Foreign keys must have a value matching a primary key value in a parent table.
  + Some foreign keys, however, can allow null values such as an offering not having an assigned faculty.
* ***Visualize relationships:*** A database diagram provides a convenient representation, highlighting primary keys, foreign keys in relationships from parent tables to related child tables.

Valid reference problems

* Orders without a customer or incorrect customer
* Order without a shipment
* Missing reference values represent valid rows (internet order without an employee) or data entered later (offering with instructor reference later).

Understand a database is a prerequisite to query formulation

- How are rows identified? PKs and CKs

- What data can be compared? Data type knowledge

- How can tables be combined? Foreign keys and relationship details (1-M relationships)

- Visualization: show the direct and indirect connections among tables

## Basics of the SQL CREATE TABLE Statement

Why have DBMS vendors created alternative visual interfaces for the create table statement?

How can a missing comma ruin your day?

### Lesson Objectives

* Write CREATE TABLE statements with column specifications including data types
* Read CREATE TABLE statements to see columns and associated data types

### CREATE TABLE statement

This statement is used to create tables in a Relational DBMS and specify details about each column in a table.

You should be able to write create table statements specifying column definitions with data types. You should also be capable of reading similar create table statements written by others.

The CREATE TABLE statement can be used to define the columns and integrity constraints of a table.

#### CREATE TABLE Syntax

CREATE TABLE is a statement in the structured query language SQL. Because SQL is an industry standard language; can be used to create tables in most DB messes.

* CREATE TABLE ( <column-list> [<constraint-list>] )
* Column list with data types and optional and inline constraints

#### Optional external constraint list

* + CONSTRAINT [ ConstraintName ] <Constraint-Spec>
  + Primary key
  + Foreign key
  + Unique
  + Check

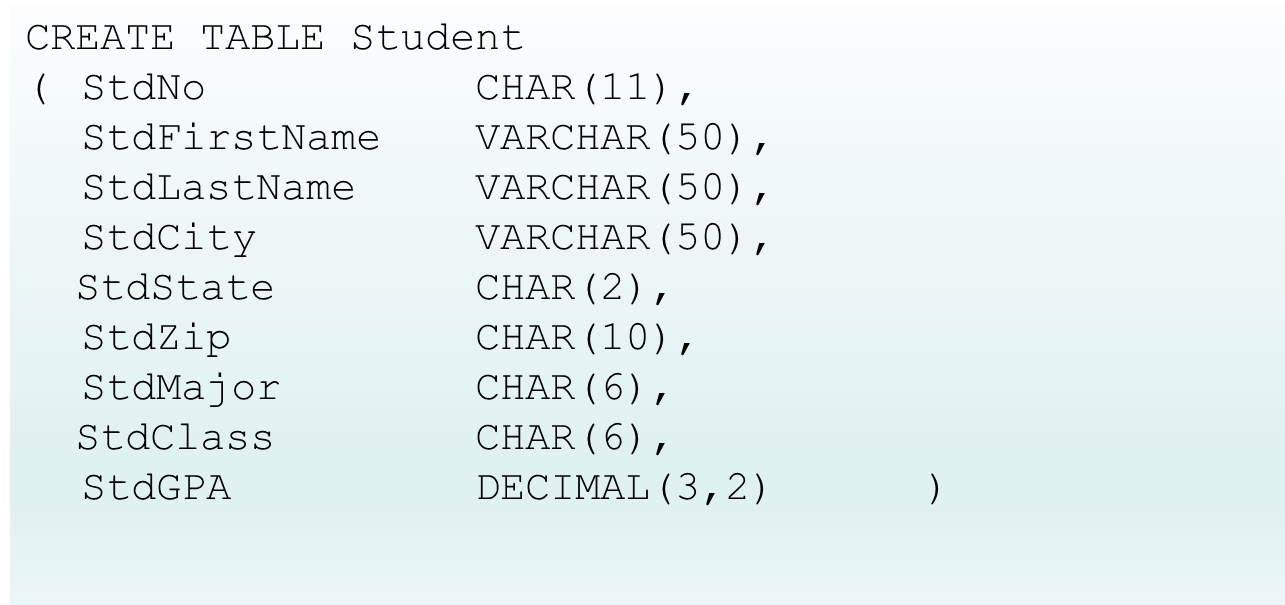
CREATE TABLE keywords followed by parentheses and column list with an optional constraint list.

Column list: list of column definitions separated by commas

* Data type
* Optional default value (DEFAULT keyword and value)
* Optional inline constraint typically NOT NULL

Constraint list:

* CONSTRAINT keyword followed by optional constraint name
* Constraint specification: keyword, column name or condition
* Constraints covered in next lecture



The statement begins with the key words CREATE TABLE. Following the CREATE TABLE keywords you need a left parenthesis. Inside the parentheses you need a list of columns, and an optional list of constraints.

In the syntax, the square brackets mean that the element is optional. CREATE TABLE ( <column-list> [<constraint-list>] )

A column list involves a column name and optional in line constraints. After the column list have an optional list of external constraints.

### Common SQL Data Types

* Char(L);
* Varchar(L)
* Integer
* Float(P)
* Decimal(W, R)
* Date/Time: Date, Time, Timestamp
* Boolean

Data types indicate the kind of data. Character, numeric, and so on and permissible operations, such as numeric operations and string operations such as subset for the column. Each data type has a name for example char for character And usually a length specification.

### Syntax Error

It is vital to pay close attention to the syntax. DBMS compilers and the software that reads SQL statements are very exacting. Here are some notes about the syntax shown in the statement:

Keywords much be spelled exactly. For example, misspelling table as tble will cause a syntax error. Database compilers, unfortunately, do not have spelling correction to help you. Each column definition is terminated by a comma.

A missing comma will result in a syntax error with a message that will not be easy to understand.

### Commun Data Type

The entire statement terminates with a right parenthesis. A key part of a column definition is the data type specification.

The number in parenthesis indicates the maximum length. The DB mess will always store the maximum length for chart columns. So you should use this data type for columns with the same length such as two character abbreviations for state names.

**Char(L)** C-H-A-R is a fixed length character string, or text. The number in parenthesis indicates the maximum length. The DB mess will always store the maximum length for chart columns. So you should use this data type for columns with the same length such as two character abbreviations for state names.

**Varchar(L)** means a variable length character string. The number in parenthesis indicates the maximum length. The DBMS will always store the length actually used for the value, not the maximum length. So you should use this data type for columns with varying length, such as student names. Note that oracle uses var char 2 instead of the sequel standard var char.

**Integer indicates** a whole number, that is, numbers without a decimal point. Use this data type for columns with numbers such as age in years.

**Float(P) indicates** a number with a floating precision, such as interest rates and scientific calculations. The precision parameter P indicates a number of significant digits.

**Decimal(W, R)** Decimal indicates a number with a fixed precision, such as monetary amounts. The W indicates the total number of digits and the R indicates the number of digits to the right of the decimal point.

Use the **DATE data type for columns with dates and times**.

Some systems support three data types, **DATE, TIME and TIMESTAMP**. Weather systems support a combined data type, DATE, storing both the date and time. Let's wrap up this lesson, giving initial details about the CREATE TABLE statement. You've learned the basic syntax of the create table statement covering column statements.

* Important definitional statement
* Data types not always portable
* Somewhat tedious specification although relatively portable
* Other interfaces for more productivity
* CREATE TABLE statement important because of relative portability
* Other interfaces make DBAs more productive

## Integrity Constraint Syntax

Why are constraint names important? Imagine that you are on call as a database administrator. An error occurs but the error name is meaningless.

The create table statement allows five different subjects in constraints. Primary key, foreign key, unique, required, and check.

Unique constraints indicate that a column, or combination of columns, as unique, that is non-duplicate values for a table. Unique constraints are not specified for primary keys. As the primary key constraint already implies uniqueness.

Unique constraints are specified for candidate keys. These are other columns besides the primary key, that are also unique at a table.

For example, student table may have STD number, as the primary key. An STD email as the candidate key specified with a unique constraint. Required constraints using the not null keywords indicate that a value must be entered for a column with no values not allowed.

Why are constraint names important?

### Objectives

* Read and write CREATE TABLE statements with PK constraints
* Read and write CREATE TABLE statements with FK constraints
* Read and write CREATE TABLE statements with simple CHECK constraints

Extend details in CREATE TABLE statement; Write syntactically acceptable statements; Specify details correctly

Be able to read and write CREATE TABLE statements containing primary key, foreign key, unique, and check constraints

The create table statement allows five different subjects in constraints:

Primary key; Foreign key, unique, required, and check.

***Required constraints are usually specified inline as they apply to individual columns, not combination of columns.***

### Unique constraints

Unique constraints indicate that a column, or combination of columns, as unique, that is non-duplicate values for a table. ***Unique constraints are not specified for primary keys. As the primary key constraint already implies uniqueness***.

Unique constraints are specified for candidate keys. These are other columns besides the primary key, that are also unique at a table. For example, student table may have STD number, as the primary key. An STD email as the candidate key specified with a unique constraint.

### Not null

Required constraints using the not null keywords indicate that a value must be entered for a column with no values not allowed.

### Check constraints

Check constraints specify conditions that a column must satisfy such as limits on a column containing a year value. Constraints may be placed inline as part of a column definition, or external after column definitions. Constraints involving more than one column must be placed external to column definitions, as you will see.

Check constraints specify conditions that a column must satisfy such as limits on a column containing a year value. Constraints may be placed inline as part of a column definition, or external after column definitions. Constraints involving more than one column must be placed external to column definitions, as you will see.

1. Let's begin with examples that isolate just the constraint part of the statement. Later parts of this lesson, we'll put the constraints into complete CREATE TABLE statements to show their placement. The first two examples to pick primary key constraints:
   * CONSTRAINT PKCourse PRIMARY KEY(CourseNo)
   * CONSTRAINT PKEnrollment PRIMARY KEY (OfferNo, StdNo)
2. The second example must be placed externally, because a primary key is a combination of two columns. Offer number and Std Number:
   * CONSTRAINT PKEnrollment PRIMARY KEY (OfferNo, StdNo)
3. The third example is a unique constraint with a unique keyword.
   * CONSTRAINT UniqueCrsDesc UNIQUE (CrsDesc)
4. The fourth example defines a foreign key constraint with a foreign key column that is offer number, and parent table offering specified in the constraint.
   * CONSTRAINT FKOfferNo FOREIGN KEY (OfferNo) REFERENCES Offering
5. The last example defines a required constraint with the NOT NULL keywords:
   * CONSTRAINT OffCourseNoReq NOT NULL

* CONSTRAINT keyword
* Optional constraint name
* Oracle syntax: MySQL has some limitations
* All of these constraints can be external or inline in Oracle and standard SQL. Typically required (NOT NULL) constraints are inline and others are external.
* Keyword(s) about constraint type

Required constraints are usually specified inline as they apply to individual columns, not combination of columns. Each example begins with a constraint keyword followed by a constraint name.

The constraint name is optional but good practice to identify the error when a constraint violation occurs.

The keyword constraint is optional for inline constraints, but required for external constraints.

This CREATE TABLE statement for the course table shows two external constraints.

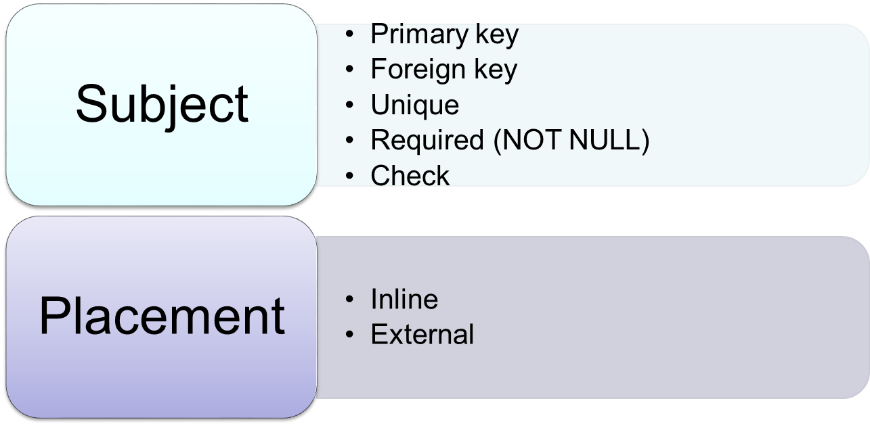
The constraint named PKcourse defines a primary key constraint.

### Constraint Overview

Constraint subject

Placement

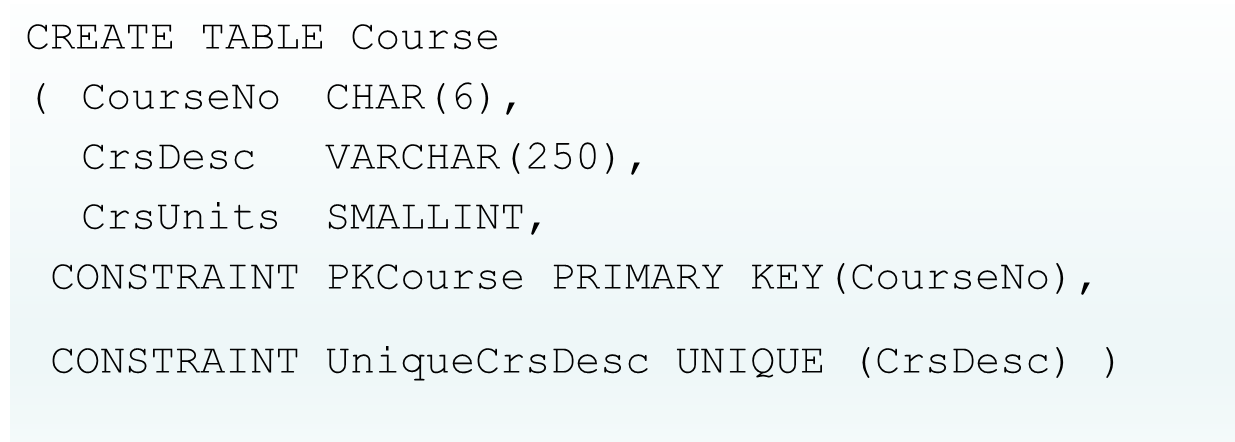
* External: after column definitions
* Inline: same line as a column definition



### Exemples

#### External PK Constraint Placement

This CREATE TABLE statement for the course table shows two external constraints. The constraint named PKcourse defines a primary key constraint. The constraint named UniqueCrs description defines a unique constraint for the Crs description column. Both constraints could be placed inline because they involve a single column.



Extended CREATE TABLE statement

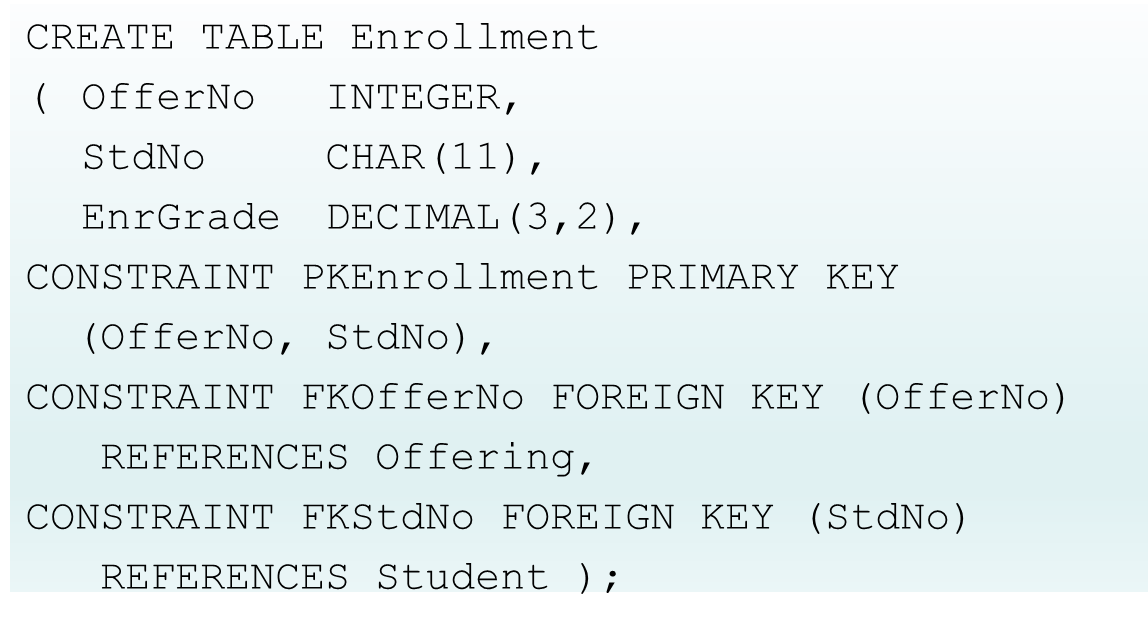
External primary key constraint: CourseNo

Candidate key: CrsDesc (course description)

Named constraints: easier to reference; PKCourse, UniqueCrsDesc

#### External FK Constraint Placement

This constraint statement for the enrollment table shows three external constraints. The constraint named PKEnrollment defines a primary key constraint for the combination of offer number and Std number. This constraint must be external as it involves multiple columns. The other external constraints involve foreign keys. These constraints can be inline or external because they involve only single columns.



Primary key:

- combination of OfferNo and StdNo

- combined PK (or composite PK)

- Use as external constraint

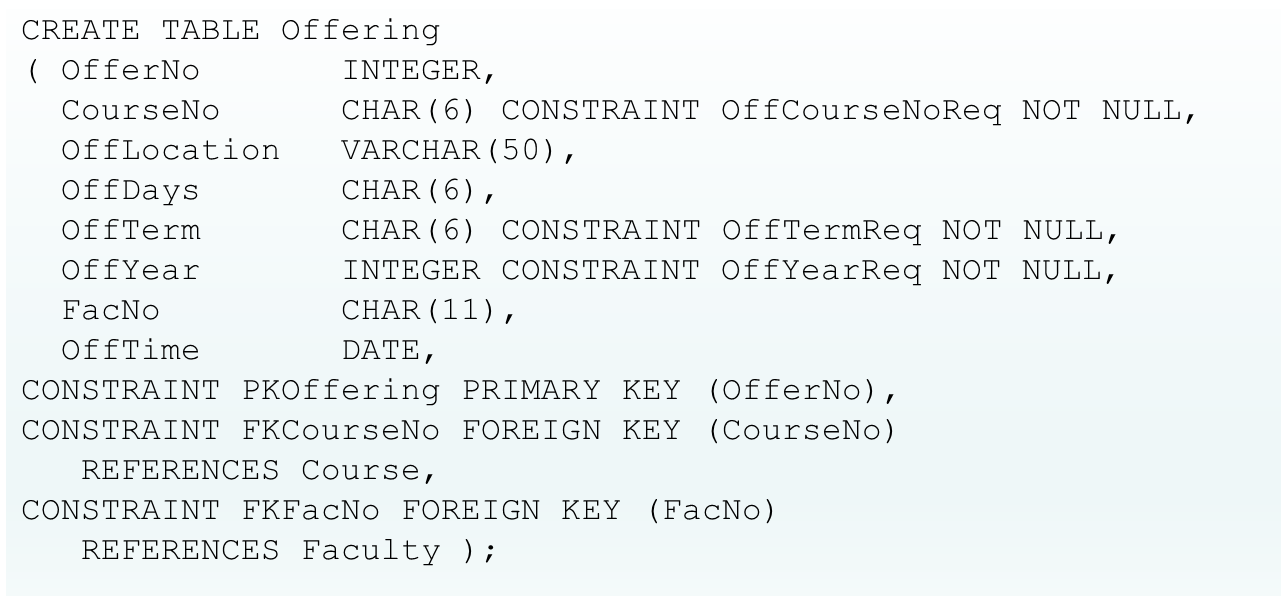
Foreign key constraints:

- OfferNo references Offering

- StdNo references Student

#### Inline Constraint Placement

The CREATE TABLE statement for the offering table shows three inline constraints and three external constraints. Required constraints with a not null key words are typically placed inline because they always involve individual columns. The constraint keyword and constraint name are optional for inline constraints. Inline constraints are written after the data type specification. The comma is necessary after the inline constraint to terminate the column definition. The three external constraints are similar to external constraints in other examples that you've seen. Check constraints, with a check keyword, define conditions that columns must satisfy. Conditions typically fall a column name, a relational operator such as greater than, and a constant, or another column name. If more than one column name appears in a check constraint, all columns must be from the table defined in the CREATE TABLE statement.



NOT NULL keywords

* Should use constraint names even for inline constraints
* Inline constraints associated with a specific column
* Easy to trace error when a constraint violation occurs

Two foreign keys:

- CourseNo: nulls not allowed

- FacNo: nulls allowed; prepare catalog before instructors are assigned; permits flexibility

#### Check Constraint Examples

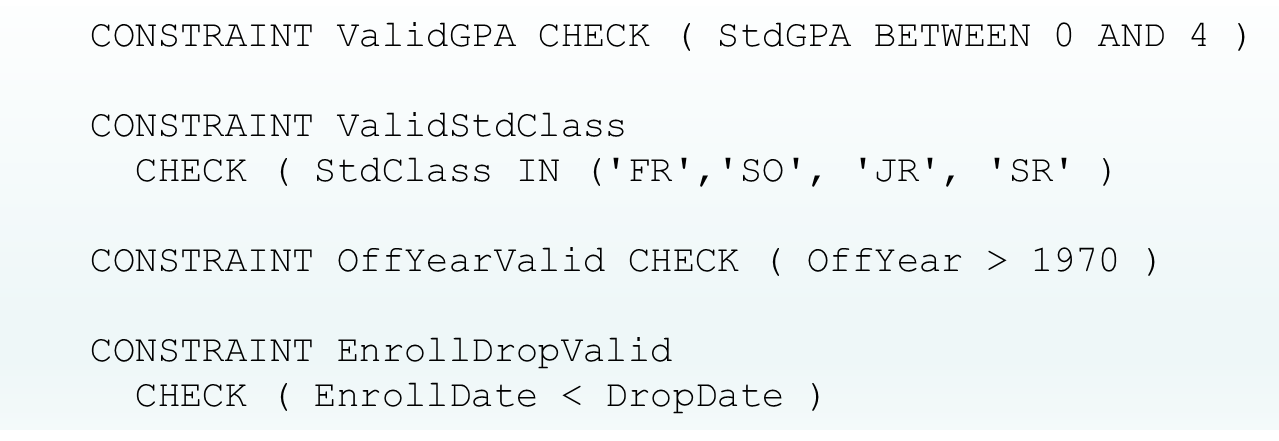
These examples depict four check constraints. The first three check constraints can be placed in line or external. The fourth check constraint must be placed external, because it involves multiple columns. Each check constraint contains the constraint keyword, constraint name, and condition inside of the parentheses.

The first CHECK constraint involves the between operator indicating that the STD GPA column must be between the values of zero and four inclusive.

The second example uses the in operator indicating that the student class column must be one of the four specified values.

The last constraint shows a condition involving two columns.

Both columns must come from the table defined in the CREATE TABLE statement. SQL compilers will not permit check constraints that involve columns from a table different than a table defined by the CREATE TABLE statement.



CHECK constraints are typically external.

First four examples show single column conditions.

Last example shows a constraint among two columns. Both columns must come from the same table. Enrollment date before drop date.

### Summary

Lets wrap up this lesson, that provided examples of constraint specification in the CREATE TABLE statement. The examples in this lesson, demonstrated primary key, foreign key, unique, required, and check constraints. Most examples were shown for external constraint placement. Required constraint examples replaced in line in the examples as typically occurs in practice. It is good practice to use constraint names to facilitate problem solving when a constraint violation occurs. A DBA on call will be helped at least initially by seeing a meaningful constraint name when a violation occurs such as a non-unique primary key value. You should also note the limitations of CHECK constraints. Only columns in the table in the CREATE TABLE statement can be used in a CHECK constraint. If you're using MySQL for the practice problems and graded problems, you should study the document on the course website containing MySQL CREATE TABLE statements. MySQL does not permit the constraint keyword in constraint names for required and checked constraints. Oracle however does not have these restrictions.

* Importance of PK and FK constraints
* Use constraint names
* CHECK constraint limitations
* MySQL syntax limitations

PK constraints ensure traceability of business entities such as customers.

FK constraints ensure connections among business entities are valid.

Constraint names help identify nature of data integrity violation. Helps specialist resolve problem more timely.

CHECK constraint limitations

* Ensure efficient execution
* Columns of the same table
* Constraints involving multiple tables cannot be done in CREATE TABLE statement (triggers instead)

CREATE TABLE statement important because of relative portability.

## Practice Problems for Module 3

The *Location* table contains several locations inside facilities. The primary keys of the tables are *CustNo* for *Customer*, *FacNo* for *Facility*, and *LocNo* for *Location*

**Customer**



***Facility***

******

***Location***



1. Write a CREATE TABLE statement for the *Customer* table. Choose data types appropriate for the DBMS used in your course. All columns are required (not null).

**Oracle**

CREATE TABLE Customer

( Custno varchar (4) CONSTRAINT CustnoNOTNULL NOT NULL,

CustFirstName varchar(50) CONSTRAINT CustnameNOTNULL NOT NULL,

CustLastName varchar(50) CONSTRAINT CustnameNOTNULL NOT NULL,

Adress varchar(50) CONSTRAINT AdressNOTNULL NOT NULL,

Internal varchar(3) CONSTRAINT InternalNOTNULL NOT NULL,

Contact varchar(50) CONSTRAINT ContactNOTNULL NOT NULL,

Phone integer CONSTRAINT PhoneNOTNULL NOT NULL,

City varchar(50) CONSTRAINT CustcityNOTNULL NOT NULL,

State varchar(2) CONSTRAINT StateNOTNULL NOT NULL,

Zip varchar(10) CONSTRAINT ZipNOTNULL NOT NULL,

CONSTRAINT PK\_Customer PRIMARY KEY(Custno) );

**MySQL**

CREATE TABLE Customer

( Custno varchar (8) CONSTRAINT NOT NULL,

CustFirstName varchar(50) CONSTRAINT NOT NULL,

CustLastName varchar(50) CONSTRAINT NOT NULL,

Adress varchar(50) CONSTRAINT NOT NULL,

Internal varchar(3) CONSTRAINT NOT NULL,

Contact varchar(50) CONSTRAINT NOT NULL,

Phone integer CONSTRAINT NOT NULL,

City varchar(50) CONSTRAINT NOT NULL,

State varchar(2) CONSTRAINT NOT NULL,

Zip varchar(10) CONSTRAINT NOT NULL,

CONSTRAINT PK\_Customer PRIMARY KEY(Custno) );

1. Write a CREATE TABLE statement for the *Facility* table. Choose data types appropriate for the DBMS used in your course. All columns are required (not null).

**oracle**

CREATE TABLE Facility (

FacNo varchar(8) CONSTRAINT FacNoNOTNUL NOT NUL,

FacName varchar(50) CONSTRAINT FacNameNameNOTNULL NOT NULL,

CONSTRAINT PK\_Facility PRIMARY KEY(Facno)

)

**Mysql**

CREATE TABLE Facility (

Facno varchar(8) CONSTRAINT NOT NUL,

FacName varchar(50) CONSTRAINT NOT NULL,

CONSTRAINT PK\_Facility PRIMARY KEY(Facno)

)

1. Write a CREATE TABLE statement for the *Location* table. Choose data types appropriate for the DBMS used in your course. *LocName* column is required (not null).

**ORACLE**

CREATE Location table (

Locno varchar(8) CONSTRAINT LocnoNOTNULL NOTNULL,

Facno varchar(8),

Locname varchar(50) CONSTRAINT LocnameNOTNULL NOT NULL,

CONSTRAINT PK\_Location PRIMARY KEY(Locno)

)

**MYSQL**

CREATE Location table (

Locno varchar(8) CONSTRAINT NOTNULL,

Facno varchar(8),

Locname varchar(50) CONSTRAINT NOT NULL,

CONSTRAINT PK\_Location PRIMARY KEY(Locno)

)

1. Identify the foreign key(s) and 1-M relationship(s) among the Customer, *Facility*, and *Location* tables. For each relationship, identify the parent table and the child table.

We have one relationship between Facility and Location:

1-M relationship: **Parent** Facility Facno (PK)- **Child** Location Facno(Fk)

1. Extend your CREATE TABLE statement from problem (3) with referential integrity constraints.

**ORACLE**

CREATE Location table (

Locno varchar(8) CONSTRAINT LocnoNOTNULL NOTNULL,

Facno varchar(8),

Locname varchar(50) CONSTRAINT LocnameNOTNULL NOT NULL,

CONSTRAINT PK\_Location PRIMARY KEY(Locno),

CONSTRAINT FK\_Facno FOREINGN KET (Facno) REFERENCES Facility(Facno)

)

**MYSQL**

CREATE Location table (

Locno varchar(8) CONSTRAINT NOTNULL,

Facno varchar(8),

Locname varchar(50) CONSTRAINT NOT NULL,

CONSTRAINT PK\_Location PRIMARY KEY(Locno),

CONSTRAINT FK\_Facno FOREINGN KET (Facno) REFERENCES Facility (Facno)

)

1. From examination of the sample data and your common understanding of scheduling and operation of events, are null values allowed for the foreign key in the *Location* table? Why or why not? Extend the CREATE TABLE statement in problem (5) to enforce the null value restrictions if any.

**Yes, nulls are allowed. Because there is no *not null* CONSTRAINT for Facno foreigh key in Location table**

**ORACLE**

CREATE Location table (

Locno varchar(8) CONSTRAINT LocnoNOTNULL NOTNULL,

Facno varchar(8) CONSTRAINT FacnoNOTNULL NOT NULL,

Locname varchar(50) CONSTRAINT LocnameNOTNULL NOT NULL,

CONSTRAINT PK\_Location PRIMARY KEY(Locno),

CONSTRAINT FK\_Facno FOREINGN KET (Facno) REFERENCES Facility(Facno)

)

**MYSQL**

CREATE Location table (

Locno varchar(8) CONSTRAINT NOT NULL,

Facno varchar(8) CONSTRAINT NOT NULL,

Locname varchar(50) CONSTRAINT NOT NULL,

CONSTRAINT PK\_Location PRIMARY KEY(Locno),

CONSTRAINT FK\_Facno FOREINGN KET (Facno) REFERENCES Facility (Facno)

)

1. Extend your CREATE TABLE statement for the *Location* table (problem 6) with a unique constraint for *LocName*. Use a named constraint clause for the unique constraint.

**ORACLE**

CREATE Location table (

Locno varchar(8) CONSTRAINT LocnoNOTNULL NOTNULL,

Facno varchar(8) CONSTRAINT FacnoNOTNULL NOT NULL,

Locname varchar(50) CONSTRAINT LocnameNOTNULL NOT NULL,

CONSTRAINT PK\_Location PRIMARY KEY(Locno),

CONSTRAINT FK\_Facno FOREINGN KET (Facno) REFERENCES Facility(Facno),

CONSTRAINT UniqueLocName UNIQUE(LocName)

)

**MYSQL**

CREATE Location table (

Locno varchar(8) CONSTRAINT NOTNULL,

Facno varchar(8) CONSTRAINT FacnoNOTNULL NOT NULL,

Locname varchar(50) CONSTRAINT NOT NULL,

CONSTRAINT PK\_Location PRIMARY KEY(Locno),

CONSTRAINT FK\_Facno FOREINGN KET (Facno) REFERENCES Facility (Facno),

CONSTRAINT UniqueLocName UNIQUE(LocName)

)

## Extra Problems for Module 3

If you want some additional practice on the CREATE TABLE statement, you can work these problems. The solution document is available in the Module 3 area of the class website.

The problems use the *Customer*, *OrderTbl*, and *Employee* tables of the simplified Order Entry database. The *Customer* table contains clients who have placed orders. The *OrderTbl* contains basic facts about customer orders. The *Employee* table contains facts about employees who take orders. The primary keys of the tables are *CustNo* for *Customer*, *EmpNo* for *Employee*, and *OrdNo* for *OrderTbl*.

**Customer**

****** ***Employee***

******

***OrderTbl***

******

1. Write a CREATE TABLE statement for the *Customer* table. Choose data types appropriate for the DBMS used in your course. Note that the *CustBal* column contains numeric data. The currency symbols are not stored in the database. The *CustFirstName* and *CustLastName* columns are required (not null).

**ORACLE**

CREATE Customer table (

CustNo varchar(10) CONSTRAINT CustNoNOTNULL NOT NULL,

CustFirstName varchar(50) CONSTRAINT CustFirstNameNOTNULL NOT NULL,

CustLastName varchar(50) CONSTRAINT CustLastNameNOTNULL NOT NULL,

CustCity varchar(50),

CustState char(4),

CustZip varchar(10),

CustBal decimal(20,2),

CONSTRAINT PK\_Customer PRIMARY KEY (CustNo));

**MYSQL**

CREATE Customer table (

CustNo varchar(10) CONSTRAINT NOT NULL,

CustFirstName varchar(50) CONSTRAINT NOT NULL,

CustLastName varchar(50) CONSTRAINT NOT NULL,

CustCity varchar(50),

CustState char(4),

CustZip varchar(10),

CustBal decimal(20,2),

CONSTRAINT PK\_Customer PRIMARY KEY (CustNo)

)

1. Write a CREATE TABLE statement for the *Employee* table. Choose data types appropriate for the DBMS used in your course. The *EmpFirstName*, *EmpLastName*, and *EmpEMail* columns are required (not null).

**ORACLE**

CREATE Employee table

(

EmpNo varchar(10) CONSTRAINT EmpNoNOTNULL NOT NULL,

EmpFirstName varchar(50) CONSTRAINT EmpFirstNameNOTNULL NOT NULL,

EmpLastName varchar(50) CONSTRAINT EmpLastNameNOTNULL NOT NULL,

EmpPhone varchar(13)

EmpEmail varchar(50) CONSTRAINT EmpEmailNOTNULL NOT NULL,

CONSTRAINT PK\_Emloyee PRIMARY KEY(EmpNo)

);

**MYSQL**

CREATE Employee table

(

EmpNo varchar(10) CONSTRAINT NOT NULL,

EmpFirstName varchar(50) CONSTRAINT NOT NULL,

EmpLastName varchar(50) CONSTRAINT NOT NULL,

EmpPhone varchar(13)

EmpEmail varchar(50) CONSTRAINT NOT NULL,

CONSTRAINT PKEmloyee PRIMARY KEY(EmpNo)

)

1. Write a CREATE TABLE statement for the *OrderTbl* table. Choose data types appropriate for the DBMS used in your course. The *OrdDate* column is required (not null).

**ORACLE**

CREATE OrderTbl table

(

OrdNo varchar(10) CONSTRAINT OrdNoNOTNULL NOT NULL,

OrdDate date CONSTRAINT OrdDateNOTNULL NOT NULL,

CustNo varchar(30),

EmpNo varchar(30),

CONSTRAINT PK\_OrderTbl PRIMARY KEY(OrdNo)

)

**MYSQL**

CREATE OrderTbl table

(

OrdNo varchar(10) CONSTRAINT OrdNoNOTNULL NOT NULL,

OrdDate date CONSTRAINT OrdDateNOTNULL NOT NULL,

CustNo varchar(30),

EmpNo varchar(30),

CONSTRAINT PK\_OrderTbl PRIMARY KEY(OrdNo)

)

1. Identify the foreign keys and 1-M relationships among the *Customer*, *Employee*, and *OrderTbl* tables. For each relationship, identify the parent table and the child table.

1-M relationship:

Parent Customer (CustNo PK) – Child OrderTbl (CustNo FK)

Parent Employee (EmpNo PK) – OrderTbl (EmpNo FK)

1. Extend your CREATE TABLE statement from problem (3) with referential integrity constraints.

**ORACLE**

CREATE OrderTbl table

(

OrdNo varchar(10) CONSTRAINT OrdNoNOTNULL NOT NULL,

OrdDate date CONSTRAINT OrdDateNOTNULL NOT NULL,

CustNo varchar(30) CONSTRAINT CustNoNOTNULL NOT NULL,

EmpNo varchar(30) CONSTRAINT EmpNoNOTNUL NOT NULL,

CONSTRAINT PK\_OrderTbl PRIMARY KEY(OrdNo),

CONSTRAINT FK\_CustNo FOREINGEN KEY (CustNo) REFERENCES Customer(CustNo),

CONSTRAINT FK\_EmpNo FOREINGEN KEY(EmpNo) REFERENCES EMPLOYEE(EmptNo)

)

**MYSQL**

CREATE OrderTbl table

(

OrdNo varchar(10) CONSTRAINT NOT NULL,

OrdDate date CONSTRAINT NOT NULL,

CustNo varchar(30) CONSTRAINT NOT NULL,

EmpNo varchar(30) CONSTRAINT NOT NULL,

CONSTRAINT PK\_OrderTbl PRIMARY KEY(OrdNo),

CONSTRAINT FK\_CustNo FOREINGEN KEY (CustNo) REFERENCES Customer(CustNo),

CONSTRAINT FK\_EmpNo FOREINGEN KEY(EmpNo) REFERENCES EMPLOYEE(EmptNo)

)

1. From examination of the sample data and your common understanding of order entry businesses, are null values allowed for the foreign keys in the *OrderTbl* table? Why or why not? Extend the CREATE TABLE statement in problem (5) to enforce the null value restrictions if any.

FOR the Customer Reference Key the foreige key it is not allowed to be null, this is because there is need to know who is ordering. But null values are allowed for the employee reference, this is an order may not have an employee assigned yet.

**ORACLE**

CREATE OrderTbl table

(

OrdNo varchar(10) CONSTRAINT OrdNoNOTNULL NOT NULL,

OrdDate date CONSTRAINT OrdDateNOTNULL NOT NULL,

CustNo varchar(30) CONSTRAINT CustNoNOTNULL NOT NULL,

EmpNo varchar(30),

CONSTRAINT PK\_OrderTbl PRIMARY KEY(OrdNo),

CONSTRAINT FK\_CustNo FOREINGEN KEY (CustNo) REFERENCES Customer(CustNo),

CONSTRAINT FK\_EmpNo FOREINGEN KEY(EmpNo) REFERENCES EMPLOYEE(EmptNo)

)

**MYSQL**

CREATE OrderTbl table

(

OrdNo varchar(10) CONSTRAINT NOT NULL,

OrdDate date CONSTRAINT NOT NULL,

CustNo varchar(30) CONSTRAINT NOT NULL,

EmpNo varchar(30),

CONSTRAINT PK\_OrderTbl PRIMARY KEY(OrdNo),

CONSTRAINT FK\_CustNo FOREINGEN KEY (CustNo) REFERENCES Customer(CustNo),

CONSTRAINT FK\_EmpNo FOREINGEN KEY(EmpNo) REFERENCES EMPLOYEE(EmptNo)

)

1. Extend your CREATE TABLE statement for the *Employee* table (problem 2) with a unique constraint for *EmpEMail*. Use a named constraint clause for the unique constraint.

**ORACLE**

CREATE Employee table

(

EmpNo varchar(10) CONSTRAINT EmpNoNOTNULL NOT NULL,

EmpFirstName varchar(50) CONSTRAINT EmpFirstNameNOTNULL NOT NULL,

EmpLastName varchar(50) CONSTRAINT EmpLastNameNOTNULL NOT NULL,

EmpPhone varchar(13)

EmpEmail varchar(50) CONSTRAINT EmpEmailNOTNULL NOT NULL,

CONSTRAINT PK\_Emloyee PRIMARY KEY(EmpNo),

CONSTRAINT Unique EmpEmail UNIQUE(EmpEmail)

);

**MYSQL**

CREATE Employee table

(

EmpNo varchar(10) CONSTRAINT NOT NULL,

EmpFirstName varchar(50) CONSTRAINT NOT NULL,

EmpLastName varchar(50) CONSTRAINT NOT NULL,

EmpPhone varchar(13)

EmpEmail varchar(50) CONSTRAINT NOT NULL,

CONSTRAINT PKEmloyee PRIMARY KEY(EmpNo),

CONSTRAINT Unique EmpEmail UNIQUE(EmpEmail)

)

## Assignment 1 Notes

Welcome to Lesson five of module three on the Relational Data Model. I'm gonna start with a trivia question that you should be able to answer if you listened to the lesson carefully. What University's athletic department is the assignment database loosely derived? Lesson five provides guidance for the practice problems and graded problems in assignment one. Both objectives are related to your ability to write syntactically correct cray table statements, that satisfy the assignment requirements and execute the statements on Oracle or MySQL. Your learning objectives for this lesson are to carefully read the assignment one requirements, and study the documentation about the Intercollegiate Athletic Database, the database used for the practice problems and the graded assignment. This diagram depicts the work associated with events managed by an athletic department at a major university. The database and associated workflow were inspired by a student project in the 1990's while I was on the faculty of the University of Washington in Seattle. The Intercollegiate Athletic Database supports the scheduling and operation of events. Customers initiate event requests with the Intercollegiate Athletic Department. Events are sometimes scheduled several months in advance. The facility and date held are recorded on an event request. If an event request is denied, no additional action is taken. If an event request is approved one or more event plans are made. Typically event plans are made for the set up, operation, and clean up of an event. An employee is assigned to manage an event plan before the plan is executed. Initially, there may not be an assigned employee. An event plan consists of one or more event plan lines. In an event plan line, the resource, location, time, and number of resources are recorded. The oracle relational diagram depicts the tables and relationships in the Intercollegiate Athletic Database. The database contains eight tables and seven relationships. The event request table is the hub of the database. An event request represents an event scheduled at a facility. For example a basketball game may be scheduled at the gymnasium. Holding an event involves plans for the allocation of resources, including personnel and equipment. The event plan table contains plans for the set up, operation and clean up of an event. Each event plan is managed by an employee, but the employee's optional initially to event plan, because an event plan is typically created before the employee is assigned. The event plan line table contains the individual resources required in event plan. Resources are assigned to specific locations of a facility. For example, guards may be required at the gates of a football stadium. The major requirement in Assignment 1 is to write CREATE TABLE statements with both acceptable syntax and conformance to required elements. The requirements indicate the data types and constraints for each table. The practice problems involve several tables in a database to help you get started. You also need to install Oracle or MySQL and execute the given SQL INSERT statements to populate the tables after the tables are created. If you have access to an Oracle server, you need to install the SQL developer software, obtain an account on the Oracle server, but not need to install the Oracle server on your own computer. The assignment requirements include constrains for primary keys, foreign keys, required columns and check constraints. Names are necessary for at least primary key and foreign key constraints, because MySQL does not support constraint names, for required and check constraints. Constraint names are not necessary for required and check constraints in the assignment. If you're working with Oracle however, I suggest that you use names for all constraints. Note that the primary key constraint for the EventPlanLine table must be external as the primary key involves two columns. Let's wrap up Lesson 5, the last lesson in Module 3. Lesson 5 has provided details about the Intercollegiate Athletic database and the requirements in assignment 1. You need to study the document about the background of the Intercollegiate Athletic database and the assignment 1 document carefully. Module three has provided practical knowledge in skills about the relational data model and creating tables in a relational DBMS. The practice problems in assignment one allow you to apply your knowledge and skills. The best test of your grasp of this material. You will gain confidence and satisfaction by using either Oracle or MySQL to complete the practice problems in assignment one. The Intercollegiate Athletic Database is used on assignment two and practice problems in modules four and five. So your effort to understand the database on assignment one will be very useful in the next parts of the course. Now to address the trivia question posed in the beginning of the lesson. The student project on which the database is derived was done by a former student as part of a database course that I taught at the University of Washington during the 1990's. The assignment database is undoubtedly a simplification of a database required to manage events at a major university. But the database is about as large and complex as can be understood in an introductory database course.

### Assignment Details

The graded problems in Module 3 provide experience with the CREATE TABLE statement.  You should execute the statements using either Oracle or MySQL.

To facilitate grading, please number the SQL statements and format them neatly. You need to show the result tables. Indicate in the beginning of your document if you used Oracle or MySQL.

If you use Oracle, you will need to use the Oracle SQL Developer to connect to an Oracle server. If you use MySQL, you will need to use the MySQL Workbench to connect to a MySQL server.

### Basic CREATE TABLE Statement Requirements

You should use the table descriptions in the Intercollegiate Database background document. You must use the same table and column names as specified in the background document. Here is some advice about data type selections.

1. You should use standard SQL data types specified in the notes except for using VARCHAR2 (an Oracle data type) instead of VARCHAR for columns containing varying length character strings. For MySQL, you should use VARCHAR for variable length strings.
2. For primary key fields (CustNo, LocNo, EventNo, PlanNo, EmpNo, ResNo, and FacNo), use the VARCHAR (or VARCHAR2 in Oracle) data type with length 8.  For consistency, corresponding foreign keys (such as EventRequest.CustNo) should also be the same data type and length.
3. For Oracle, you should use the DATE data type for the columns involving dates or times. The EventPlanLine.TimeStart and EventPlanLine.TimeEnd columns will store both date and time data so you should use the DATE data type. In MySQL use the DATE data type for columns with just date details (date columns in the EventRequest and EventPlan tables) and DATETIME for columns with date and time details (time columns in the EventPlanLine table).
4. Use CHAR(1) for the Customer.Internal column as Oracle does not provide a BOOLEAN data type.  MySQL has the Boolean data type, but I suggest that you use CHAR(1) instead.

### Constraints

After writing the basic CREATE TABLE statements, you should modify the statements with constraints. The CONSTRAINT clauses can be either inline in a column definition or separate after column definitions except where noted. You should specify a meaningful name for each CONSTRAINT clause.

* For each primary key, you should specify a PRIMARY KEY constraint clause. For single column primary keys (CustNo, LocNo, EventNo, PlanNo, EmpNo, ResNo, and FacNo), the constraint clause can be inline or external. For multiple column primary keys (combination of PlanNo and LineNo), the CONSTRAINT clause must be external.
* For each foreign key, you should specify a FOREIGN KEY constraint clause. The constraint clauses can be inline or separate.
* Define NOT NULL constraints for all columns except eventplan.empno, EventRequest.DateAuth, EventRequest.BudNo, and EventPlan.Notes.  Make sure that you define NOT NULL constraints for the PK of each table. Because of MySQL syntax limitations for NOT NULL constraints (inline with no constraint name and no CONSTRAINT keyword), you should define inline NOT NULL constraints.
* Define a named CHECK constraint to restrict the eventrequest.status column to have a value of “Pending”, “Denied”, or “Approved”. You can use the IN operator in this constraint. In MySQL, the syntax does not allow the CONSTRAINT keyword and a constraint name for CHECK constraints. You should use the CHECK keyword followed the condition enclosed in parentheses.
* Define named CHECK constraints to ensure that the resource.rate and eventrequest.estaudience are greater than 0. In MySQL, you cannot use a constraint name and the CONSTRAINT keyword for CHECK constraints. In MySQL, the syntax does not allow the CONSTRAINT keyword and a constraint name for CHECK constraints. You should use the CHECK keyword followed the condition enclosed in parentheses.
* Define a named CHECK constraint involving EventPlanLine.TimeStart and EventPlanLineTimeEnd. The start time should be smaller (chronologically before) than the end time. This CHECK constraint must be external because it involves two columns. In MySQL, the syntax does not allow the CONSTRAINT keyword and a constraint name for CHECK constraints. You should use the CHECK keyword followed the condition enclosed in parentheses.

### Populating Tables

The course website contains a text file containing SQL INSERT statements to populate the tables. You need to create the tables before inserting rows in each table. You need to insert rows in parent tables before child tables that reference parent tables. The INSERT statements in the file are in a proper order for populating the tables.

### Initial CREATE TABLE Statements

To facilitate your work, you can use the CREATE TABLE statements you created in the practice assignment in module 03 for the Customer, Facility, and Location tables. Thus, you only need to write CREATE TABLE statements for the remaining five tables (ResourceTbl, Employee, EventRequest, EventPlan, and EventPlanLine). You still need to execute the CREATE TABLE statements to create all of the tables and the INSERT statements to populate all tables.

### Submission

The submission requirements involve CREATE TABLE statements and evidence that you executed the statements and created the tables in Oracle or MySQL. You will submit 5 documents with each document containing a CREATE TABLE statement and screen snapshot to indicate that you created the table in Oracle or MySQL. In each document, you should neatly format your CREATE TABLE statement so that it can be easily graded. You should use the same table and column names as specified in the ICA database background document. You should not put any identifying details about yourself in your submitted document. For the screen snapshot, you need to capture a screen showing most columns and rows of the populated table. You can use a feature of the Oracle or MySQL client to show the rows in a table. Alternatively, you can execute an SQL statement (for example, SELECT \* FROM ResourceTbl) to show the columns and rows.

You should submit the documents in this order.

* Employee
* ResourceTbl
* EventRequest
* EventPlan
* EventPlanLine

# WEEK III : Basic Query Formulation with SQL

Welcome to lesson 1 of module 4 on basic query formulation SQL. I'm gonna start with two questions for you. One a trivia question and the other an important question about the impact of standards commercially. Why is SQL sometimes pronounced sequel? How does compliance with the SQL standard compare to compliance with web standards such as HTML5, the markup language for web documents? Module 4 builds on the skills and knowledge in module 3. Module 3 emphasized concrete skills and knowledge about the relational data model, using the create table statement, and understanding the tables and relationships in existing databases. These skills are essential for developing query formulation skills that are emphasized in modules 4 and 5. This lesson provides your initial background about query formulation, so the learning objectives are modest. I want you to briefly explain three types of statements in SQL. As a more conceptual objective, I want you to reflect about the level of conformance to the SQL standard among DBMS vendors. Especially the issues of portablity versus vendor control and innovation. The Structured Query Language, SQL has evolved over 40 years as a language containing statements for database definition, control and manipulation. In Module 3, you used the create table statement. Modules 4 and 5 emphasize the select statement for data manipulation, the most important statement in SQL. The Structured Query Language has a colorful history. SQL began life as the SQUARE language in IBM system R project. The SQUARE Language was somewhat mathematical in nature. After conducting human factors experiments, the IBM research team revised the language and renamed it SEQUEL, a follow up to SQUARE, and then renamed it SEQUEL 2. Its current name, SQL, resulted from legal issues surrounding the name SEQUEL. Because of this naming history a number of database professionals, particularly those working during the 1970s pronounce the name as sequel rather than SQL. With the force of IBM behind SQL, many an imitators use some variant of SQL. It may seem surprising but IBM was not the first company to commercialize SQL. Until a standards effort developed in the 1980s, SQL was in a state of commercial confusion, with many variants implemented. The standards efforts by the American National Standards Institutes, the International Organization for Standards, and the International Electrotechnical Commission have restored some order. Although SQL was not initially the best database language developed, the standards efforts have improved the language, as well as standardized its specification. SQL can be used in two contexts, standalone and embedded. In the Stand-alone context, the user submits SQL statements with the use of a specialized editor, such as the SQL developer. In the embedded context, an executing program submits SQL statements, and the DBMS sends results back to the program. The program includes SQL statements along with statements of the host program language such as Java or Visual Basic. SQL was designed as a complete data language. Covering database definition, manipulation, and control. Definition statements create objects such as tables. Manipulation statements retrieve or change rows. Control statements ensure proper usage of a database, such as row integrity and security. Database administrators typically use most of the database definition and control statements you have already used to create statements in module 3. Module 4 covers basic aspects of the SELECT statement. Module 5 covers extensions to the SELECT statement, as well as the INSERT, UPDATE, and DELETE statements. The weakness of the SQL standards is the lack of conformance testing. Until 1996, The US Department of Commerce's National Institute of Standards and Technology conducted conformance tests to provide assurance that government software can be ported among conforming DBMSs. Since 1996, DBMS vendor claims have substituted for independent conformance testing. Even for core SQL, the major vendors lack support for some features and provide proprietary support for other features. With the optional parts conformance has much greater variance. Writing a portal SQL code requires careful study for core SQL. But it's not possible for optional parts of SQL. In comparison, compliance of web standards, such as HTML 5, is strong. HTML 5 is the markup language for web pages. The World Wide Web Consortium, the major governing body of the web, has a validation service. There is also web standards project with a variety of standard compliance test pages called Acid3. Let's wrap up this lesson giving you an initial background on the SQL standard. SQL is a broad language with many statements but weak conformance among DBMS vendors. Weak conformance means reduced portability across DBMSs and higher switching costs. The DBMS industry has more vendor power compared to the web environment. The DBMS industry may argue that weak conformance allows more vendor innovation at a modest cost of somewhat higher switching costs. Modules 4 and 5 will cover details of the SELECT statement and the query formulation process. You will need lots of practice to master query formulations and the SELECT statement.

CREATE table EMPLOYEE

( EMPNO varchar2(8) constraint EMPNONOTNULL not null,

EMPNAME varchar2(50) constraint EMPNAMENOTNULL not null,

DEPARTMENT Varchar2(30) Constraint DEPARTMENTNOTNULL Not Null,

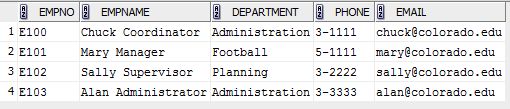
PHONE varchar(15),

EMAIL Varchar2(50) Constraint EMAILNOTNULL Not Null,

Constraint Pk\_Employee Primary Key (Empno),

CONSTRAINT UniqueEmpemail UNIQUE (email));

SELECT \* FROM EMPLOYEE;



Create Table Resourcetbl (

RESNO varchar2(8) constraint RESNONOTNULL not null ,

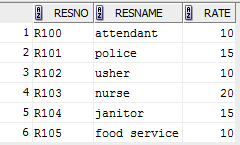
Resname Varchar2(50) Constraint ResnameNotNull Not Null,

Rate Integer Constraint RateNotNull Not Null,

constraint PK\_RESOURCETBL primary key(RESNO),

CONSTRAINT UniqueResname UNIQUE (Resname) );

SELECT \* FROM ResourceTbl;



Create Table Eventplan (

Planno Varchar2(8) Constraint Planno Not Null,

EventNo Varchar2(8) Constraint EventplanNoNOTNULL Not Null,

WorkDate date Constraint WorkDate Not Null,

Notes Varchar2(50),

Activity Varchar2(50) Constraint Activity Not Null,

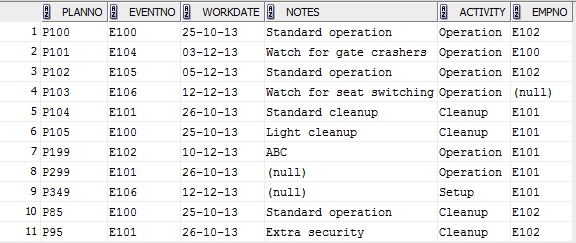
Empno Varchar2(8),

Constraint Pk\_Eventplan Primary Key(Planno,Eventno),

CONSTRAINT Fk\_EventNo FOREIGN KEY ( EventNo) References Eventrequest( EventNo)

);

SELECT \* FROM Eventplan;



Create Table Eventplanline (

PlanNo Varchar2(8) Constraint PlanNoNotNull Not Null,

LineNo INTEGER Constraint LineNoNotnul Not Null,

Timestart Date Constraint TimestartNotNull Not Null,

Timeend Date Constraint TimeendNotNull Not Null,

Numberfld Integer Constraint NumberfldNotNull Not Null,

Locno Varchar2(8) Constraint LinelocNoNotNull Not Null,

Resno Varchar2(8) Constraint LineResNoNotNull Not Null,

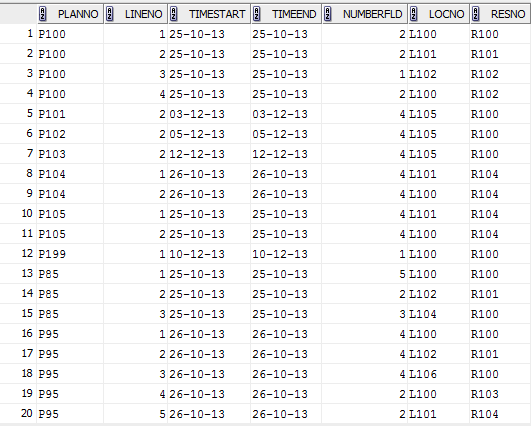
Constraint Pk\_Eventplanline Primary Key (Lineno,PlanNo),

constraint FK\_RESNO foreign key (RESNO ) references RESOURCETBL(RESNO),

Constraint Fk\_Locno Foreign Key (Locno ) References Location(LocNo ),

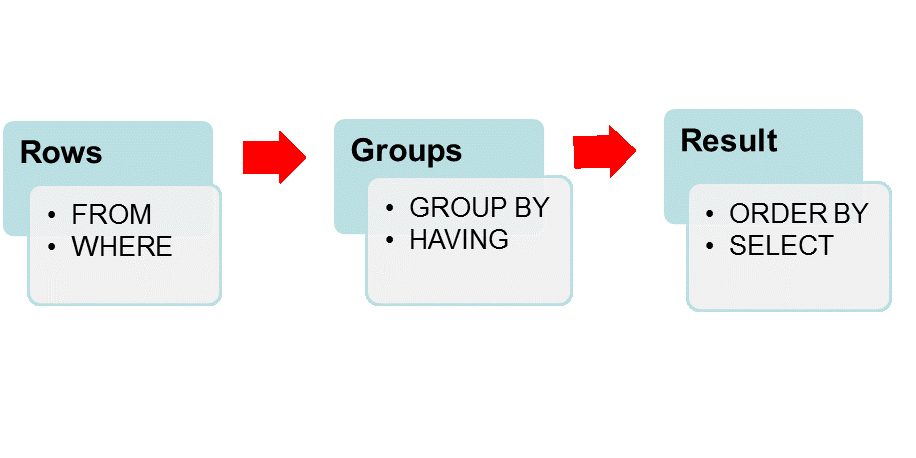
Constraint TimeValid Check (Timestart < Timeend) );

SELECT \* FROM Eventplanline;



## SQL Overview video lecture

Welcome to lesson 1 of module 4 on basic query formulation SQL. I'm gonna start with two questions for you. One a trivia question and the other an important question about the impact of standards commercially. Why is SQL sometimes pronounced sequel? How does compliance with the SQL standard compare to compliance with web standards such as HTML5, the markup language for web documents? Module 4 builds on the skills and knowledge in module 3. Module 3 emphasized concrete skills and knowledge about the relational data model, using the create table statement, and understanding the tables and relationships in existing databases. These skills are essential for developing query formulation skills that are emphasized in modules 4 and 5. This lesson provides your initial background about query formulation, so the learning objectives are modest. I want you to briefly explain three types of statements in SQL. As a more conceptual objective, I want you to reflect about the level of conformance to the SQL standard among DBMS vendors. Especially the issues of portablity versus vendor control and innovation. The Structured Query Language, SQL has evolved over 40 years as a language containing statements for database definition, control and manipulation. In Module 3, you used the create table statement. Modules 4 and 5 emphasize the select statement for data manipulation, the most important statement in SQL. The Structured Query Language has a colorful history. SQL began life as the SQUARE language in IBM system R project. The SQUARE Language was somewhat mathematical in nature. After conducting human factors experiments, the IBM research team revised the language and renamed it SEQUEL, a follow up to SQUARE, and then renamed it SEQUEL 2. Its current name, SQL, resulted from legal issues surrounding the name SEQUEL. Because of this naming history a number of database professionals, particularly those working during the 1970s pronounce the name as sequel rather than SQL. With the force of IBM behind SQL, many an imitators use some variant of SQL. It may seem surprising but IBM was not the first company to commercialize SQL. Until a standards effort developed in the 1980s, SQL was in a state of commercial confusion, with many variants implemented. The standards efforts by the American National Standards Institutes, the International Organization for Standards, and the International Electrotechnical Commission have restored some order. Although SQL was not initially the best database language developed, the standards efforts have improved the language, as well as standardized its specification. SQL can be used in two contexts, standalone and embedded. In the Stand-alone context, the user submits SQL statements with the use of a specialized editor, such as the SQL developer. In the embedded context, an executing program submits SQL statements, and the DBMS sends results back to the program. The program includes SQL statements along with statements of the host program language such as Java or Visual Basic. SQL was designed as a complete data language. Covering database definition, manipulation, and control. Definition statements create objects such as tables. Manipulation statements retrieve or change rows. Control statements ensure proper usage of a database, such as row integrity and security. Database administrators typically use most of the database definition and control statements you have already used to create statements in module 3. Module 4 covers basic aspects of the SELECT statement. Module 5 covers extensions to the SELECT statement, as well as the INSERT, UPDATE, and DELETE statements. The weakness of the SQL standards is the lack of conformance testing. Until 1996, The US Department of Commerce's National Institute of Standards and Technology conducted conformance tests to provide assurance that government software can be ported among conforming DBMSs. Since 1996, DBMS vendor claims have substituted for independent conformance testing. Even for core SQL, the major vendors lack support for some features and provide proprietary support for other features. With the optional parts conformance has much greater variance. Writing a portal SQL code requires careful study for core SQL. But it's not possible for optional parts of SQL. In comparison, compliance of web standards, such as HTML 5, is strong. HTML 5 is the markup language for web pages. The World Wide Web Consortium, the major governing body of the web, has a validation service. There is also web standards project with a variety of standard compliance test pages called Acid3. Let's wrap up this lesson giving you an initial background on the SQL standard. SQL is a broad language with many statements but weak conformance among DBMS vendors. Weak conformance means reduced portability across DBMSs and higher switching costs. The DBMS industry has more vendor power compared to the web environment. The DBMS industry may argue that weak conformance allows more vendor innovation at a modest cost of somewhat higher switching costs. Modules 4 and 5 will cover details of the SELECT statement and the query formulation process. You will need lots of practice to master query formulations and the SELECT statement.



Step 1: FROM clause (cross product and join operators)

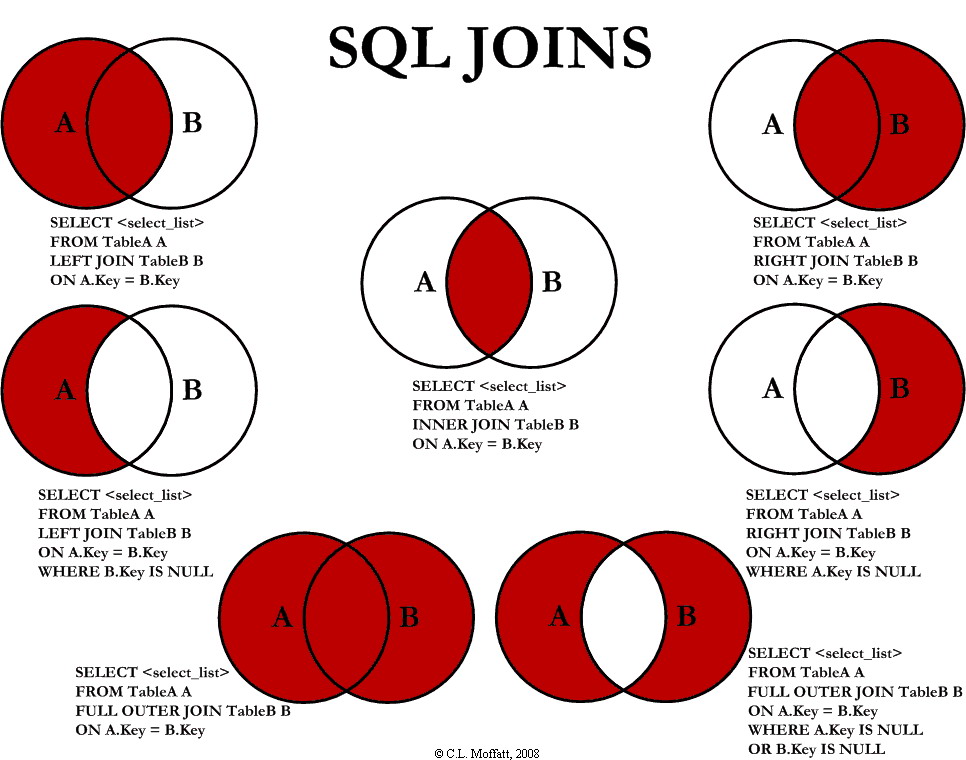
Step 2: WHERE clause (row conditions)

Step 3: GROUP BY clause (sort on grouping columns, compute aggregates)

Step 4: HAVING clause (group conditions)

Step 5: ORDER BY clause

Step 6: eliminate columns not in SELECT



# Week IV

### Lesson Objectives

* Gain context for other database development modules
* Explain goals of database development
* Explain the position of this module in the database development process

***Why is database development challenging and exciting?***

### Information System



Information system:

- Accepts data from its environment, processes data, and produces output data

for decision making

- Interacts with environment

- Database provides the long-term memory

- Database is a key component but not the only component

- Other components: inputs, outputs, processes, software, hardware, people

- Developing info system more than db development

Student Loan System:

- Environment: lenders, students, government agencies

### Data Bases

Databases are essential components of many information systems. The role of a database is to provide a long-term memory for an information system. The long-term memory contains entities and relationships.

For example, the database for a student loan system contains data about students, loans, and payments so that the statements, cash disbursements, and delinquency notices can be generated.

Databases are not the only components of information systems. Information systems also contain people, procedures, input data, output data, software and hardware. Thus, developing an information system involves more than developing a database.

However, in many development efforts for business information systems, database development is the most important activity. For business information systems the models for processes and environment interaction are usually created after the data model. Thus, the remaining modules in this course emphasize database development concept and skills. Broadly.

The goal of database development is to create a database that provides an important resource for an organization. To fulfill this broad goal, a database should:

* provide a vocabulary for a large community of users;
* support organizational policies;
* contain high-quality data;
* and provide efficient access.

Achieving a common vocabulary that facilitates communication is not easy. Developing a database requires compromise to satisfy a large community of users. In some sense, a good database designer shares some characteristics with a good politician.

A good politician often finds solutions with which everyone finds something to agree and unfortunately disagree.In establishing a common vocabulary, a good database designer also finds similar imperfect solutions. Forging compromises can be difficult. But the results can improve productivity, customer satisfaction, and other measures of organizational performance.

A database contains business rules to support organizational policies. Defining business rules is the essence of defining semantics or meaning of a database.

*For example, in an order entry system, an important rule is an order must precede a shipment.*

**A database can contain an integrity constraint to support this rule**.

### Importance of defining business rules

Defining business rules enables a database to actively support organizational policies. This active role contrasts with a more passive role that databases have in establishing a common vocabulary.

### Data base quality

The importance of data quality is analogous to the importance of product quality in manufacturing. Poor product quality can lead to loss of sales, lawsuits, and customer dissatisfaction. Because data are the product of an information system, data quality is equally important.

Poor data quality can lead to poor decision-making about communicating with customers, identifying repeat customers, tracking sales, and resolving customer problems. For example, communicating with customers can be difficult if addresses are outdated or customer names are inconsistently spelled on different orders. Even if the other design goals are met, a slow performing database will not be used.

Thus, finding in an efficient implementation is paramount. However, an efficient implementation should respect the other goals as much as possible. An efficient implementation that compromises the meaning of the database or data quality may be rejected by database users.

### Broad Goals of Database Development

A database provides a common vocabulary for an organization. Before a common database is implemented, different parts of an organization may have different terminology. For example, there may be multiple formats for addresses, multiple ways to identify customers, and different ways to calculate interest rates. After a database is implemented, communication can improve among different parts of an organization. Thus, a database can unify an organization by establishing a common vocabulary.

Define business rules to support organizational policies.

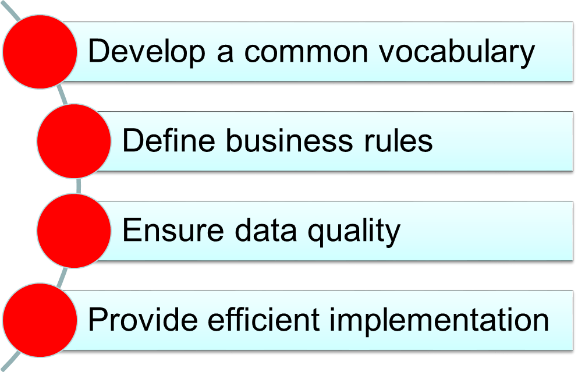
Ensure data quality

* Tradeoff of quality level and improved decision making
* Important for data warehouses for business intelligence
* Providing business rules achieves some level of quality
* Must provide ways to measure quality

Efficient implementation

* Overrides other concerns
* Not studied in this course
* Studied in the course about relational database support for data warehouses

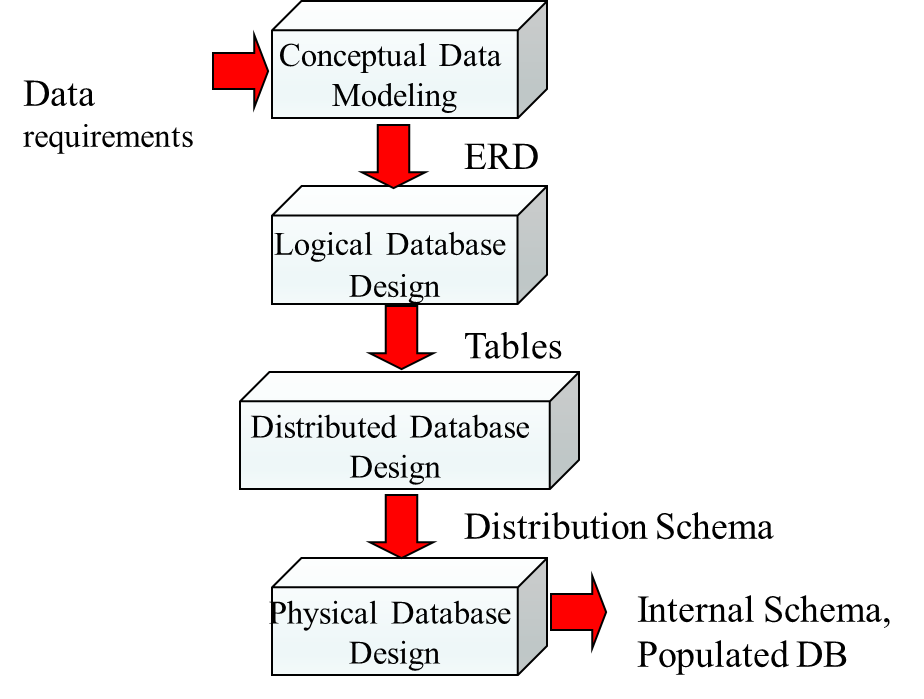
The goal of the database development process is to produce an operational database for an information system. To produce an operational database, you needed to find data models and populate that is supplied with data, the database.



This diagram shows important phases in the database development process. The first two phases are concerned with the information content of the database.

While the last two phases are concerned with efficient implementation.

### Database Development Phases



## Basic ERD notation

### Lesson Objectives

* Explain cardinality notation in an ERD
* Explain differences between ERD notation and relational database diagram
* Objectives: understand notation
* - Entity types, relationships, attributes
* - Cardinalities
* - Use notation precisely
* - Differentiate ERD notation from Oracle notation: slight differences
* Data modeling is challenging
* - Ambiguity: part science, part art
* - Opportunity for some creative problem solving
* - Emphasis on database design not database usage

Why have ERD standards not been developed and adopted

### Basic Symbols



#### Entity type:

- Collection of things of interest: persons, places, things, events

- Contains attributes: like columns

- Primary key

- Entity: instance or member of an entity type

#### Relationship:

- Named association among entities: name is significant (gives it more status)

- Similar to a foreign key in relational model except for name and cardinalities

- Usually between two entity types: can involve multiple entity types and one

- Bidirectional:

- Can be used to navigate in both directions

- Two names

- Course to Offering: Has, Provides

- Offering to Course: IsProvidedFor

- Which name to use: try to use active verb; not always possible

#### Attribute:

- Properties of entity types or relationships

- Data type to indicate the kind of values and permissible operations on the attribute

- Shown inside entity type or next to relationship

### the major differences between an ERD and a relational database diagram.

* ERDs have three basic elements. Entity types, relationships, and attributes.
* Entity types represent collections of physical things such as books, people, and places, as well as events such as payments.
* An entity is a member or instance of an entity type.
* In the Crow's Foot Notation, as well as in many other notations, rectangles denote entity types.
* In this ERD, the course entity type represents a set of courses in the database. Attributes are properties of entity types or relationships.
* An entity type should have a primary key as well as other descriptive attributes.
* Attributes are shown inside an entity type rectangle.
* Underlining indicates that the attribute or attributes serves as the primary key of the entity type.;
* Relationships are named associations among entity types.
* In the Crow's Foot Notation, relationship names appear in the line connecting the entity types involved in a relationship.
* In this ERD, the Has relationship shows that the Course and Offering entity types are directly related or connected.
* Relationships store associations in both directions. For example, the Has relationship shows the offerings for a given course and the associated course for a given offering. Informally,
* ERDs have a natural language correspondence.

Entity types can correspond to nouns in relationships to verbs or prepositional phrases connecting nouns. In this sense one can read an entity relationship diagram as a collection of sentences.



For example, this ERD can be read as course **has** offerings. You should use the natural language correspondence as a guide rather than as a strict rule.

***For large ERDs you will not always find a good natural language correspondence for all parts of a diagram***.

### Cardinalities

Cardinalities constrain the number of objects that participate in a relationship. To depict the meaning of cardinalities, an instance diagram is useful. This diagram shows a set of courses, Course 1, Course 2, Course 3, a set of offerings, Offering 1, Offering 2, Offering 3, Offering 4, and connections between the two sets. Course



Definition:

- a constraint on the number of entities that participate in a relationship

- Specify the minimum and maximum cardinalities in both directions

Instance diagram:

- Shows occurrences of entity types (entities)

- Useful to understand some relationships (similar to sample table usage)

- Lines show relationships among entities

- Course1 related to Offering1, Offering2, and Offering3

- Course2 related to Offering4

- Course3 not related to any offerings

- Course related to a minimum of 0 and maximum of many

- Offering: each related to exactly one course

### Cardinality Notation



Symbols:

- Oval: means 0

- Perpendicular line: means 1

- Crow's foot: means many (0 or more); unconstrained

- Some drawing tools support exact cardinalities (numbers)

Placement:

- Inside symbol: minimum cardinality

- Outside symbol: maximum cardinality

- Interpret the far cardinality symbols: near the other entity type

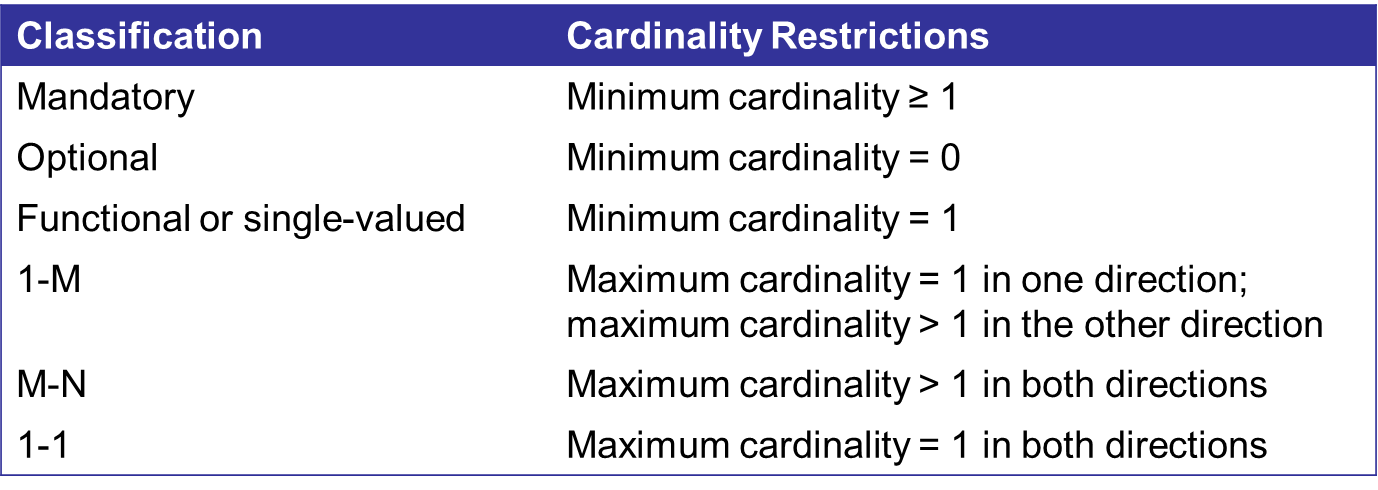
- Course is related to a min of 0 and max of many offerings

- Offering is related to a min of 1 and max of 1 courses (exactly one)

The Crow's Foot Notation uses three symbols to represent cardinalities. The crow's foot symbol, that is two angled lines and one straight line, denotes many, that's zero or more, related entities. In this ERD, the crow's foot symbol near the offering entity type means that a course can be related to many offerings. The circle means a cardinality of zero, while a line perpendicular to the relationship line denotes a cardinality of one. To depict minimum and maximum cardinalities, the cardinality symbols are placed adjacent to each entity type in a relationship.

The minimum cardinality symbol appears on the inside, toward the relationship name, while the maximum cardinality symbol appears on the outside, towards the entity type. In this ERD, a course is related to a minimum of zero offerings, that is a circle in the inside position, in a maximum many offerings, that is a crow's foot in the outside position. Similarly, an offering is related to exactly one, that is one and only one course, as shown by the single vertical lines in both inside and outside positions.

### Important Cardinalities



Cardinalities are classified by common values for a minimum and maximum cardinality. This table shows two classifications for minimum cardinalities.

A minimum cardinality of one or more indicates a mandatory relationship. For example, participation in the has relationship is mandatory for each offering, due to the minimum cardinality of one. A mandatory relationship makes the entity type existence dependent on the relationship. The offering entity type depends on the has relationship, because the offering entity cannot be stored without a related course entity.

In contrast, a minimum cardinality of zero indicates an optional relationship. For example, the has relationship is functional for offering, because an offering entity can be related to a maximum of one course.

The word function comes from mathematics, where a function gives one value.

A relationship that has a maximum cardinality of 1 in one direction, and more than 1, that's many, in the other direction, is called 1-M, read one to-many relationship. The has relationship shown in this lesson is one to many.

A relationship that has a maximum cardinality of more than 1 in both directions is known as an M to M, or many to many relationship. In this ERD the Team Teaches relationship allows multiple professors to join and teach the same offering. **Many to many relationships are common in business databases to represent the connection between parts and suppliers, authors and books, and skills and employees.**

For example, a part can be supplied by many suppliers, and a supplier can supply many parts. Less common are one to one relationships in which the maximum cardinality equals one in both directions. For example, the Works In relationship in this ERD, allows a faculty to be assigned to one office, and an office to be occupied by at most one faculty.



TeamTeaches:

- M-N

- Optional in both directions

WorksIn:

- 1-1

- Optional: office can be empty

- Mandatory: faculty must be assigned to an office

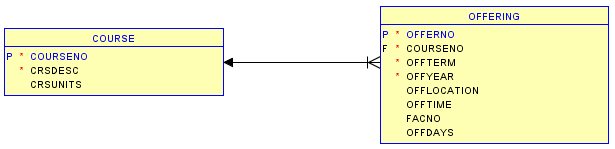
- 1-1 relationships are specialized.

### The differences between ERD notation and relational database diagrams.

ERD notation



Oracle Relational Model Diagram



Preference rather than one model being more powerful

Differences:

- Named relationships: no name in relational model (FKs instead)

- ERD does not need FKs: redundant with relationship

- Child side of relationship: M cardinality is implied by lack of FK. Minimum cardinality is not changeable. Course has a minimum of one offering is not enforced in the relational model.

- Different symbol on the parent side of a relationship: arrow instead of cardinality symbols.

- Solid line denotes a mandatory relationship in the relational model diagram, not id dependency.

- Relationships with attributes: not permitted in the relational model

- No M-N relationships in the relational model: must use associative table with FKs from 2 parent tables

## Relationship variations I

What is the difference between existence dependency and identification dependency?

These relationship variations do not occur often in practice. However, when occurring, these variations are important parts of a data model.

### Lesson Objectives

The objectives in this lesson involve two specialized types of relationships, identification dependency and many-to-many relationships with attributes.

* Explain an example involving identification dependency
* Apply relationship equivalency between M-N relationship and associative entity type
* Appreciate specialized relationships but resist temptation to overuse them

***Typical novice mistakes in data modeling are to overuse specialized relationships.***

In some business situations, an entity type may not have its own primary key. Entity types without their own primary key must borrow part or all of their primary key from other entity types. Entity types that borrow part or their entire primary key are known as weak entity types.

### Identification Dependency

A relationship that provides a component of the primary key is known as an identifying relationship. Thus, an identification dependency involves a weak entity type in one or more identifying relationships. Identification dependency occurs because some entities are closely associated with other entities.



For example, a room does not have an identity separate from its building, because a room is physically contained in a building. You can reference a room only by providing its associated building identifier. In the ERD for buildings and rooms, the room entity type is identification-dependent on the building entity type in the contains relationship. A solid relationship line indicates an identifying relationship.

For weak entity types, the underlying attribute, if present, is part of the primary key, but not the entire primary key. Thus, the primary key of room is a combination of BldgId from the building entity type and RoomNo from the room entity type.

Concept:

- Some entity types borrow part or entire PK

- Specialized concept: important when it occurs but not too common

- Similar to FK part of PK in relational model

- Closely related entities: physical containment

- Room is physically contained in a building

- Identification of room includes building

- Others: country-state, order-orderline

Symbols:

- Weak entity:

- Borrows part or all of PK

- Diagonal lines in the corners

- Identifying relationship:

- Solid line

- Indicates the source of PK

- Ambiguity if entity type participates in more than one relationship

Example:

- PK of Room is a combination of RoomNo (local key) and BldgID (borrowed attribute)

- Cardinality of weak entity in the identifying relationship must be 1-1

- Room cannot exist unless associated building exists

- Identification dependency involves existence dependency:

- Weak entity is existent dependent on other entity

- Also borrows part of all of PK

One of the differences between an ERD and a relational database diagram is that relationships in ERD can have attributes. This situation typically occurs with many-to-many relationships.

M-N Relationships with Attributes



In a many-to-many relationship, attributes are associated with a combination of entity types, not just one of the entity types.

Relationships are first class citizens

- Can have attributes just like entity types

- Most typical for M-N relationships

- Attribute depends on both entity types, not just one entity type

- 1-M relationships with attributes is controversial

Example:

- EnrGrade: grade recorded for a student in a particular course

- Depends on the combination of Student and Offering

-EnrGrade is not part of the Student or Offering entity types

M-N Relationships with Attributes (II)

AuthOrder: the order in which the author’s name appears in the title of a book

- Record order of authors: important in publishing disciplines

- AuthOrder is part of the Writes relationship (combination of Author and Book)

- AuthOrder is not part of the Author or Book entity types

Qty: quantity of part supplied by a supplier; quantity varies by part and supplier

If an attribute is associated with only one entity type, then it should be part of that entity type, not the relationship. In this ERD the attribute EnrGrade is associated with a combination of a student offering, not either one alone.

For example, the EnrolsIn relationship records the fact that student with student number 123779993 has a grade of 3.5 in the offering with offer number of 1256. To provide clarification about many-to-many relationships with attributes, these partial ERDs show more examples. In Example A, the attribute quantity, or Qty, represents the quantity of a part supplied by a given supplier. In Example B, the attribute AuthOrder represents the order in which the author's name appears in a title of the book.

### M-N Relationship Equivalency Rule

To improve your understanding of many-to-many relationships, you should know an important equivalence for many-to-many relationships. An M-N, or many-to-many relationship, can be replaced by an associative entity type and two identifying one-to-many relationships. This associative entity type style is similar to the representation in a relational database diagram with an associative table containing the combined primary key, consisting of two foreign keys.

* Replace M-N relationship
  + Associative entity type
  + Two identifying 1-M relationships
* M-N relationship versus associative entity type

Some ERD notations do not directly support M-way relationships such as the Crow’s Foototation

Replace with associative entity and M identifying 1-M relationships

If you feel more comfortable with the one-to-many style, then use it. In terms of the ERD, the many-to-many and one-to-many styles have the same meaning.

Relationship Equivalency Example





As an example of the many-to-many relationship equivalency role, the EnrolsIn relationship is replaced by two identifying relationships, registers and grants, and an associative entity type enrollment. The relationship name EnrolsIn has been changed to a now enrollment to follow the convention for nouns for entity type names. There is one situation when the one-to-many style is preferred to many-to-many style. When a many-to-many relationship must be related to other entity types and relationships, you must use the one-to-many style. Since this situation is not common, the choice of a many-to-many relationship or an associative entity type is largely a matter of preference.

## Relationship Variations II

The relationship variations covered in Lesson 4 do not occur often in practice. However, when occurring, these variations are important parts of a data model. The objectives in this lesson involve two specialized types of relationships, self-identifying relationships and M-way relationships.

The most important objective is to appreciate specialized relationships but avoid temptation to overuse them.

Typical novice mistakes in data modeling are to overuse specialized relationships.

### ERD Notation for Self-Referencing Relationships

A self-referencing relationship involves connections among members of the same set. Self-referencing relationships are sometimes called reflexive relationships, because they are like a reflection in a mirror. Relationship connects entity type to itself, the position of cardinalities is not important: relationship involves the same entity type. Otherwise, nothing is different about self-referencing relationships

Key point:

- 1-M vs. M-N

- Use instance diagrams to help you reason



This partial ERD shows two self-referencing relationships involving the faculty and course entity types. Both relationships involve two entity types that are the same, faculty for supervises and course for prerequisite tool. Note both relationships start and terminate with the same entity type. These relationships depict important concepts in the university database.

The supervises relationship depicts an organizational chart, while the prerequisite relationship depicts course dependencies that can affect a student's course planning. Self-referencing relationships occur in a variety of business situations.

Any set that can be visualized as a hierarchy can be represented as a self-referencing relationship. Typical examples include hierarchical charts of accounts, genealogical charts, part designs, and transportation routes.

In these examples, self-referencing relationships are an important part of the database. For self-referencing relationships, it is important to distinguish between one-to-many and many-to-many relationships.

### Instance Diagrams for Self-Referencing Relationships

An instance diagram can help you understand the difference. Basic idea:

Associations among members of the same set, Employees: supervisory relationships; Courses: prerequisite structures, Specialized concept: important when occurs but not too common

Instance diagram:

Useful to depict self-referencing relationships; 1-M self-referencing: at most one upward connection (traditional org chart); M-N self-referencing: more than one upward connection; IS461 has multiple prerequisites (IS480 and IS460); IS320 is the prerequisite for multiple courses

The hierarchy in Part A shows an instance diagram for the supervises relationship. Notice that each faculty has at most one superior.



For example, Faculty 2 and Faculty 3 have Faculty 1 as a superior. Therefore, supervises is a one-to-many relationship, because each faculty can't have at most one supervisor.

In contrast, there is no such restriction in the instance diagram for the prerequisite relationship shown in Part B. For example, IS461 has two prerequisites, IS480 and IS460, while IS320 is a prerequisite to both IS480 and IS460.

### 1.1.1 Associative Entity Types for M-way Relationships

Therefore, PreReqTo is a many-to-many relationship, because a course can be a prerequisite to many courses, and a course can have many prerequisites. An M-way relationship involves an association of more than two entity types.

The crow's foot notation only supports binary relationships, so M-way relationships cannot be directly represented. Instead, an M-way relationship is indirectly represented as an associated entity type and a collection of one-to-many relationships.

The typical value for M is three, as relationships involving more than three entity types are very rare in practice. Three-way relationships are not common practice either but important when occurring. In this ERD, three one-to-many relationships link the associative entity type uses to the part, supplier, and project entity types.

The uses entity type is associative, because its role is to connect other entity types. Because associative entity types provide a connecting role, they are sometimes given names using active verbs. In addition, associative entity types are always weak as they must borrow the entire primary key. For example, the uses entity type obtains its primary key through the three identifying relationships.

The issue of when to use an M-way associative entity type can be difficult to understand. If a database only needs to record pairs of facts, an M-way associative entity type is not needed. For example, if a database only needs to record who supplies a part and what projects use a part, then the M-way associative entity type should not be used. In this case, there should be binary relationships between supplier and part and between project and part. You should use an M-way associative entity type when the database should record combinations of three or more entities rather than just combinations of two entities.



For example, if the database needs to record which supplier provide parts on specific projects, an M-way associative type is needed. Now, let's wrap up Lesson 4 about relationship variations for ERDs. This lesson covered two important variations, self-identifying relationships and M-way relationships. Self-referencing relationships support business requirements in which members of a set are represented as a hierarchy.

Typical examples include hierarchical charts of accounts, genealogical charts, part designs, and transportation routes. An M-way relationship represents a business situation involving a combination of more than two entities. Because the crow's foot notation only represents binary relationships in a social entity type in a collection of one-to-many, identifying relationships are used.

These relationships are not common in practice but important when they occur. However, typical novice mistakes are to overuse specialized relationships, so make sure you understand them before using them in practice. M-way relationships are especially error-prone because of the complexity of the business situations involving more than two entities.

## Exercise Problems for Module 6

The exercise problems in Module 6 provide practice using the Crow’s Foot notation. You are encouraged to use the ER *Assistant* or other drawing tool to complete the exercise problems in module 6. If you want to use the ER Assistant, I encourage you to watch at least the first part of the software demonstration about the ER Assistant, available in module 7.

1. Draw an ERD containing *Student* and *Paper* entity types connected by a 1-M relationship. The *Student* entity type should have attributes for *StdNo* (primary key), *StdFirstName*, *StdLastName*, *StdAdmitSemester*, *StdAdmitYear*, and *StdEnrollStatus* (full or part-time). The Paper entity type should have attributes for *PaperNo* (primary key), *PaperTitle*, *PaperSubmitDate*, *PaperAccepted* (yes or no), and *PaperType* (first, second, proposal, or dissertation). Add a 1-M relationship from *Student* to *Paper*.
2. Extend the ERD with an *Evaluator* entity type and an M-N relationship between *Paper* and Evaluator. The Evaluator entity type should have attributes for *EvalNo* (primary key), *EvalFirstName*, *EvalLastName*, *EvalEmail*, and *EvalOffice*. The M-N relationship should have attributes for *EvalDate*, *EvalLitReview* (1 to 5 rating), *EvalProbId* (1 to 5 rating), *EvalTechWriting* (1 to 5 rating), *EvalModelDev* (1 to 5 rating), *EvalOverall* (1 to 5 rating), and *EvalComments*.
3. Transform the M-N relationship from problem 9 into an associative entity type and identifying relationships.